百度智能云

AIHC 文档



【版权声明】

版权所有©百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司。 未经本公司书面许可，任何单位和个人不得擅自摘抄、复制、传播本文档内容，否则本公司有权依法追究法律责任。

【商标声明】



和其他百度系商标，均为百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司的商标。 本文档涉及的第三方商标，依法由相关权利人所有。未经商标权利人书面许可，不得擅自对其商标进行使用、复制、修改、传播等行为。

【免责声明】

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导。 如您购买本文档介绍的产品、服务，您的权利与义务将依据百度智能云产品服务合同条款予以具体约定。本文档内容不作任何明示或暗示的保证。

目录

目录 2
功能发布记录 10
产品描述 17
介绍 17
核心概念 18
产品优势 18
使用限制 19
应用场景 19
快速入门 19
快速开始体验百舸核心功能 19
快速创建大模型训练任务 21
快速部署推理服务 26
Qwen3.8-Flash-Next 31
DeepSeek-V4-Pro-0813 31
Kimi-K3快速开始卡片描述 32
GLM-5.3-Flash 33
GLM-5.3 34
DeepSeek-V4.1-Flash 35
产品定价 36
计费概述 36
计费方式 36
操作指南 37
快速开始 37
概述 37
Nvidia Isaac Lab 快速部署 38
Nvidia Isaac Sim 快速部署 41
ManiSkill 快速部署 43
LeRobot 数据格式转换 44
Isaac Sim 4.5.0 可视化实践 45
Robotwin 2.0 快速实践 57
Isaac Sim 可视化 61
Isaac Lab遥操环境搭建和图形可视化 74
Apollo-BEV-Train 78
Isaac工作流--动作数据合成&扩增、模仿学习 81
Apollo-Model-Deployment 85
Apollo-BEV-Model-Export 88
BEVFormer-加速版 88
BEVFusion加速版 90
Isaac工作流--使用 GR00T-Dreams 生成训练数据 92
快速部署 OpenClaw（Clawdbot） 94

RLinf：pi0.5模型的强化学习训练与评估 101
LingBot-Depth训推 107
LingBot-World训推 112
LingBot-VLA训推 117
LingBot-VA训推 121
GEAR-SONIC在开发机中训练 125
DreamZero在开发机中训练 127
WBC-AGILE在开发机中训练 129
FastWAM模型训推 132
RLinf v0.3 基于 LIBERO 的训练 135
具身智能-Isaac Lab 3.0 141
在百舸上使用lingbot-va加速镜像训练视觉-动作模型 146
在百舸上使用 openpi (JAX) 加速镜像训练 pi0.5 模型 148
在百舸上使用 Motus 加速镜像训练 VLA 模型 150
在百舸上使用 WSA 加速镜像训练 VLA 模型 152
在百舸上使用 Mind2Act 评测镜像评测视觉-动作模型 157
在百舸上使用 SparseBEV++ 加速镜像训练 3D 目标检测模型 160
在百舸上使用 Wall-X 加速镜像训练 VLA 模型 162
全托管资源池 164
创建全托管资源池 164
管理全托管资源池 166
创建和管理资源队列 171
资源队列管理详情 172
CCR镜像免密拉取功能 176
碎片治理 176
安全组规则 178
GPU虚拟化和MIG虚拟化 179
队列空闲资源出借 181
调度策略与超时重排 183
负载类型配置与使用 186
资源报表 188
模型缓存加速配置 190
资源池监控 191
资源规格模板 193
RapidFS Master托管实例缓存加速 195
资源队列CPU节点弹性伸缩 199
开发机系统盘持久化配置 203
队列子用户资源配额管理 205
查看资源池、队列、节点操作云审计记录 207
资源池产品事件 210
删除资源池 212

自运维资源池 213
创建自运维资源池 213
管理自运维资源池 215
创建节点 216
管理节点 216
查看监控大盘 219
前提条件 222
前提条件 224
镜像本地缓存管理 227
故障自愈 228
队列管理 230
碎片治理 232
开启GPU显存共享 234
CCR镜像免密拉取功能 236
资源规格模板 236
开发机 238
创建开发机 238
管理开发机 239
网络配置 241
存储挂载与使用 249
制作开发机镜像 253
模型开发环境 256
开发机监控 260
在开发机中将模型部署为在线服务 261
开发机拉取海外镜像 262
在开发机中安装工具引导 263
分布式训练 266
训练任务生命周期 266
创建分布式训练任务 267
管理分布式训练任务 268
查看训练详情 270
训练框架说明 270
配置训练任务的消息通知 277
使用闲时资源 277
训练任务优先级管理 278
配置训练任务低利用率自动停止策略 279
在训练任务中使用云存储 280
训练任务可观测 281
训练容错 290
RDMA的使用 291

训练任务Hang检测 295
使用百舸智能助手进行排队任务的诊断排查 298
使用百舸智能助手进行失败任务的故障排查 299
BCCL：百度自研高性能集合通信库 299
FlashCkpt：大模型训练高性能ckpt保存 301
通用环境变量列表 303
在线服务部署 304
在线服务部署 304
在线服务部署 304
多服务流量分流 307
服务自动扩缩容 308
自定义监控及扩缩容指标 309
日志查询及采集 312
AI调用示例 314
推理参数说明 315
服务性能压测 316
在线调试 320
健康检查 321
滚动更新与优雅退出 322
工作流 322
概述 322
创建工作流 323
使用YAML定义工作流 324
运行工作流 332
开发任务模板 334
工作流模板示例 345
预置工作流模板 349
AI数据处理 388
算子列表 388
简介 388
AIHC-Daft算子开发使用指南 585
数据集管理 591
创建数据集 591
管理数据集 592
导入数据到数据集 593
管理数据集存储容量 596
公共数据集 597
自定义 Lance 格式数据集 608
模型管理 612
模型创建及管理 612
权限管理 612
权限概述 612

配置IAM预置权限策略 615
配置IAM自定义权限策略 616
配置预置RBAC权限策略 617
预留实例券 618
概述 618
预留实例券创建 619
预留实例券续费 620
模型评测 622
概述 622
创建评测 622
评测详情 & 结果对比 623
创建自定义推理镜像的评测任务 624
具身评测自定义推理镜像接口说明 625
湖管理 626
湖管理快速入门 626
湖管理权限管理 629
表管理 632
API参考 639
API功能更新记录 639
概述 639
通用说明 640
服务域名 646
请求头和错误返回 646
全托管资源池相关接口 648
资源池 648
队列 652
节点 660
自运维资源池相关接口(随产品下线中) 661
资源池 661
队列 664
节点 670
分布式训练相关接口 671
创建训练任务 671
查询训练任务列表 673
查询训练任务详情 676
删除训练任务 693
更新训练任务 694
查询训练任务事件 694
查询训练任务日志 695
查询训练任务Pod事件 696
停止训练任务 696

查询训练任务监控 696
查询训练任务所在节点列表 697
获取训练任务WebTerminal地址 697
批量停止训练任务 698
错误码 699
在线服务部署相关接口 699
创建服务 699
拉取服务列表 702
获取服务状态 704
查询服务详情 705
升级服务 707
服务扩缩容 709
配置公网访问 710
拉取服务变更记录 710
查询服务变更详情 711
删除服务 712
拉取服务pod列表 712
摘除Pod流量 713
删除Pod并重建 714
获取实例组列表 714
错误码 716
查询推理服务日志 716
获取推理服务WebTerminal地址 717
创建多角色服务 717
拉取多角色服务列表 719
查询多角色服务详情 720
删除多角色服务 721
查询多角色服务访问信息 722
开发机相关接口 723
创建开发机 723
变更配置 726
删除开发机 728
查询开发机详情 729
查询开发机列表 730
查询开发机事件 731
开启实例 733
停止实例 734
定时停止实例 734
制作开发机镜像 735
查询镜像任务详情 735
错误码 736

数据集相关接口 736
创建数据集 736
删除数据集 737
修改数据集 738
获取数据集详情 738
获取数据集列表 739
获取数据集版本详情 740
获取数据集版本列表 741
创建数据集版本 741
删除数据集版本 742
模型管理相关接口 742
创建模型 742
删除模型 743
修改模型 743
获取模型详情 744
获取模型列表 744
获取模型版本详情 745
获取模型版本列表 746
新建模型版本 746
删除模型版本 747
工作流相关接口 747
查询工作流模版详情 747
查询工作流运行列表 748
创建工作流模版 749
更新工作流模版 750
删除工作流运行实例 751
停止工作流运行 752
运行工作流 752
重试工作流运行 753
数据结构 753
模型评测相关接口 772
查询评测集的默认配置 772
查询模型的默认配置 773
查询可评测的模型及其评测集 774
查询回放视频播放地址 775
查询评测结果 776
删除评测任务 778
创建评测任务 778
查询评测任务列表 780
查询评测任务详情 782

停止评测任务 784
数据结构 784
SDK参考 788
GO SDK 788
概述 788
安装SDK工具包 788
初始化 789
资源池相关 791
分布式训练相关 792
在线服务部署相关 796
开发机相关 801
接口文档 801
数据集相关 805
JAVA SDK 808
概述 808
安装SDK工具包 808
初始化 808
数据集相关 811
Python SDK 818
概述 818
安装SDK工具包 818
初始化 819
资源池相关 820
分布式训练相关 823
在线服务部署相关 833
数据集相关 845
工作流相关 850
常用工具 858
AIHC CLI 858
概览 858
安装AIHCCLI 858
配置AIHCCLI 859
通过CLI使用百舸服务 862
版本变更记录 887
AIHC Skill 887
概述 887
具身智能 888
最佳实践 902
使用产品事件设置告警通知规则 902

功能发布记录

发布时间	功能概述
2026-10	● 资源池： - 监控整体升级优化 - 预留实例券购买流程体验提升 - 新增故障自愈事件推送 ● 开发机： - 开发机列表显示开发机最近一次关机时间且支持排序 - 支持同时挂载 eRDMA 和 RDMA 设备 - 运行中的开发机详情界面展示所在节点ID ● 分布式训练： - Ray任务新增时间线 - Ray任务新增Head节点容错 - 训练任务默认开启hang检测 - 训练支持虚拟化卡任务的自动停止策略
2026-09	● 资源池： - 全托管资源池遍历策略升级 - 百舸支持全预付和半预付的预留实例券 ● 开发机： - 新增产品事件功能 - 新增开发机实例节点ID、实例IP显示 - 新增导出开发机列表能力 - 存储挂载新增支持PFS L3 ● 分布式训练： - 优化任务进入WebIDE和WebShell时在新标签页打开 - 创建任务OpenAPI/CLI新增支持闲时资源 - 任务告警信息增加任务创建人 - RayJob 预制大盘&Dashboard 支持展示Prometheus指标 - 优化Ray任务失败时默认展示submitter日志 - 存储挂载新增支持PFS L3
2026-08	● 快速开始： - 新增RLinf v0.3：RPBZZZ6 加速版等开发环境快速部署模板 - 新增MiniMax-H3、Qwen3.8-27B、Qwen3.8-27B（FP8）等开源模型快速部署模板 ● 资源池： - 更新资源池依赖产品开通时机 - 队列添加节点时支持CPU节点跨可用区 - 监控指标治理优化 - 新增产品事件查看功能 ● 开发机： - 开发机支持挂载PFS L3 - 开发机镜像检查优化 - 开发机配置策略校验优化，管理权限统一校验开发机所有者 - CCR镜像地址支持跨账号使用 ● 分布式训练： - 任务详情页查询耗时优化 - 训练任务支持基于优先级的抢占 - Rayjob默认不往head节点调度任务 - 自运维资源池支持自动GC Pod - CLI工具适配Ray任务创建 ● 在线服务部署： - 全托管推理支持预付费及续费 - 推理服务支持使用全托管资源池闲时资源
2026-07	● 快速开始： - 新增RLinf v0.3、Cosmos3：RPBZZZ6 加速版、SparseDrive加速版、FastWAM加速版、OpenPI-0.5：RPBZZZ6加速版等开发环境快速部署模板 - 新增Hy3（FP8）、Hy3、Laguna-M.1（FP8）、Laguna-M.1、Laguna-XS-2.1（FP8）、Laguna-XS-2.1、Laguna-S-2.1（FP8）、Laguna-S-2.1、Kimi-K2.6（W8A8）、Step-3.7-Flash（FP8）、Step-3.7-Flash、DeepSeek-V4-Flash-0731等开源模型快速部署模板 ● 资源池： - 新增批量自动续费功能 - 百舸资源池购买面板优化 - 全托管&自运维增加节点工作负载详情 ● 开发机： - 开发机 OpenAPI 支持 CFS 作为持久化系统盘 - 优化制作镜像失败后容器解冻 - 开发机数据集和存储挂载路径校验 - 优化BLB 服务状态检查 ● 分布式训练： - 训练任务列表支持复制获得任务链接

	- 训练任务支持分享功能 ● 在线服务部署： - 多角色服务部署、完善分布式锁 - 推理默认开启碎片迁移
2026-06	● 快速开始： - openpi0.5/DreamZero/Lingbot-VLA：RPBZZZ6 加速版 ● 资源池： - 队列维度增加对于子用户的资源配额管理限制 - 队列资源视图的优化 - 提交任务排队诊断能力的优化 - 资源报表&监控的升级优化 ● 开发机： - 新增支持使用CFS作为持久化存储 ● 分布式训练： - 训练任务支持分享能力，可将指定任务分享至第三方账号；被分享账号可查看任务基础信息、运行日志及监控指标，适用于团队协作评审、跨人员问题排查场景 - 提供训练任务对比功能，可视化展示多任务间配置参数、运行状态差异，适用于调参效果对比、实验结果复盘。 ● 在线服务部署： - 在线服务部署列表增加【只看我的】筛选功能 - 在线服务部署日志内容支持前缀匹配的模糊检索方式 ● 数据工程&AI资产管理： - 数据下载全新上线，支持加速下载huggingface源站的数据集/模型权重
2026-05	● 快速开始： - 新增MiniCPM-V-4.6-Thinking、MiniCPM-V-4.6、Qwen3-ASR-0.6B、Qwen3-ASR-1.7B系列模型快速部署 - 新增Dream Zero、WBC-AGILE、GEAR-SONIC加速训练 - 上线 Rlinf v0.2 版本、Lerobot v0.5.1 版本 ● 资源池： - 全托管资源池支持通过RapidFS Master托管实例缓存加速 - 全托管资源池的资源队列支持CPU节点的弹性伸缩 ● 开发机： - 闲时关机最多支持天数提升到3天 - 支持挂载RapidFS存储及RapidFS数据集 - 开发机公钥支持更多数量并支持修改不重启开发机 ● 分布式训练： - 日志下载支持异步下载、时间选择 ● 工作流： - 支持拖拉连线编排子任务串行、并行关系 - 支持从任务列表中选择方式添加子任务节点 ● 数据工程&AI资产管理： - 公共数据集下载支持断点续传
2026-04	● 快速开始： - 快速开始新增BEHAVIOR-1K、RoboCasa以及 Isaac Sim 5.1版本的仿真软件部署实践 - 新增GLM-5.1、DeepSeek-V4、Seed、Hunyuan等系列模型快速部署 ● 资源池： - 监控大盘优化，增加CPU/内存/网络等监控指标 - 支持自动化的碎片治理 - 支持修改预付费实例统一到期时间 ● 开发机： - 支持批量停止、删除开发机 - 支持挂载数据湖存储加速工具 RapidFS - 低利用率关机等自动操作在开发机中有提醒 - 支持在WebIDE扩容云磁盘 - 支持更换开发机镜像 ● 分布式训练： - 训练任务新增支持 Rayjob类型任务 - 训练任务支持 hang 的感知以及解析，大幅降低分布式训练 Hang 问题排查成本 - 全托管资源池训练任务增加 VPC 网络监控指标 - 训练任务日志查询，支持跳转日志页首/页尾 ● 在线服务部署： - 支持对大语言、多模态模型性能压测 - 支持通过泛域名访问Web服务 - 支持将环境变量敏感信息加密 ● 工作流： - 工作流功能模块面向所有用户全量开放，不需要再申请白名单 - 定时运行工作流支持设置并发策略（允许、禁止、替换） - 支持output参数，可以在下游节点中使用上游节点的输出作为参数
	● 快速开始： - 新增快速开始搭建Libero、Simpler 仿真环境并调用

2026-03	- 快速开始新增LingBot-Depth、LingBot-World、LingBot-VLA、LingBot-VA系列LingBot模型 - 新增支持GPU卡部署GLM-5、MiMo-V2-Flash、MiniMax-M2.5、InternVL3-38B、DeepSeek-Coder-V2-Instruct、Qwen3-Coder-Next、Qwen3-30B-A3B-x-2507等模型 - 支持P800部署BEVformer、DeepSeek-V3.2（W8A8）、DeepSeek-V3.2-Exp（W8A8）、GLM-5、Qwen3-Coder-Next等模型 ● 资源池： - 全托管资源池的资源队列支持空闲资源出借功能 - 全托管资源池增加资源报表功能 - 排队中的任务增加排队诊断功能 - 支持RapidFS推理模型缓存加速相关配置功能 - 支持预留实例券功能 ● 开发机： - 制作镜像支持免密到推送CCR - WebIDE中支持实时查看CPU、内存、GPU、CDS资源利用率 - 开发机内执行 top、free 等命令调试时支持查看开发机容器的资源使用情况 - 资源池开发机全局配置中支持关闭制作镜像功能 - 开发机内置Tensorboard快速启动 - P800的开发机支持低利用率关机 ● 在线服务部署： - 支持推理服务在线调试 - 部署的公共模型权重加载加速 - 存储挂载支持Alluxio存储 ● 工作流：工作流模块全新发布 - 支持使用YAML方式编排工作流 - 支持串行、并行、菱形任务编排 ● AI资产管理： - pytorch、sglang等多个预置镜像新增社区最新版本 - 新增ray-cpu、ray-gpu预置镜像
2026-02	● 快速开始： - 新增基于RLinf的Pi0.5模型强化学习训练与评测的最佳实践 - 支持快速部署Mixtral-8x22B、MiMo-VL-7B、Baichuan2、GLM-4.6、GLM-4.7、Kimi-K2.5等模型 ● 资源池： - 全托管资源池的队列支持负载类型的配置，提升精细化管理能力 - 全托管&自运维资源池支持虚拟化的物理队列的碎片治理功能 ● 开发机： - WebIDE中支持查看历史执行命令的操作时间 - WebIDE提供vim、bash默认配置文件 - WebIDE新增Code Spell Checker、Markdown Editor 等插件 - 开发机中的进程OOM不会重启开发机 - 资源池全局配置支持关闭创建开发机时申请共享CPU/内存 - 支持在创建开发机时不申请百舸云盘 ● AI资产管理： - 公共数据集上线RoboMIND2.0系列数据集
2026-01	● 快速开始： - 快速在百舸部署apollo 智驾BEV模型全流程仿真验证环境（BEV模型训练、训练模型导出、推理），快速部署无缝对接 Apollo 自动驾驶系统 - 支持快速部署Qwen3-VL、GLM-4、GLM-Z1、PaddleOCR-VL、gemma-3等系列模型 - 新增 NVIDIA Isaac GR00T Blueprint 和 NVIDIA Isaac GR00T-Dreams的最佳实践，支持用户在百舸快速构建 ● 资源池： - 全托管资源池队列支持多种不同套餐规格同时添加到同一物理队列中 - MIG产品化能力的持续升级，支持控制台开启、变更、关闭MIG模式。 - 资源视图升级：增加节点可挂载云磁盘数的视图，并增加相关的事前诊断能力。 - 支持配置资源规格模板，负载侧根据资源规格模板配置资源。 ● 分布式训练任务： - 新增训练任务优先级规则配置功能。队列管理员可在队列详情中为不同用户或角色设定可使用的最高任务优先级，从而保障高优任务优先获得资源。 - 分布式训练任务，支持使用 eRDMA 加速训练 - 分布式训练任务日志查询，支持同时查询全部实例的日志 - 新增Cosmos-Transfer 2.5模型的训练模版，可一键发起模型训练 ● 在线服务部署： - 支持设置Pod长时间未成功调度的告警通知策略 - 服务调用信息优化，按模型类型提供调用示例 - 流量策略支持关闭平台提供的默认鉴权 ● 开发机 - 在开发机使用Podman打镜像支持 - isaac RDP 可视化方案支持 linux 远程客户端 - 支持本地IDE连接百舸开发机实例 - 支持查看制作镜像失败的记录 - 支持在指定的节点创建开发机实例 ● AI资产管理： - 预置镜像管理优化按模块显示等多项使用体验优化
	● 快速开始： - 新增基于 RDP协议 的 Isaac Sim 的可视化方案，相比已有的VNC 方案，性能和使用体验更优

2025-12	- 新增 Isaac Lab 快速开始卡片，指导客户使用 Isaac lab 搭建遥操系统采集数据 - 支持在Nvidia GPU 单机部署推理加速的Wan2.1模型 - 支持Qwen3 系列模型在昆仑芯P800快速部署 ● 资源池： - 自运维资源池支持节点管理，方便用户查看节点的资源信息； - 全托管资源池MIG队列支持监控； - 全托管资源池支持A100和A800的混合调度； - 监控整体优化升级； ● 分布式训练任务： - 训练对接 BOS 对象存储的组件升级至 2.0， 相比 1.0 提供更稳定、更高性能的存储体验 - 训练任务列表支持定时刷新能力 - 新增Qwen2.5VL模型（P800芯片）、Wan2.1 训练模版，可一键发起模型训练 ● 在线服务部署： - 在线服务部署提供开源模型单机、分布式、PD分离部署模版，降低模型部署的复杂度 - 部署推理服务时支持部署推理调度器，指定调度策略 ● 开发机： - pip、apt替换百度软件源 - 支持通过PFS持久化安装conda和pip包在模型训练时共享数据 - WebIDE加载速度提速 - 支持指定工作目录 - 支持WebIDE探活、保活 - 支持查看云磁盘/百舸云盘用量，仅扩容云磁盘/百舸云盘 - 支持非root用户登录WebIDE ● AI资产管理： - 新增镜像管理模块，可视化展示平台预置镜像，优化预置镜像选择体验
2025-11	● 快速开始： - 新增RoboTwin2.0 工具的一键部署，快速构建具身开发环境 - Pelican-VL1.0多模态模型，内置开发教程指导客户快速启动开发 - 新增Qwen2.5VL、Qwen2.5模型的训练模版，可一键发起模型训练 ● 资源池 - 支持广州地域使用百舸平台 - 全托管资源池能力增强：新增对队列中各类工作负载（包括开发机、训练任务、推理服务）的统一视图，提升资源使用透明度与可预期性。 - 遍历策略优化，支持超时重排，解决已入队大任务阻塞后续小任务入队调度问题 ● 分布式训练任务： - 任务提交前，自动检查任务申请的资源是否满足当前队列的资源容量，给出可调度性提示 - TFJob类型的任务支持设置告警 - 任务Tensorboard日志存储，新增支持对接CFS文件存储 ● 在线服务部署： - 支持MIG虚拟化队列部署推理服务 - 服务列表支持导出 - 支持调用自定义域名的服务 - 支持批量管理服务实例 ● 开发机： - 支持MIG虚拟化队列创建开发机 - WebIDE增加守护进程，自动恢复机制提升稳定性 - 支持免释放重启
2025-10	● 快速开始： - 一键部署isaac sim/lab开发环境支持可视化 - Qwen3-8B、Qwen3-14B模型支持使用P800加速芯片快速部署推理服务 ● 资源池： - 全托管资源池的队列支持GPU虚拟化能力，用户开启GPU虚拟化后，可以申请0——1小数卡的任务，提升资源利用率。 ● 分布式训练任务 - 训练模版新增具身模型RDT2训练模版，可一键发起模型训练 - 提交任务选择队列时，可直观查看队列资源使用情况，快速选择合适的队列进行任务提交。当前支持全托管资源池队列和自运维资源池物理队列 ● 在线服务部署： - 服务支持开启删除保护，防止在线服务被误操作删除 - 支持在全托管资源池虚拟化部署推理服务 - 支持为服务设置标签，用于服务筛选和统计 ● AI资产管理： - 新增PhysicalAI系列具身智能公共数据集，支持在平台加速下载到自有BOS存储桶
	● 快速开始： - 支持DeepSeek-V3.2-Exp、Kimi-K2-Instruct-0905、Qwen3系列模型在线服务快速部署 - 支持Isaac Lab 2.2.0、Isaac Sim 5.0 开发环境新版本部署支持一键部署Label Studio标注平台开发环境，降低数据管理和数据标注成本 ● 资源池： - 支持云盘配额管理 - 全托管资源池：新增队列规格限制策略——队列限制“单芯单可用区” - 全托管与自运维资源池、队列与节点：优化资源视图展示逻辑，综合纳入系统预留、组件占用及故障节点的资源核算，提升资源数据准确性与可观测性 ● 分布式训练任务： - 训练模版新增具身模型pi0.5、OpenVLA的训练模版，可一键发起模型训练

2025-09	- 训练任务存储，支持对接数据湖存储加速工具RapidFS，依靠对象存储BOS作为数据湖存储底座，加速模型的训练 - Pytorch分布式训练框架，新增自动注入NPROC_PER_NODE环境变量，用户可在容器内直接引用 - 日志查询页面，支持单次加载最多1000条日志，提升日志查询体验 - 训练任务名称最多支持255字符 ● 在线服务部署： - 支持流量策略功能，多个服务按比例分流，满足灰度发布、异构资源部署等场景需求 - 发布多角色服务功能，支持在多角色服务组中部署PD分离服务 - PD分离部署DeepSeek服务支持根据Kv Cache感知调度 - 实例新增"排队中"状态，服务列表可查看不同状态的实例数信息 - 推理监控支持查看服务Token调用量 ● 开发机： - 大盘监控上线磁盘和共享存储监控指标，新增开发机资源相关环境变量，大大提升资源可观测能力，资源用量更清晰 - 全托管资源池开发机上线百舸云盘，服务更稳定，客户对资源的管理更统一 - 网关升级为云原生网关，统一托管，降低客户资源占用的同时，更加稳定、安全、可靠 - 提升BLB IP组配额至50，更多开发机能同时复用一个BLB，降低客户资源成本 - 开发机创建、更新支持设置环境变量和启动脚本，支持更多开发方式的同时，提升开发效率和开发体验 - 镜像管理支持镜像地址和拉取命令一键复制，镜像管理更高效 ● 权限管理： - 百舸平台IAM系统策略更新：新增CFS、RapidFS、BLS相关权限策略 ● AI资产管理： - 提供具身智能、LLM等场景30+热门开源数据集快速下载 - 新增BOS类型与PFS L2类型数据集相互转储 - PFS L2类型数据集支持生命周期管理，支持按周期和路径条件自动化管理PFS中文件生命周期，提升PFS存储利用率
2025-08	● 快速开始： - 新增DeepSeek-V3.1快速部署 - 具身智能场景新增Isaac Lab、Isaac Sim快速开始应用在开发机一键启用 ● 资源池： - 全托管资源池新增队列工作负载tab页面（仅支持训练任务） ● 分布式训练任务： - 训练任务列表页面，支持展示任务运行期间GPU利用率指标，快速查看当前任务资源使用情况 - 训练任务名称支持重名 - 提供镜像是否支持RDMA的验证文档，可参考验证镜像是否支持RDMA - 训练任务支持挂载BOS对象存储类型的数据集 ● 在线服务部署： - 云原生网关升级，部署服务前需创建网关实例，网关实例支持多个服务复用 - 使用云原生网关部署的服务支持设置IP黑白名单、QPS限流 - 多个服务支持复用四层传输负载均衡BLB实例 - 部署服务支持查看所选队列的总卡数和当前剩余可用卡数 ● 开发机： - 支持免CDS方案 - 支持共享CPU/内存 - 预置镜像全面支持RDMA - 支持ccr免密拉取、提前校验密码推送
2025-07	● 快速开始： - 新增面向具身智能、自动驾驶场景的8个快速开始应用，包括：模型SparseDrive/Uniad、工具Maniskill/LeRobot等 - 新增ERNIE-4.5系列0.3B/0.3B-Base/21B-A3B/21B-A3B-Base模型、ERNIE-4.5-VL系列28B-A3B、28B-A3B-Base模型快速部署 - Qwen2.5 系列大语言模型、 Qwen2.5 vl 模型在昆仑芯资源中快速部署 ● 资源池： - 全托管资源池支持队列节点转让 - 自运维资源池队列开启GPU虚拟化后支持创建子队列 ● 分布式训练任务： - 训练任务支持挂载BOS类型的数据集 ● 在线服务部署： - 部署在线服务时支持CCR免密登录 - 在线服务日志持久化默认前60天免费 - 在线服务支持查看文心系列模型的推理性能和流量监控 ● 开发机 - 支持CDS扩容 - 预置镜像新增CUDA最新版本 - webIDE支持tensorboard profiler能力 - 10余项细节体验优化 ● 权限管理：百舸不同权限角色对于资源池详情页面的二级页面是否可见进行升级优化： - 自运维资源池：队列管理员&队列开发成员对于节点管理、全局配置、资源池拓扑、监控、事件不可见；队列管理只能看到自己所在的队列 - 全托管资源池：队列管理员&队列开发成员对于节点管理、全局配置、监控、变更记录不可见。
	● 快速开始：新增模型快速部署（文心ERNIE-4.5系列、π0系列具身智能、Llama-3.1-Nemotron系列） ● 资源池： - 自运维资源池：节点批量操作（封锁、解除封锁、移出集群）、物理队列筛选 - 全托管资源池：队列批量移出节点/封锁节点、节点管理筛选优化、列表字段补全（可用区/ID）、支持编辑【备注】字段；全局配置：CCR免密镜像、推理服务公网访问开关、开发机全局配置

2025-06	• 分布式训练任务：新增【NVIDIA GR00T N1.5】训练模版；支持设置任务最长运行时长、多任务TensorBoard对比、挂载【对象存储BOS】；任务日志默认对接【日志服务BLS】（免费60天）；监控新增VPC网络带宽/收发包、CPU/内存利用率 • 在线服务部署：支持查看服务/实例事件；新增服务维度资源/性能监控大盘；支持按状态筛选服务 • 开发机：新增时间线/ssh/节点IP字段展示；托管资源池支持全局配置（细化到用户/用户组）；ccr镜像拉取免密 • 权限管理：百舸接入【Cprom权限】及【AIHCAssetFullControl】策略
2025-05	• 快速开始：新增DeepSeek-R1-0528系列、Qwen3系列模型快速部署。 • 资源池：增加卡时统计监控大盘，支持按照子用户和工作负载类型进行卡时统计。 • 分布式训练：支持TensorFlow框架，用户可创建tensorflowjob的训练任务。 • 分布式训练：支持设置任务的可见范围，支持设置队列内可见和仅创建人可见，实现任务实例级别的权限控制。 • 开发机：新增开发机配额管理选项，支持管理员by资源池维度限制成员创建开发机数量、GPU卡数、CPU核数、内存量、CDS用量，提升资源利用率。 • 开发机：新增支持开发机所有者变更、公共开发机、按所有者搜索开发机、列表字段管理等能力。 • 开发机：全面接入云审计，支持用户常用操作、api调用记录、查询、审计。 • 开发机：常见错误可读性优化，支持CDS计费显示，帮助用户更清晰用量与成本，支持BOS免密挂载，新增10多项页面交互细节优化。
2025-04	• 快速开始：提供快速开始入口，支持主流大语言模型基于vLLM、SGlang加速框架快速部署。 • 分布式训练：创建训练任务支持指定节点调度； • 分布式训练：任务容错后，支持查询历史的worker信息。 • 分布式训练：新增 rdt、openpi模型的训练模版，可一键发起模型训练。 • 开发机：新增支持托管资源池；上线GPU虚拟化能力；支持多pfs挂载、数据集挂载、存储挂载权限管理；支持开发机所有权管理、公共开发机能力。 • 开发机：基础环境全量升级，所有客户支持最新版开发机；开发机状态流转优化，流转状态更清晰；流量接入支持BLB和EIP复用，流量成本降低90%。 • 在线服务：优化PD分离部署的服务管理体验，增加服务组为开启PD分离部署的服务提供统一的流量入口
2025-03	• 开发机：提升节点可挂载CDS配额，增加单节点可创建开发机实例数量；webIDE支持文心快码，提升开发效率；其它优化8项。 • 分布式训练支持高精度的RDMA网络监控能力，最高支持10ms精度，为故障诊断排除、训练性能调优等提供数据支撑。 • 分布式训练支持使用CFS存储，可指定CFS子目录进行挂载。 • 分布式训练AIAK-Training-LLM 加速模版，新增 DeepSeek-V3/R1 模型的微调模版，包含满血版 V3 和 R1 模型，以及基于 DeepSeek-R1 蒸馏后的六个小模型，可一键发起全系列 DeepSeek 模型精调。 • 在线服务支持分布式推理部署，充分利用分布式计算资源，满足大规模模型推理的性能和规模要求。 • 在线服务支持PD分离部署DeepSeek全尺寸模型，降低延迟，提升吞吐。
2025-02	• 托管资源池：托管资源池是百舸提供的免运维资源池，托管资源池下，用户无需关注资源池和算力资源的运维管理，由平台保障运维稳定性，为用户提供稳定、可靠、易用的算力管理服务。 • 碎片治理功能：通过对任务诊断与重调度，有效解决分布式训练、小规模推理等场景下的非整机任务导致的资源碎片问题，实现资源高效回收与复用。 • 开发机：支持notebook+vscode集成部署，免EIP跳转；上线新版权限管理、监控等模块；联动升级10+项产品细节体验提升。 • 在线服务：通过自定义部署的在线推理服务支持通过AI网关接入流量并提供服务鉴权能力。 • 工具市场：新增Deepseek-R1应用快速部署镜像，新增联网搜索工具镜像。
2025-01	• 新增工作流功能，支持视频处理工作流模板、数据处理组件，支持使用已有组件编排工作流处理任务（白名单）。 • 开发机完成改版上线，支持开关机持久化数据，内置vscode、jupyterLab等webide，支持开发镜像管理与训练任务联动，支持低利用率关机等一系列服务于开发体验提升、资源利用率提升的重要优化。 • 训练容错支持无条件重试能力，在训练异常时自动重调度任务尝试恢复，避免偶发性异常导致的训练中断问题。 • 百舸平台权限管理升级，支持资源池、队列资源的细粒度的权限配置，满足多团队使用的资源隔离场景。 • 支持CPU资源创建和纳管，支持创建纯CPU工作负载，用于数据处理、开发等场景。
2024-12	• 新增模型管理，支持自定义模型创建，用户可以将自有模型在模型管理中统一管理（12.27）。 • 创建分布式训练任务，支持用户自定义设置任务的提交时间。 • 训练任务历史日志，支持关键字检索&上下文查询&日志展示字段可配置。 • 百度自研高性能通信库BCCL易用性增强，平台支持用户在创建训练任务时，一键注入BCCL通信库。 • 训练任务资源监控，支持任务维度的PFS并行文件存储的带宽和IOPS指标监控。 • 训练任务预置启动超时时间，默认1h。启动超过1h的任务，平台会自动将任务直接置为失败，仅支持Pytorchjob。 • 在线服务部署支持定时、弹性扩缩容实例。 • 支持调度诊断功能，对于长期排队中的任务（排队超过5min）可执行调度诊断，定位任务排队原因并提供解决方案，提升问题解决效率。 • 支持物理队列资源转让功能，支持选择节点和目标队列进行资源转让，无需手动从物理队列移出移入节点。 • 对于新建资源池和新建节点场景，百舸提供可用性检查能力，检查资源池和节点是否异常。 • 推理加速支持大语言模型、多模态模型等，提供更丰富的模型选择。
	• 自定义服务支持定时自动扩缩容、根据自定义业务指标弹性扩缩容。 • 新增数据集管理，支持数据集创建、权限管理以及在训练任务中挂载数据集。

2024-11	• 新增开发机可见范围，支持所有者可见、公开可见。 • 新增开发机、在线服务部署、工具市场支持管理员控制创建时是否可以开启外网。 • 训练容错，支持快速识别和感知训练NCCL/CUDA日志中的典型异常，快速容错，减少故障恢复时间。 • 自定义服务容错，GPU卡级别故障实例能够快速在健康节点重建。 • 基于百度云高性能集合通信库BCCL，新增支持训练hang场景的诊断。 • 轻量服务更名为工具市场，开源工具支持部署到通用资源池，上线模型评测、模型微调、语音合成等工具。 • 添加轻量计算实例无需重装操作系统。
2024-10	• 资源调度策略升级，新增Fifo调度策略，实现按照任务创建时间进行严格排队，解决遍历策略下大任务长期无法获得资源的问题。 • 任务消息通知，新增 任务发生容错、任务所在节点异常 的通知事件。 • 训练容错，支持基于日志关键字的容错能力，满足自定义训练异常进行容错的场景。 • 轻量计算实例可添加CPU服务器和边缘GPU云服务器。 • 自定义服务支持挂载本地盘。
2024-09	• 模型部署参数增加，提高模型请求补偿项可选性，优化结束符号超时问题支持 • Multi lora推理服务热更新，保持原有服务稳定。 • 开发机支持自定义镜像。 • 工具市场支持挂载CFS存储、可部署到边缘BEC云服务器。 • 支持通过百舸控制台创建弹性队列、层级队列。可基于队列不同配额设置实现训推混部、潮汐混部，提升资源利用效率。 • 支持以自定义镜像的方式部署在线服务，提供可配置的健康检查、滚动更新策略，保证服务稳定性、细粒度的流量控制及版本管理。 • 训练任务消息通知，新增短信、电话、企业微信/钉钉、自定义webhook通知方式。 • 训练任务实时日志支持关键字检索。 • 提供CLI命令行，支持资源池/节点/训练任务的管理。 • 创建中、排队中的任务支持停止操作，新增记录停止人的功能。 • 训练任务使用PFS，支持指定PFS子目录挂载。
2024-08	• 新增开发机模块，为开发者提供的在线编译、调试代码和模型开发环境。 • 新增工具市场功能，集成主流AI开源项目和百舸自研产品镜像模版，可面向中小企业提供一键快速部署AI场景所需工具和工具托管能力，以快速搭建开发调试环境。
2024-07	• 百舸控制台支持创建NPU资源池，支持NPU、英伟达同一资源池混合部署，统一管理。 • 创建资源池时，网络模式支持选择VPC-ENI，提供更灵活的容器网络配置能力。
2024-06	• 发布百舸OpenAPI，支持资源池、队列、训练任务的管理。 • 百舸资源视图新增空闲资源统计功能，支持查看集群实时空闲资源分布情况及历史趋势。 • 训练任务性能监控，新增 lm loss指标。 • 训练任务集合通信带宽监控大盘，支持按照通信类型进行带宽汇聚分析、新增 Rankld 信息。 • llama、qwen系列的模型部署推理服务时支持Multi lora，可以将多个lora模型和base模型合并部署，提升资源利用率。 • 支持通过实例和服务维度查看推理服务业务监控。 • 推理服务新增支持Qwen2系列模型。
2024-05	• 新增AI加速套件，整合AIAK训练、推理加速镜像及训推加速过程中所需的一系列工具，轻松一键获取。 • 推理服务支持实例反亲和部署，避免单节点故障导致服务不可用。 • 推理服务实例弹性伸缩，在业务负载出现波峰或波谷时，可以根据业务流量自动调整实例数量，实现动态管理线上服务的计算资源，从而保障业务平稳运行并提高资源利用率。 • 推理服务支持设置防抢占策略，开启后实例不会被训练任务抢占。 • 发布队列维度可观测大盘，支持监控队列维度GPU利用率、分配率、超发率等关键指标。 • 训练任务支持用户自定义配置CPU、内存以及共享内存资源配额，用户可根据业务需求按需配置。
2024-04	• 训练加速模型支持新增 Qwen1.5 和 Mixtral 系列模型。 • 推理加速模型支持Qwen 1.5系列，InternLM2-20B以及Mixtral-7Bx8等模型。 • 训练任务性能监控，训练吞吐指标兼容动态seq-length场景。
2024-03	• 支持训练任务Timeline，便于用户快速查询任务运行的重要时间节点。 • 基于百度云集合通信库BCCL，提供训练任务集合通信带宽可观测能力，为故障诊断排除、训练性能调优等提供数据支撑。 • 基于AIAK-Training 训练加速工具包，提供训练任务吞吐、训练分阶段耗时等性能监控指标，便于用户实时掌握训练的性能。 • 支持RDMA Tor拓扑感知功能，减少训练过程中跨TOR通信的频率，提升训练效率。
	• 训练任务集成Tensorboard能力，支持通过Tensorboard查看训练结果分析报告。

2024-02	• 任务资源监控新增GPU NVLink带宽、RDMA网络收发包等指标；节点维度监控新增磁盘/CPU/内存等资源统计，便于用户直观监控任务所在节点的运行情况。 • 百舸平台全面支持昇腾芯片，允许用户基于昇腾芯片快速搭建资源池，并支持节点健康状态的实时监测功能。同时支持部署和管理基于昇腾芯片的训练任务，提供训练可观测和自动容错等基础功能，满足用户在百舸上使用昇腾芯片进行业务部署的场景。
2024-01	• 提供训练任务资源监控大盘，支持快速查看任务/实例/GPU/节点维度的资源使用情况。 • 提供训练任务事件列表，支持查询任务以及关联资源在启动、运行阶段的变更情况，有助于日常观察任务变更以及定位问题。 • 提供资源池拓扑功能，支持查看资源池节点和任务的可视化拓扑。
2023-12	• 提供资源视图大盘和组件稳定性大盘，支持快速查看资源使用情况和组件关键指标。 • 支持设置/修改任务优先级，按优先级高低进行任务调度。 • 支持资源池事件中心，统一记录资源池所有事件，并持久化保存，方便问题排查和追溯。
2023-11	• 提供CheckPoint加速能力，降低保存Checkpoint的时间开销。 • 训练任务支持一键重跑，提高任务提交效率。 • 训练任务日志一键导出，快速下载任务日志，提高问题排查效率。
2023-10	• 推理服务新增Llama2、ChatGLM2、Baichuan1&2模型。 • 提供资源视图列表，支持集群/节点/GPU卡资源分配和空闲状态查看&分析。
2023-09	• 支持资源池节点故障感知&隔离，自动感知故障并封锁故障节点。 • 支持RDMA Test，表单化执行benchmark测试，以评估集群网络性能。
2023-08	• 提供开源数据集加速下载、开源模型权重加速下载以及数据转储。 • 支持大模型推理部署服务，提供主流开源大模型快速部署和加速。
2023-07	• 支持任务自动容错，主动感知硬件故障，自动重启任务。 • 支持MPI类型任务，可基于MPI框架直接运行任务。 • 支持本地盘存储方案，支持本地盘或文件系统以HostPath方式挂载到资源池并使用。
2023-06	• 提供NCCL检测功能，表单化执行benchmark测试，无需手动创建nccl test任务。 • 提供训练任务模板，支持主流开源大模型一键训练加速。 • 支持训练任务状态感知、训练效果LOSS异常感知，通过短信、邮件等方式向告警接收人发送任务信息。
2023-05	• 百舸异构计算平台AIHC正式发布，支持资源池创建和管理，Pytorch任务创建和管理。

产品描述

介绍

产品介绍

百度百舸·AI 计算平台（AI Heterogeneous Compute，简称 AIHC）是面向大规模深度学习的高性能云原生 AI 计算平台，为模型算法专家和运维专家提供全面的集群运维支持和任务全生命周期管理，同时具备训练 / 推理加速、故障容错、智能故障诊断等高级功能，万卡单任务有效训练时长可达 98% 以上，大幅提升算力使用效率，助力企业在大模型时代的业务转型。

产品架构

百度百舸AI计算平台的产品架构图如下所示：



百度百舸·AI 计算平台包括 AI 基础设施、AI 资源管理、AI 工程和模型加速能力。

AI基础设施

百舸 AI 基础设施层作为平台的基石，整合了高性能计算（昆仑芯 / GPU）、高性能存储（PFS、BOS 等）、高性能网络（大规模 HPN、万卡级别 RDMA 网络等），为上层应用提供稳定、高效且可弹性扩展的硬件与系统支持。

AI 资源管理

百舸 AI 资源管理层是实现算力高效利用的核心。用户可根据业务场景选择全托管或自运维资源池，通过弹性多级队列与智能负载调度技术，结合碎片治理和节点免运维等能力，灵活分配计算任务并最大化资源利用率。

AI 工程

AI 工程聚焦于 AI 开发的全生命周期管理，提供从数据准备到 AI 资产（如数据集、模型、镜像）管理的完整工具链，为开发者构建高效、规范的一站式机器学习环境。

模型加速能力

面向社区开源热门模型，百舸通过模型预置与训推加速提供开箱即用的产品能力，降低模型开发与部署成本。

- 预置 80+ 热门模型，覆盖大语言模型（LLM）、多模态模型、智驾模型（自动驾驶）、具身智能模型等。
- 训推加速效果：LLM 训练加速 20%+，多模态加速 30%，智驾模型加速 100%+。

核心概念

概述

本文档介绍与百度百舸计算平台相关的核心概念，包括全托管资源池、自运维资源池、轻量资源、队列、节点、实例、PFS 文件系统、CPROM 监控、开发机、分布式训练、任务和部署服务，并结合控制台实际入口说明常用资源与任务的查看、创建和管理方式。

全托管资源池

全托管资源池是百舸提供的免运维 AI 计算资源池。在全托管资源池下，用户无需关注资源池和算力资源的运维管理，由百舸平台保障运维稳定性，为用户提供稳定、可靠、易用的算力管理服务。

使用引导：创建全托管资源池

自运维资源池

自运维资源池是百舸为您提供的稳定、高性能的大规模分布式训练、在线服务部署环境。在自运维资源池中，您可创建多个资源池，一个资源池对应一个 CCE集群，可以关联一个 PFS实例 和一个 CPROM实例。在资源池中，用户可以创建多个队列，从而在不同的队列中处理不同业务的工作任务。通常情况下，企业内部一个独立的团队或者项目建议创建一个独立的资源池，实现团队间或项目间的资源隔离。

使用引导：创建自运维资源池

轻量资源

轻量资源实例是百舸提供的灵活、易用、高性能的 AI 开发、模型部署环境。轻量服务主要面向中小企业和开发者，无需运维和管理集群，开箱即用。您可将已购买的百度云云服务器添加到轻量资源中用于部署轻量服务，助力您使用更便携更高效的在云端开发测试环境。

使用引导：轻量计算实例

队列

队列是资源池中部分资源的集合，用于工作负载（训练任务、在线服务）的运行。一个资源池中可创建多个队列。用户购买资源池后可将资源池划分成若干个队列，并使用队列中的资源处理不同业务的工作负载。

全托管资源池资源队列是算力资源分配调度的单元，所有使用全托管资源池算力的工作负载都需要提交到对应的资源队列中才能获得算力分配，使用方式可参考创建和管理资源池队列。

自运维资源池创建成功后会默认生成 default 队列，可使用当前资源池的所有计算资源，使用方式可参考自运维资源池队列管理。

节点

在百度百舸平台中，节点（Node）可以看作是计算、存储和容器等服务的一个集合，用户可以根据实际需求选择合适的节点来执行 AI 任务。通过使用节点，可以有效地管理和扩展应用程序的部署，并提供高度可靠的服务。

实例

实例可以理解为一台虚拟的服务器，包含 CPU、内存等最基础的计算组件，是云服务器呈献给您的实际操作实体。百舸中轻量计算的实例一般指的是 BCC 云服务器或 BEC 边缘云服务器实例，详情可查看 BCC实例、BEC边缘云服务器。

PFS 文件系统

并行文件存储服务PFS（Parallel Filesystem Service），是百度智能云提供的完全托管、简单可扩展的并行文件存储系统，针对高性能计算场景提供亚毫秒级的访问能力和高 IOPS 的数据读写请求能力。同时，百度智能云 PFS 提供简单、易操作的接口，免去部署、维护费用的同时，最大化提升您的业务效率。

CPROM 监控

百度智能云Prometheus监控服务（Cloud Managed Service for Prometheus，简称 CProm）是完全对接开源 Prometheus 监控系统，针对云原生场景提供的全托管、高可用、可扩展的 Prometheus 监控服务。

开发机

开发机 是百舸平台为开发者提供的在线编译、调试代码和模型开发的模块。开发机支持使用 WebIDE 在线开发，或通过 SSH 连接开发机远程开发，也提供了持久化的共享文件系统用于存储开发中的数据。开发机在关机后，平台会释放开发机的算力，通过备份开发机可以保存之前的操作、下载的数据和配置环境等。

分布式训练

分布式训练 是百舸提供的灵活、稳定、高性能的机器学习训练环境。支持多种算法框架，能够处理大规模的分布式深度学习任务，同时也支持自定义算法框架。

任务

任务是百舸计算平台中最基本的执行单元，一个任务通常指一个 AI 模型的分布式训练作业。

部署服务

百舸平台提供更灵活的部署方式，您可以将通过 Docker 构建的镜像使用此功能部署为推理服务，将模型文件或代码挂载到服务实例中。使用方式可参考自定义部署服务。

产品优势

简单易用

百度百舸·AI 计算平台整合百度智能云完善的基础设施和 AI 集群能力，提供全面、智能的高性能云原生 AI 计算平台。平台所有功能均支持在百度智能云控制台页面操作，您可以在控制台完成从资源创建到大模型训练任务提交的全流程操作。通过使用集成预置部署模板和推理镜像，帮助企业分钟级、低成本完成推理服务部署。

性能强劲

百度百舸·AI 计算平台提供经过百度内部验证的 AIAK 加速能力，帮助您在 AI 计算场景下实现更快的性能。训练性能提升 30%，推理性能提升 60%，您可以基于百度 AI 加速能力构建符合自身业务需求的训练任务，提升训练效率和效果。

稳定可靠

百舸平台内置集群容错能力。在长时间运行训练任务场景下，平台通过容错能力进行自动重启和重新调度，有效解决掉卡和任务 hang 问题，提升训练稳定性，保障训练任务不中断。集群级集合通信库 BCCL 支持故障精准定位和慢节点发现，并在大模型训练场景的可观测性、故障诊断、稳定性等方面进行优化，保障不因集群故障浪费时间。大规模快速重调度和任务初始化能力支持万卡任务分钟级恢复。

智能高效

百舸平台提供全面、智能的大模型训练推理业务生命周期管理，不仅可以快速搭建大规模训练集群，在训练过程中大幅缩短运行时间，腾出更多卡时，同时也提供强大的日志智能管理能力、节点智能检测能力和监控大盘展示能力，并集成 TensorBoard 功能，帮助您高效发现和解决任务及资源层面的问题，提高训练效果。

使用限制

本文将介绍在使用百度百舸·AI计算平台过程中可能会涉及到云产品以及对应的限制项，方便您在使用前根据需要统一调整，避免因限制导致的使用问题。

通用限制

限制项	限制
白名单	因当前依赖产品限制，使用百舸平台前需开通相应云产品白名单。 请在创建资源池前，联系百度技术支持人员，并确保当前账户已完成以下操作，否则无法创建和使用资源池。 *云服务器BCC：开通A800机型套餐白名单，更多信息请查看云服务器BCC* *并行文件存储服务PFS ：开通PFS产品白名单并创建可用的PFS实例*

GPU/NPU套餐支持

百舸平台上可直接创建的GPU/NPU套餐如下，其他型号逐步完善中。

产品形态	GPU/NPU型号	套餐名称	支持镜像&驱动&CUDA版本
EBC/EHC	Nvidia A800	ehc.lgn5.c128m1024.8a800.8re.4d	Ubuntu 20.04 & Ubuntu 22.04 + 535.161.08 + 12.2.0 & 12.3.2
BCC	Nvidia A800	bcc.lgn5c.c116m978.8a800.8ib.2d	Ubuntu 20.04 & Ubuntu 22.04 + 535.161.08 + 12.2.0
BCC	Nvidia H800	bcc.lgn6tcef.c180m2938.8h800.8re.4d	Ubuntu 20.04 & Ubuntu 22.04 + 535.161.08 + 12.2.0
BCC	Nvidia A100	bcc.gn5.c112m896.8A100-40g	Ubuntu 20.04 & Ubuntu 22.04 + 535.161.08 + 12.2.0
BCC	Nvidia A100	bcc.lgn5c.c116m978.8a100.8ib.2d	Ubuntu 20.04 & Ubuntu 22.04 + 535.161.08 + 12.2.0
BCC	Nvidia L20	bcc.gn5i.c112m476.4l20	Ubuntu 20.04 & Ubuntu 22.04 + 535.161.08 + 12.2.0
EBC/EHC	Nvidia A800	ehc.lgn5.c128m1024.8a800.8re.4d	Ubuntu 22.04 + 535.161.08 + 12.3.2
EBC/EHC	Nvidia H20	ebc.lgn7t.c208m2048.8h20.4d	Ubuntu 22.04 + 550.127.08 + 12.5.1

分布式训练

限制项	限制
任务管理	a. 通过k8s api提交的任务不支持复制、重跑操作 b. 训练平台仅支持Pytorch、MPI框架任务的提交
可观测	集合通信带宽监控暂不支持MPIJob
稳定性	训练容错能力暂不支持MPIJob

开发机

限制项	限制
资源配额	开发机创建在已有百舸资源池的指定队列中，与训练任务、推理服务共享资源池GPU/CPU/内存等资源，资源配额受用户队列资源配额限制。
环境保存	当前开发机以容器形式存在，开发机重启、关机、异常退出、所在节点（集群计算节点）异常时，容器会销毁或缩容，对应环境会一并销毁。 开发机发生重启、关机、异常时以下内容会丢失： 1. 未保存的文件 2. 非镜像中预制的软件和依赖 3. 用户对开发机设置的用户账号、权限（user权限等）
文件保存	开发机系统目录为临时存储，在发生重启、关机、异常退出时保存在系目录的文件会丢失，建议用户文件不要放置在系统目录，而是放置在挂载的共享存储中

应用场景

大模型预训练

百舸平台提供大模型预训练的基础环境。您可以利用百舸平台提供的强大底层算力、高速网络和高性能存储，一站式搭建大模型预训练环境。平台支持多种主流训练框架，同时提供 AI 加速能力，帮助加速大模型预训练过程。

- 强大的算力资源：百舸平台提供多种 GPU 算力资源，包括 A100、A800 等，满足大模型训练场景的算力需求。
- 高速网络：百舸平台默认开启 RDMA 高速互联网络，保障大模型训练网络性能。
- 高性能存储：百舸平台使用百度云并行文件系统 PFS 作为高性能存储，为模型训练数据保驾护航。
- 支持多种训练框架：百舸平台支持 PyTorch、MPI 等主流训练框架，您可以提交并运行支持对应框架的训练任务。
- 百度验证的 AI 加速能力：百舸平台提供经百度内部验证的 AI 加速能力，实现大模型训练场景性能提升。

模型微调

您可以在百舸平台上对已经训练好的模型进一步调整参数，以适应特定任务或领域。通常情况下，微调会利用已有的模型参数和特定训练数据进行。

- 灵活的任务管理：百舸平台提供灵活的任务管理能力，可对任务进行复制、停止、删除等操作，方便在模型微调过程中调整参数。
- 强大的日志管理：百舸平台提供强大的日志管理能力、监控大盘展示能力，并集成 TensorBoard 功能。您可以通过日志、监控和 TensorBoard 实时查看任务运行情况，调整模型参数，提高训练效果。

快速入门

快速开始体验百舸核心功能

概述

快速开始对百舸分布式训练、在线服务部署、开发机等进行了模板封装，涵盖 LLM、AIGC、计算机视觉等多种 AI 场景，帮助您零代码、高效快捷地部署、训练、开发开源大模型、具身智能模型等，从而快速上手百舸功能，验证业务需求。

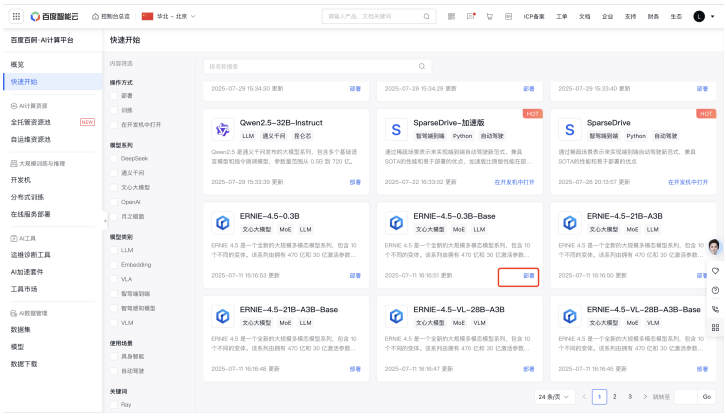
本文带你通过快速开始快速上手理解使用百舸分布式训练、在线服务部署、开发机三个核心功能模块。

前提条件

1. 使用主账号开通百舸平台
2. 登录百舸控制台，购买资源池，并确保已创建可用的资源队列。
3. 如需部署在线服务，请提前准备可用的 AI 原生网关。

模型部署

1. 登录百舸控制台，在左侧导航栏单击快速开始，搜索并找到对应模型选项卡，然后单击部署。



2. 配置部署参数。在部署配置页面已为您预置了默认参数，直接单击部署 > 确定。



3. 提交后，页面跳转到在线服务部署列表。等待几分钟，当服务状态为运行中时，代表部署成功。
4. 调用模型。

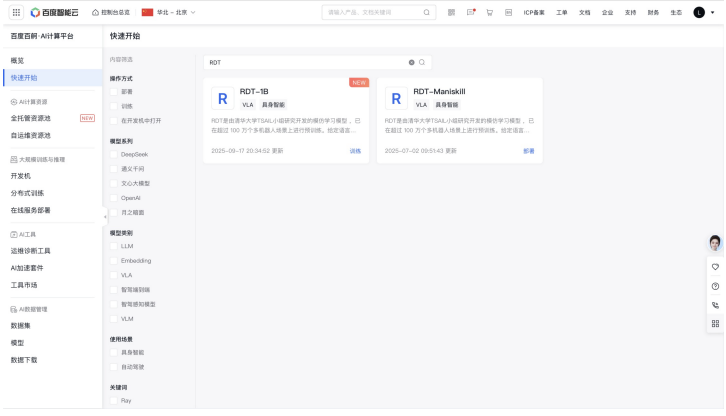
   在服务列表页面，单击 **【查看调用信息】** 获取调用地址和 Token。

   常用的调用方式如下：

```
curl --location '<访问地址>/v1/chat/completions' \
--header 'Content-Type: application/json' \
--header 'Authorization: <TOKEN>' \
--data '{
    "model": "ERNIE-4.5-300B-A47B",
    "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}],
    "max_tokens": 1024,
    "temperature": 0
}'
```

模型训练

1. 在左侧导航栏单击快速开始，搜索并找到RDT-1B选项卡，然后单击训练。



配置训练任务参数。只需配置如下关键参数，其他参数默认即可。



下表给出了不同模型默认参数和数据集的情况下，训练所依赖的最低配置：

模型名称训练方式GPU资源robotics-diffusion-transformer/rdt-1bPretrain8卡，H800/A800等资源robotics-diffusion-transformer/rdt-1bFinetune1-8卡，H800/A800等资源

3. 单击 **【确定】** 后，页面跳转到分布式训练任务列表。训练任务进入创建中状态，需要等待几分钟，当任务状态为运行中时开始训练。

4. 查看训练任务并等待训练完成。在训练过程中，任务详情页面将展示任务日志、事件及指标监控。待训练执行完成后，模型将存储到设置的 BOS 目录中并自动注册到模型管理。

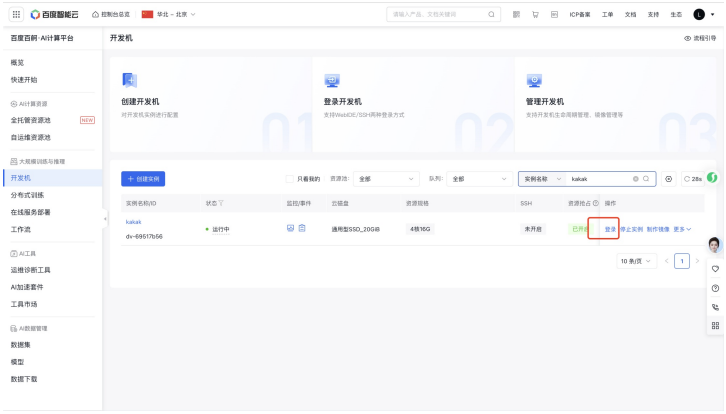
在开发机中打开

1. 在左侧导航栏单击快速开始，搜索并找到Isaac Lab选项卡，然后单击在开发机中打开。

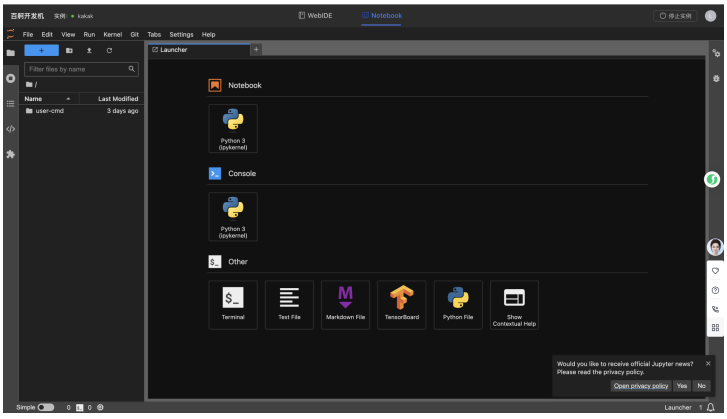
配置启动参数。只需配置如下关键参数，其他参数默认即可。



2. 点击确定后会跳转到开发机列表，需要等待几分钟，当处于运行中时开发机可以正常使用。



3. 点击登录，可以登录开发机使用开发机内置的WebIDE和Notebook进行开发调试。



快速创建大模型训练任务

概述

本文档介绍使用百度百舸·AI 计算平台的准备工作和创建流程。首次使用百度百舸·AI 计算平台时，您可以参考本文完成资源池创建、队列创建、分布式训练任务创建以及监控查看等关键流程。

主要操作流程

1. 注册百度智能云账号，并完成实名认证。请参考 注册 和 实名认证。
2. 登录 百度百舸·AI计算管理控制台。
3. 创建资源池。
4. 创建队列。
5. 创建分布式训练任务。
6. 查看监控。

使用准备

在创建大模型训练任务前，您需要创建资源池，百舸为您提供了全托管资源池和自运维资源池，您可根据您的业务规模和需求选择资源池类型。

全托管资源池 为您提供稳定、易用的免运维资源池，您可更专注于模型训练；自运维资源池 为您提供稳定、高性能的计算集群，您可在平台全方位管理您的计算资源。

因当前产品限制，创建资源池需开通相应白名单。创建前可通过 工单 联系百度支持人员，确保当前账户已完成以下操作，否则无法创建和使用资源池。我们将在后续版本优化这一操作。

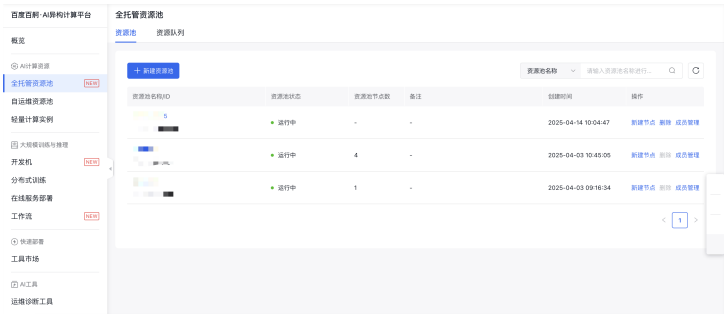
- 云服务器 BCC：
  - 开通 A800 机型套餐白名单。
- 并行文件存储服务 PFS：
  - 开通 PFS 产品白名单。
  - 创建可用的 PFS 实例。
  - 开通 ParentDir 白名单。

创建流程

第一步 创建资源池

资源池是百舸内部管理 AI 资源的基础单元，一个资源池对应一个 CCE 集群（计算服务）、一个 PFS 实例（存储服务）和一个 CProm 实例（监控服务）。本文以创建全托管资源池为例。

注意： 创建资源池会创建或关联计算、存储、监控等云资源，可能产生费用。提交订单前，请确认付费方式、节点数量、节点规格、账户余额和协议内容。1. 登录 百度百舸·AI计算平台 控制台，进入 **AI 计算资源-> 全托管资源池**，单击 **【创建资源池】** 进入 创建全托管资源池 页面。



3. 配置基本信息。

- 资源池名称
- 备注
- 选择地域和可用区



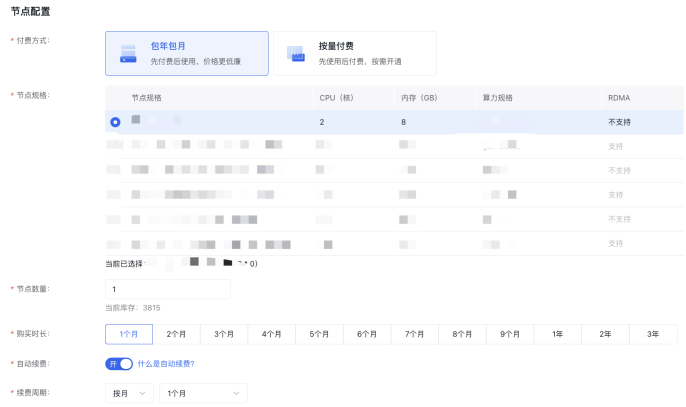
4. 配置网络信息

- 配置资源池网络
- 选择节点子网和容器子网
- 选择安全组
- 填写Cluster IP网段



5. 配置节点信息

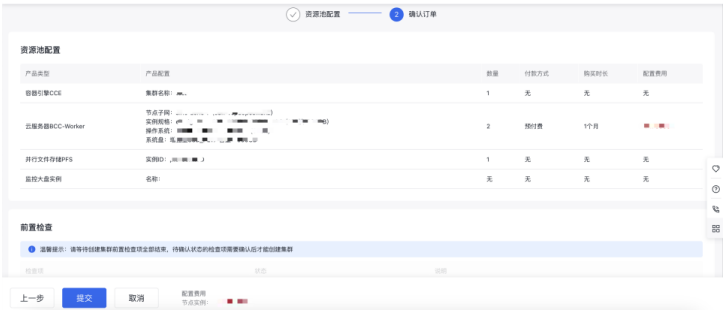
- 选择付费方式
- 选择节点规格
- 选择节点数量
- 选择使用时间
- 选择续费方式



6. 选择关联服务 按需选择并行文件系统PFS和监控实例，监控实例也可后续再开通。



7. 确认订单。 确认资源池配置信息，然后单击提交。



您也可以选择创建自运维资源池进行后续的大模型训练，创建请参考：创建自运维资源池。

第二步：创建队列

队列是一个资源池中部分资源的集合，用于工作负载（训练任务、在线服务、开发机）的运行。本文提供创建全托管资源池队列的操作方式。

1. 登录百度百舸·AI 计算平台控制台后，进入 **AI 计算资源->全托管资源池**，切换到 **【资源队列】** 页签，单击 **【创建资源队列】** 进入创建页面。



2. 在队列管理页面，点击 **创建队列**，并填写相关信息。

- 队列基本信息。
- 资源中节点分配信息。
- 若资源池中无节点，可以点击 **添加节点** 进行新增。



3. 确定创建。 完成上述配置后，单击完成即可完成资源池中队列的创建。

如您选择自运维资源池创建队列，请参考：队列管理。

第三步：创建分布式训练任务

任务是百舸 AI 计算平台中最基本的执行单元，一个任务通常指一个 AI 模型的训练作业。

1. 进入 **大规模训练与推理->分布式训练** 任务列表页面，单击 **【创建任务】**。

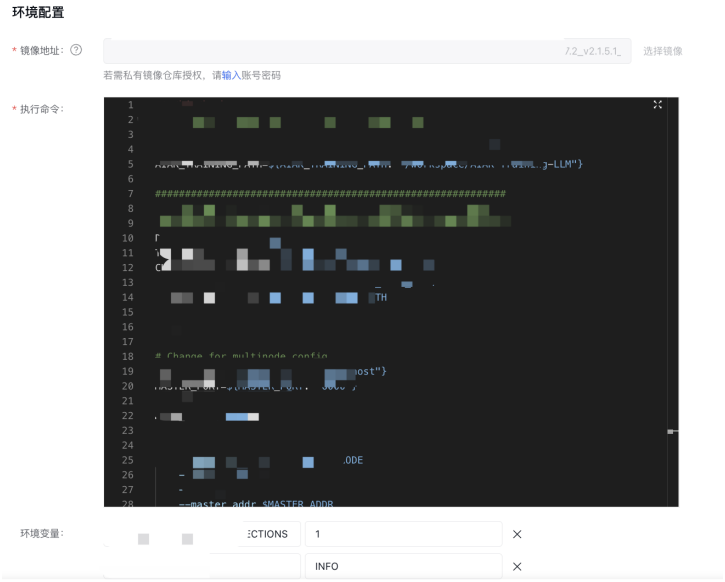
2. 配置任务基础信息。

- 填写任务名称
- 选择资源池：资源池类型和对应资源池
- 选择队列：资源池下对应的队列
- 选择任务优先级
- 选择训练框架
- 选择是否开通日志持久化
- 选择创建方式：支持自定义创建和AIAK训练加速模板创建，选择模板后将自动填充好运行训练作业的配置信息，您可以按需修改。



3. 环境配置

- 选择镜像地址，如您任务创建方式选择AIAK训练加速模板创建，则无需再次选择镜像地址；
- 填写执行命令，您也可以基于AIAK加速训练加速模板中执行命令进行修改；
- 设置环境变量



4. 配置资源

- 选择实例数
- 选择是否申请加速芯片、加速芯片类型
- 填写CPU
- 填写共享内存
- 选择是否开启RDMA



5. 配置数据源

- 根据关联文件系统填写挂载路径。



6. 选择容错/诊断

- 选择是否配置任务hang检测
- 选择是否开启容错和容错策略



7. 高级配置：百舸训练任务为您提供更多个性化高级配置

- 选择是否开启Tensorboard，如开启则需填写日志路径
- 选择是否开启任务退出后自动删除，如开启则需配置保留时长
- 选择是否开启告警，如开启则配置告警策略
- 选择是否定时提交训练任务，如开启则需指定定时提交时间



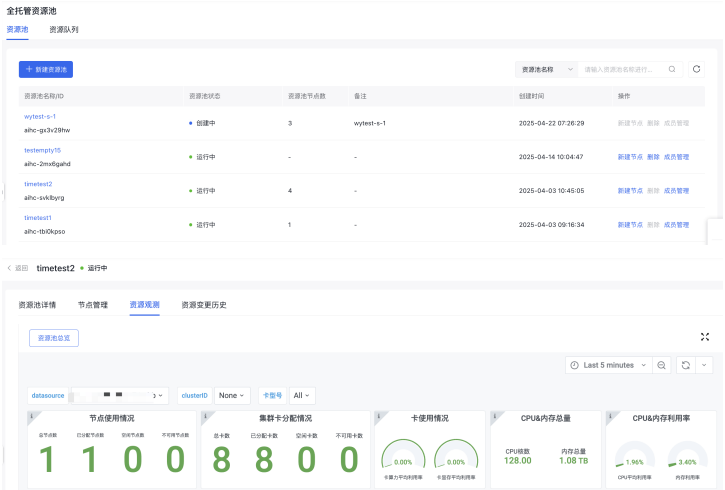
8. 填写完成后，点击 **完成**，提交任务。至此，您已成功运行一个大模型训练任务。

第四步：查看训练任务详情

全托管资源池需要接入CPROM监控实例可查看资源池监控，可在列表页点击资源池名称，进入资源池详情页，点击 **快速接入**，创建监控实例。



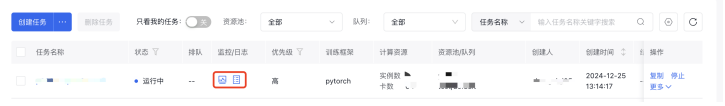
若在创建资源池阶段已接入CPROM监控实例，可直接在资源池详情中，点击 **资源观测** 查看监控。



自运维资源池需要创建CPROM监控实例可进行监控查看，接入方式 接入监控实例并启用采集任务

训练任务各项监控和观测内容

用户可以通过在分布式训练列表页点击 **监控和日志** 按钮，或点击任务名称进入任务详情页，点击 **任务监控/任务日志** 查看任务的资源使用情况和任务日志。



百舸平台为训练任务提供了全面的可观测内容，用户可以多维度查看训练任务的各项指标： 任务日志、 任务事件、 任务资源监控、 任务性能监控、 任务集合通信带宽监控、 训练效果监控Tensorboard。

快速部署推理服务

概述

本文档介绍如何在百度百舸·AI 计算平台快速部署推理服务。在首次使用百度百舸·AI 计算平台时，您可以按照本文档完成资源池创建、资源队列创建、自定义推理服务部署，以及服务升级、手动扩缩容等操作。

主要操作流程

1. 注册百度智能云账号，并完成实名认证。
2. 登录 百度百舸·AI 计算管理控制台。
3. 创建资源池和资源池队列。
4. 部署推理服务。
5. 根据业务需要执行服务升级、手动扩缩容、监控等更多操作。

前提条件

1. 已注册百度智能云账号，并完成实名认证。具体操作请参考 注册 和 实名认证。2. 已根据业务规模和资源管理需求选择资源池类型。

全托管资源池 为您提供稳定、易用的免运维资源池，您可更专注于模型训练和推理服务部署；自运维资源池 为您提供稳定、高性能的计算集群，您可在平台全方位管理计算资源。

因当前产品限制，创建资源池需开通相应白名单。创建前可通过 工单 联系百度支持人员，确保当前账号已完成以下操作，否则无法创建和使用资源池：

- 云服务器 BCC：
  - 开通 A800 机型套餐白名单。
- 并行文件存储服务 PFS：
  - 开通 PFS 产品白名单。
  - 创建可用的 PFS 实例。
  - 开通 ParentDir 白名单。

3. 已在 容器引擎 CCE 的 **Helm->Helm 模板** 中，在百度智能云模板中安装 `cce-inference-controller` 到 `kube-system` 命名空间，并将 `replicaCount` 修改为 `1`。

Helm 模板对应 Helm 社区的 Helm Chart 概念，是 Helm 定义的一种打包格式，用来描述一组相关的 Kubernetes 资源。Helm 能够帮助您管理自定义的 Kubernetes 资源集合，从而快速实现复杂应用的定义、安装和升级。

3. 准备部署自定义服务所需的信息：

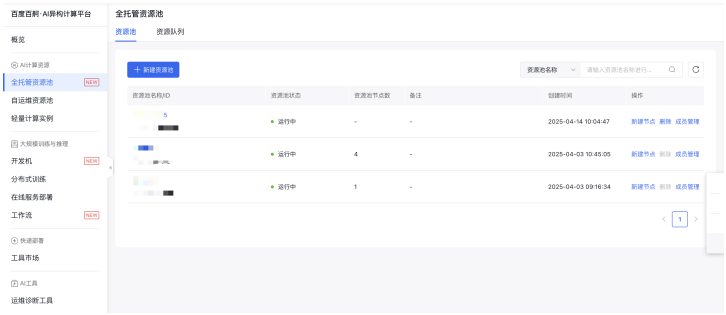
- 服务镜像地址，例如： `registry-vpc.cn-beijing.baidu.com/xxx/yyy:zzz`。
- 镜像的启动运行命令，例如： `/data/eas/ENV/bin/python /data/eas/app.py`。
- 镜像中进程监听的网络端口号，例如： `8000`。

创建流程

第一步：创建资源池

资源池是百舸内部管理异构资源的基础单元，一个资源池对应一个 CCE 集群（计算服务）、一个 PFS 实例（存储服务）和一个 CProm 实例（监控服务）。本文以创建全托管资源池为例。

1. 登录 百度百舸·AI 计算平台 控制台，在控制台导航中选择 **AI 计算资源->全托管资源池**，进入全托管资源池列表后，单击 **【新建资源池】**。您也可以直接访问 新建资源池。



3. 配置基本信息。

- 资源池名称
- 备注
- 选择地域和可用区



4. 配置网络信息

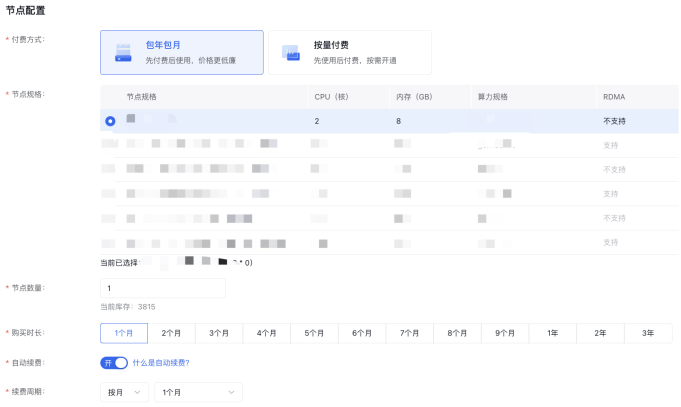
- 配置资源池网络
- 选择节点子网和容器子网
- 选择安全组

- 填写Cluster IP网段



5. 配置节点信息

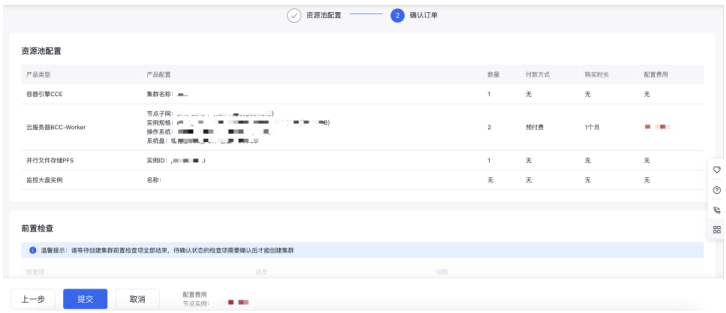
- 选择付费方式
- 选择节点规格
- 选择节点数量
- 选择使用时间
- 选择续费方式



6. 选择关联服务 按需选择并行文件系统PFS和监控实例，监控实例也可后续再开通。



7. 确认订单。 确认资源池配置信息，然后单击提交。



您也可以选择创建自运维资源池进行后续的大模型训练和推理服务部署。创建方式请参考 创建自运维资源池。

第二步：创建队列

队列是一个资源池中部分资源的集合，用于训练任务、在线服务、开发机等工作负载的运行。本文提供创建全托管资源池队列的操作方式。

1. 在百度百舸·AI 计算平台控制台导航中选择 **AI 计算资源->全托管资源池**，进入全托管资源池页面后切换到 **资源队列** 页签，单击 **【创建资源队列】**。



2. 在队列管理页面，点击 **创建队列**，并填写相关信息。

- 队列基本信息。
- 资源中节点分配信息。
- 若资源池中无节点，可以点击 **添加节点** 进行新增。



4. 确认队列名称、资源池、节点来源、负载类型、调度策略和队列状态等必填项已完成配置后，单击 **【提交】**。如页面提示必填项缺失，请补充完成后再次提交。

如您选择创建了自运维资源池，您可以在自运维资源池下创建队列。具体操作请参考 队列管理。

第三步：部署服务

百舸平台提供灵活的部署方式，您可以将通过 Docker 构建的镜像部署为推理服务，并将模型文件或代码挂载到服务实例中。

1. 登录 百度百舸·AI 计算平台 控制台，在控制台导航中选择 **大规模训练与推理->在线服务部署**，进入在线服务列表后，单击 **【部署服务】**。



2. 配置服务基础信息

- 填写服务名称
- 选择资源池：资源池类型和对应资源池队列
- 选择资源规格
- 选择是否开启分布式推理
- 是否开启PD分离
- 选择是否开启服务容错，开启服务容错，当节点或加速芯片故障时会封锁此节点，并自动驱逐坏卡或故障节点的实例在健康节点上重建
- 配置优先级



3. 配置环境信息

- 选择服务镜像
- 填写镜像的启动命令

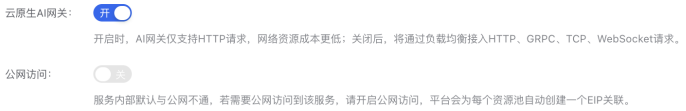
- 如需开启公网需要选择端口
- 添加环境变量
- 选择存储卷设置容器挂载目标路径
- 可选择开启健康检查自动检测并恢复异常状态的容器



4. 配置流量接入信息。

- 选择是否开启云原生AI网关，开启时，AI网关仅支持HTTP请求，网络资源成本更低；关闭后，将通过负载均衡接入HTTP、GRPC、TCP、WebSocket请求。
- 选择是否开启公网访问，服务内部默认与公网不通，若需要公网访问到该服务，请开启公网访问配置访问方式，选择一个弹性公网IP。

流量接入



5. 根据需要选择高级配置。

- 配置最小实例数
- 选择是否开启日志持久化，开启日志持久化功能会将日志持久化保存到日志服务（BLS）中，日志存储、读取、写入和索引会产生费用。
- 配置优雅退出时间
- 选择是否开启资源碎片迁移
- 滚动更新 支持为服务设置滚动更新策略，需要输入滚动更新过程中最大不可用和最大超量实例数比例。

（1）最大不可用：滚动更新过程中不可用实例的数量占预期实例数的百分比，范围1%-100%。示例：预期实例数为10，最大不可用为20%，则更新过程中可用的实例数最少有8个。

（2）最大超量：滚动更新过程中超出预期的实例数量占预期实例数的百分比，范围1%-100%。 示例：预期实例数为10，最大超量为10%，则更新过程中实例数最多可达11个。

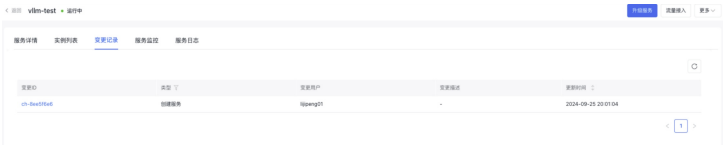
- 设置共享内存大小



第四步：升级服务和手动扩缩容

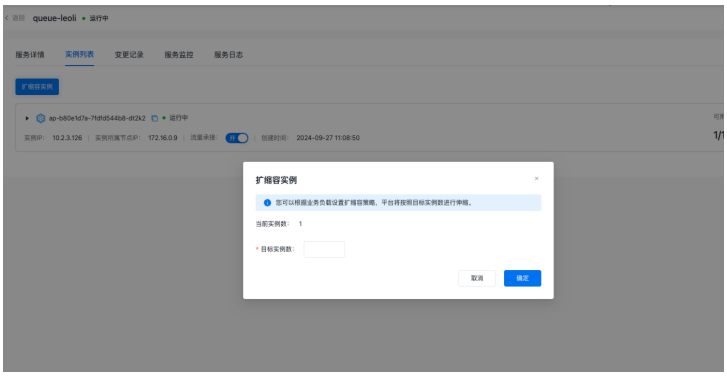
1. 升级服务

服务部署成功后，您可以通过 **升级服务** 修改镜像版本、存储挂载、健康检查、滚动更新策略等信息，填写本次变更内容的描述。 平台会记录详细的版本变更记录，您可在服务详情中查看版本变更记录。



2. 手动扩缩容

服务部署成功后，如果您的服务流量有变化，您可以进行扩缩容操作以满足您的流量负载要求。



更多操作

- 实例摘流
- 流量接入
- 自定义监控及扩缩容指标
- 自定义服务自动扩缩容
- 日志查询及采集
- 模型部署服务
- 服务弹性伸缩

Qwen3.8-Flash-Next

模型介绍

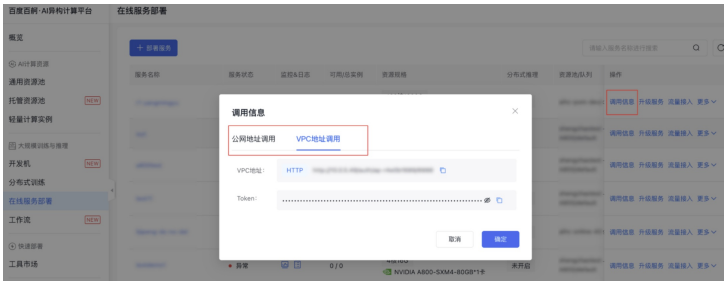
Qwen3.8-Flash-Next 是阿里于2026年8月发布的多模态MoE模型，作为Qwen4架构的预览版。参数量125B（激活参数6B），外加51B N-gram嵌入与4B MTP模块。原生支持262K上下文，可通过YaRN扩展至1M Token，支持图像与视频理解。

模型特性

- **核心能力**：代码编写、智能体任务、专业工作显著增强，SWE-bench Pro达62.5分
- **MoE架构**：512个专家，每次激活10个路由专家+1个共享专家
- **长上下文**：混合注意力机制（Gated DeltaNet + Qwen稀疏注意力），1M Token下吞吐为Qwen3.7-Plus的8.6倍
- **思维控制**：支持`enable_thinking`、`preserve_thinking`、`reasoning_effort`参数
- **视觉语言**：原生支持图像和视频理解

API调用

1. 服务部署成功后，可在服务列表查看调用信息



2. 调用示例

```
 curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: <TOKEN>" \
-d '{
  "model": "Qwen3.8-Flash-Next",
  "messages": [{"role": "user","content": [{"type": "text", "text": "请描述这张图片的内容"},{"type": "image_url", "image_url": {"url": "http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/dog_e941a21.jpeg"}}]},
  "max_tokens": 1024,
  "temperature": 0.7
}'
```

DeepSeek-V4-Pro-0813

模型介绍

DeepSeek-V4-Pro-0813 是 DeepSeek 于 2026 年 8 月 13 日发布的旗舰 MoE 模型（V4 系列正式版，取代 4 月的 V4-Pro 预览版）。参数量 1.6T（激活参数 49B），附带 MTP 模块与 3 层 DSpark 投机解码草稿模块。原生支持 1M 上下文，最大输出 384K Token，仅支持文本输入。

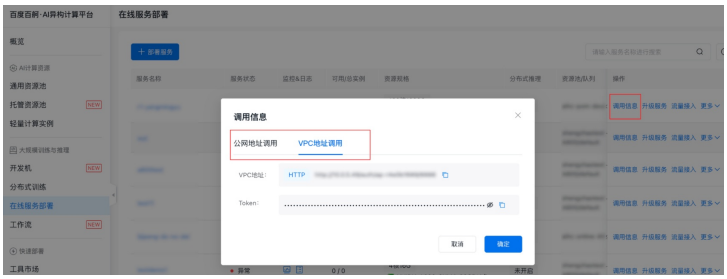
模型特性

- **核心能力**：智能体、代码工程与工具使用显著增强，SWE-bench Verified 达 96.4%（开源权重第一），Terminal-Bench 2.1 达 87.9 分
- **MoE 架构**：384 个路由专家 + 1 个共享专家，每次激活 6 个路由专家，FP4（专家）+ FP8（其余）混合精度
- **长上下文**：混合注意力机制（Compressed Sparse Attention + Heavily Compressed Attention），1M 上下文下单 Token 算力约为 V3.2 的 27%，KV Cache 仅约 10%
- **思维控制**：默认开启思考模式

- **模态支持**：纯文本，暂不支持图像与视频理解

API调用

1. 服务部署成功后，可在服务列表查看调用信息



2. 调用示例

```
 curl -X POST '<访问地址>/v1/chat/completions' \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer <TOKEN>' \
-d '{
    "model": "DeepSeek-V4-Pro-0813",
    "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}],
    "max_tokens": 1024,
    "temperature": 0.7
}'
```

Kimi-K3快速开始卡片描述

模型介绍

Kimi-K3 是月之暗面开源的原生多模态智能体模型，也是该系列目前能力最强的一代。模型总参数量 **2.8T**、激活参数量 **104B**，构建在 Kimi Delta Attention（KDA）与 Attention Residuals （AttnRes）之上，原生具备视觉理解能力，支持 **1M（1,048,576）token 上下文**。 它是业界首个开放权重的 3T 级模型，面向长周期编程、知识工作与复杂推理等前沿场景。

核心特性

- **全新架构**：基于 Kimi Delta Attention（KDA）与 Attention Residuals（AttnRes） ， 并通过 Stable LatentMoE 框架提升 MoE 稀疏度——从 896 个专家中激活 16 个 ， 相比 Kimi K2 获得约**2.5 倍**的整体扩展效率提升。
- **长周期编程**：在极少人工干预下可持续进行长时间工程会话，能在超大代码库中导航并编排终端工具 ， 覆盖 GPU kernel 优化、编译器开发、视觉在环的游戏开发、CAD 乃至芯片设计等场景。
- **智能体知识工作**：支持端到端的知识工作产出，可生成带交互式可视化、组件与仪表盘的深度研究报告 ， 以及动效设计与视频剪辑，由其原生多模态架构驱动。
- **原生多模态与长上下文**：在同一模型内理解文本、图像与视频，支持 1M token 上下文窗口。
- **开放前沿权重**：完整权重以 Kimi K3 License 开源发布。

模型概要

项	值
架构	混合专家模型（MoE）
总参数量	2.8T
激活参数量	104B
层数	93
稠密层数量	1
注意力层构成	69 层 KDA + 24 层 Gated MLA
注意力隐藏维度	7168
注意力头数	96
Latent MoE 维度	3584
MoE 隐藏维度（每个专家）	3072
专家数量	896
每 token 选择的专家数	16
共享专家数量	2
词表大小	160K
上下文长度	1,048,576
注意力机制	KDA & Gated MLA
激活函数	SiTU-GLU
视觉编码器	MoonViT-V2
视觉编码器参数量	401M
量化	MXFP4 权重 / MXFP8 激活（量化感知训练）
支持模态	文本、图像

API调用

服务部署成功后，可在服务列表查看调用信息。

基础对话

⚠ **max_tokens 建议不低于 1024**。Kimi-K3 默认以思考模式运行， 若预算过小会全部被思考内容占满，导致 `content` 为空。

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <TOKEN>" \
-d '{
    "model": "Kimi-K3",
    "messages": [{"role": "user", "content": "请简要回答人类是否能登上火星?"}],
    "max_tokens": 1024,
    "temperature": 0.6
}'
```

返回中思考过程在 message.reasoning_content，最终回答在 message.content。 **流式调用时需要同时消费 delta.reasoning_content 和 delta.content**， 只读 delta.content 会拿不到内容。

多模态（图像理解）

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <TOKEN>" \
-d '{
    "model": "Kimi-K3",
    "messages": [{"role": "user", "content": [
        {"type": "text", "text": "请描述这张图片的内容"},
        {"type": "image_url", "image_url": {"url": "http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/dog_e941a21.jpeg"}}]}],
    "max_tokens": 1024
}'
```

工具调用

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <TOKEN>" \
-d '{
    "model": "Kimi-K3",
    "messages": [{"role": "user", "content": "今天山东济南的天气如何"}],
    "tools": [{"type": "function", "function": {
        "name": "get_weather",
        "description": "Get the current weather in a given location",
        "parameters": {"type": "object", "properties": {
            "location": {"type": "string", "description": "City and state"},
            "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}},
            "required": ["location"]}}}],
    "tool_choice": "auto",
    "max_tokens": 1024
}'
```

工具调用命中时 finish_reason 为 tool_calls，参数在 message.tool_calls[].function.arguments。

GLM-5.3-Flash

模型介绍

GLM-5.3-Flash 是智谱 GLM-5 系列中**首个原生多模态模型**。模型总参数量 **320B**、 激活参数量仅 **18B**，在各项基准与真实业务负载上全面超过上一代 GLM-5.2，而价格仅为其 **十分之一**；在编程与智能体基准上接近 Claude Opus 4.8 的水平。

该模型基于全新训练的基座，架构与训练方案围绕能力和效率重新设计：首次在 GLM 系列中引入 **稀疏注意力与线性注意力的混合架构**，在保持精确长上下文能力的同时大幅降低长上下文服务成本； 并采用 **mHC（Manifold-Constrained Hyper-Connections）**进一步提升扩展效率。 配合最新的 30T token 多模态预训练语料，GLM-5.3-Flash 以更少的算力提供更强的智能。

核心特性

- **原生多模态**：GLM-5 系列首个原生多模态模型，同一模型内理解文本与图像。
- **混合注意力架构**：首次在 GLM 系列引入稀疏注意力 + 线性注意力的混合设计， 显著降低长上下文推理成本，同时保持精确的长上下文能力。
- **mHC 提升扩展效率**：采用 Manifold-Constrained Hyper-Connections，提高模型扩展效率。
- **极高性价比**：总参数 320B 但激活仅 18B，基准表现超过 GLM-5.2，价格约为其 1/10。
- **超长上下文**：支持 1M（1,048,576）token 上下文。
- **思考预算可控**：通过 reasoning_effort 参数控制是否输出思考过程。

模型概要

项	值
架构	混合专家模型（MoE）
总参数量	320B
激活参数量	18B
层数	45
隐藏维度	4096
注意力头数	64
专家数量	288
每 token 选择的专家数	8
注意力机制	稀疏注意力 + 线性注意力混合
特殊结构	mHC（Manifold-Constrained Hyper-Connections）
上下文长度	1,048,576
词表大小	154,880
量化	FP8（E4M3）
支持模态	文本、图像

API调用

服务部署成功后，可在服务列表查看调用信息。

基础对话

GLM-5.3-Flash 默认以最高思考预算运行，思考过程在 message.reasoning_content， 最终回答在 message.content。建议 max_tokens 不低于 2048，避免预算被思考内容占满。

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3-Flash",
    "messages": [{"role": "user", "content": "请用中文简要回答：为什么天空是蓝色的？"}],
    "max_tokens": 2048,
    "temperature": 0.6
}'
```

流式调用时需要同时消费 delta.reasoning_content 和 delta.content， 只读 delta.content 会拿不到思考过程。

控制思考过程

用 reasoning_effort 控制：**不传或传 max 会输出思考过程；传 low 或 high 则不输出** （reasoning_content 为空，回答更快）。传其它值会返回参数校验错误。

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3-Flash",
    "reasoning_effort": "low",
    "messages": [{"role": "user", "content": "用一句话打个招呼"}],
    "max_tokens": 512
}'
```

多模态（图像理解）

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3-Flash",
    "messages": [{"role": "user", "content": [
        {"type": "text", "text": "请描述这张图片的内容"},
        {"type": "image_url", "image_url": {"url": "http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/dog_e941a21.jpeg"}}]}],
    "max_tokens": 2048
}'
```

工具调用

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3-Flash",
    "messages": [{"role": "user", "content": "今天山东济南的天气如何"}],
    "tools": [{"type": "function", "function": {
        "name": "get_weather",
        "description": "Get the current weather in a given location",
        "parameters": {"type": "object", "properties": {
            "location": {"type": "string", "description": "City and state"},
            "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]},
            "required": ["location"]}}}],
    "tool_choice": "auto",
    "max_tokens": 2048
}'
```

工具调用命中时 finish_reason 为 tool_calls，参数在 message.tool_calls[].function.arguments。

GLM-5.3

模型介绍

GLM-5.3 是智谱面向**工程编码与智能体（Agent）场景**的最新旗舰模型。它与上一代 GLM-5.2 使用**同一个基座模型**，全部能力提升来自后训练，因此在保持同等部署成本的前提下， 复杂编码与长程任务能力有明显跃升。

官方数据显示：在智谱自建的 Z.ai Code Bench 上，GLM-5.3 相比 GLM-5.2 提升约 **50%**； 在 Terminal Bench 3.0（28.3 vs GLM-5.2 的 4.6）、Agents' Last Exam、AutomationBench、 GDPval-AA v2 等公开基准上达到**开源权重模型的最优水平**，是构建长程 Coding Agent 与 自主智能体的理想基座。

架构上，GLM-5.3 采用 **MLA 低秩注意力 + DSA 稀疏注意力**的混合专家（MoE）设计， 并在权重中自带 **MTP（Multi-Token Prediction）层**，可直接开启投机解码显著提升生成速度； 支持 **100 万 token** 上下文。

核心特性

- **工程编码与智能体能力领先**：面向真实工程任务优化，在终端任务、长程自主执行、 自动化流程等基准上取得开源权重模型最优成绩。
- **与 GLM-5.2 同基座、纯后训练提升**：部署形态和资源需求与 GLM-5.2 一致， 升级不需要改变部署规格。
- **DSA 稀疏注意力 + MLA**：显著降低长上下文推理的显存与计算开销。
- **自带 MTP 投机解码**：权重内含 MTP 层，开启后实测平均接受 3.35 个草稿 token， 单流生成速度可达 300+ tokens/s。
- **超长上下文**：支持 1,048,576（1M）token。
- **思考过程可控**：通过 reasoning_effort 参数控制是否输出思考过程。

模型概要

项	值
架构	混合专家模型（MoE），MLA + DSA 稀疏注意力
层数	78
隐藏维度	6144
注意力头数	64
专家数量	256（另含 1 个共享专家）
每 token 选择的专家数	8
投机解码	权重自带 MTP 层（1 层）
上下文长度	1,048,576
词表大小	154,880
量化	FP8（E4M3，block 128×128）
支持模态	文本

API调用

服务部署成功后，可在服务列表查看调用信息。

基础对话

GLM-5.3 默认以最高思考预算运行，思考过程在 `message.reasoning_content`，最终回答在 `message.content`。建议 `max_tokens` 不低于 2048，避免预算被思考内容占满。

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3",
    "messages": [{"role": "user", "content": "请用中文简要回答：为什么天空是蓝色的？"}],
    "max_tokens": 2048,
    "temperature": 0.6
}'
```

流式调用时需要同时消费 `delta.reasoning_content` 和 `delta.content`，只读 `delta.content` 会拿不到思考过程。

控制思考过程

用 `reasoning_effort` 控制：**不传或传 `max` 会输出思考过程；传 `low` 或 `high` 则不输出**（`reasoning_content` 为空，回答更快）。传其它值会返回参数校验错误。

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3",
    "reasoning_effort": "low",
    "messages": [{"role": "user", "content": "用一句话打个招呼"}],
    "max_tokens": 512
}'
```

多轮对话场景建议显式传 `chat_template_kwargs: {"clear_thinking": true}`，让上一轮的思考内容在下一轮被清除。

工具调用

```
curl -X POST "<访问地址>/v1/chat/completions" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer token" \
-d '{
    "model": "GLM-5.3",
    "messages": [{"role": "user", "content": "今天山东济南的天气如何"}],
    "tools": [{"type": "function", "function": {
        "name": "get_weather",
        "description": "Get the current weather in a given location",
        "parameters": {"type": "object", "properties": {
            "location": {"type": "string", "description": "City and state"},
            "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}},
            "required": ["location"]}}}],
    "tool_choice": "auto",
    "max_tokens": 2048
}'
```

工具调用命中时 `finish_reason` 为 `tool_calls`，参数在 `message.tool_calls[].function.arguments`。

长上下文

服务默认以 131072（128K）上下文启动，实测 4.5 万 token 的输入可正常检索作答。 需要更长上下文时，可在部署时调大启动命令里的 `--context-length`（模型上限 1,048,576），并相应评估显存。

DeepSeek-V4.1-Flash

模型简介

DeepSeek-V4.1-Flash 是 DeepSeek 发布的 MoE 大模型（原生 fp8 权重），面向高并发、长上下文的在线推理场景，支持最长 1048576（1M）token 上下文，自带 DSpark 三级草稿，配合投机解码可显著提升单流吞吐。模型自带视觉模块（多模态），可处理图文输入。权重约 475 GiB。

部署步骤

1. 在百舸「快速开始」中选择 DeepSeek-V4.1-Flash 卡片。
2. 选择加速芯片，规格按上表自动带出，确认队列有不少于 4 张空闲卡。
3. 提交部署。首次冷启动需从 BOS 拉取约 475 GiB 权重并加载，耗时较长（分钟级到小时级，视 BOS 冷读速度），启动探针已为冷读预留充足窗口，耐心等待服务就绪即可。

4. 服务就绪后，通过网关地址 + 鉴权 token 调用，model 字段填 DeepSeek-V4.1-Flash。

调用示例

基础对话（将 <GATEWAY>、<TOKEN> 替换为部署后分配的网关地址与鉴权 token）：

```
curl -s https://<GATEWAY>/v1/chat/completions \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer <TOKEN>' \
-d '{
  "model": "DeepSeek-V4.1-Flash",
  "messages": [{"role": "user", "content": "请用中文简要解释：为什么天空是蓝色的？"}],
  "max_tokens": 2048,
  "temperature": 0.6
}'
```

function call：

```
curl -s https://<GATEWAY>/v1/chat/completions \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer <TOKEN>' \
-d '{
  "model": "DeepSeek-V4.1-Flash",
  "messages": [{"role": "user", "content": "今天山东济南的天气如何？"}],
  "tools": [{"type": "function", "function": {"name": "get_weather", "description": "Get the current weather in a given location", "parameters": {"type": "object", "properties": {"location": {"type": "string"},
"unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}}, "required": ["location"]}}}],
  "tool_choice": "auto"
}'
```

注意事项

- model 名称必须填 DeepSeek-V4.1-Flash（与卡片一致），否则按名调用会 404。
- 该模型的思考内容并入正文 content 返回，没有单独的 reasoning_content 通道；按常规对话解析即可。
- 不建议强制模型"只给最终答案、不要推理"：实测在禁推理约束下多步算术易出错，放开后恢复正常。涉及计算、推理的提问建议允许模型分步作答。
- 已验证能力：中文无乱码、function call、流式输出、监控指标、长上下文检索（1M 上下文内暗号可准确召回）。
- 首次部署冷启动较慢属正常现象，根因是大体积权重首次 BOS 冷读；同节点二次部署因页缓存变热会明显加快。

产品定价

计费概述

计费项

百舸平台的计费项由节点和云盘两部分组成，它们独立计费。

计费项	计费说明	计费方式
百舸节点	以百舸节点的形式提供，包括 CPU、内存和 GPU，收取节点规格费用。	包年包月（预付费） 按量付费（后付费）
百舸云盘	安全可靠的高弹性存储服务，按容量和类型收费。	按量付费（后付费）

其他使用资源（如 PFS、CFS、CCR、CProm、BLS 等）付费详情可以参见相关资源价格文档。

计费模式

百舸平台的资源提供灵活的计费模式，用户可根据任务需求选择。

计费模式	资源付费方式	说明
包年包月	预付费	按订单的购买周期计费。
按量付费	后付费	按分钟计费，每小时出账，支持转为包年包月。

支付方式

购买百舸平台的资源时，您可以选择以下支付方式：

- 百度智能云账户余额：可在管理控制台的财务总览处查看当前余额。
- 在线支付：支持多种主流在线支付渠道。
- 代金券支付：请确保代金券的状态为使用中且订单类型为通用。

计费方式

包年包月（预付费）

计费规则

包年包月（预付费）计费方式，需要提前一次性支付所选时间段内的费用。可选择 1～9 个月或 1～5 年进行付费，计费周期自实例创建日起按所选单位周期计算。该模式价格相较于按量付费（后付费）模式更低廉，适合业务稳定的长期需求。

计费价格

实例包年包月（预付费）价格信息请联系百度支持人员。

到期提醒和处理

- 到期提醒：服务到期前 7 天，系统会给您发送即将到期提醒通知。
- 到期后处理：到期后立即停服，系统会发送到期停服通知。数据为您保留7天，期间不收取费用，7天内未充值则释放，释放前1天和释放时系统都会发送释放通知。

按量付费（后付费）

计费规则

按量付费（后付费）计费方式，是按实例的实际使用量进行收费，按分钟计费，不足 1 分钟按 1 分钟计费，每小时出账。按量付费（后付费）不需要提前支付费用，但需要保证账户余额大于 100 元。该模式比包年包月（预付费）模式价格高一些，适合业务瞬间有较大变化的场景，资源使用完即可立即释放，降低成本。

计费价格

实例按量付费（后付费）价格信息请联系百度支持人员。

预留实例券

预留实例券介绍

预留实例券（Reserved Instance Voucher，简称 RI）是一种能够抵扣按量付费节点账单的抵扣权益，同时提供资源的预留保障。可以通过承诺在一定期限内（如 1 年、3 年等）使用特定配置的资源，来获取相比按量付费更低的折扣价格。

如果有长期稳定的 AI 训练或推理业务，或者对资源供应的确定性有极高要求，开通预留实例券是优化成本和保障交付的最佳选择。

适用范围

适用范围概念优势可用区级实例券生效后将自动匹配并抵扣相同 AZ、实例规格以及 EHC 集群（仅适用 EHC 类型实例）的按量计费实例账单支持资源预留，购买预留实例券后，百度云将保证您账号下在对应的 AZ 中至少能有与 RI 相同的实例规格 * 台数的按量计费资源。

付费方式

RI 当前仅支持 **0 预付** 的付费方式。

付费类型定义适用场景0预付购买时无需支付费用，费用按小时扣除（承诺消费）。不希望初期投入资金，但仍需长期资源保障的场景。

资源抵扣

实例券生效后将自动匹配相同 AZ、实例规格的按量付费（后付费）实例进行抵扣，如您购买的是 EHC 实例规格，则实例券会自动匹配同 EHC 集群的实例。您无需再为被抵扣的实例额外付费。在预留实例券到期后，按量付费（后付费）实例将按照实例费用出账并恢复计费。

付费方式比较

付费方式	价格折扣	资源保障	计费解耦	资金压力	灵活性
包年包月	高折扣	强保障	实例与账单绑定	高（需一次性支付）	不可提前释放
按量付费	无折扣，价格最高	无保障	实例与账单绑定	低（用多少付多少）	随时释放
预留实例券（RI）	中高折扣（视付款方式而定）	强保障（AZ 级锁定库存）	账单与实例解耦，自动匹配	灵活（支持 0 预付）	券与实例分离，实例可释放重建

操作指南

快速开始

概述

概述

本文档介绍百度百舸快速开始模块的功能说明、计费与权限要求，以及通过快速开始一键部署模型的操作流程。快速开始模块集成了众多 AI 开源社区及百舸官方优质的模型、AI 工具，支持零代码快速实现模型部署、模型开发等能力，帮助用户更快、更高效、更便捷地体验百舸平台核心产品能力。

功能介绍

快速开始提供了丰富的模型库，涵盖 LLM、AIGC、计算机视觉等多种 AI 场景。您可以对模型进行一键训练和部署，从而快速验证业务需求。

计费说明

注意： 快速开始本身不收费，但通过快速开始进行模型训练或部署操作时，会使用用户账号下已有的资源，用户仅需要为已有资源付费。

权限说明

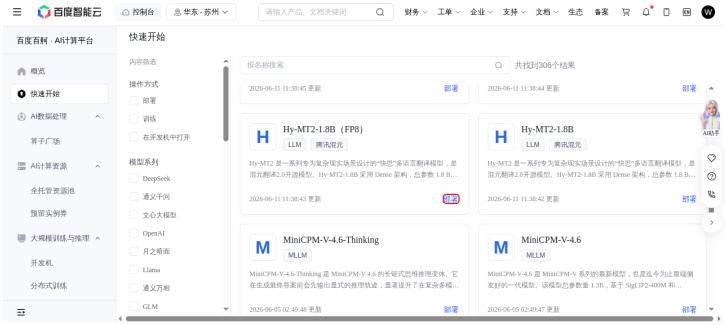
快速开始本身不需要额外授权，但通过快速开始进行模型训练或部署操作时，需要开通相关资源权限。

通过快速开始进行模型部署操作时，会在**【在线服务部署】** 产品模块下拉起对应的模型服务部署任务，因此需要开通在线服务部署相关权限。

一键部署模型

操作步骤如下。

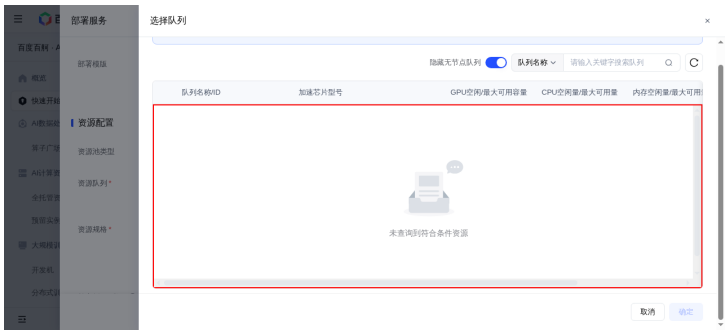
1. 在**【快速开始】** 页面，通过**内容筛选** 勾选**【部署】**，选择支持部署操作的模型模板，单击模板卡片上的**【部署】**。



2. 在**【部署服务】** 抽屉中确认系统自动带入的服务名称、推理引擎、部署模版等信息，并按页面要求选择资源池类型、资源队列、资源规格、单实例 Pod 数、模型加载加速和 AI 原生网关等配置项。



3. 在**资源队列** 字段中选择与当前模板兼容的队列。若资源队列无可选项，请先准备符合模型部署要求的资源队列，再继续选择资源规格和 AI 原生网关。



4. 确认配置无误后，单击**【确定】** 提交部署任务。

5. 部署提交成功后，选择**大规模训练与推理->在线服务部署->在线服务**，查看已部署的服务。您可以在服务列表中对服务执行查看、编辑、更新等后续操作。



Nvidia Isaac Lab 快速部署

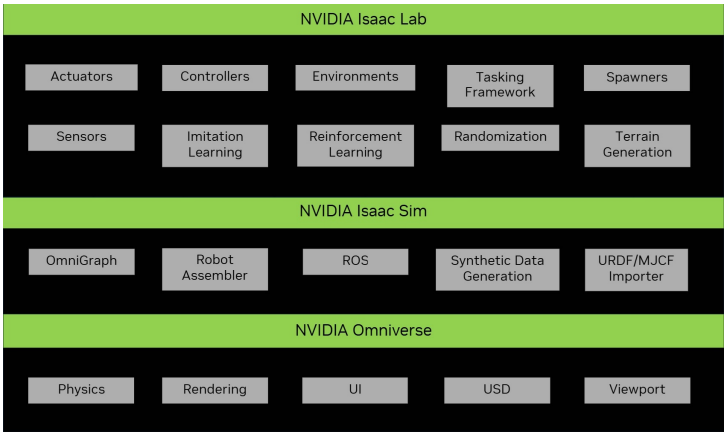
工具介绍

> Isaac Lab 是由 NVIDIA 推出的基于Isaac Sim的开源机器人研究框架，基于 GPU 加速实现精确仿真，统一和简化强化学习、模仿学习和运动规划等机器人研究工作流程，推动机器人 sim-to-real（仿真到现实）的高效迁移。

Isaac Lab 是由 NVIDIA 推出的基于Isaac Sim的开源机器人研究框架，基于 GPU 加速实现精确仿真，统一和简化强化学习、模仿学习和运动规划等机器人研究工作流程，推动机器人 sim-to-real（仿真到现实）的高效迁移。

其优势主要包括：

1. **机器人形态支持全面：**兼容**16+ 主流机器人形态**，涵盖机械臂（如 Franka）、四足（如 ANYmal）、人形机器人（如 Atlas）等；
2. **训练环境丰富：**提供 **30+ 开箱即用环境**，这些环境支持主流强化学习框架训练，如RSL RL、SKRL、RL Games或Stable Baselines等；
3. **支持复杂对象仿真：**支持刚体、铰接系统和柔性物体等复杂仿真对象；
4. **多传感器系统支持：**支持RGB/深度/分割相机、相机注释、IMU、接触传感器、光线投射器。



部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	8核	建议按表单默认值及以上
内存	64G	建议按表单默认值及以上
GPU	**Isaac Lab 2.1.0 :** A10 / A100 / A800 / H20 / H800 / L20 / 支持RT Core的GPU卡 ×1 **Isaac Lab 2.2.0 :** L20/支持RT Core的GPU卡 ×1	按需
CDS	按需	按需
其它	无	无

注意：部署该实例的集群需要具备访问公网的能力，配置方式见：https://cloud.baidu.com/doc/CCE/s/plmk196bx

> Isaac Lab 2.2.0 依赖于 Isaac Sim 5.0，该仿真环境在渲染时使用了 DLSS（超采样技术，Deep Learning Super Sampling），因此推荐使用支持RT Core的GPU卡。

服务网卡配置

由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置HTTP & HTTPS网络代理支持。服务网卡配置参考

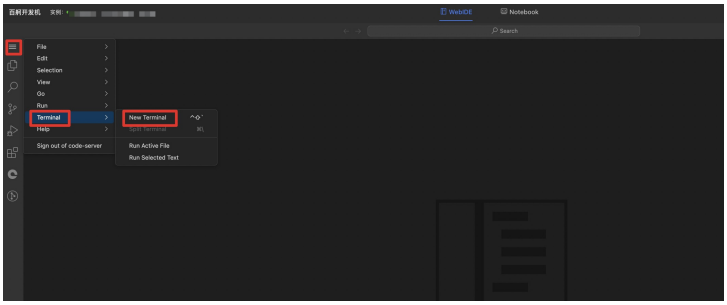
使用说明

本文以 **快速启动预定义训练任务**和**创建Project并训练推理** 两个案例，介绍Isaac Lab的使用过程。

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，打开Terminal界面：



主要路径介绍

- isaac lab源码路径：/workspace/isaaclab

快速启动训练任务

Isaac Lab官方提供了丰富的Tutorials 供用户学习和试用，执行如下命令，展示 Isaac Lab 中所有可用的强化学习 (RL) 和指令学习 (IL) 任务

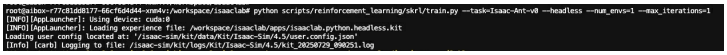
```
 cd /workspace/isaaclab
./isaaclab.sh -p scripts/environments/list_envs.py --keyword <search_term>
```



以预定义环境Isaac-Ant-v0为例，执行如下命令启动训练。这里设置的num_envs和max_iterations仅为了快速验证，--headless确保代码在无GUI的环境下正常运行：

```
python scripts/reinforcement_learning/skrl/train.py --task=Isaac-Ant-v0 --headless --num_envs=1 --max_iterations=1
```

训练输出包含运行环境、仿真场景配置、观测空间、终止条件、奖励函数、强化学习模型定义，以及日志保存路径等信息，如下所示：



举例：使用强化学习训练智能体

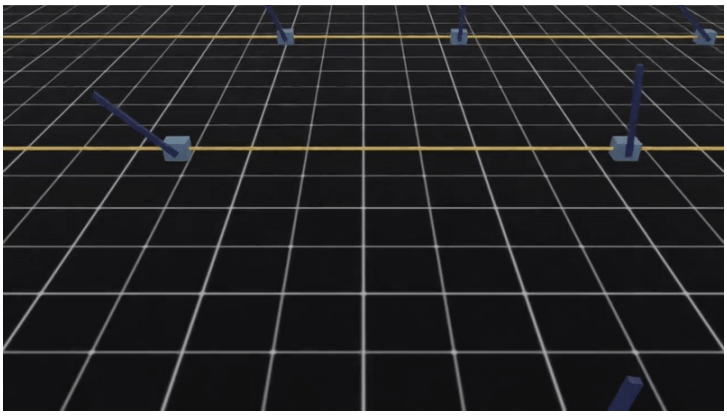
我们从 Stable-Baselines3 训练一个 PPO 智能体来解决倒立平衡任务.官方链接:Training with an RL Agent

```
 cd /workspace/isaaclab
###### 执行训练（ --headless指定无gui，--video指定渲染视频,录制智能体在训练过程中行为视频）
./isaaclab.sh -p scripts/reinforcement_learning/sb3/train.py --task Isaac-Cartpole-v0 --num_envs 64 --headless --video
```

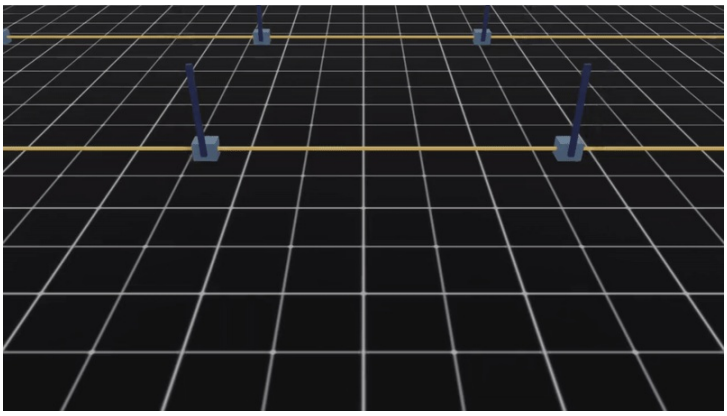
视频已保存到该./logs/sb3/Isaac-Cartpole-v0/<run-dir>/videos/train目录。您可以使用任何视频播放器打开这些视频。

可视化结果如下：

优化前：



优化后：



创建新的Project

提示：新建Project的train.py和play.py都提供了丰富的参数设置，可以通过--help查看，其中，--info可以看到更详细的输出信息，可用于运行环境错误分析。

环境创建&注册

执行如下命令，选择任务类型、Project属性和工作流等即可完成一个Project的创建。

```
./isaaclab.sh --new
```

以当前创建的project为例，task type选择external，workslow选择轻量的Direct | single-agent，然后确定RL库和算法等。这里我们将Project name设置为test，下文如无特殊说明，test一般指代Project name。

```
[INFO] Running template generator...

? Task type: External
? Project path: /workspace/
? Project name: test

      RL environment features support according to Isaac Lab workflows

| Environment feature                          | Direct | Manager-based |
|----------------------------------------------|--------|---------------|
| Single-agent                                 | yes    | yes           |
| Multi-agent                                  | yes    | no            |
| Fundamental/composite spaces (apart from 'Box') | yes | no            |

? Isaac Lab workflow: Direct | single-agent

                    Supported RL libraries

| RL/training feature          | rl_games | rsl_rl | skrl                  | sb3      |
|------------------------------|----------|--------|-----------------------|----------|
| ML frameworks                | PyTorch  | PyTorch | PyTorch, JAX          | PyTorch  |
| Relative performance         | ~1X      | ~1X    | ~1X                   | ~0.03X   |
| Algorithms                   | PPO      | PPO    | AMP, IPPO, MAPPO, PPO | PPO      |
| Multi-agent support          | yes      | no     | yes                   | no       |
| Distributed training         | yes      | no     | yes                   | no       |
| Vectorized training          | yes      | yes    | yes                   | no       |
| Fundamental/composite spaces | no       | no     | yes                   | no       |

? RL library: skrl
? RL algorithms for skrl: AMP, PPO

Validating specification...
Generating external project...
  |-- Copying repo files...
  |-- Copying scripts...
  |-- Copying extension files...
  |-- Generating tasks...
  |    |-- Generating 'Template-Test-Direct-v0' task...
  |-- Copying vscode files...
Setting up git repo in /workspace/test path...
  |  Initialized empty Git repository in /workspace/test/.git/

--------------------------------------------------------------------------------
Project 'test' generated successfully in /workspace/test path.
See /workspace/test/README.md to get started!
--------------------------------------------------------------------------------
```

```
.
├── README.md
├── scripts
│   ├── list_envs.py
│   ├── random_agent.py
│   ├── skrl
│   │   ├── play.py
│   │   └── train.py
│   └── zero_agent.py
└── source
    └── test
        ├── config
        │   └── extension.toml
        ├── docs
        │   └── CHANGELOG.rst
        ├── pyproject.toml
        ├── setup.py
        └── test
            ├── __init__.py
            ├── tasks
            │   ├── __init__.py
            │   └── direct
            │       ├── __init__.py
            │       └── test
            │           ├── __init__.py
            │           ├── agents
            │           │   ├── __init__.py
            │           │   ├── skrl_amp_cfg.yaml
            │           │   └── skrl_ppo_cfg.yaml
            │           ├── test_env.py
            │           └── test_env_cfg.py
            └── ui_extension_example.py

11 directories, 20 files
```

按照如上方式，会创建一个cartpole（倒立摆）基准测试，可用于快速确认环境配置的正确性。在路径test/source/test/test/tasks/direct/test下可以看到**init**.py，该初始化函数用于自定义环境的注册：

```
 import gymnasium as gym

from . import agents

gym.register(
    id="Template-Test-Direct-v0",
    entry_point=f"{__name__}.test_env:TestEnv",
    disable_env_checker=True,
    kwargs={
        "env_cfg_entry_point": f"{__name__}.test_env_cfg:TestEnvCfg",
        "skrl_amp_cfg_entry_point": f"{agents.__name__}:skrl_amp_cfg.yaml",
        "skrl_cfg_entry_point": f"{agents.__name__}:skrl_ppo_cfg.yaml",
    },
)
```

然后，执行如下命令将其安装为python模块，便于后续启动训练和推理：

```
 cd /workspace/test
python -m pip install -e source/test
```

成功安装后，使用如下命令可以看到该环境已经注册到环境列表中，后面会使用Task Name作为task参数启动训练和推理：

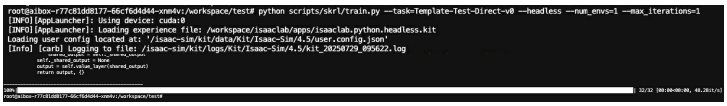
```
 cd /workspace/test
python scripts/list_envs.py
```



启动训练

执行如下命令启动训练，类似地，这里设置的num_envs和max_iterations仅为了快速验证。训练结束后，checkpoints会保存在logs目录下。

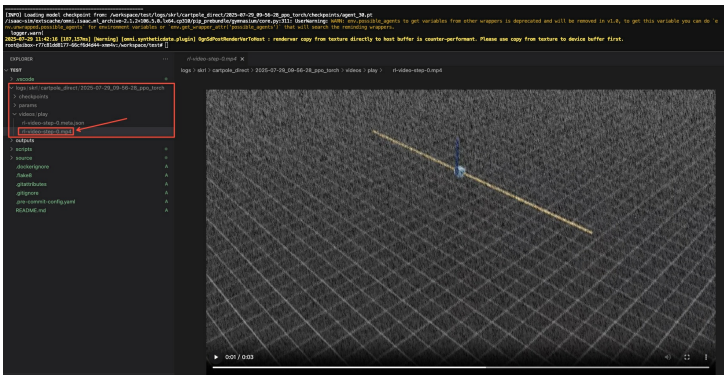
```
 cd /workspace/test
python scripts/skrl/train.py --task=Template-Test-Direct-v0 --headless --num_envs=1 --max_iterations=1
```



启动推理

执行如下命令启动训练后模型的推理。参数 --video 可以确保在推理任务结束后，在logs/videos目录下保存一个仿真视频：

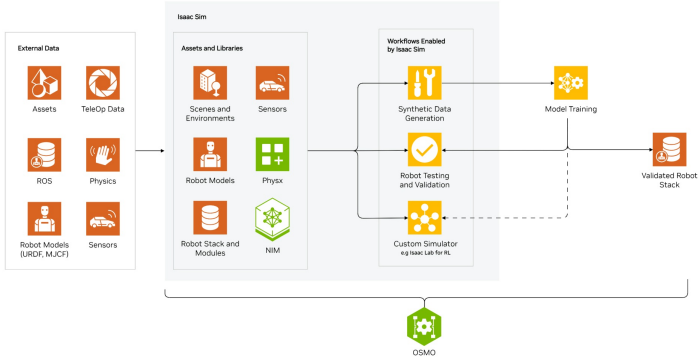
```
 cd /workspace/test
python scripts/skrl/play.py --task=Template-Test-Direct-v0 --headless --num_envs=1 --video
```



Nvidia Isaac Sim 快速部署

工具介绍

> NVIDIA Isaac Sim是基于NVIDIA Omniverse构建的机器人仿真平台。通过GPU加速物理引擎来实现精确仿真，并支持大规模、多传感器RTX渲染。集成数据生成、强化学习、ROS集成和数字孪生模拟等端到端工作流程。



NVIDIA Isaac Sim是基于NVIDIA Omniverse构建的机器人仿真平台，支持从URDF、MJCF和CAD等

常见格式导入机器人系统。通过GPU加速物理引擎来实现精确仿真，并支持大规模、多传感器RTX渲染。集成数据生成、强化学习、ROS集成和数字孪生模拟等端到端工作流程。其优势包括：

- **资产导入&导出：**支持从非USD格式导入机器人和环境，或将其导出为非USD格式；
- **机器人微调：**优化机器人的物理精度、计算效率或真实感；
- **机器人仿真：**提供全流程运动控制工具链，如控制器、运动生成和运动学求解器以及策略集成接口；
- **传感器系统：**支持RTX和物理学传感器。

部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	建议8核及以上	按需
内存	建议64G及以上	按需
GPU	**Isaac Sim 4.5.0 :** A10 / A100 / A800 / H20 / H800 / L20 / 支持RT Core的GPU卡 ×1 **Isaac Sim 5.0.0 及以上版本:** L20/支持RT Core的GPU卡 ×1	按需
CDS	按需	按需
其它	无	无

> Isaac Sim 5.0及以上版本的仿真渲染依赖DLSS（超采样技术，Deep Learning Super Sampling），因此推荐使用支持RT Core的系列显卡。
>
> **服务网卡配置** 由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置HTTPS网络代理支持。服务网卡的配置参考

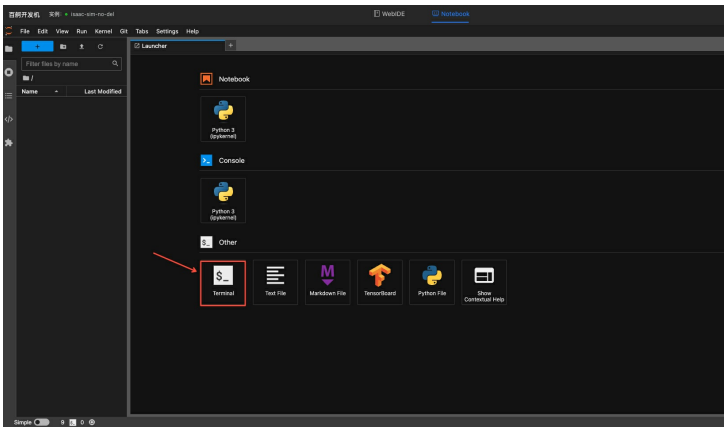
使用说明

本文以纯python脚本形式启动两个任务为例，简单介绍Isaac Sim的使用。

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置。

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，打开Terminal界面：



主要路径介绍

- Isaac Sim USD资产挂载路径：/mnt/bos/isaac-sim，该目录下存有Issac Sim预置的部分USD资产。

快速开始

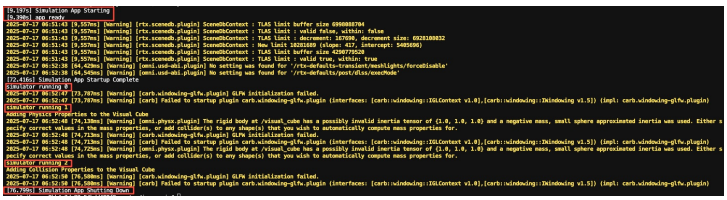
立方体碰撞模拟 该任务主要模拟了不同属性的立方体，并逐步为其中一个视觉立方体添加物理属性（刚体属性）和碰撞属性的过程。

执行如下命令即可启动任务：

```
cd ~/workspace/quick_start
/isaac-sim/python.sh my_getting_started.py
```

> 相比standalone_examples/tutorials/getting_started.py，my_getting_started.py设置当前仿真环境为headless模式。

运行结果如下所示：



2.2 机械臂&小车运控模拟 该任务主要模拟了机械臂和小车在同一环境下的运动控制。

执行如下命令即可启动任务：

```
cd ~/workspace/quick_start
/isaac-sim/python.sh my_getting_started_robot.py
```

> 相比standalone_examples/tutorials/getting_started_robot.py，my_getting_started_robot.py的改动主要在以下三个地方：
>
> 1. 设置当前仿真环境为headless模式；
> 2. 用USD挂载路径代替函数get_assets_root_path()，避免网络访问限制导致任务运行失败；

3. 用GroundPlane(prim_path="/World/GroundPlane", z_position=0)代替my_world.scene.add_default_ground_plane()，创建地面对象，避免网络访问限制导致任务运行失败。

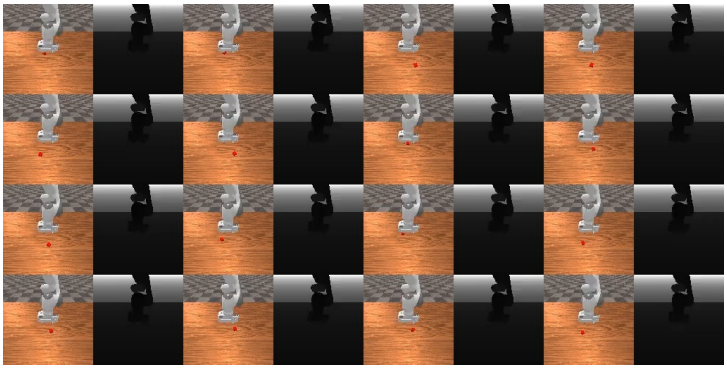
运行结果如下所示：



ManiSkill 快速部署

工具介绍

ManiSkill是一个由SAPIEN支持的**开源机器人模拟与训练框架**，通过GPU并行化技术实现高吞吐仿真和数据采集，支持多样化机器人任务和仿真-现实闭环迁移，并提供丰富的预置学习算法。



ManiSkill是一个由SAPIEN支持的机器人**模拟**和**训练**开源框架。其特点包括：

1. **超高速数据采集。**基于 GPU 并行化技术，在 NVIDIA 4090 GPU 上可实现 30,000+ FPS 的 RGBD + 语义分割数据采集，大幅提升视觉数据生成效率。
2. **GPU 并行化异构仿真**
    1. **状态型合成数据生成**：支持高吞吐量的仿真数据采集（如物体位姿、物理状态）。
    2. **异构场景并行**：每个并行环境可运行**完全不同的物体场景**（如随机物体布局、多样化任务配置）。
3. **多样化任务覆盖**
    1. **机器人形态**：人形机器人、移动机械臂、单臂机器人等。
    2. **任务类型**：桌面操作（抓取/推挤）、清洁/绘图、灵巧操作（如精细物体操控）。
4. **开发者友好 API**
    1. 通过**面向对象设计**抽象 GPU 内存管理，提供简洁的任务构建接口，降低开发复杂度。
5. **仿真-现实闭环**
    1. **Real2Sim**：将真实策略部署到 GPU 仿真中，实现 **100 倍加速**的性能评估。
    2. **Sim2Real**：支持将在仿真中训练的策略无缝迁移到真实机器人。
6. **预置机器人学习基线**
    1. **强化学习**：PPO、SAC、TD-MPC2 等。
    2. **模仿学习**：行为克隆（BC）、扩散策略（Diffusion Policy）。
    3. **大模型支持**：视觉语言动作模型（VLAs），如 **Octo**、**RDT-1B**、**RT-x**。

官方文档：https://maniskill.readthedocs.io/en/v3.0.0b21/user_guide/getting_started/quickstart.html#

部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	按需	建议8核及以上
内存	按需	建议32GB及以上
GPU	按需	本案例主要以A800为实践（推荐）
CDS	按需	按需
其它	无	无

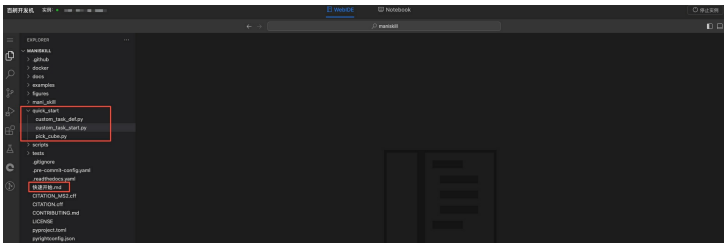
使用说明

本文以GPU并行仿真和创建并启动自定义任务为例，介绍ManiSkill的使用过程

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置。

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，打开/root/workspace/maniskill，即可看到给出的示例代码目录quick_start：

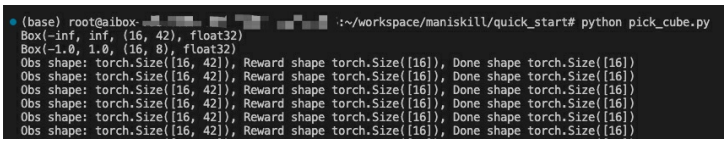


启动GPU并行化仿真任务

该示例主要介绍ManiSkill的并行化仿真功能，打开IDE终端，执行如下命令：

```
 ##### 首先确保在正确的python环境下
##### conda activate base
cd /root/workspace/maniskill/quick_start
python pick_cube.py
```

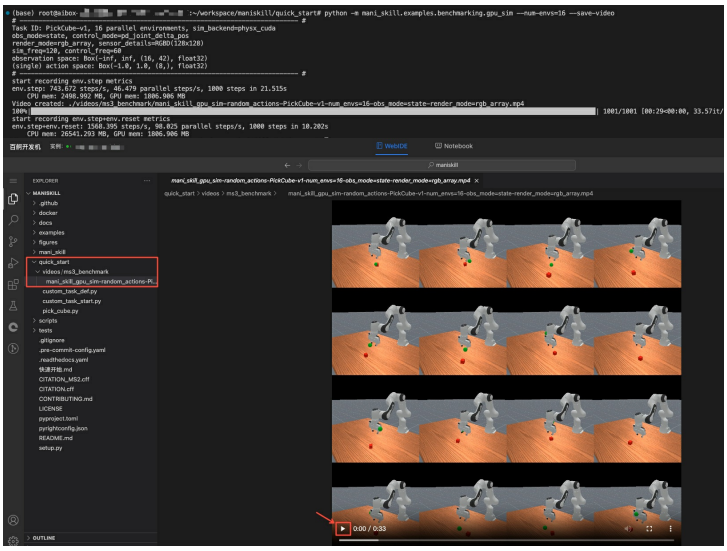
可以看到结果如下所示，具体地，对于任务进行过程中的obs、reward等中间输出，可以自行修改代码查看：



为了可视化地查看任务，可以直接执行如下命令：

```
 ##### 首先确保在正确的python环境下
##### conda activate base
python -m mani_skill.examples.benchmarking.gpu_sim --num-envs=16 --save-video
```

任务完成/超时后，可看到渲染得到的.mp4视频，点击播放即可查看：



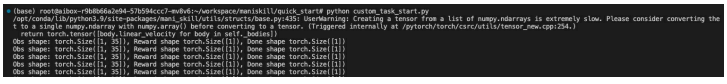
构建并启动自定义任务

示例代码quick_start/custom_task_def.py定义了一个简单的"推立方体"的任务，也即以强化学习的方式训练机械臂从立方体的后面将其推动到目标区域。代码中已经包含必要的注释以帮助理解一个自定义任务的构建，这里不再赘述。

quick_start/custom_task_start.py的代码逻辑与quick_start/pick_cube.py比较类似，可参考。

执行如下命令启动任务：

```
 ##### 首先确保在正确的python环境下
##### conda activate base
cd /root/workspace/maniskill/quick_start
python custom_task_start.py
```



具体对于任务进行过程中的obs、reward等中间输出，可以自行修改代码查看。

LeRobot 数据格式转换

工具介绍

> LeRobot 是 Hugging Face 推出的、基于 PyTorch的开源具身智能学习框架。目前「快速开始」主要支持数据集格式转换功能，支持将HDF5、RLDS等多种格式的数据集转换为LeRobotDataset格式。

它的主要特征在于：

1. **多模态算法集成。**融合模仿学习、强化学习及 Transformer 架构，支持机器人通过视觉等多模态数据学习操作逻辑（如模仿人类抓取、自主强化试错）。
2. **全流程开发工具链**
    1. **预训练模型**：提供开箱即用的机器人控制模型，降低开发门槛；

2. **数据集与模拟器**：内置多传感器时序数据集，兼容 Gazebo 等仿真环境，支持虚拟调试与算法迭代；
3. **硬件兼容性**：模块化设计适配机械臂、人形机器人等硬件，开源硬件文档助力系统搭建。

3. **具身智能典型场景。**专注物体操作（如抓取、传递、堆叠）与空间导航，在 AlohaTransferCube（物体传输）、PushT（物体操纵）等任务中实现高效交互控制。

部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	8核	建议按表单默认值及以上
内存	32GB	建议按表单默认值及以上
GPU	目前数据转换功能无GPU要求	目前数据转换功能无GPU要求
CDS	按需	按需
其它	无	无

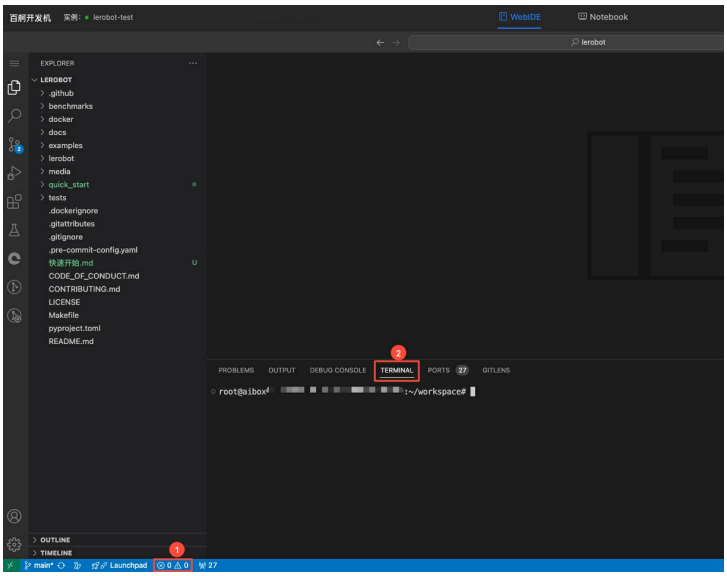
使用说明

本文以 amandlek/mimicgen_datasets/robot/square_d1_iiwa.hdf5 的数据转换为例，介绍LeRobot数据转换功能的使用过程

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，并打开VScode中的terminal 。



启动数据转换脚本

以转换amandlek/mimicgen_datasets下的mimicgen/robot/square_d0_iiwa.hdf5为例：

- 转换脚本路径：lerobot/quick_start/data_convert.py，以下为脚本中引用的文件
  - 源数据集路径src_root：/mnt/dataset/mimicgen/robot
  - 源数据集文件h5_f：square_d0_iiwa.hdf5
  - 目标数据集路径tgt_path：/root/workspace/lerobot/quick_start/output/

在terminal中执行数据转换脚本

```
 cd /root/workspace/lerobot/quick_start/
python data_convert.py
```

脚本执行完成即可查看转换结果

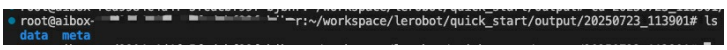
温馨提醒

1. 脚本data_convert.py中有详细的注释以帮助理解，实际业务可以根据自己的需要进行更改
2. 每次脚本启动在linux命令和脚本data_convert.py内都会默认重置清空目标文件夹下的历史数据，实际业务请根据需要调整

查看转换结果

在转换输出路径/root/workspace/lerobot/quick_start/output/{TIMESTAMP}/下，可以看到：

- data：转换后数据
- meta：转换元信息



Isaac Sim 4.5.0 可视化实践

案例介绍

> NVIDIA Isaac Sim是基于NVIDIA Omniverse构建的机器人仿真平台，本文是在开发机上基于isaac sim实现可视化能力的实践教程

NVIDIA Isaac Sim是基于NVIDIA Omniverse构建的机器人仿真平台，支持从URDF、MJCF和CAD等常见格式导入机器人系统。通过GPU加速物理引擎来实现精确仿真，并支持大规模、多传感器RTX渲染。集成数据生成、强化学习、ROS集成和数字孪生模拟等端到端工作流程。其优势包括：

- **资产导入&导出：**支持从非USD格式导入机器人和环境，或将其导出为非USD格式；
- **机器人微调：**优化机器人的物理精度、计算效率或真实感；
- **机器人仿真：**提供全流程运动控制工具链，如控制器、运动生成和运动学求解器以及策略集成接口；
- **传感器系统：**支持RTX和物理学传感器。

官方地址：

https://docs.isaacsim.omniverse.nvidia.com/4.5.0/index.html

部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	建议8核及以上	按需
内存	建议64G及以上	按需
GPU	Isaac Sim 4.5.0：A800 / L20 / 支持RT Core的NVIDIA GPU卡 1卡及以上	按需
CDS	按需	按需
其它	无	无

使用需知：

1. Isaac Sim 4.5.0的仿真渲染依赖DLSS（超采样技术，Deep Learning Super Sampling），因此推荐使用GeForce RTX系列显卡（A800没有RT core，测试效果有噪点问题，不推荐使用）
2. 本案例使用L20等Ada Lovelace架构三代RT Core的显卡，且使用的BLB网络带宽为50Mbps时，无噪点，无卡顿，表现更为符合期望
3. isacc官方要求，显卡驱动版本需要高于535.129 ，如果Ubuntu版本>=22.04.5 kernel 6.8.0-48-generic，则驱动至少要求在535.216.01以上
4. 535驱动版本中（550无此问题），含535.255及以上的驱动版本，存在一个已知的显卡驱动判断错误，需要添加--/rtx/verifyDriverVersion/enabled=false参数，跳过显卡驱动判断（操作方式见后续步骤）
5. 官方未对cuda和cudnn有限制，不同的cuda版本和cudnn版本是否对可视化界面的表现有影响未验证
6. 由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置HTTPS网络代理支持访问。服务网卡的配置参考

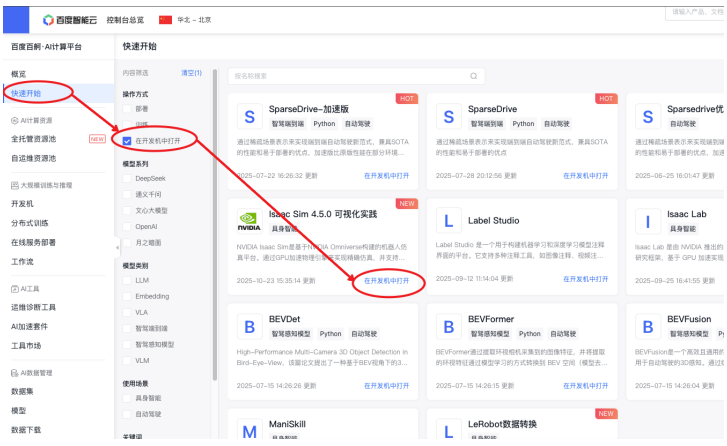
使用说明

本文会先介绍如何【启动可视化客户端】，然后以【执行官方四足和人形可操作演示】、【宇树GR1机器人简易操作演示】两个例子，简单介绍Isaac Sim的使用。

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置

1. 创建与登录开发机

1.1 步骤一：从快速开始创建一个【Isaac Sim 4.5.0 可视化实践】开发机实例：



1.2 步骤二：填写表单

- GPU选择： 建议使用 NVIDIA L20
- 其它选项值建议参考截图表单填写(官方推荐8核64G)，保证运行所需的基础资源配置
- **启用SSH**，填入创建好的公钥，选择支持**公网访问**的BLB（建议选择网络带宽≥50Mbps的BLB，带宽越大越好，越少共享复用越好）
- 其它表单项默认即可



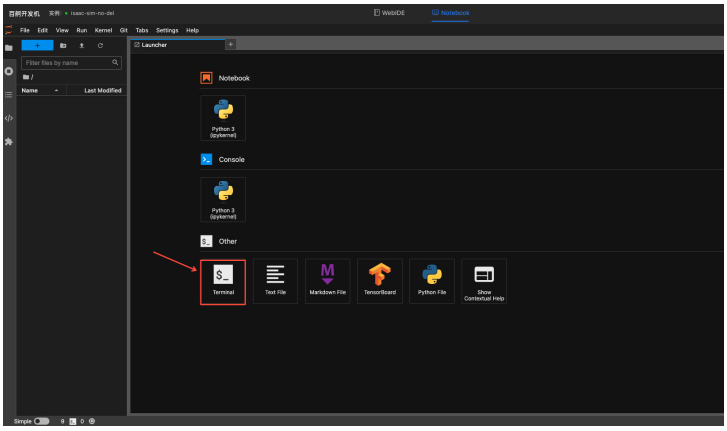
1.3 步骤三：创建并登录开发机 填写完表单，提交后，在开发机列表可见创建的开发机，等待资源调度并部署成功后，点击登录，即可进入开发机webIDE



主要路径介绍

- Isaac Sim USD资产挂载路径：/mnt/bos/isaac-sim，该目录下存有Isaac Sim预置的部分USD资产。

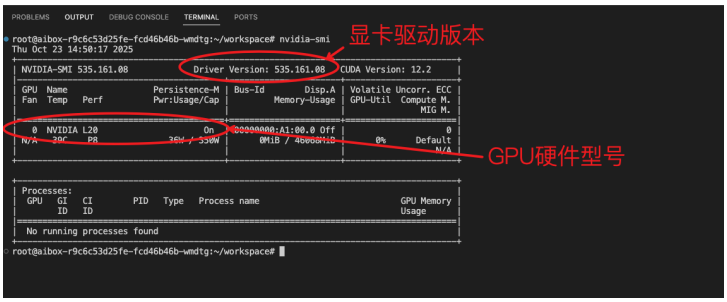
1.4 验证GPU硬件和显卡驱动版本 根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，打开Terminal界面：



在打开的Terminal界面内输入以下命令，确认NVIDIA的GPU硬件和显卡驱动版本

```
nvidia-smi
```

执行结果查看，红圈中是显卡驱动版本和GPU硬件型号



请参考我们下面提出的建议，选择驱动版本

1. isacc官方要求，显卡驱动版本需要高于535.129 ，如果Ubuntu版本>=22.04.5 kernel 6.8.0-48-generic，则驱动至少要求在535.216.01以上

2. 535驱动版本中（550无此问题），含535.255及以上的驱动版本，存在一个已知的显卡驱动判断错误，需要添加--/rtx/verifyDriverVersion/enabled=false参数，跳过显卡驱动判断（操作方式见后续步骤）

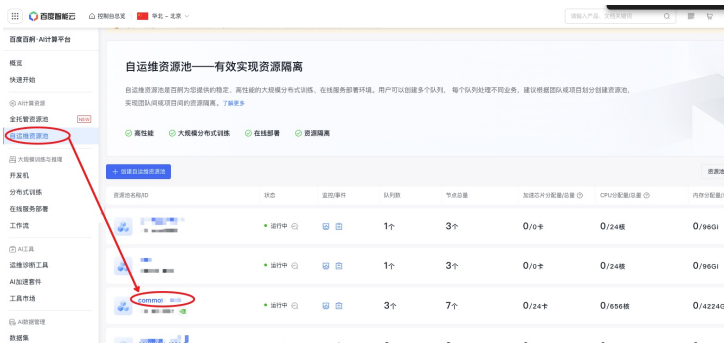
3. 官方未对cuda和cudnn有限制，不同的cuda版本和cudnn版本是否对可视化界面的表现有影响未验证

1.5 节点调整显卡驱动版本，即重装系统（如果驱动版本满足，可跳过） 如显卡驱动不符合建议，则建议重装节点驱动： 建议使用百舸的驱动推荐版本，如果推荐的驱动无法满足业务需求，可进入BCC进行重装操作，有更多的版本选择

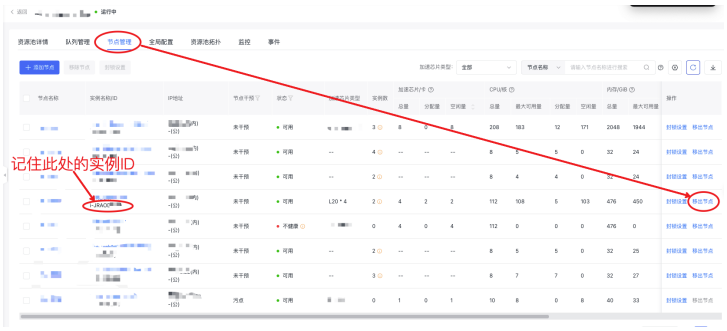
重装步骤： 从资源池中移除节点 -> 在bcc中重装系统并调整显卡驱动版本 -> 将节点重新加入资源池

备注：因为全托管资源池使用的推荐驱动版本和Isaac sim客户端的适配性没有问题，所以此处只覆盖自运维资源池的重装步骤

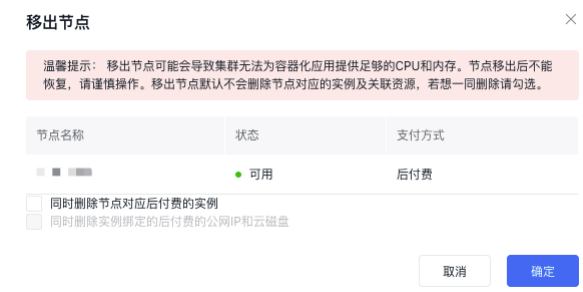
1.5.1 从资源池中移除节点 1、在百舸页面中，选择开发机所在的资源池，点击进入详情界面



2、点击希望重装驱动的节点（eg：如果GPU卡选择了L20，则需要更新所有L20的节点），记录下来**实例ID**（整个流程都会用到），操作移出节点，弹出窗口



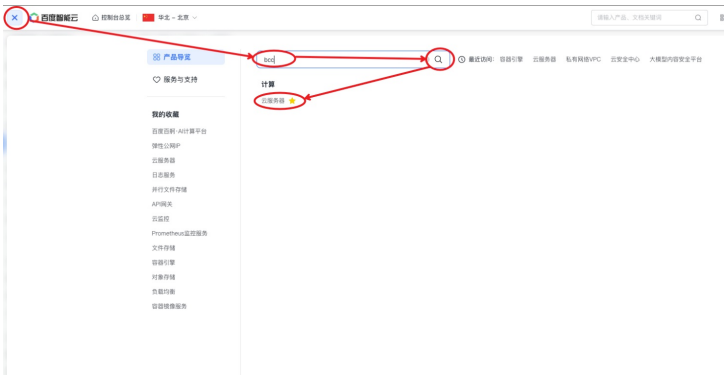
3、在下面弹窗中，无需勾选复选框，直接点击确定即可



4、等待移出完成（需要刷新页面或者用右上角刷新按钮才能看到状态更新）



1.5.2 在bcc中重装系统并调整显卡驱动版本 1、进入bcc：点击左上角，打开导航页，搜索bcc，然后点击【云服务器】，进入bcc界面



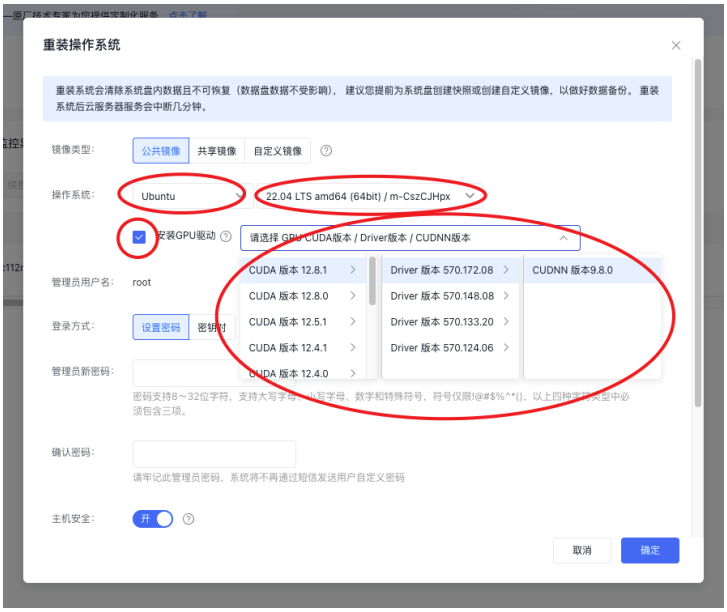
2、找到对应节点，并操作

使用刚才记录的**实例ID**，在bcc中搜索，并操作【重装操作系统】（重装操作系统只有移出节点之后才能操作）



3、系统和驱动版本选择

重装操作系统，建议选择Ubuntu的22.04版本作为os，然后选择目标Driver版本，cuda版本和CUDNN版本可按需选择，对isaac sim可视化客户端没有影响



4、设置root密码，点击确定，开始重装系统



5、安全验证，输入短信验证码



6、开始重装后，此处状态会显示为【重建中】，等到状态变回【运行中】，即可将节点重新加入资源池

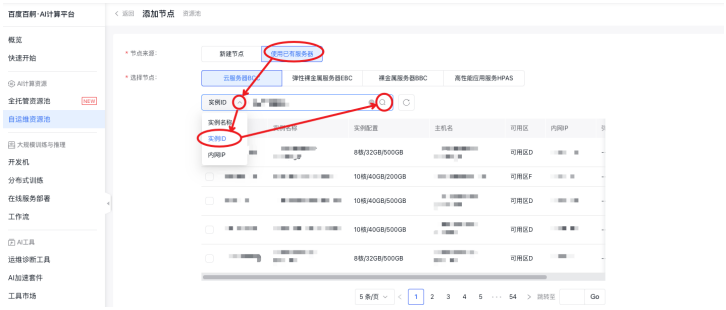


节点页面

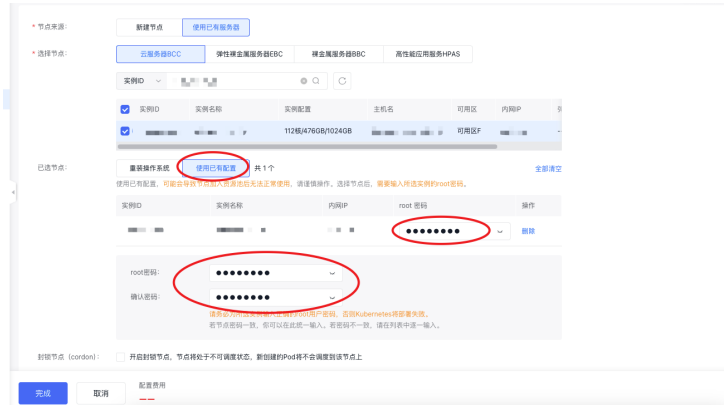
1.5.3 节点重新加入资源池 1、回到百舸资源池，节点管理界面，点击添加节点按钮，进入添加



2、点击【使用已有服务器】，通过**实例ID**，找到我们刚刚重装的机器（有时实例不在【云服务器BCC】选项卡中，可以尝试【弹性裸金属服务器EBC/裸金属服务器BBC/高性能应用服务HPAS】这几个选项卡都找一下有没有）



3、注意选择【使用已有配置】，不要再重装操作系统以免覆盖掉我们刚才选择的显卡驱动，输入刚才重装时设置的root密码，点击完成，开始加入



4、等待节点加入完成，状态为未干预，且可用，即可开始使用



2. 启动客户端

2.1 启动vnc远程桌面 2.1.1 启动vnc服务端 在Terminal界面内输入以下命令，开启vnc服务

```
##### 镜像中TurboVNC服务器，可以使用以下命令设置或修改密码（会被提示可以设置一个 view-only 密码，可以跳过）；
##### 如果不在这一步设置一个密码，在实际启动vncserver的时候就会被要求现设置密码
##### /opt/TurboVNC/bin/vncpasswd ~/.vnc/passwd

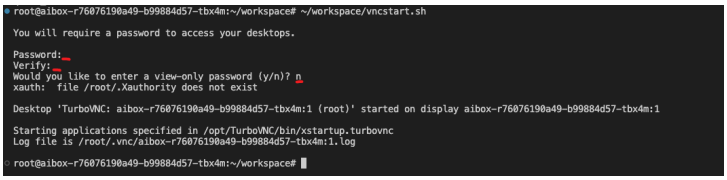
##### 如果之前没有设置过turboVNC的密码，这里会要求设置密码和只读密码（当前先不设置只读密码）例如密码： aihclite，建议只读
##### /opt/TurboVNC/bin/vncserver :1 -geometry 1920x1080 -depth 24  #参数1 会默认选择端口5901

##### 每次在停止开发机之前，都要用这个命令关闭vnc服务端
##### 如果没有关闭，则下次开机再想启动vnc服务，就会碰上【/tmp/.X1-lock】和【/tmp/.X11-unix/X1】两个文件锁，需要删除了文件锁之后才能再次开启vnc服务端
##### /opt/TurboVNC/bin/vncserver -kill :1

##### 使用下面的脚本启动vnc客户端，客户端会自动清理锁文件
##### 但如果:1已经在运行一个vnc服务，那么脚本会提示【A VNC server is already running as :1】并退出，不做任何事情
~/workspace/vncstart.sh
##### 如果希望修改启动参数，例如修改默认分辨率为1024*768，或者将24色修改为16色，则可以在【/opt/TurboVNC/bin/vncserver -kill :1】关掉之前的vnc服务之后，使用下面带参数的命令重新启动vnc服务
##### ~/workspace/vncstart.sh -geometry 1024*768 -depth 16
```

初次开启vnc服务，会要求输入密码（密码限制了8位长度），以及选择是否添加只读密码

按以下操作完成密码设置，建议不添加只读密码（建议**选n**）



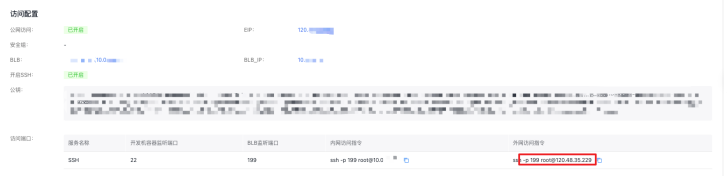
这个步骤完成后，vnc远程桌面就会在开发机的5901端口等待响应

为安全等考量，不建议将5901直接公开到外网上，下面会演示如何通过ssh将端口投射到本地，然后通过本地连接远程桌面

2.1.2 使用本地ssh将开发机的5901端口投射到本地 本地（Windows或Mac）命令行启动ssh作为转发通道（在使用vnc期间，命令行的ssh需要持续开启）：

```
##### 用本地环境的命令行执行这个ssh，-L是将开发机的5901监听连接到本地，后面的root、ip和-p部分根据实际开发机给出的公网访问命令执行
ssh -L 5901:127.0.0.1:5901 root@开发机公网IP地址 -p 开发机公网端口（可在开发机详情页复制）
```

开发机详情页公网IP和公网端口位置（在创建开发机时，选择开启ssh，并选择一个能连上公网的BLB，才能看到下面的部分）：

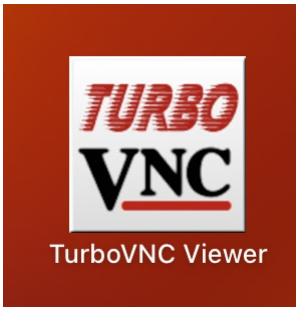


2.1.3 下载TurboVNC客户端到本地并安装，并链接vnc服务 TurboVNC客户端下载地址：

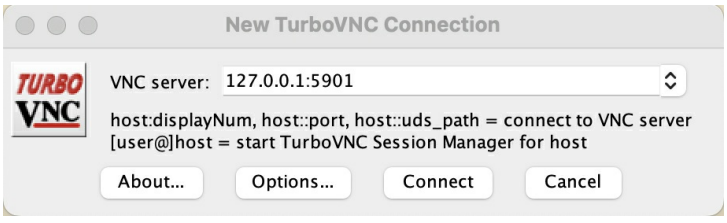
https://github.com/TurboVNC/turbovnc/releases 找到相应版本（建议选择3.2版本），点击展开版本卡片最下方的Assets链接，可以看到各OS下载地址。当前版本TurboVNC Viewer v3.2 (20250506)。

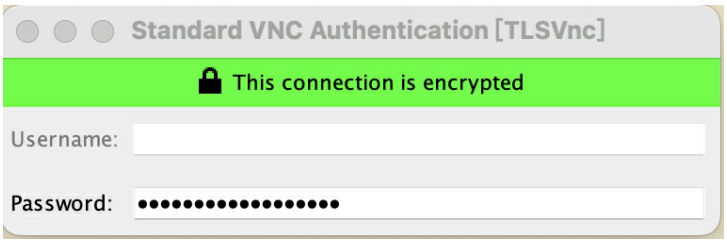


下载安装完成后，会有两个快捷图标，点击如下图标：

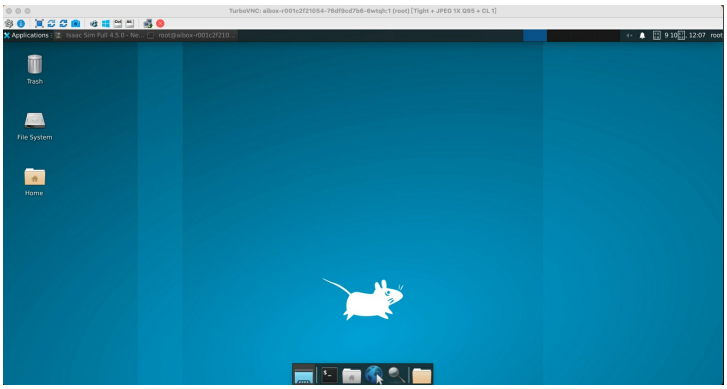


点击启动后，点击connect进行远程连接：

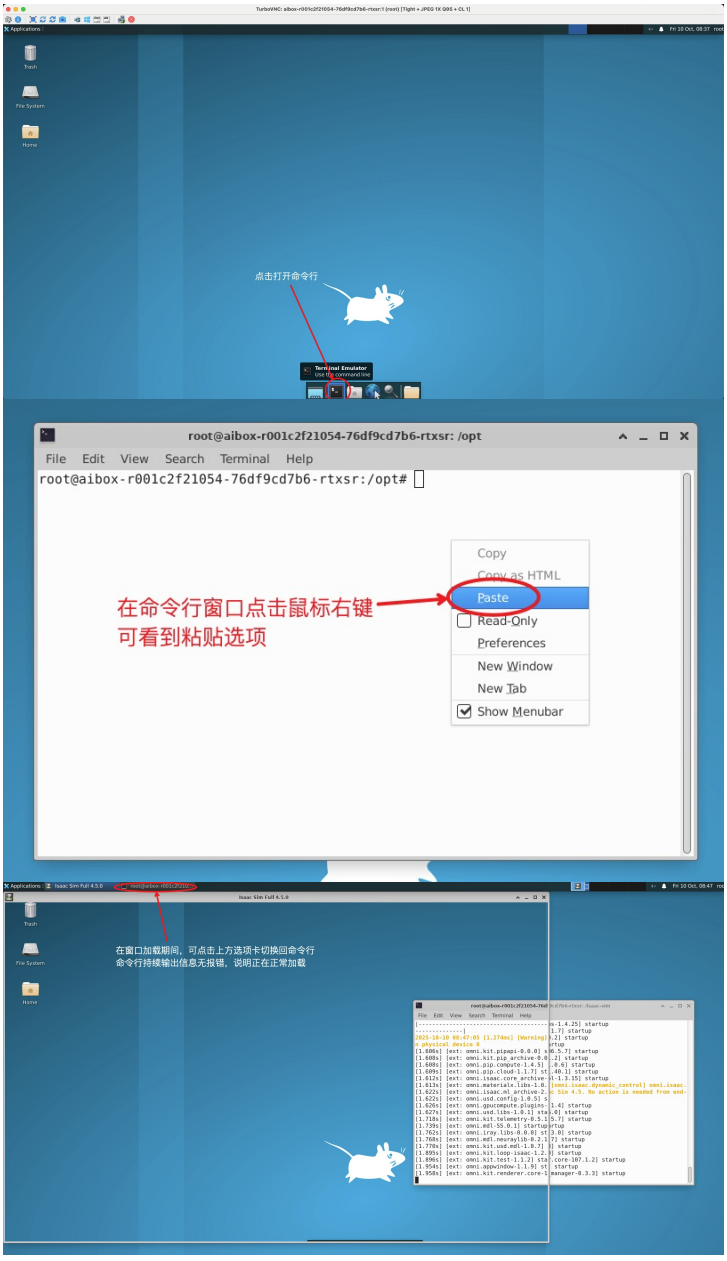




连接成功后：



2.2 启动isaac-sim客户端 在上面已经打开的vnc界面中，执行如下操作，从命令行启动isaac-sim客户端IsaacSim可视化界面：



可视化界面为先出现窗口边框，加载过程结束后再出现窗口内容，创建开发机第一次加载可能会需要几分钟(下载一些服务依赖)，之后再启动一般需要十秒左右。另外，IsaacSim在初始化和使用过程中可能会访问外网，**建议设置HTTP和HTTPS网络代理**

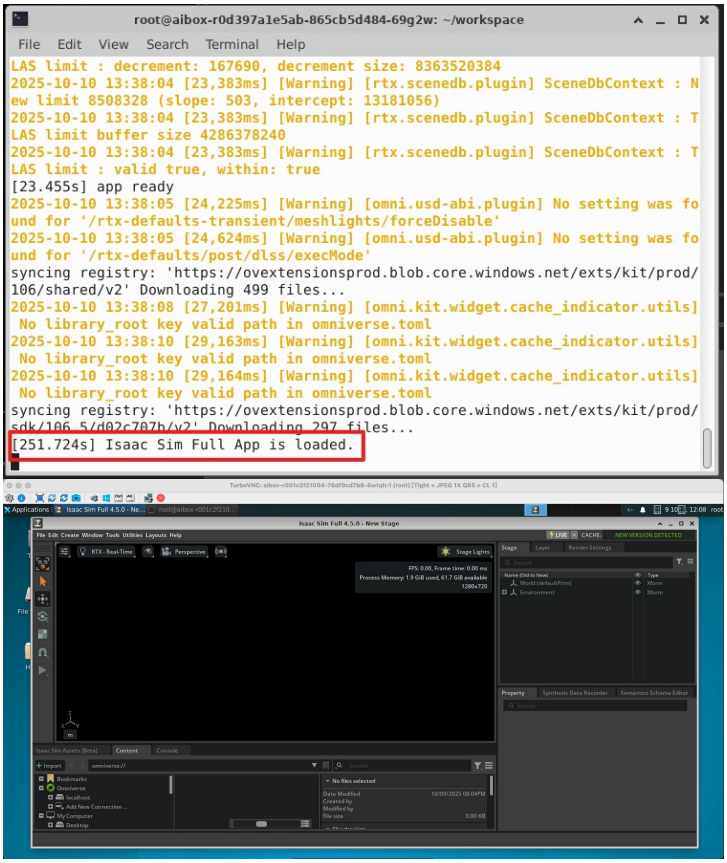
```
##### 先设置网络代理，再启动isaac-sim服务
/isaac-sim/isaac-sim.sh --allow-root
```

请使用nvidia-smi命令查看显卡驱动版本，如显卡驱动版本为535.261.03则建议使用

```
 ##### 先设置网络代理，再启动isaac-sim服务
/isaac-sim/isaac-sim.sh --allow-root --/rtx/verifyDriverVersion/enabled=false
```

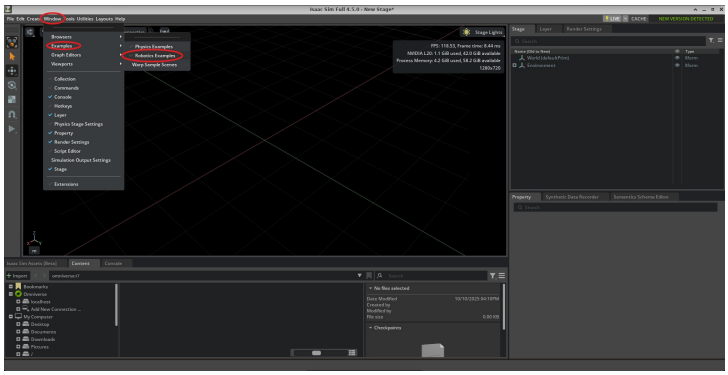
启动结果：

命令行出现红框内字样，说明启动成功



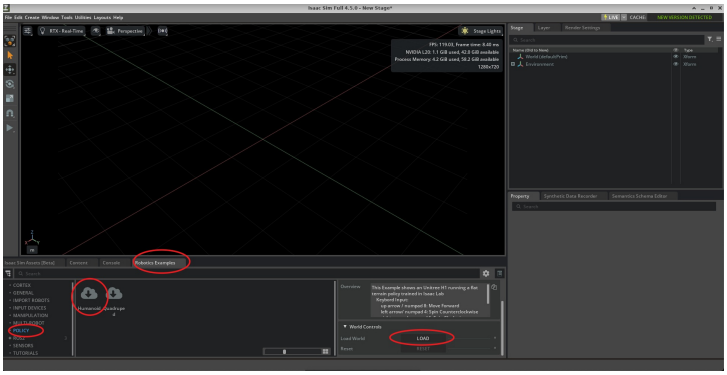
3. 案例演示

3.1 执行官方四足和人形可操作演示 3.1.1 步骤一：开启机器人案例选项卡



3.1.2 步骤二：加载

Humanoid人形机器人案例 点击Load按钮加载，Reset按钮重置案例（或选择右侧的Quadruped四足机器人案例，也是点击Load按钮加载，Reset按钮重置案例）



一般来说，对一个模型或者脚本点击Load按钮，客户端窗口会停止响应一些时间进行网络加载

但在开发机刚新建时，这里的加载时间过长，也有几率是客户端卡死

如果客户端卡死，则可以点击右上角的x关闭客户端后，在命令行再次执行命令启动，一般一到两次之后，一旦有一次Load成功，之后客户端就不会再卡死



3.1.3 步骤三：操作机器人 窗口缩放操作：

- 在窗口内使用鼠标滚轮

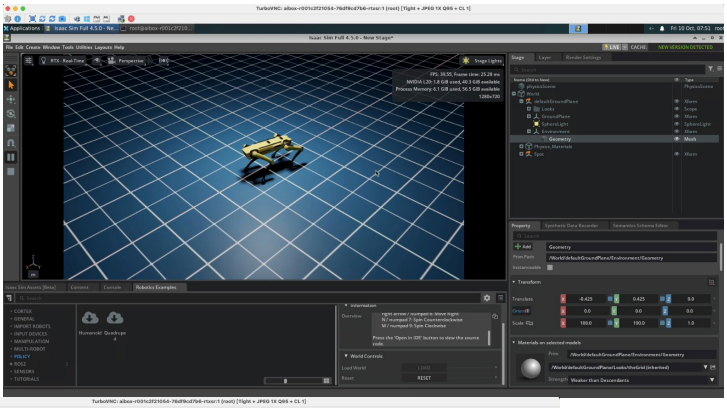
Humanoid人形机器人操作：

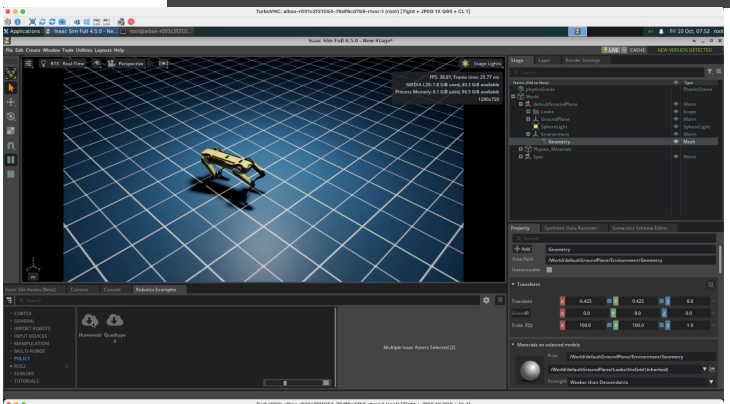
- 前进：向上箭头/数字键 8
- 左转：左箭头/数字 4
- 右转：右箭头/数字 6

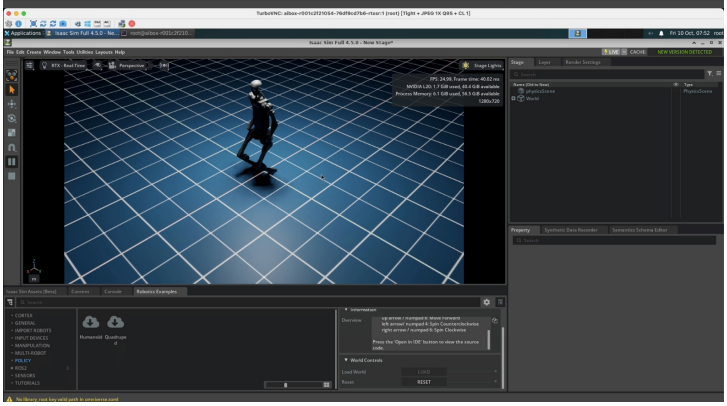
Quadruped四足机器人操作：

- 前进：向上箭头/数字键 8
- 后退：后退箭头/数字 2
- 向左移动：左箭头/数字 4
- 向右移动：右箭头/数字键 6
- 左转：N / NUM 7
- 右转：M / NUM 9

3.1.4 运行效果图片&视频



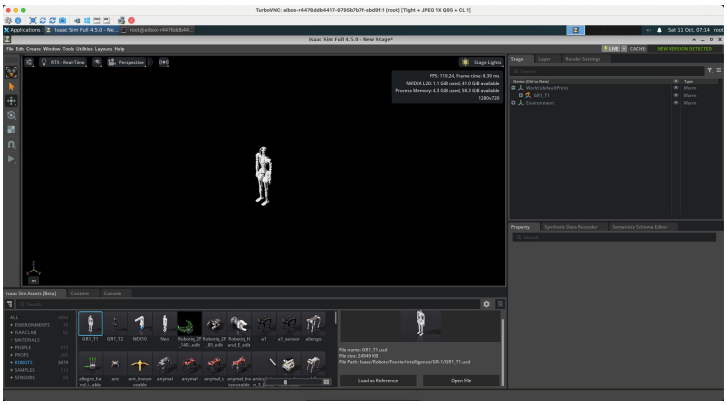




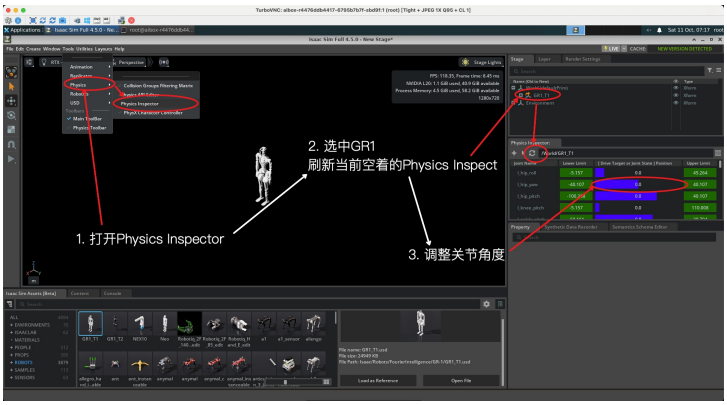
3.2 宇树GR1机器人简易操作演示 3.2.1 步骤一： 在Assets中找到GR1机器人，并加载到工作空间中 操作步骤：



操作结果：

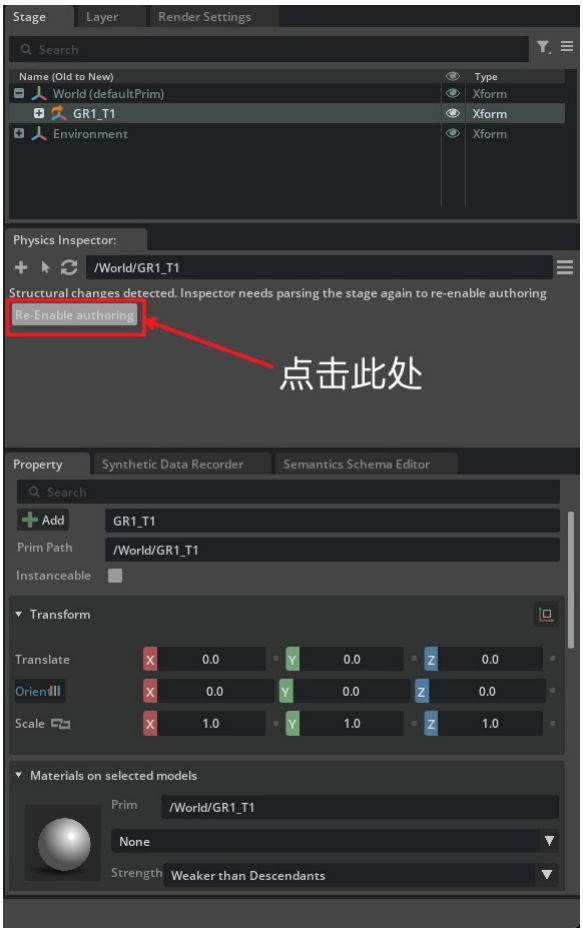


3.2.2 步骤二： 使用Physics Inspector工具，对关节进行简易操作 操作步骤：

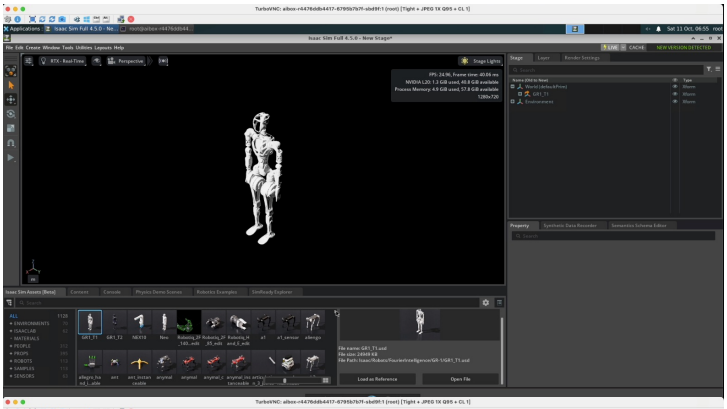


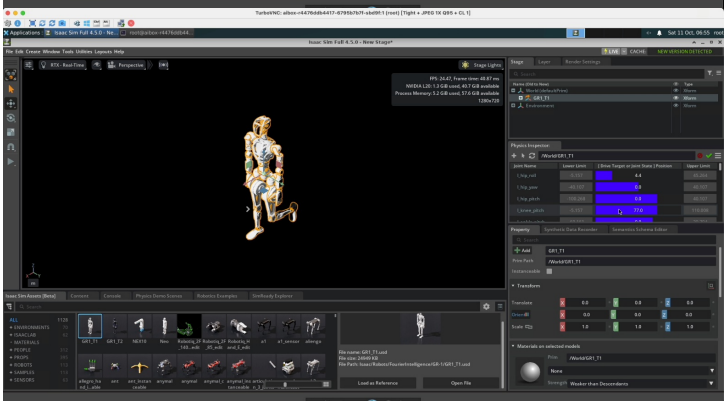
额外步骤：

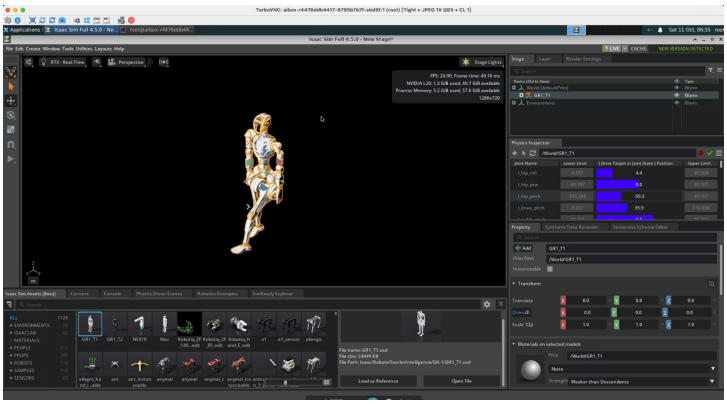
如果按照上面操作后，Physics Inspector显示成下面的样子，则需要点击一下【Re-Enable authoring】按钮，就会显示关节角度界面


Stage
Layer
Render Settings
Search
Name (Old to New)
Type
World (defaultPrim)
Xform
GR1_T1
Xform
Environment
Xform
Physics Inspector:
/World/GR1_T1
Structural changes detected. Inspector needs parsing the stage again to re-enable authoring
Re-Enable authoring
点击此处
Property
Synthetic Data Recorder
Semantics Schema Editor
Search
Add
GR1_T1
Prim Path
/World/GR1_T1
Instanceable
Transform
Translate
Orient
Scale
Materials on selected models
Prim
/World/GR1_T1
None
Strength
Weaker than Descendants

操作结果图片&视频：







Robotwin 2.0 快速实践

Robotwin 2.0 快速实践

RoboTwin2.0 是一款面向机器人学习的仿真与数据工具集，可快速构建虚拟机器人环境、生成高质量训练数据，并支持模仿学习等算法的开发与验证。

工具介绍

RoboTwin2.0 是一款面向机器人学习的仿真与数据工具集，可快速构建虚拟机器人环境、生成高质量训练数据，并支持模仿学习等算法的开发与验证。

开发机配置 & 最佳实践建议

组件类别	部署建议
CPU	8 核及以上
内存	64G 及以上
GPU	L20等带有rtx core的芯片
存储	按需

使用说明

本文会先介绍如何【创建开发机】，然后以【DP】、【ACT】模型部署为例，简单介绍Robotwin 2.0的使用。

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置

创建与登录开发机 1.1 步骤一：从快速开始创建一个【Robotwin 2.0】开发机实例： 在百度百舸.快速开始，找到 RoboTwin 2.0点击右侧【在开发机中打开】在右侧弹出菜单中填写开发机配置信息：

- 基本信息
  - 实例名称：输入您为开发机命名的名称，例如：dev-RoboTwin
  - 资源池类型/资源池/队列：根据您的资源池类型选择已创建的资源池和队列
  - 资源规格：建议选择 A800 显卡并配置 8 核 64G 内存
  - 云磁盘：推荐预留100GiB保证开发机正常运行
  - 存储挂载：用于挂载云存储资源（PFS/BOS/CFS等），此处选择存储类型为PFS的资源，默认带出存储配置区，填写源路径与挂载路径，例如：源路径：/RoboTwin，挂载路径：/mnt/pfs/
- 访问配置&高级配置
  - 可以使用默认配置，此处不做修改

完成后点击确定，即可完成开发机创建。

1.2 步骤二：登录开发机 填写完表单，提交后，在开发机列表可见创建的开发机，等待资源调度并部署成功后，点击登录，即可进入开发机NoteBook操作终端。

进入切换环境并预载数据资产 2.1 切换至 RoboTwin 环境，并进入预置 RoboTwin 目录：

```
 conda activate RoboTwin
cd /root/RoboTwin/
```

2.2 下载 RoboTwin 资产（因为网络因素，请使用 http_proxy 保证下载，后续涉及下载出现问题都可以使用此方法）：

```
 export http_proxy=http://xxx.xxx.xxx.xxx:xxxx
export https_proxy=http://xxx.xxx.xxx.xxx:xxxx
bash script/_download_assets.sh
unset http_proxy
unset https_proxy
```

2.3 （可选）数据生成

RoboTwin 官方虽然在 HuggingFace 上传了数据，但仍建议用户生成视频来作为数据集，可执行如下代码生成：

```
 ##### 生成数据工具加载
apt-get update
apt-get install -y xvfb
apt-get install -y libglvnd0 libglvnd-dev
##### 参考格式 xvfb-run -s "-screen 0 1400x900x24" bash collect_data.sh ${task_name} ${task_config} ${gpu_id}
##### beat_block_hammer意为由机械臂举起锤子敲击桌子上生成方块
##### task_config 必须是 'demo_clean' 或者 'demo_randomized'
xvfb-run -s "-screen 0 1400x900x24" bash collect_data.sh beat_block_hammer demo_randomized 0
```

最终结果会生成于data/${task_name}/${task_config}文件夹下，生成视频如下：

[附件] 后续训练都是用此步骤生成数据进行验证，task_name填写说明可见文末附录，task_config 配置可参考/root/RoboTwin/task_config/路径下相关文件

使用 RoboTwin 进行模型训练 RoboTwin 2.0 中已经预置了一些模型，用户可以快速进行训练，部署与验证相关模型。

3.1 ACT 3.1.1 进入ACT 目录并安装相关环境

```
 cd policy/ACT

pip install pyquaternion pyyaml rospkg pexpect mujoco==2.3.7 dm_control==1.0.14 opencv-python matplotlib einops packaging h5py ipython

cd detr && pip install -e . && cd ..
```

3.1.2 数据预处理，将RoboTwin 2.0 的数据转换为 ACT 训练用格式

```
 ##### bash process_data.sh ${task_name} ${task_config} ${expert_data_num}
bash process_data.sh beat_block_hammer demo_randomized 50
```

3.1.3 训练

运行如下代码进行训练：

```
 ##### bash train.sh ${task_name} ${task_config} ${expert_data_num} ${seed} ${gpu_id}
bash train.sh beat_block_hammer demo_randomized 50 0 0
```

可以通过调整/root/RoboTwin/policy/ACT/train.sh中参数修改训练参数

```
 ##### !/bin/bash
task_name=${1}
task_config=${2}
expert_data_num=${3}
seed=${4}
gpu_id=${5}

DEBUG=False
save_ckpt=True

export CUDA_VISIBLE_DEVICES=${gpu_id}

python3 imitate_episodes.py \
    --task_name sim-${task_name}-${task_config}-${expert_data_num} \
    --ckpt_dir ./act_ckpt/act-${task_name}/${task_config}-${expert_data_num} \
    --policy_class ACT \
    --kl_weight 10 \
    --chunk_size 50 \
    --hidden_dim 512 \
    --batch_size 8 \
    --dim_feedforward 3200 \
    --num_epochs 6000 \
    --lr 1e-5 \
    --save_freq 2000 \
    --state_dim 14 \
    --seed ${seed}
```

3.1.4 评估

```
 ##### bash eval.sh ${task_name} ${task_config} ${ckpt_setting} ${expert_data_num} ${seed} ${gpu_id}
bash eval.sh beat_block_hammer demo_randomized demo_randomized 50 0 0
```

评估文件在eval_result文件夹下：

[附件] **3.2 DP** 3.2.1 安装环境

```
 cd policy/DP
pip install zarr==2.12.0 wandb ipdb gpustat dm_control omegaconf hydra-core==1.2.0 dill==0.3.5.1 einops==0.4.1 diffusers==0.11.1 numba==0.56.4 moviepy imageio av matplotlib termcolor sympy
pip install -e .
```

3.2.2 数据预处理，将RoboTwin 2.0 的数据转换为 DP 训练用格式

```
 ##### bash process_data.sh ${task_name} ${task_config} ${expert_data_num}
bash process_data.sh beat_block_hammer demo_randomized 50
```

3.2.3 训练

action_dim为维度，此处aloha-agilex为双臂 6 自由度+夹爪，共 14 维度。

```
 ##### bash train.sh ${task_name} ${task_config} ${expert_data_num} ${seed} ${action_dim} ${gpu_id}
bash train.sh beat_block_hammer demo_randomized 50 0 14 0
```

3.2.4 评估 使用如下代码进行评估

```
 bash eval.sh beat_block_hammer demo_randomized demo_clean 50 0 0
```

附录：task_name

参考：官网说明文档

task_name	task_description	中文翻译
beat_block_hammer	Pick up the hammer and use it to beat the block on the table once. The hammer is placed at a fixed position on the table, but the block is generated randomly on the table. If the block's x coordinate(dim 0) is greater than 0, use the right arm to grasp the hammer, else use the left arm. To beat the block, you should place the hammer on the block's functional point(i.e., use the place_actor API to align the hammer's contact point with the block's functional point). Note: You don't need to Lift the hammer after beating the block, and you don't need to open the gripper or return the arm to origin position.	拿起锤子并用它敲击桌子上的方块一次。锤子固定在桌子上的某个位置，但方块在桌子上随机生成。如果方块的x坐标大于0，使用右臂抓取锤子，否则使用左臂。要敲击方块，应将锤子放置在方块的功能点上（即使用place_actor API将锤子的接触点与方块的功能点对齐）。注意：敲击方块后不需要抬起锤子，也不需要打开夹爪或将手臂返回初始位置。

pick_dual_bottles	Use both arms to simultaneously pick up the bottle1 and bottle2 and move them to the front target locations, with the bottle1 on the left and the bottle2 on the right. Note: You don't need to open gripper and don't put down the bottles at the end.	使用双臂同时拿起bottle1和bottle2，并将它们移动到前方的目标位置，bottle1在左边，bottle2在右边。注意：不需要打开夹爪，最后也不需要放下瓶子。
pick_diverse_bottles	Use both arms to simultaneously pick up the diverse bottles and move them to the front target locations, with the bottle1 on the left and the bottle2 on the right. No need to put the bottles down. In which the bottles may be lying down. Note: You don't need to open gripper and don't put down the bottles at the end.	使用双臂同时拿起不同的瓶子，并将它们移动到前方的目标位置，bottle1在左边，bottle2在右边。不需要放下瓶子。瓶子可能是倒下的状态。注意：不需要打开夹爪，最后也不需要放下瓶子。
handover_block	There are two blocks on the desk. Use the left arm to grab the block and move it to the handover point, then use right arm to grab the block and open the gripper of left arm simulaniously. Use right arm move block on the target block. Note: You should first pass the block to the right gripper and close right gripper, then open the left gripper.	桌上有两个方块。使用左臂抓取方块并移动到交接点，然后使用右臂抓取方块并同时打开左臂夹爪。使用右臂将方块移动到目标方块上。注意：应先将方块传递给右夹爪并关闭右夹爪，然后打开左夹爪。
stack_blocks_two	Use the gripper to pick up block1 and move block 1 to the target position. Then pick up block 2 and place it on the block 1. If block1's x coordinate(dim 0) is greater than 0, use right arm to stack the block1, else use the left arm, and same for the block2. Note: You need to call the get_avoid_collision_pose function to avoid collisions when the left and right arms move alternately. For example, if the previous action uses the left arm and the next action uses the right arm, you need to move the left arm after release gripper to avoid collisions, vice versa. The pre-dis of stacked blocks may be smaller.	使用夹爪拿起block1并将block1移动到目标位置。然后拿起block2并将其放置在block1上。如果block1的x坐标大于0，使用右臂堆叠block1，否则使用左臂，block2同理。注意：当左右臂交替移动时，需要调用get_avoid_collision_pose函数避免碰撞。例如，如果前一个动作使用左臂，下一个动作使用右臂，需要在释放夹爪后移动左臂以避免碰撞，反之亦然。堆叠方块的预抓取距离可能较小。
stack_blocks_three	Use the gripper to pick up block1 and move block 1 to the target position. Then pick up block 2 and place it on the block 1, and finally pick up block3 and place it on the block2. If block1's x coordinate(dim 0) is greater than 0, use right arm to stack the block1, else use the left arm. And same for the block2 and block3. Note: You need to call the get_avoid_collision_pose function to avoid collisions when the left and right arms move alternately. For example, if the previous action uses the left arm and the next action uses the right arm, you need to move the left arm after release gripper to avoid collisions, vice versa. The pre-dis of stacked blocks may be smaller.	使用夹爪拿起block1并将block1移动到目标位置。然后拿起block2并将其放置在block1上，最后拿起block3并将其放置在block2上。如果block1的x坐标大于0，使用右臂堆叠block1，否则使用左臂，block2和block3同理。注意：当左右臂交替移动时，需要调用get_avoid_collision_pose函数避免碰撞。例如，如果前一个动作使用左臂，下一个动作使用右臂，需要在释放夹爪后移动左臂以避免碰撞，反之亦然。堆叠方块的预抓取距离可能较小。
place_container_plate	Use both arms to pick up the container and place it in the plate. If the container's x coordinate(dim 0)is greater than 0, use right arm to grasp the right side of the container, then pick up the container and place it in the plate. Else use the left arm grasp the left side of the container, then pick up the container and place it in the plate. Note: You may need to close the jaws tightly to pick up the container.	使用双臂拿起容器并将其放入盘子中。如果容器的x坐标大于0，使用右臂抓取容器的右侧，然后拿起容器并将其放入盘子中。否则使用左臂抓取容器的左侧，然后拿起容器并将其放入盘子中。注意：可能需要紧紧关闭夹爪来拿起容器。
place_empty_cup	Use both arms to pick up the empty cup and place it on the coaster. If the cup's x coordinate(dim 0) is greater than 0, use right arm to grasp the cup, then pick up the cup and place it on the coaster, else use the left arm grasp the the cup, then pick up the cup and place it on the coaster. Note: You may need to close the jaws tightly to pick up the cup. Pre-dis for grabbing and placing cups may be smaller. The distance of lifting the cup may be smaller.	使用双臂拿起空杯子并将其放在杯垫上。如果杯子的x坐标大于0，使用右臂抓取杯子，然后拿起杯子并将其放在杯垫上，否则使用左臂抓取杯子，然后拿起杯子并将其放在杯垫上。注意：可能需要紧紧关闭夹爪来拿起杯子。抓取和放置杯子的预抓取距离可能较小。抬起杯子的距离可能较小。
place_shoe	Pick up the shoe and place it on the target block. And the head of the shoe should be towards the left side. The shoe is randomly placed on the table, if the shoe's x coordinate(dim 0) is greater than 0, use right arm to grasp the shoe, else use the left arm grasp the shoe.	拿起鞋子并将其放在目标方块上。鞋头应朝向左侧。鞋子随机放置在桌子上，如果鞋子的x坐标大于0，使用右臂抓取鞋子，否则使用左臂抓取鞋子。
place_dual_shoes	Left shoe and right shoe are randomly generated on the desktop, one on the left and one on the right. Use left and right arms to pick up two shoes simultaneously. And put down them on the shoe box respectively. The head of the shoe should be towards the left side. Left shoe should be placed on the point0 of shoe box, and right shoe should be placed on the point1 of shoe box. Note: You may need to put the shoes in order to avoid left and right arm collisions. Avoiding collisions needs to be done before place shoes. Pre-dis for grabbing and placing shoes may be smaller.	左鞋和右鞋随机生成在桌面上，一只在左边，一只在右边。使用左右臂同时拿起两只鞋子，并分别将它们放在鞋盒上。鞋头应朝向左侧。左鞋应放置在鞋盒的point0上，右鞋应放置在鞋盒的point1上。注意：可能需要按顺序放置鞋子以避免左右臂碰撞。避免碰撞需要在放置鞋子之前完成。抓取和放置鞋子的预抓取距离可能较小。
adjust_bottle	Pick up the bottle on the table headup with the correct arm. Move the arm upward by 0.1 meters along z-axis, and place the bottle at target pose. Note: You should keep gripper closed when placing the bottle.	使用正确的手臂拿起桌子上的瓶子（瓶口朝上）。沿z轴向上移动手臂0.1米，并将瓶子放置在目标姿态。注意：放置瓶子时应保持夹爪关闭。
blocks_ranking_rgb	Place the red block, green block, and blue block in the order of red, green, and blue from left to right, placing in a row. Pick and place each block to their target positions. Note: You should move end effector back to origin after placing each block to avoid collisions. You can place the red block, the green block, and the blue block in the order.	将红色方块、绿色方块和蓝色方块按照红、绿、蓝的顺序从左到右排成一行。逐个拿起并放置每个方块到其目标位置。注意：放置每个方块后应将末端执行器移回原点以避免碰撞。可以按照红色方块、绿色方块、蓝色方块的顺序放置。
blocks_ranking_size	There are three blocks on the table, the color of the blocks is random, move the blocks to the center of the table, and arrange them from largest to smallest, from left to right. Pick and place each block to their target positions. Note: You should move end effector back to origin after placing each block to avoid collisions. You can place the smallest block, the middle block, and the largest block in the order.	桌上有三个方块，方块颜色随机，将方块移动到桌子中央，并按从大到小、从左到右的顺序排列。逐个拿起并放置每个方块到其目标位置。注意：放置每个方块后应将末端执行器移回原点以避免碰撞。可以按照最小方块、中间方块、最大方块的顺序放置。
click_bell	Click the bell's top center on the table. Move the top of bell's center and close gripper. And move the gripper down to touch the bell's top center. Note: You can change some API parameters to move above the bell's top center and close the gripper. You can use self. grasp_actor() to simulate the action of touch and click. self.grasp_actor() is only used to move the top center of the bell and close the gripper. So you must use same pre_grasp_dis and grasp_dis as the click_bell task. You don't need to lift the bell after clicking it, and you don't need to open the gripper or return the arm to origin position.	点击桌子上铃铛的顶部中心。移动到铃铛顶部中心并关闭夹爪。然后向下移动夹爪接触铃铛顶部中心。注意：可以更改一些API参数移动到铃铛顶部中心上方并关闭夹爪。可以使用self.grasp_actor()模拟触摸和点击动作。self.grasp_actor()仅用于移动到铃铛顶部中心并关闭夹爪。因此必须使用与click_bell任务相同的pre_grasp_dis和grasp_dis。点击铃铛后不需要抬起铃铛，也不需要打开夹爪或将手臂返回初始位置。
grab_roller	Use both arms to grab the roller on the table. Grasp the roller with both arms simultaneously at different contact points. And lift the roller upper by 0.15 meters by moving both arms upward simultaneously.	使用双臂抓取桌子上的滚筒。同时用双臂在不同接触点抓取滚筒。通过同时向上移动双臂将滚筒抬起0.15米。
lift_pot	Use both arms to lift the pot. Grasp the pot with both arms at specified contact points. And lift the pot by moving both arms upper by 0.15 meters. Note: The pre_grasp_dis is very small when grasping the pot.	使用双臂抬起锅。在指定接触点用双臂抓取锅。通过向上移动双臂将锅抬起0.15米。注意：抓取锅时预抓取距离非常小。
move_can_pot	There is a can and a pot on the table. Use one arm to pick up the can and move it to beside the pot. Grasp the can, and move the can upward. Place the can near the pot at target pose. Note: You don't need to return the arm to origin position.	桌子上有一个罐子和一个锅。使用一只手臂拿起罐子并将其移动到锅旁边。抓取罐子，向上移动罐子。将罐子放置在锅附近的目标姿态。注意：不需要将手臂返回初始位置。
move_playingcard_away	Use the arm to pick up the playing card and move it to left or right. Grasp the playing cards with specified arm, and then move the playing cards horizontally(right if right arm, left if left arm). Note: You should open gripper to release the playing cards after moving them.	使用手臂拿起扑克牌并将其向左或向右移动。用指定手臂抓取扑克牌，然后水平移动扑克牌（右臂向右移动，左臂向左移动）。注意：移动扑克牌后应打开夹爪释放扑克牌。
move_stapler_pad	Use appropriate arm to move the stapler to a colored mat. Grasp the stapler with specified arm, and move the arm upward. Place the stapler at target pose with alignment constraint.	使用适当的手臂将订书机移动到彩色垫子上。用指定手臂抓取订书机，向上移动手臂。将订书机放置在具有对齐约束的目标姿态上。
click_alarmclock	Click the alarm clock's center of the top side button on the table. Move the top of bell's center and close gripper. And move the gripper down. Note: You can change some API parameters to move above the alarm clock's top center and close the gripper(grasp_actor). You can use self.grasp_actor() to simulate the action of touch and click	点击桌子上闹钟顶部按钮的中心。移动到闹钟顶部中心并关闭夹爪。然后向下移动夹爪。注意：可以更改一些API参数移动到闹钟顶部中心上方并关闭夹爪（grasp_actor）。可以使用self.grasp_actor()模拟触摸和点击动作。
dump_bin_bigbin	Grab the small bin and pour the balls into the big bin. If the small bin is on the right side of the table, grasp the deskbin with right arm, and place the deskbin at middle pose. Then return right arm to origin while simultaneously grasping with left arm. If deskbin is on left side, directly grasp with left arm. Perform pouring actions 3 times. Note: The gripper should be closed when pouring the balls into the big bin. You should use self.delay(6) in the end of the task to wait for the pouring actions to complete. Don't use functional point for the deskbin, use self.place_actor()to place the deskbin at middle pose.	抓起小垃圾桶将球倒入大垃圾桶中。如果小垃圾桶在桌子右侧，用右臂抓取桌面垃圾桶，将其放置在中介姿态。然后同时用左臂抓取时将右臂返回原点。如果桌面垃圾桶在左侧，直接用左臂抓取。执行倒球动作3次。注意：将球倒入大垃圾桶时应关闭夹爪。应在任务结束时使用self.delay(6)等待倒球动作完成。不要使用桌面垃圾桶的功能点，使用self.place_actor()将桌面垃圾桶放置在中介姿态。

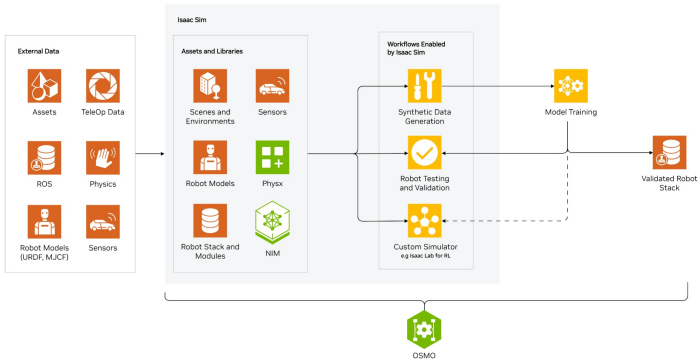
handover_mic	Use one arm to grasp the microphone on the table and handover it to the other arm. Move the grasping arm to the microphone's position and grasp it. Move the handover arm to the middle position for handover. Move the handover arm to grasp the microphone from the grasping arm. Move the grasping arm to open the gripper.	使用一只手臂抓取桌子上的麦克风并将其交接给另一只手臂。移动抓取手臂到麦克风位置并抓取。移动交接手臂到中介位置进行交接。移动交接手臂从抓取手臂处抓取麦克风。移动抓取手臂打开夹爪。
hanging_mug	Use left arm to pick the mug on the table, rotate the mug and put the mug down in the middle of the table, use the right arm to pick the mug and hang it onto the rack. Move the grasping arm to the mug's position and grasp it. Move the grasping arm to a middle position before hanging. Grasp the mug with the hanging arm, and move the grasping arm back to its origin. Move the hanging arm to the target pose and hang the mug. Note: grasping arm is left arm, hanging arm is right arm. The target pose for hanging the mug is the functional point of the rack.	使用左臂拿起桌子上的杯子，旋转杯子并将杯子放在桌子中央，使用右臂拿起杯子并将其挂在架子上。移动抓取手臂到杯子位置并抓取。在悬挂前移动抓取手臂到中介位置。用悬挂手臂抓取杯子，并将抓取手臂移回原点。移动悬挂手臂到目标姿态并悬挂杯子。注意：抓取手臂是左臂，悬挂手臂是右臂。悬挂杯子的目标姿态是架子的功能点。
move_pillbottle_pad	Use one arm to pick the pillbottle and place it onto the pad. Grasp the pillbottle. Get the target pose for placing the pillbottle, and place the pillbottle at the target pose.	使用一只手臂拿起药瓶并将其放在垫子上。抓取药瓶。获取放置药瓶的目标姿态，并将药瓶放置在目标姿态。
place_a2b_left	Use appropriate arm to place object on the left of target object. Grasp the object with specified arm. And get target pose and adjust x position to place object to the left of target object. Place the object at the adjusted target object position. Note: You can decrease the x position of target pose by 0.13 to place object to the left of target object.(target_pose[0]-= 0.13)	使用适当的手臂将物体放置在目标物体的左侧。用指定手臂抓取物体。获取目标姿态并调整x位置将物体放置在目标物体左侧。将物体放置在调整后的目标物体位置。注意：可以将目标姿态的x位置减少0.13来将物体放置在目标物体左侧。
place_a2b_right	Use appropriate arm to place object on the right of target object. Grasp the object with specified arm. And get target pose and adjust x position to place object to the right of target object. Place the object at the adjusted target object position. Note: You can increase the x position of target pose by 0.13 to place object to the right of target object.(target_pose[0]+= 0.13)	使用适当的手臂将物体放置在目标物体的右侧。用指定手臂抓取物体。获取目标姿态并调整x位置将物体放置在目标物体右侧。将物体放置在调整后的目标物体位置。注意：可以将目标姿态的x位置增加0.13来将物体放置在目标物体右侧。
place_bread_basket	If there is one bread on the table, use one arm to grab the bread and put it in the basket. If there are two breads on the table, use two arms to simultaneously grab up two breads and put them in the basket. Grasp the bread. If there is one bread, place the bread into the basket. If there is two breads, place left bread into the basket, and place right bread into the basket when move left arm back to origin. Note: You should move the arm back to origin after placing the bread to avoid collisions.	如果桌子上有一个面包，使用一只手臂抓起面包放入篮子中。如果桌子上有两个面包，使用双臂同时抓起两个面包放入篮子中。抓取面包。如果有一个面包，将面包放入篮子中。如果有两个面包，将左面包放入篮子中，并在移动左臂回原点时将右面包放入篮子中。注意：放置面包后应将手臂移回原点以避免碰撞。
place_bread_skillet	If there is one bread on the table, use one arm to grab the bread and put it into the skillet. Grasp the skillet and bread simultaneously with dual arms. Get the functional point of the skillet as placement target for the bread. Place the bread onto the skillet.	如果桌子上有一个面包，使用一只手臂抓起面包放入煎锅中。同时用双臂抓取煎锅和面包。获取煎锅的功能点作为面包的放置目标。将面包放在煎锅上。
place_can_basket	Use one arm to pick up the can and place it into the basket. Use the other arm to lift up the basket. Grasp the can with the specified arm. Place the can at the selected position into the basket. Lift the basket with the opposite arm. The height of lifting the basket is 5 cm.	使用一只手臂拿起罐子并将其放入篮子中。使用另一只手臂抬起篮子。用指定手臂抓取罐子。将罐子放置在篮子中的选定位置。用另一只手臂抬起篮子。抬起篮子的高度为5厘米。
place_cans_plasticbox	Use dual arm to pick and place cans into plasticbox. Grasp both objects with dual arms. Place left object into plastic box at target point 1, and then left arm moves back to origin while right arm places object into plastic box at target point 0. Grasp the second can with the right arm and place it into the plastic box at target point 0. Right arm moves back to original position. Note: You should use left arm to grasp object 1 and right arm to grasp object 2. Don't set pre_dis_axis to fp, because the pre_dis_axis is not used in this task.	使用双臂拿起罐子并将其放入塑料盒中。用双臂抓取两个物体。将左物体放入塑料盒的目标点1，然后左臂移回原点同时右臂将物体放入塑料盒的目标点0。用右臂抓取第二个罐子并将其放入塑料盒的目标点0。右臂移回原始位置。注意：应使用左臂抓取物体1，右臂抓取物体2。不要将pre_dis_axis设置为fp，因为此任务中不使用pre_dis_axis。
place_fan	Grab the fan and place it on a colored pad. Grasp the fan with the selected arm. Place the fan to the target pose. Note: The height of lifting the fan is small. Fan have front and back, so you should use constraint'align'to align the fan's front with the pad's front.	抓起风扇并将其放在彩色垫子上。用选定手臂抓取风扇。将风扇放置到目标姿态。注意：抬起风扇的高度较小。风扇有前后之分，因此应使用'align'约束将风扇的前部与垫子的前部对齐。
place_burger_fries	Use dual arm to pick the hamburg and frenchfries and put them onto the tray. Dual grasp of hamburg and french fries. Get target poses from tray for placing. And place hamburg on tray, then place french fries on tray while moving the arm that placed hamburg back to origin. Note: Use left arm to grasp hamburg and right arm to grasp french fries. The target pose for placing hamburg and french fries is the functional point 0 and 1 of the tray respectively.	使用双臂拿起汉堡和薯条并将它们放在托盘上。同时抓取汉堡和薯条。从托盘获取放置的目标姿态。将汉堡放在托盘上，然后在移动放置汉堡的手臂回原点的同时将薯条放在托盘上。注意：使用左臂抓取汉堡，右臂抓取薯条。放置汉堡和薯条的目标姿态分别是托盘的功能点0和1。
place_mouse_pad	Grasp the mouse and place it on a colored pad. Grasp the mouse with the selected arm. Place the mouse at the target location. Note: The mouse have front and back, so you should use constraint'align'to align the mouse's front with the pad's front.	抓取鼠标并将其放在彩色垫子上。用选定手臂抓取鼠标。将鼠标放置在目标位置。注意：鼠标有前后之分，因此应使用'align'约束将鼠标的前部与垫子的前部对齐。
place_object_basket	Use one arm to grab the target object and put it in the basket, then use the other arm to grab the basket, and finally move the basket slightly away. Grasp the object with the specified arm. Place the object at the selected position into the basket. Lift the basket with the opposite arm. Note: You should not open the gripper after lifting the basket. The height of lifting the basket is 5 cm.	使用一只手臂抓起目标物体放入篮子中，然后使用另一只手臂抓起篮子，最后将篮子稍微移开。用指定手臂抓取物体。将物体放置在篮子中的选定位置。用另一只手臂抬起篮子。注意：抬起篮子后不应打开夹爪。抬起篮子的高度为5厘米。
place_object_scale	Use one arm to grab the object and put it on the scale. Grasp the object with the selected arm. Place the object on the scale. Note: Don't use functional point_id and pre_dis_axis='fp', because the object can be any object that is specified in the task.	使用一只手臂抓起物体并将其放在秤上。用选定手臂抓取物体。将物体放在秤上。注意：不要使用functional point_id和pre_dis_axis='fp'，因为物体可以是任务中指定的任何物体。
place_object_stand	Use appropriate arm to place the object on the stand. Grasp the object with the specified arm. Place the object onto the display stand. Note: Don't use functional_point_id and pre_dis_axis='fp', because the object can be any object that is specified in the task.	使用适当的手臂将物体放置在架子上。用指定手臂抓取物体。将物体放置在展示架上。注意：不要使用functional_point_id和pre_dis_axis='fp'，因为物体可以是任务中指定的任何物体。
place_phone_stand	Pick up the phone and put it on the phone stand. Grasp the phone with specified arm. Place the phone onto the stand's functional point and align the points.	拿起电话并将其放在电话架上。用指定手臂抓取电话。将电话放置在架子的功能点上并对齐点。
press_stapler	Use one arm to press the stapler. Move arm to the position of the stapler and close the gripper. Move the stapler down slightly. Note: You can use self.grasp_actor() to simulate the action of move to the position of stapler or pressing the stapler. The stapler should be pressed at the top center.	使用一只手臂按压订书机。移动手臂到订书机位置并关闭夹爪。将订书机稍微向下移动。注意：可以使用self.grasp_actor()模拟移动到订书机位置或按压订书机的动作。订书机应在顶部中心被按压。
rotate_qrcode	Use arm to catch the qrcode board on the table, pick it up and rotate to let the qrcode face towards you. Grasp the QR code with specified pre-grasp distance. Place the QR code at the target position. Note: The QR code have front and back, so you should use constraint'align'to align the QR code's front with the target position. Don't use functional point of the QR code when placing it.	使用手臂抓取桌子上的二维码板，拿起并旋转让二维码面向你。用指定的预抓取距离抓取二维码。将二维码放置在目标位置。注意：二维码有正反面之分，因此应使用'align'约束将二维码的正面与目标位置对齐。放置二维码时不要使用其功能点。
scan_object	Use one arm to pick the scanner and use the other arm to pick the object, and use the scanner to scan the object. Move the scanner and object to the gripper. Get object target pose and place the object. Move the scanner to align with the object. Note: The object target pose is dependent on the arm used to grasp the object. The scanner should be placed at a distance of 0.05 meters from the functional point of the object. You should not open the gripper after placing the object and scanner.	使用一只手臂拿起扫描器，另一只手臂拿起物体，然后用扫描器扫描物体。将扫描器和物体移动到夹爪处。获取物体目标姿态并放置物体。移动扫描器与物体对齐。注意：物体目标姿态取决于用于抓取物体的手臂。扫描器应放置在距离物体功能点0.05米处。放置物体和扫描器后不应打开夹爪。
stack_bowls_three	Stack the three bowls on top of each other. Move bowl 1 to the target pose, then move bowl 2 above bowl 1, and finally move bowl 3 above bowl 2. Note: The target pose of bowl 2 is at 5 cm above bowl 1, and the target pose of bowl 3 is at 5 cm above bowl 2. All target pose is np.ndarray([x, y, z]), so you should concatenate the quaternion later.	将三个碗叠放在一起。将碗1移动到目标姿态，然后将碗2移动到碗1上方，最后将碗3移动到碗2上方。注意：碗2的目标姿态在碗1上方5厘米处，碗3的目标姿态在碗2上方5厘米处。所有目标姿态都是np.ndarray([x, y, z])，因此稍后应连接四元数。
stack_b	Stack the two bowls on top of each other. Move bowl 1 to the target pose, then move bowl 2 above bowl 1. Note: The	将两个碗叠放在一起。将碗1移动到目标姿态，然后将碗2移动到碗1上方。注

owls_two	target pose of bowl 2 is at 5 cm above bowl 1. All target pose is np.ndarray([x, y, z]), so you should concatenate the quaternion later.	意：碗2的目标姿态在碗1上方5厘米处。所有目标姿态都是np.ndarray([x, y, z])，因此稍后应连接四元数。
stamp_seal	Use one arm to pick the stamp and place it on the target block. Grasp the seal with specified arm. Place the seal on the target block. Note: Don't set pre_dis_axis to fp, because the pre_dis_axis is not used in this task.	使用一只手臂拿起印章并将其放在目标方块上。用指定手臂抓取印章。将印章放置在目标方块上。注意：不要将pre_dis_axis设置为fp，因为此任务中不使用pre_dis_axis。
shake_bottle_horizontally	Shake the bottle horizontally with proper arm. Grasp the bottle with specified arm. Shake the bottle horizontally by moving the arm left and right.	用适当的手臂水平摇晃瓶子。用指定手臂抓取瓶子。通过左右移动手臂水平摇晃瓶子。
shake_bottle	Shake the bottle with proper arm. Grasp the bottle with specified arm. Shake the bottle by moving the arm up and down.	用适当的手臂摇晃瓶子。用指定手臂抓取瓶子。通过上下移动手臂摇晃瓶子。
put_bottles_dustbin	Use one arm to pick the bottle and put it into the dustbin. Grasp the bottle with specified arm. Place the bottle into the dustbin. Note: Don't set pre_dis_axis to fp, because the pre_dis_axis is not used in this task.	使用一只手臂拿起瓶子并将其放入垃圾桶中。用指定手臂抓取瓶子。将瓶子放入垃圾桶中。注意：不要将pre_dis_axis设置为fp，因为此任务中不使用pre_dis_axis。
turn_switch	Use one arm to click the switch. Close the gripper before clicking the switch. Then move the arm to the switch and click it. Note: You can use grasp_actor() to simulate the action of clicking the switch.	使用一只手臂点击开关。点击开关前关闭夹爪。然后移动手臂到开关处并点击它。注意：可以使用grasp_actor()模拟点击开关的动作。
open_laptop	Open the laptop with one proper arm. Grasp the laptop with specified arm. Open the laptop by moving the arm up.	用适当的手臂打开笔记本电脑。用指定手臂抓取笔记本电脑。通过向上移动手臂打开笔记本电脑。
open_microwave	Open the microwave with one proper arm. Grasp the handle of the microwave with specified arm. Pull the handle to open the microwave.	用适当的手臂打开微波炉。用指定手臂抓取微波炉把手。拉动把手打开微波炉。
put_object_cabinet	Use one arm to open the cabinet, and use another arm to pick the object and put it into the cabinet. Grasp the cabinet handle with one arm and open the cabinet. Grasp the object with the other arm and place it into the cabinet.	使用一只手臂打开柜子，另一只手臂拿起物体并将其放入柜子中。用一只手臂抓取柜子把手并打开柜子。用另一只手臂抓取物体并将其放入柜子中。

Isaac Sim 可视化

案例介绍

> NVIDIA Isaac Sim是基于NVIDIA Omniverse构建的机器人仿真平台，本文是在开发机上基于isaac sim实现可视化能力的实践教程



NVIDIA Isaac Sim是基于NVIDIA Omniverse构建的机器人仿真平台，支持从URDF、MJCF和CAD等常见格式导入机器人系统。通过GPU加速物理引擎来实现精确仿真，并支持大规模、多传感器RTX渲染。集成数据生成、强化学习、ROS集成和数字孪生模拟等端到端工作流程。其优势包括：

- **资产导入&导出：**支持从非USD格式导入机器人和环境，或将其导出为非USD格式；
- **机器人微调：**优化机器人的物理精度、计算效率或真实感；
- **机器人仿真：**提供全流程运动控制工具链，如控制器、运动生成和运动学求解器以及策略集成接口；
- **传感器系统：**支持RTX和物理学传感器。

官方地址：

https://docs.isaacsim.omniverse.nvidia.com/4.5.0/index.html

部署方案

目前百舸平台提供了两种Isaac sim可视化的方案：

- 基于 RDP 远程桌面的部署方式：支持密码鉴权，通用性较高。
- NV 官方 webviewer可视化方案：基于 WebRTC协议，性能较好，不支持鉴权（推荐配合云的安全组使用）

选型建议：如果仅使用 Isaac sim 软件，推荐 webviewer 方式，性能较好；如果还有一些其他可视化操作的需求，可以使用 RDP 方式；也可以组合使用，如用户通过 webviewer 的方式使用 Isaac sim，也可以再启动 RDP桌面 进行一些其他可视化的操作

> - 不推荐使用两种方式同时启动 Isaac sim
> - webview 方式仅支持 Isaac sim 5.0 版本，不支持 4.5 版本

部署环境要求

	部署要求	最佳实践
CPU	建议8核及以上	按需
内存	建议64G及以上	按需
GPU	Isaac Sim：L20等 支持RT Core的NVIDIA GPU卡 1卡及以上	按需
CDS	按需	按需
其它	无	无

使用需知：

1. Isaac Sim 的仿真渲染依赖DLSS（超采样技术，Deep Learning Super Sampling），因此推荐使用GeForce RTX系列显卡
2. 本案例使用L20等Ada Lovelace架构三代RT Core的显卡，且使用的BLB网络带宽为50Mbps时，无噪点，无卡顿，表现更为符合期望
3. isacc官方要求，显卡驱动版本需要高于535.129 ，如果Ubuntu版本>=22.04.5 kernel 6.8.0-48-generic，则驱动至少要求在535.216.01以上
4. 535驱动版本中（550无此问题），含535.255及以上的驱动版本，存在一个已知的显卡驱动判断错误，需要添加--/rtx/verifyDriverVersion/enabled=false参数，跳过显卡驱动判断（操作方式见后续步骤）
5. 官方未对cuda和cudnn有限制，不同的cuda版本和cudnn版本是否对可视化界面的表现有影响未验证
6. 由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置HTTPS网络代理支持访问。服务网卡的配置参考
7. 建议在距离您最近的地域创建开发机实例使用当前方案。如果远程桌面不流畅，建议配置 RDP 比特率为16位

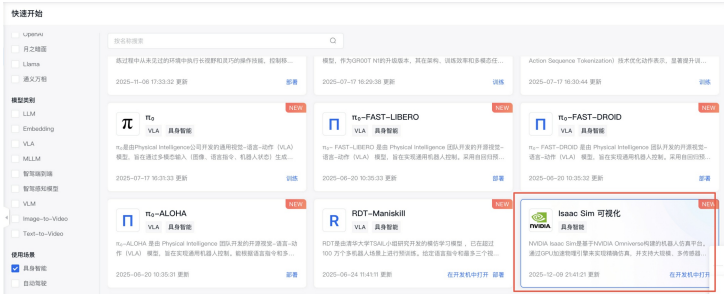
使用说明

本文会先介绍如何【启动可视化客户端】，然后以【执行官方四足和人形可操作演示】、【宇树GR1机器人简易操作演示】两个例子，简单介绍Isaac Sim的使用。

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置

1. 创建与登录开发机

1.1 从快速开始创建一个【Isaac Sim可视化】开发机实例



点击在开发机中打开，创建开发机，主要配置如下：

- 基本信息
  - 版本选择：支持 Isaac Sim 5.0 和 4.5 版本
- 资源配置：
  - GPU选择： 建议使用 NVIDIA L20
  - CPU/内存：推荐8核64G及以上，保证运行所需的基础资源配置
- 启动自定义端口，选择支持**公网访问**的BLB（建议选择网络带宽≥50Mbps的BLB，带宽越大越好，越少共享复用越好）
- 新增访问端口，具体如下:

服务名称	开发机监听端口	BLB 监听端口	传输协议	用途
rdp	3389（固定）	随机分配即可	TCP	RDP 启动必需
streaming-udp	随机分配，需要与**BLB 监听端口**一致	随机分配，需要与**开发机监听端口**一致	UDP	需要保证**开发机监听端口**和**BLB 监听端口**一致，随机分配即可 webviewer必需
streaming-tcp	49100（固定）	随机分配即可	TCP	webviewer 必需
webviewer	8080（固定）	随机分配即可	TCP	webviewer 必需

> Isaac sim 4.5 版本仅支持 RDP 的方式启动，不支持 Webviewer 方式，无需设置streaming-udp、streaming-tcp、webviewer服务

1.2 登录开发机

填写完表单，提交后，在开发机列表可见创建的开发机，等待资源调度并部署成功后，点击登录，即可进入开发机webIDE

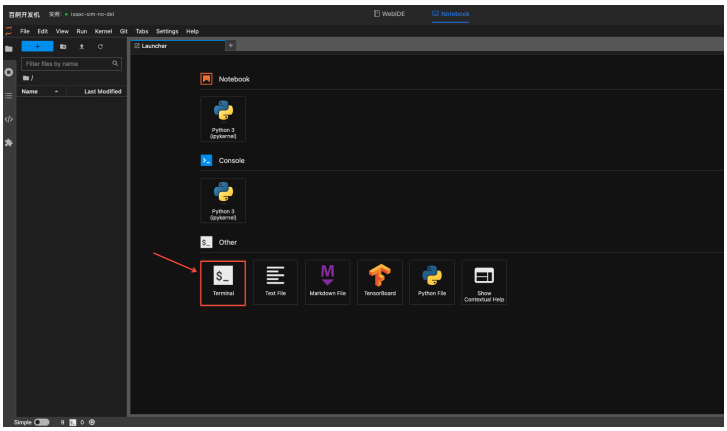


主要路径介绍

- Isaac Sim USD资产挂载路径：/mnt/bos/isaac-sim，该目录下存有Isaac Sim预置的部分USD资产。

1.3 验证GPU硬件和显卡驱动版本

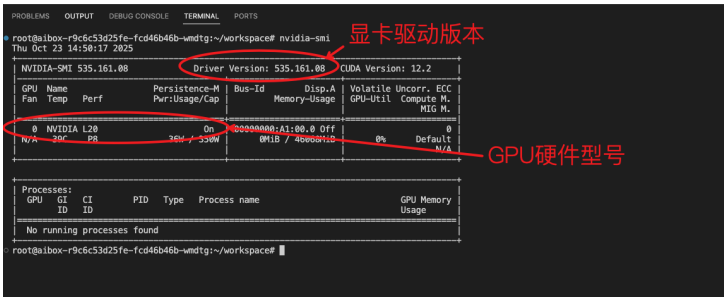
根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，打开Terminal界面：



在打开的Terminal界面内输入以下命令，确认NVIDIA的GPU硬件和显卡驱动版本

```
nvidia-smi
```

执行结果查看，红圈中是显卡驱动版本和GPU硬件型号



请参考我们下面提出的建议，选择驱动版本

1. isacc官方要求，显卡驱动版本需要高于535.129 ，如果Ubuntu版本>=22.04.5 kernel 6.8.0-48-generic，则驱动至少要求在535.216.01以上

2. 535驱动版本中（550无此问题），含535.255及以上的驱动版本，存在一个已知的显卡驱动判断错误，需要添加--/rtx/verifyDriverVersion/enabled=false参数，跳过显卡驱动判断（操作方式见后续步骤）

3. 官方未对cuda和cudnn有限制，不同的cuda版本和cudnn版本是否对可视化界面的表现有影响未验证

如显卡驱动不符合建议，建议联系售后支持同学重装节点驱动

2. 通过 RDP 方式配置访问Isaac Sim环境

登录开发机，进入开发机webIDE，打开Terminal界面

2.1 安装并启动 RDP 服务

使用如下指令安装并启动 RDP 服务

```
 ##### !/bin/bash

set -e

echo "===== 1. 安装桌面环境和 xrdp ====="
apt-get update
apt-get install -y xfce4 xfce4-goodies xrdp dbus-x11 libvulkan1

echo "===== 2. 配置 xrdp 使用 xfce4 ====="
echo "xfce4-session" > ~/.xsession
chmod +x ~/.xsession

##### 修改 startwm.sh 启动 xfce4
if ! grep -q "startxfce4" /etc/xrdp/startwm.sh; then
    sed -i 's|^test -x /etc/X11/Xsession.*|# &\nstartxfce4|' /etc/xrdp/startwm.sh
fi

echo "===== 3. 配置 Vulkan 环境变量 ====="
##### 写入 bashrc（确保 RDP 终端也能用 GPU Vulkan）
grep -q "VK_ICD_FILENAMES" ~/.bashrc 2>/dev/null || {
    echo 'export VK_ICD_FILENAMES=/etc/vulkan/icd.d/nvidia_icd.json' >> ~/.bashrc
    echo 'export LD_LIBRARY_PATH=/usr/local/nvidia/lib64:$LD_LIBRARY_PATH' >> ~/.bashrc
}

echo "===== 4. 启动 xrdp 服务 ====="
service dbus start 2>/dev/null || true
service xrdp restart

echo ""
echo "===== RDP 配置完成 ====="
echo ""
echo "连接方式："
echo "  1. 打开 RDP 客户端（Windows: mstsc, macOS: Microsoft Remote Desktop）"
echo "  2. 输入服务器 IP:3389"
echo "  3. 用户名: root"
echo ""

##### 检查是否设置了密码
if ! passwd -S root 2>/dev/null | grep -q " P "; then
    echo "[提示] 请设置 root 密码用于 RDP 登录："
    echo "  passwd root"
    echo ""
fi
```

2.2 设置RDP远程桌面登陆的账号密码

使用如下指令配置密码

```
passwd root
```

2.3 通过桌面客户端访问 在本地电脑或者需要连接开发机Isaac Sim环境的电脑。

2.3.1 MAC/Windows远程桌面

从app store安装Windows App。



本次安装后，

1. 在Windows APP 中依次点击右上角的**加号** 和 **添加电脑** 菜单，新建远程连接

2. 将从开发机复制的 RDP 的公网地址加端口（公网地址可从开发机详情页的自定义端口配置区域获取）输入到访问地址栏





3. 输入2.1步骤中中设置的用户名和密码

4. 登陆远程桌面

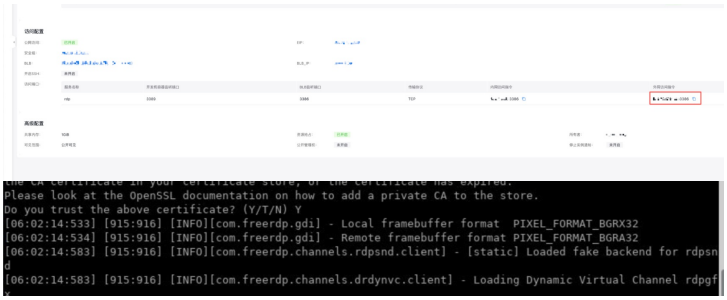
2.3.2 Linux远程桌面

1. 安装FreeRDP：

```
apt update && apt install -y freerdp2-x11
```

2. 启动FreeRDP连接服务：

```
##### 启动xfreerdp 用户名和密码为2.1设置RDP远程桌面登陆的账号密码中设置的用户名和密码
xfreerdp /v:<IP>:3386 /u:root /p:密码
```

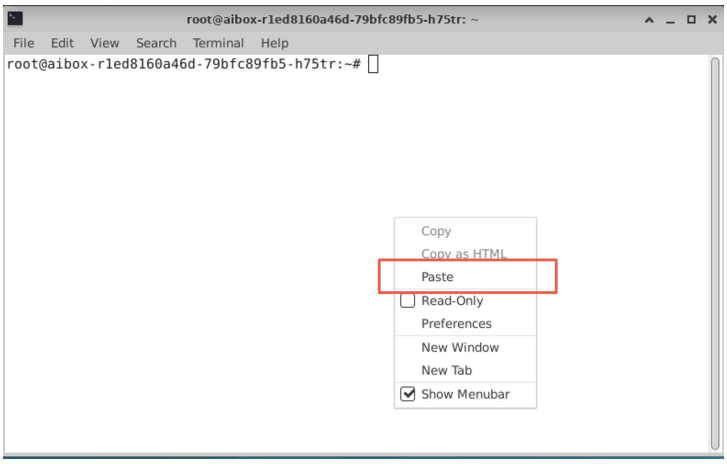


3. 启动后可以看到远程桌面，可能需要输入密码，输入的密码使用在2.1设置RDP远程桌面登陆的账号密码中设置的密码即可登录。

2.4 启动isaac-sim

在上面已经打开的RDP界面中，执行如下操作，从命令行启动isaac-sim客户端IsaacSim可视化界面：





可视化界面为先出现窗口边框，加载过程结束后再出现窗口内容，创建开发机第一次加载可能会需要几分钟(下载一些服务依赖)，之后再启动一般需要十秒左右。另外，IsaacSim在初始化和使用过程中可能会访问外网，**建议设置HTTP和HTTPS网络代理**

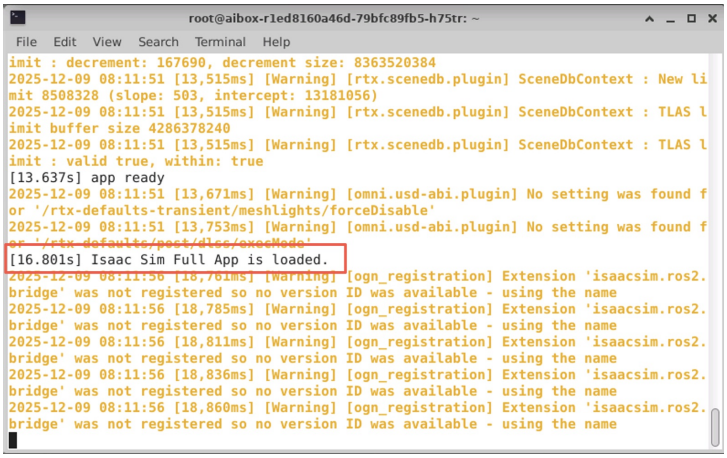
```
 ##### 先设置网络代理，再启动isaac-sim服务
/isaac-sim/isaac-sim.sh --allow-root
```

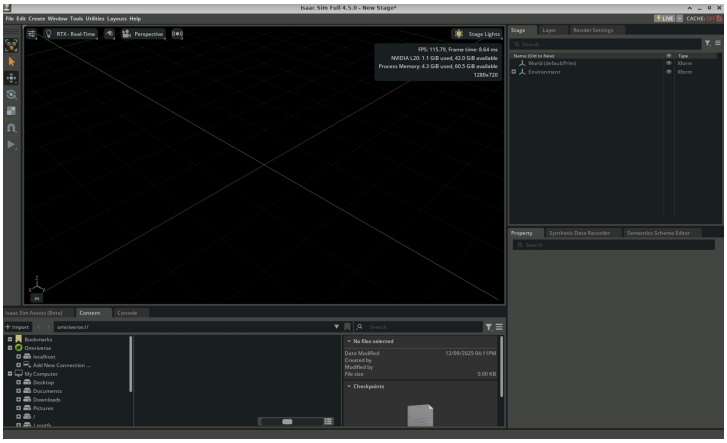
请使用nvidia-smi命令查看显卡驱动版本，如显卡驱动版本为535.261.03则建议使用

```
 ##### 先设置网络代理，再启动isaac-sim服务
/isaac-sim/isaac-sim.sh --allow-root --/rtx/verifyDriverVersion/enabled=false
```

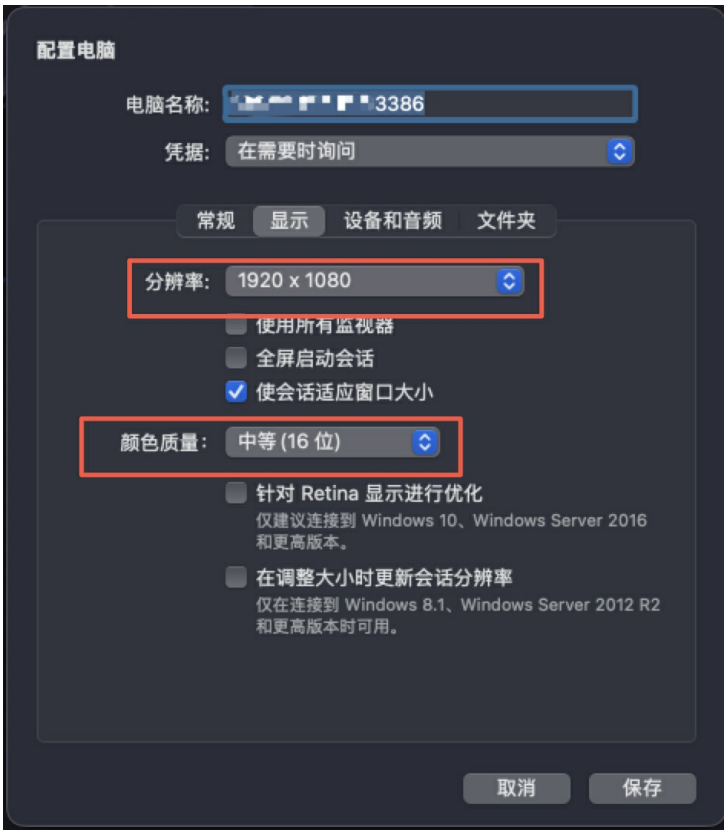
启动结果：

命令行出现红框内字样，说明启动成功





如果您的远程桌面在运行过程中出现卡顿现象，建议您降低远程连接的分辨率和颜色质量



3. 通过 webviewer 方式配置访问Isaac Sim环境

登录开发机，进入开发机webIDE，打开Terminal界面

3.1 启动webviewer和isaac sim streaming

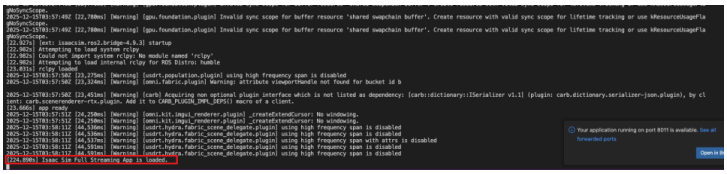
- 启动webviewer服务

```
##### --out-streaming-ip 当前开发机的外网IP地址
##### --inner-streaming-port 开发机webviewer-tcp端口的【开发机容器监听端口】字段
##### --out-webrtc-tcp-port 开发机streaming-tcp端口的【BLB监听端口】字段
~/start_webviewer.sh --out-streaming-ip <IP地址> --inner-webviewer-port <开发机容器监听端口> --out-streaming-tcp-port <BLB监听端口>
##### 使用下面的命令可关闭webviewer服务
~/stop_webviewer.sh
```

- 启动isaac sim streaming服务端，并查看日志确认完全启动

```
##### --out-streaming-ip 当前开发机的外网IP地址
##### --streaming-udp-port 开发机streaming-udp端口的【开发机容器监听端口】和【BLB监听端口】字段
##### --assets-root 开发机以bos方式加入的Assets文件夹，isaac sim 5.0可直接使用【/mnt/bos/isaac-sim/Assets/Isaac/5.0】这个值
~/start_isaac_streaming.sh --out-streaming-ip <IP地址> --streaming-udp-port <端口号> [可选： --assets-root /mnt/bos/isaac-sim/Assets/Isaac/5.0]
##### 使用下面的命令可关闭isaac sim的streaming服务
##### ~/stop_isaac_streaming.sh
##### 持续查看日志，直到确认完全启动，完全启动判定见下图
tail -f /root/isaac-sim.streaming.log
```

确认完全启动方式： 看到日志中出现这个字样



注意： 每次启动isaac sim的streaming服务，只能连接一个客户端（每个使用webviewer登录的都会被视为一个客户端），如果前面的人没有关闭客户端，后面访问会出现空白页面。可以通过可重启streaming服务强制访问（注意，这会杀死当前正在进行的未保存工作）

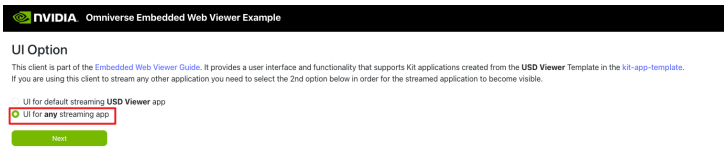
重启streaming服务： 使用~/stop_isaac_streaming.sh关闭，再用~/start_isaac_streaming.sh启动（webviewer不需要重启）

3.2 使用webviewer连接Isaac Sim streaming服务

1. 在开发机详情页中找到webviewer-port端口，复制【外网访问指令】，内容应是【IP:端口】



2. 在chrome浏览器打开，出现下面的选择界面，选择下面的【UI for **any** streaming app】选项，再点击Next



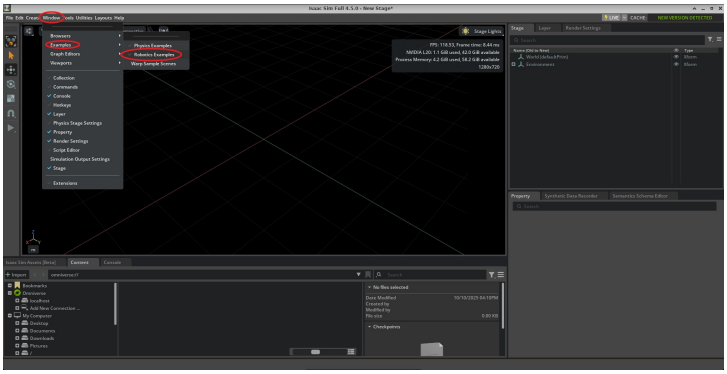
3. 由于 webviewer 的方式不支持密码鉴权，推荐配置 LB 的安全组来控制来源 IP 范围，避免服务泄露。您可以在开发机详情找到开发机使用的 BLB，跳转到BLB 页面配置安全组信息。请参考：BLB 支持安全组

4. 案例演示

以 RDP 启动 Isaac sim 为例：

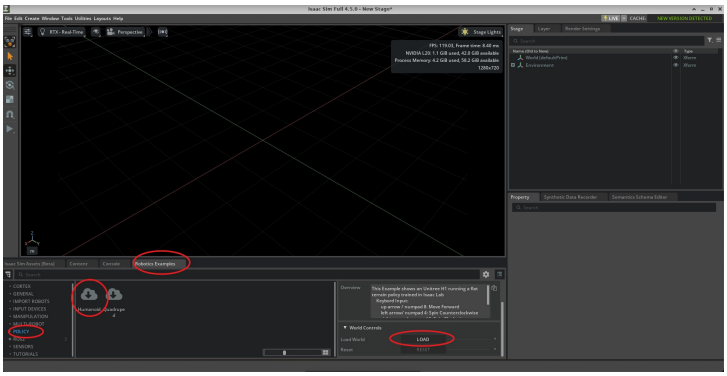
4.1 执行官方四足和人形可操作演示

4.1.1 步骤一：开启机器人案例选项卡



4.1.2 步骤二：加载Humanoid人形机器人案例

点击Load按钮加载，Reset按钮重置案例（或选择右侧的Quadruped四足机器人案例，也是点击Load按钮加载，Reset按钮重置案例）



一般来说，对一个模型或者脚本点击Load按钮，客户端窗口会停止响应一些时间进行网络加载

但在开发机刚新建时，这里的加载时间过长，也有几率是客户端卡死

如果客户端卡死，则可以点击右上角的x关闭客户端后，在命令行再次执行命令启动，一般一到两次之后，一旦有一次Load成功，之后客户端就不会再卡死



4.1.3 步骤三：操作机器人

窗口缩放操作：

- 在窗口内使用鼠标滚轮

Humanoid人形机器人操作：

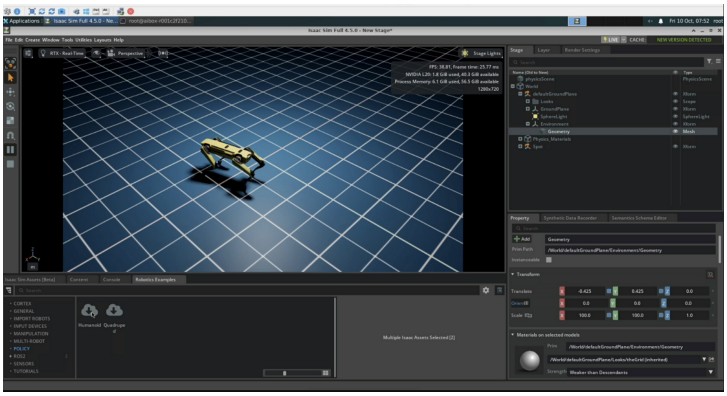
- 前进：向上箭头/数字键 8
- 左转：左箭头/数字 4
- 右转：右箭头/数字 6

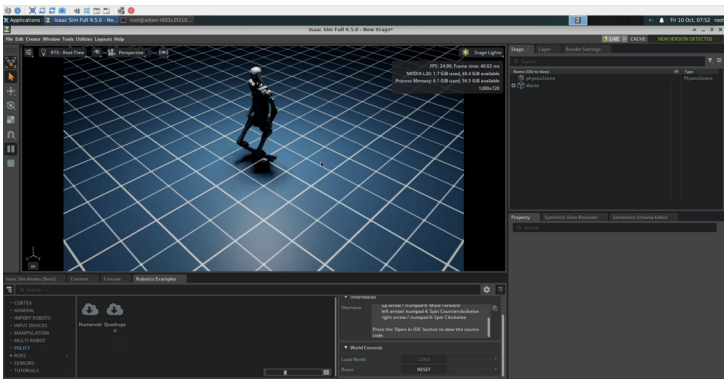
Quadruped四足机器人操作：

- 前进：向上箭头/数字键 8
- 后退：后退箭头/数字 2
- 向左移动：左箭头/数字 4
- 向右移动：右箭头/数字键 6

- 左转：N / NUM 7
- 右转：M / NUM 9

4.1.4 运行效果图片&视频

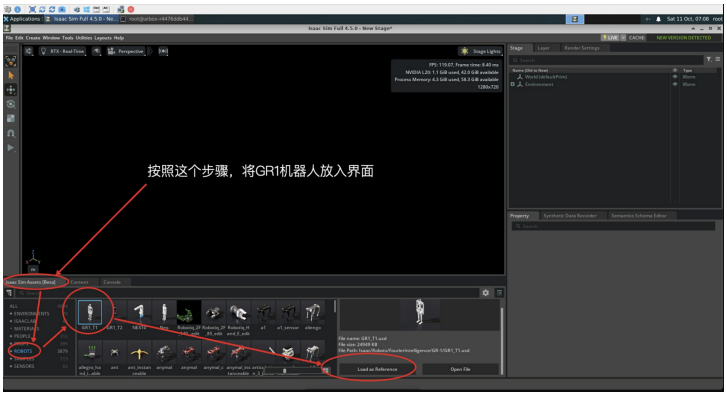




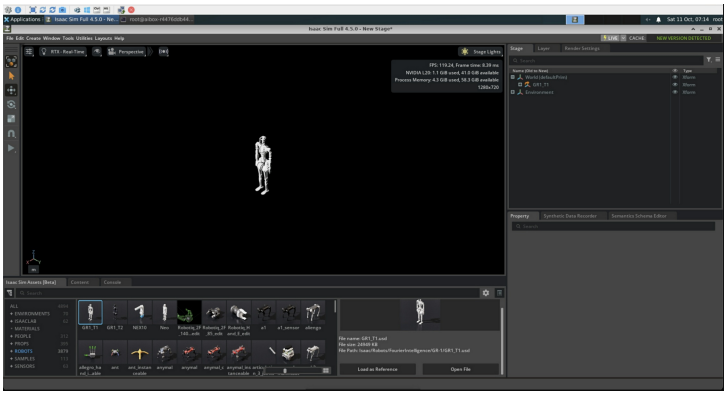
4.2 宇树GR1机器人简易操作演示

4.2.1 步骤一： 在Assets中找到GR1机器人，并加载到工作空间中

操作步骤：

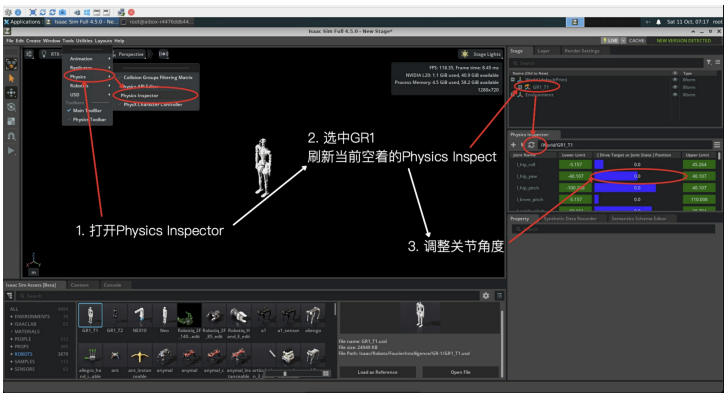


操作结果：



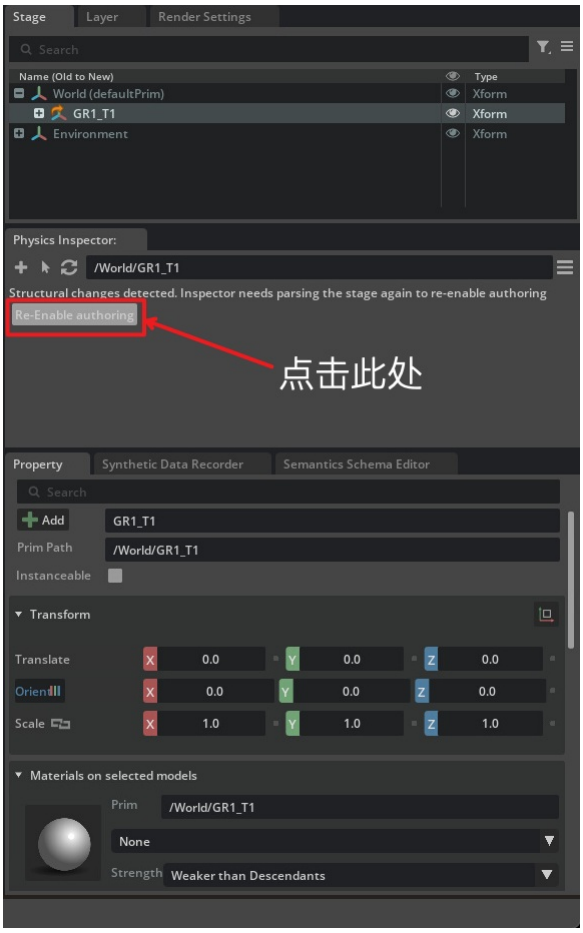
4.2.2 步骤二： 使用Physics Inspector工具，对关节进行简易操作

操作步骤：

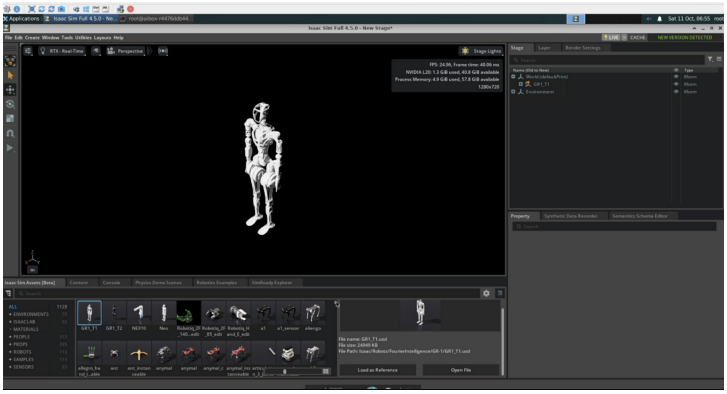


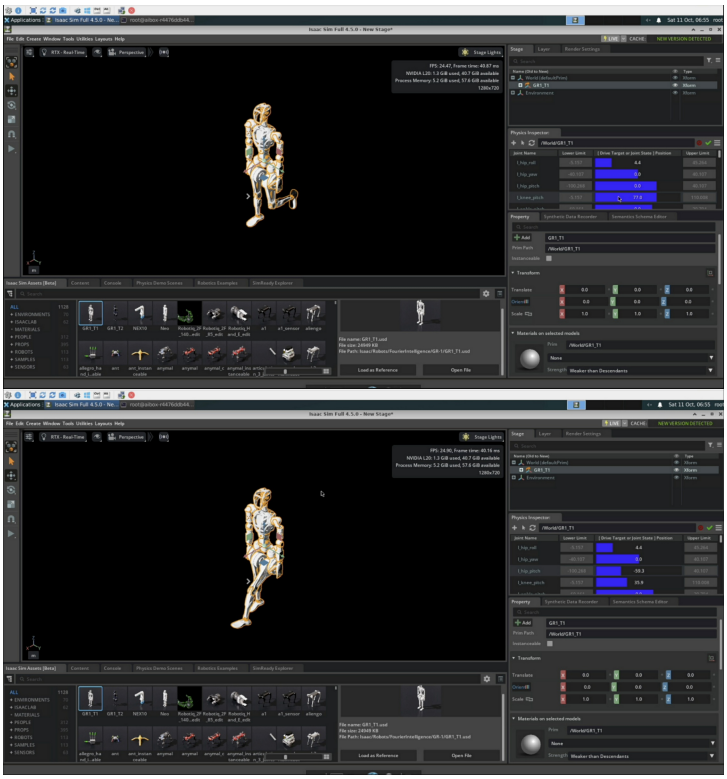
额外步骤：

如果按照上面操作后，Physics Inspector显示成下面的样子，则需要点击一下【Re-Enable authoring】按钮，就会显示关节角度界面



操作结果图片&视频：





4.3 Replicator 数据生成

Replicator，是一种用于生成具身智能训练数据的框架 这个框架可使用数据资产，根据要求，生成不同场景、不同情况、随机化的大量训练数据（生成结果可包含场景图片 + 精准标注） **数据资产**可使用多种开源标准，例如通用场景描述(USD)、PhysX、材质定义语言(MDL) 基于官方公开的示例，其大致工作流程为：

步骤	内容
1. 定义场景	基于USD构建或加载一个3D场景
2. 配置生成规则	设置相机、灯光、随机化参数（如物体、纹理、位置的随机范围）
3. 执行与记录	运行脚本，Replicator会自动在场景中采样大量随机状态，驱动渲染引擎和标注器，生成成对的图像和标注文件
4. 输出	结果以标准格式（如RGB图像、边界框JSON等）保存，可直接用于主流的AI训练框架（如PyTorch, TensorFlow）

下面我们将展示几个官方的例子 这两个示例均是 Isaac Sim官网示例，两个示例各有侧重，从不同方面示范了如何使用Replicator插件

示例1：物品数据集合成

官方链接：Object Based Synthetic Dataset Generation

这是一个使用 Isaac Sim 和 Replicator 生成以物体为中心的合成数据集的示例，该脚本会在一个预定义的区域（由不可见的碰撞墙围成）内，生成带有标签的物体和干扰物体，并从多个相机视点捕捉场景。同时，该脚本还演示了如何随机化相机姿态、对物体施加随机速度，以及如何触发自定义事件来随机化场景。这些随机化器可以是基于 Replicator 的，也可以是基于自定义的 Isaac Sim/USD API 的，并且可以在特定时间触发。

示例 1-2：GUI模式

```
 ##### 创建工作目录（若已存在则跳过）
mkdir -p ~/isaacsim_test
##### 进入工作目录，生成结果就会在当前目录下生成
cd /root/isaacsim_test/

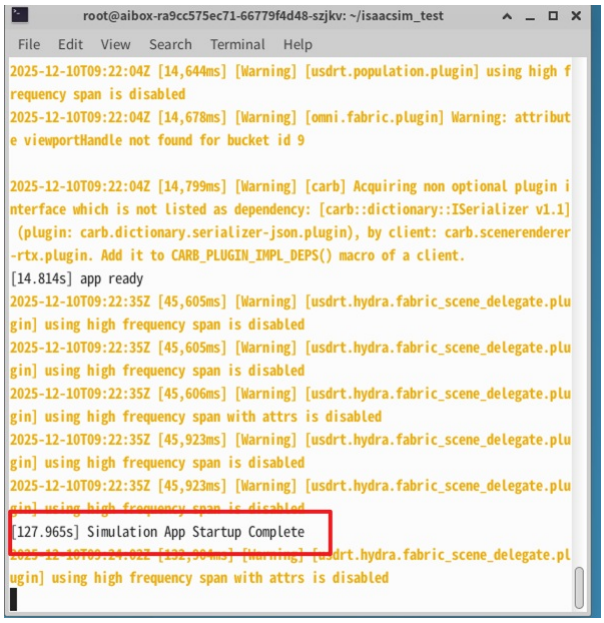
##### 下面的操作请注意根据Isaac Sim的不同版本加载不同的命令

##### 启动，使用的Assets为bos挂载盘，Isaac Sim 4.5.0版本
/isaac-sim/python.sh /isaac-sim/standalone_examples/replicator/object_based_sdg/object_based_sdg.py --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/4.5"

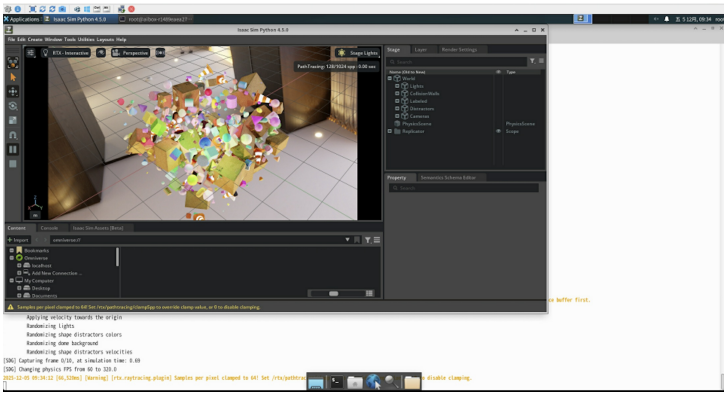
##### 启动，使用的Assets为bos挂载盘，Isaac Sim 5.0.0版本
/isaac-sim/python.sh /isaac-sim/standalone_examples/replicator/object_based_sdg/object_based_sdg.py --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/5.0"
```

1. 执行上面的文件后，会出现GUI界面

2. 需要等待一段时间，直到命令行出现【Simulation App Startup Complete】字样，之后GUI界面内会开始加载各种信息



3. 图像中的物品会在一个规划好区域内，随机运动 => 快要碰撞（距离小于阈值） => 随机重置运动（每次重置的时候会卡顿），这样反复循环，直到结束模拟



生成结果文件夹结构（当前工作目录下的_out_obj_based_sdg_pose_writer文件夹）：

```
 ##### 会生成30个结果，每个结果中有两个图片和一个json文件，000000-000029，这里就都用0000XX代替
##### 其中0000XX.png是没有标注的图片，0000XX_overlay.png则会给目标物品进行标注
_out_obj_based_sdg_pose_writer/
├── 0000XX.json
├── 0000XX_overlay.png
├── 0000XX.png
└── metadata.txt

0 directories, 91 files
```

示例 1-2：Headless模式

步骤 1：创建工作目录并拷贝官方脚本

执行以下命令，进入工作目录：

```
 ##### 创建工作目录（若已存在则跳过）
mkdir -p /root/isaacsim_test
##### 进入目标目录
cd /root/isaacsim_test/
```

步骤 2：新建配置

1. 执行以下命令，生成配置文件

2. 这个配置文件是以/isaac-sim/standalone_examples/replicator/object_based_sdg/config/object_based_sdg_centerpose_config.yaml 为基础，添加上headless: true，来达到无GUI模式的目的

3. 并且加上renderer: RaytracedLighting，来启动自然光源渲染方式

```
 cat > ./object_based_sdg_centerpose_config_headless.yaml << 'EOF'
launch_config:
  renderer: RaytracedLighting
  headless: true
writer_type: PoseWriter
writer_kwargs:
  output_dir: _out_obj_based_sdg_pose_writer_centerpose
  format: centerpose
  write_debug_images: true
  skip_empty_frames: false
EOF
```

步骤 3：运行脚本生成合成数据

执行以下命令启动数据生成（指定配置文件和 3D 资产路径）：

```
##### 下面的操作请注意根据Isaac Sim的不同版本加载不同的命令

##### 启动，使用的Assets为bos挂载盘，并使用刚才修改/生成的文件作为配置文件，Isaac Sim 4.5.0版本
/isaac-sim/python.sh /isaac-sim/standalone_examples/replicator/object_based_sdg/object_based_sdg.py \
 --config="./object_based_sdg_centerpose_config_headless.yaml" \
 --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/4.5"

##### 启动，使用的Assets为bos挂载盘，并使用刚才修改/生成的文件作为配置文件，Isaac Sim 5.0.0版本
/isaac-sim/python.sh /isaac-sim/standalone_examples/replicator/object_based_sdg/object_based_sdg.py \
 --config="./object_based_sdg_centerpose_config_headless.yaml" \
 --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/5.0"
```

结果查看生成的数据存储路径：/root/isaacsim_test/_out_obj_based_sdg_pose_writer_centerpose，也就是当前执行目录下创建_out_obj_based_sdg_pose_writer_centerpose文件夹

示例2 自助移动机器人导航数据集合成

官方链接：Randomization in Simulation – AMR Navigation

使用 Isaac Sim 与 Replicator 从模拟环境中采集合成数据的示例（AMR 导航），本教程旨在演示如何设置一个 Isaac Sim 模拟场景，并结合 omni.replicator 扩展，利用多样化的随机化技术来采集合成数据，将多种随机情况混合在一起。

示例 1-1：GUI 模式

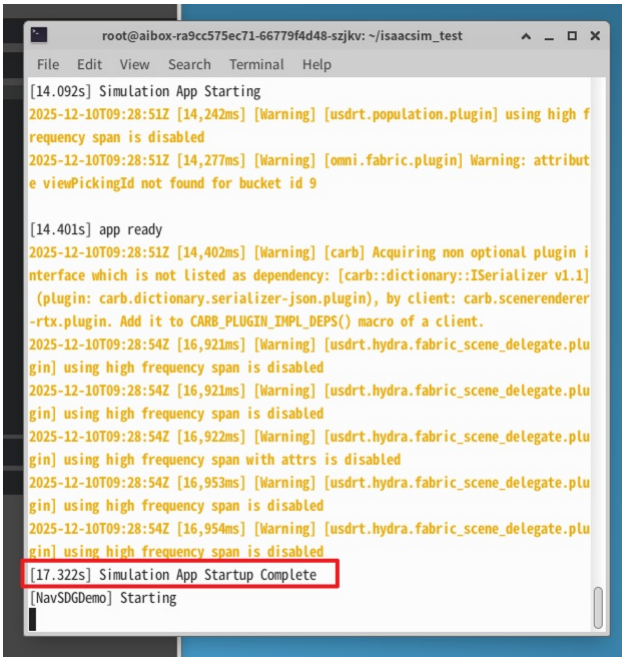
```
##### 创建工作目录（若已存在则跳过）
mkdir -p ~/isaacsim_test
##### 进入工作目录，生成结果就会在当前目录下生成
cd /root/isaacsim_test/

##### 下面的操作请注意根据Isaac Sim的不同版本加载不同的命令

##### 启动，使用的Assets为bos挂载盘，Isaac Sim 4.5.0版本
/isaac-sim/python.sh /isaac-sim/standalone_examples/replicator/amr_navigation.py --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/4.5"

##### 启动，使用的Assets为bos挂载盘，Isaac Sim 5.0.0版本
/isaac-sim/python.sh /isaac-sim/standalone_examples/replicator/amr_navigation.py --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/5.0"
```

1. 执行上面的文件后， 等到命令行内出现【Simulation App Startup Complete】字样后，之后GUI界面内会开始加载各种信息



2. gui界面开始播放后，可以在Isaac Sim Python窗口内使用下面的操作

    1. 按住鼠标右键可操控场景镜头朝向（镜头改变方向和鼠标移动方向相同）

    2. 鼠标滚轮控制镜头缩放

3. 生成结果文件夹结构（当前工作目录下的_out_obj_based_sdg_pose_writer文件夹）：

```
##### 会生成9个结果，每个结果中有两个图片文件，0000-0008，这里就都用000X代替
##### 其中left_sensor文件夹是左侧镜头的图片，right_sensor文件夹是右侧镜头的图片
_out_nav_sdg_demo/
├── left_sensor
│   └── rgb
│       ├── rgb_000X.png
├── metadata.txt
└── right_sensor
    └── rgb
        ├── rgb_000X.png

4 directories, 19 files
```

示例 2-2：Headless模式

按照下述操作执行

```
##### 创建工作目录（若已存在则跳过）
mkdir -p ~/isaacsim_test
##### 进入工作目录
cd /root/isaacsim_test/
##### 复制官方例子，并且把启动模式从gui图像模式替换为无GUI命令行模式
cp /isaac-sim/standalone_examples/replicator/amr_navigation.py ./
sed -i 's/simulation_app = SimulationApp(launch_config={"headless": False})/simulation_app = SimulationApp(launch_config={"headless": True})/g' ./amr_navigation.py

##### 下面的操作请注意根据Isaac Sim的不同版本加载不同的命令

##### 启动，使用的Assets为bos挂载盘，Isaac Sim 4.5.0版本
/isaac-sim/python.sh ./amr_navigation.py --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/4.5"

##### 启动，使用的Assets为bos挂载盘，Isaac Sim 5.0.0版本
/isaac-sim/python.sh ./amr_navigation.py --/persistent/isaac/asset_root/default="/mnt/bos/isaac-sim/Assets/Isaac/5.0"
```

Isaac Lab遥操环境搭建和图形可视化

案例介绍

本文以 Isaac lab 2.1.0 版本为例，指导客户如何搭建遥操系统采集数据以及图形的可视化

Isaac Lab提供了SE(2)和SE(3)空间的命令接口用于机器人控制：

- SE(2) teleoperation: 返回线性x-y速度和偏航角速度
- SE(3) teleoperation: 返回6维向量表示姿态变化
- 注意: 目前Isaac Lab Mimic仅在Linux上支持。

部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	建议8核及以上	按需
内存	建议64G及以上	按需
GPU	L20等 支持RT Core的NVIDIA GPU卡 1卡及以上	按需
CDS	按需	按需
其它	无	无

使用需知：

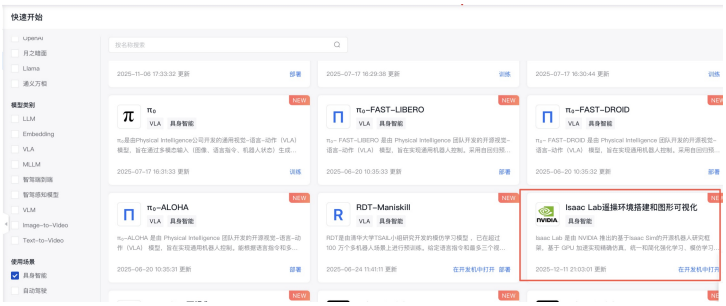
1. 由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置HTTPS网络代理支持访问。服务网卡的配置参考

2. 建议在距离您最近的地域创建开发机实例使用当前方案。如果远程桌面不流畅，建议配置 RDP 比特率为16位

使用说明

1. 创建与登录开发机

1.1 创建开发机实例



点击在开发机中打开，创建开发机，主要配置如下：

- 资源配置：
  - GPU选择： 建议使用 NVIDIA L20
  - CPU/内存：推荐8核64G及以上，保证运行所需的基础资源配置
- 启动自定义端口，选择支持**公网访问**的BLB（建议选择网络带宽≥50Mbps的BLB，带宽越大越好，越少共享复用越好）
- 新增访问端口：
  - 服务名称：rdp
  - 开发机容器监听端口：3389
  - BLB监听端口：随机分配即可

1.2 登录开发机

填写完表单，提交后，在开发机列表可见创建的开发机，等待资源调度并部署成功后，点击登录，即可进入开发机webIDE



2. 配置访问远程桌面

登录开发机，进入开发机webIDE，打开Terminal界面

2.1 设置RDP远程桌面登陆的账号密码

使用chpasswd工具设置RDP远程桌面登陆的账号密码

```
echo "root:xrdp-aihc" | chpasswd
```

其中，

- root：为设置用户名。
- xrdp-aihc：为登陆密码，请替换为你的密码。
- chpasswd：Linux 系统中一个用于**批量设置或者更新用户密码**的实用命令行工具，支持从标准输入或指定的文件中读取用户名和密码的组合，然后自动完成密码修改

2.2 启动RDP服务

启动 RDP 服务

```
/usr/bin/supervisord -c /etc/supervisor/conf.d/xrdp.conf
```

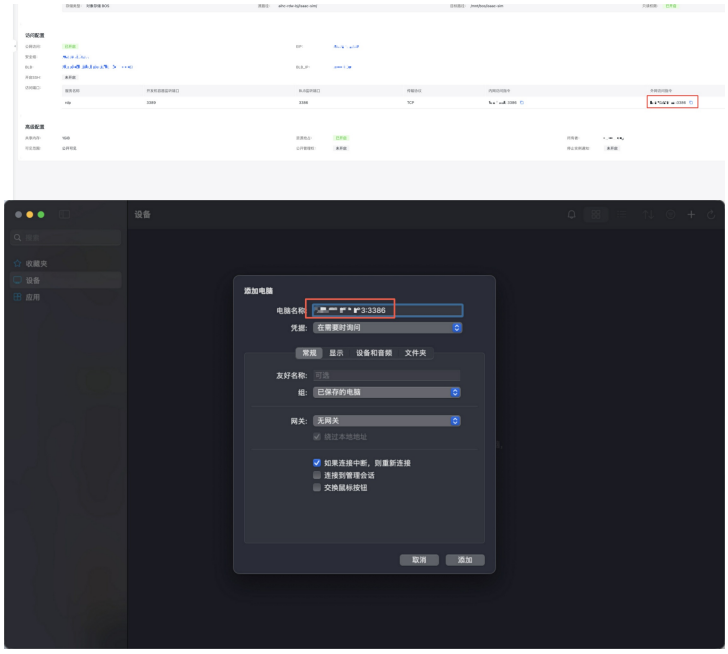
2.3 通过桌面客户端访问

在本地电脑或者需要连接开发机Isaac SIm环境的电脑，安装Windows App。以Mac为例，从app store安装

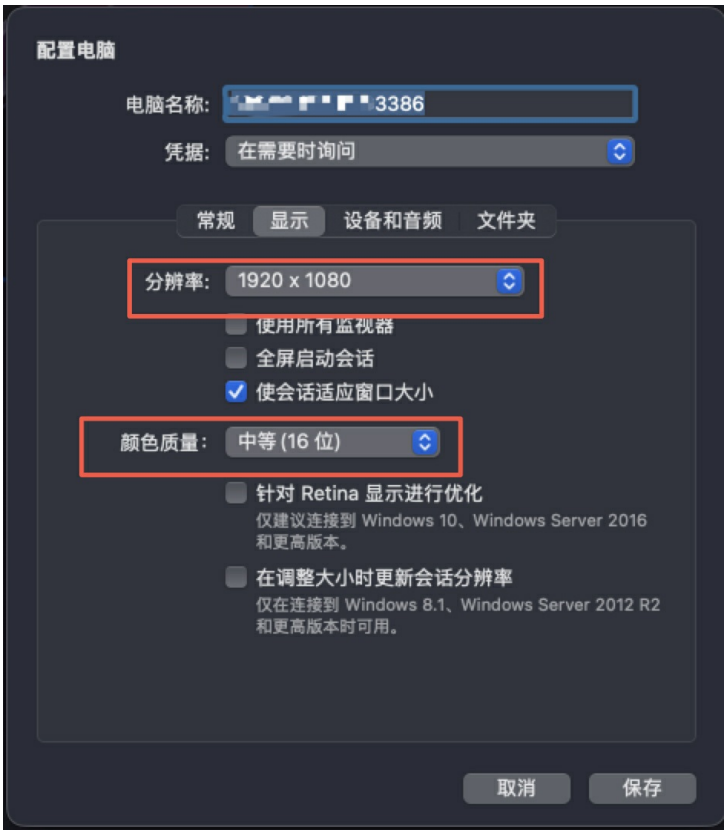


本次安装后，

1. 在Windows APP 中依次点击右上角的**加号** 和 **添加电脑** 菜单，新建远程连接
2. 将从开发机复制的 RDP 的公网地址加端口（公网地址可从开发机详情页的自定义端口配置区域获取）输入到访问地址栏



3. 输入2.1步骤中中设置的用户名和密码
4. 登陆远程桌面
5. 如果您的远程桌面在运行过程中出现卡顿现象，建议您降低远程连接的分辨率和颜色质量



3. 案例演示

登录到远程桌面，打开远程桌面的命令行工具



3.1 使用键盘进行遥操

以Isaac-Stack-Cube-Franka-IK-Rel-v0为例，建议启动前建议设置HTTP和HTTPS网络代理。

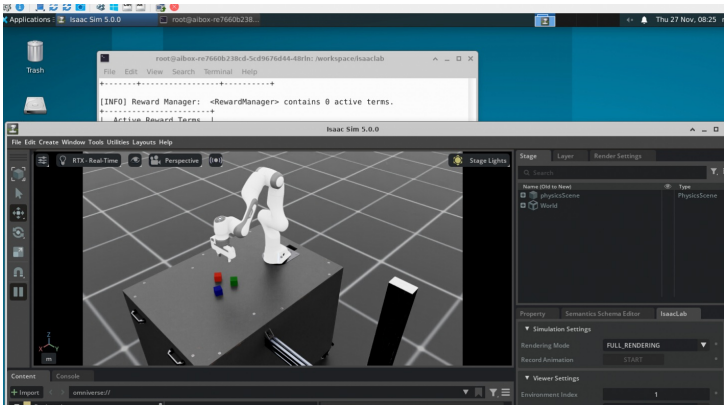
```
 mkdir -p datasets

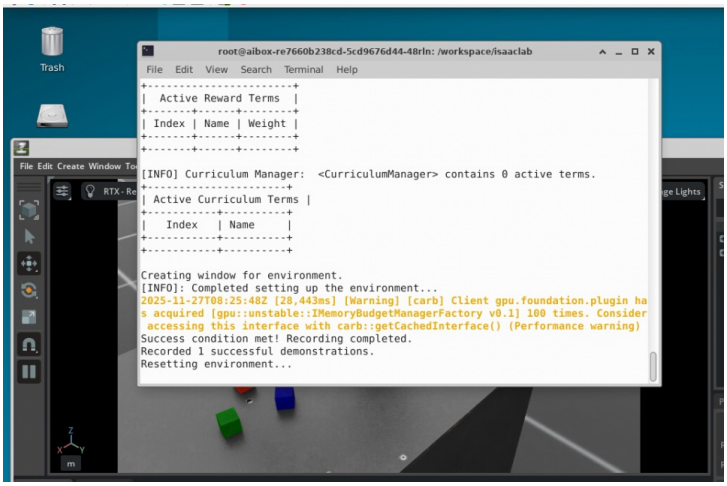
#可通过 ./isaaclab.sh -p scripts/environments/list_envs.py 查看环境列表,根据需要切换环境

#使用键盘操作启动 Isaac-Stack-Cube-Franka-IK-Rel-v0 任务
./isaaclab.sh -p scripts/tools/record_demos.py \
  --task Isaac-Stack-Cube-Franka-IK-Rel-v0 \
  --teleop_device keyboard \
  --dataset_file ./datasets/dataset.hdf5 \
  --num_demos 10
```

- 参数详解
  - 必需参数
    - --task TASK: 指定要运行的任务环境名称（如Isaac-Stack-Cube-Franka-IK-Rel-v0）
  - 可选参数
    - --teleop_device TELEOP_DEVICE: 指定遥操作设备类型（keyboard, spacemouse, handtracking等）
    - --dataset_file DATASET_FILE: 指定保存演示数据的文件路径（HDF5格式）
    - --step_hz STEP_HZ: 控制演示收集的频率（每秒步数）
    - --num_demos NUM_DEMOS: 指定要收集的演示episode数量
    - --num_success_steps NUM_SUCCESS_STEPS: 指定成功episode所需的最小步数
    - --enable_pinocchio: 启用Pinocchio动力学库

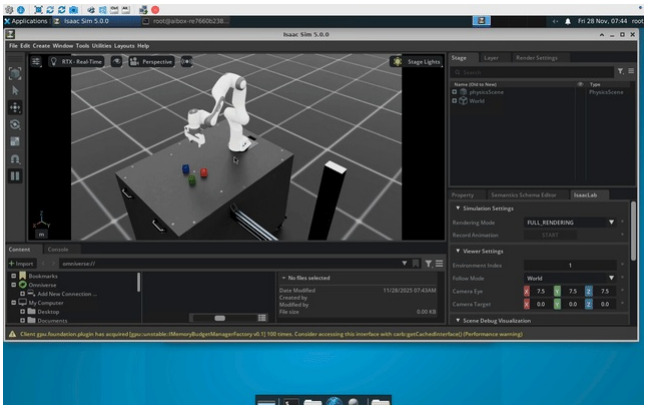
- 操作指南
  - SE(3) 键盘控制器：Se3Keyboard
    - 重置所有指令：R
    - 切换夹爪（打开/关闭）：K
    - 沿x轴移动机械臂：W/S
    - 沿y轴移动机械臂：A/D
    - 沿z轴移动机械臂：Q/E
    - 绕x轴旋转机械臂：Z/X
    - 绕y轴旋转机械臂：T/G
    - 绕z轴旋转机械臂：C/V





验证回放数据

```
./isaaclab.sh -p scripts/tools/replay_demos.py \
  --task Isaac-Stack-Cube-Franka-IK-Rel-v0 \
  --dataset_file ./datasets/dataset.hdf5
```



3.2 使用空间鼠标进行遥操

以Isaac-Stack-Cube-Franka-IK-Rel-v0为例，建议启动前建议设置HTTP和HTTPS网络代理。

```
 #使用SpaceMouse(空间鼠标)操作 启动 Isaac-Stack-Cube-Franka-IK-Rel-v0 任务
./isaaclab.sh -p scripts/tools/record_demos.py \
  --task Isaac-Stack-Cube-Franka-IK-Rel-v0 \
  --teleop_device spacemouse \
  --dataset_file ./datasets/dataset.hdf5 \
  --num_demos 10
```

- 操作指南
  - SE(3) SpaceMouse 控制器：Se3SpaceMouse
    - 重置所有指令：右键单击
    - 切换夹爪（打开/关闭）：单击 SpaceMouse 左键
    - 沿 x/y 轴移动机械臂：倾斜 SpaceMouse
    - 沿 z 轴移动机械臂：推或拉 SpaceMouse
    - 旋转机械臂：扭转 SpaceMouse

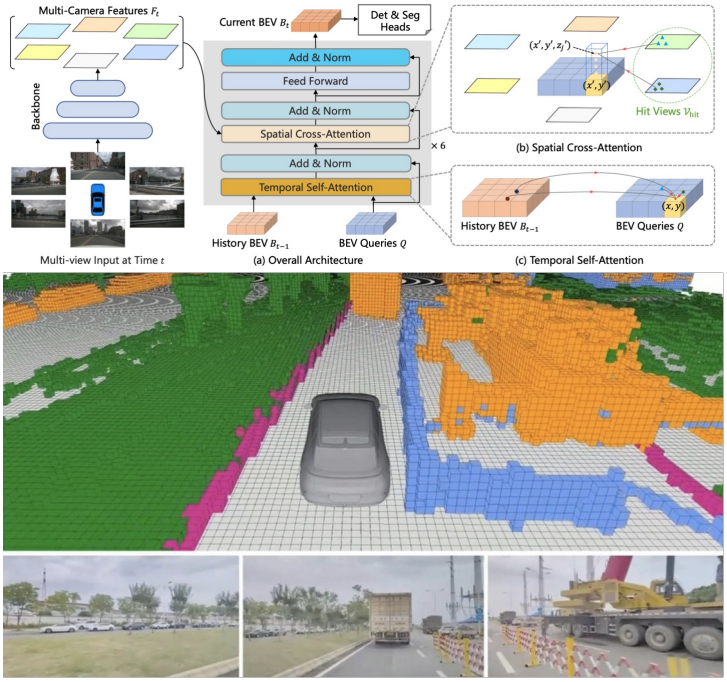
同时，Isaac lab 可通过Apple Vision Pro 遥操收集数据，具体请参考：https://isaac-sim.github.io/IsaacLab/main/source/overview/imitation-learning/teleop_imitation.html

Apollo-BEV-Train

为加速 BEV感知模型从研发到验证的闭环，Apollo 自动驾驶平台提供了一套**开箱即用**的工具链，涵盖**模型训练**（Apollo-BEV-Train） → **模型导出**（Apollo-BEV-Model-Export） → **模型验证**（Apollo-Model-Deployment）三大核心阶段。用户可按此顺序依次使用三个标准化环境，高效完成 BEV 模型的开发与评估。

介绍

近些年来，基于纯视觉+BEV的目标检测和占据栅格任务称为自动驾驶的研究热点。2022年提出的BEVFormer网络，使用环视图像作为输入，使用Transformer架构生成BEV特征，完成目标检测任务。占据栅格任务是一种通过预测三维栅格来表达真实世界的任务，如下图所示，栅格代表对应位置被占据，不同颜色代表不同的类别。占据栅格任务可以很好地处理自动驾驶中面临的异形障碍物问题，为此受到广泛的研究。



Apollo 11.0发布了纯视觉模型Apollo-vision-Net，具有如下的特性：

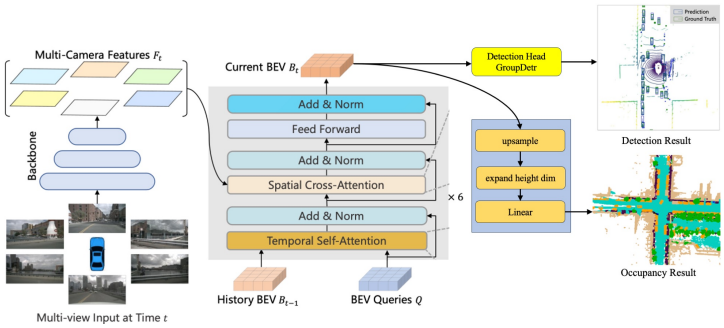
1. 【 更先进 】将视觉 bev 目标检测 + occ 占用网络的主流感知范式引入 Apollo 开源框架
2. 【 效果好 】在业界经典模型基础之上进行优化，各自的效果均超越业界经典模型效果。目标检测 mAP 较 bevformer（2022 ECCV）提升 6.74% ，occ miou较OccNet（2023 ICCV）提升 2.39%
3. 【 性能高 】模型共享backbone，多任务联合训练优化，在单个 jetson orin 平台上可达到 5hz 的推理帧率，在效果与性能上均取得1+1>2的效果

Apollo-vision-Net 介绍

网络结构

Apollo-vision-Net的整体网络结构如下所示：

- 输入环视图像，经过image backbone提取图像特征
- Transformer encoder部分
  - 使用temporal self-attention融合当前帧bev queries和历史帧bev queries
  - 使用spatial cross-attention融合bev queries和图像特征
  - 经过6个encoder layer后输出bev queries
- detection head：将bev queries作为输入，使用groupdetr网络进行目标检测
- occ head：对bev queries进行上采样，随后扩充高度方向特征，最后使用linear层预测每个栅格类别信息



我们对Apollo-vision-Net进行了如下优化，显著提升了目标检测、占据栅格的分数以及模型性能：

- image backbone：使用深度估计数据（Toyota DDAD15M）预训练的DLA-34替换ResNet-50，降低了模型复杂度同时提升了效果
- image neck：使用SecondFPN网络替换单尺度FPN网络，提升了模型整体效果
- detection head：使用GroupDETR替换DETR，在不增加耗时的前提下，显著提升目标检测效果
- occ head：在Transformer encoder部分使用低分辨率bev queries（50*50*），*在occ head部分再上采样至高分辨率（200*200），可大幅提升模型性能
- occ loss：将occ focal loss weight从1.0提升至10.0，引入affinity loss和lovasz-softmax loss，大幅提升occ检测效果

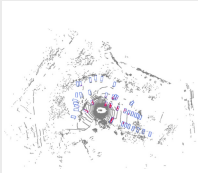
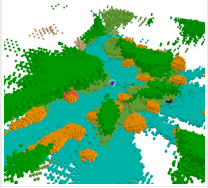
定量结果

在Nuscenes数据集上，Apollo-vision-Net目标检测mAP分数超越bevformer-tiny（2022 ECCV）6.74%，超越OccNet-R50（2023 ICCV）2.39%

	目标检测mAP （val dataset）	占据栅格miou (OpenOcc val dataset)
bevformer-tiny （2022 ECCV）	25.2%	-
OccNet-R50 （2023 ICCV）	-	19.48%
Apollo-vision-Net (ours)	31.94% （↑ 6.74）	21.87% （↑ 2.39）

定性结果

Nuscenes数据集结果

图像	目标检测结果	occ结果
		

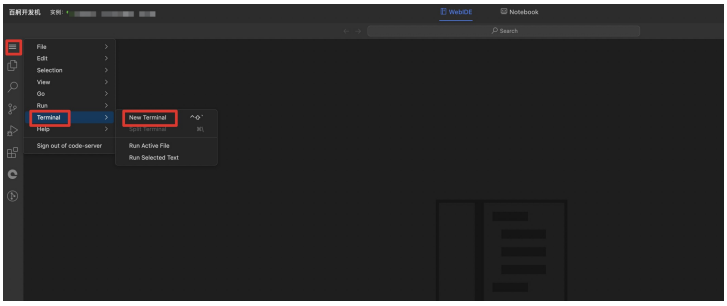
百度自动驾驶数据集结果 为了进一步验证效果，我们使用百度自动驾驶数据对Apollo-vision-Net进行训练，同时对occ的分辨率进行提升（0.5m*0.5m*0.5m->0.2m*0.2m*0.2m）可以看到Apollo-vision-Net可以提供复杂城市道路场景下的准确目标检测和occ检测。

部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	按需	建议按表单默认值及以上
内存	按需	建议按表单默认值及以上
GPU	24G以上显存	建议24G以上显存
CDS	按需	按需

模型训练使用说明

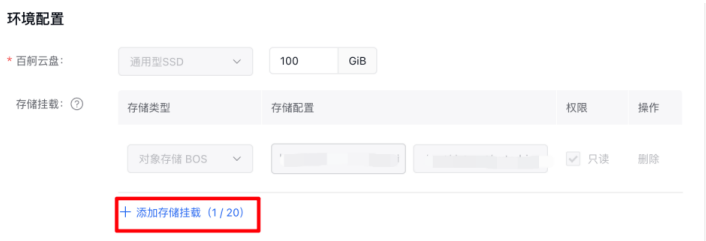
创建与登录开发机 根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，并打开VScode中的terminal



- 代码保存路径：/root/Apollo-Vision-Net
- 数据默认挂载路径：/mnt/dataset/nuscenes_data

数据集处理

存储挂载配置 默认在bos挂载nuscenes数据集



1. 挂载PFS或CFS

在创建或更新开发机时，在存储挂载配置中选择集群对应的 PFS 或 CFS 存储，并设置自定义挂载路径（示例路径：`/mnt/pfs/nuscenes_data`）可以在创建或者更新开发机配置时，选择存储挂载，选择集群对应的

PFS或者CFS，可自定义挂载路径  **数据集迁移** 创建数据集目录结构

```
cd /mnt/pfs/nuscenes_data
mkdir -p bev_data/nuscenes
```

从bos上迁移数据集至PFS文件

```
cp -rv  bos挂载路径/apollo_nuscenes_bev_data/nuscenes/*  /mnt/pfs/nuscenes_data/bev_data/nuscenes/
cp -rv  bos挂载路径/apollo_nuscenes_bev_data/can_bus/*  /mnt/pfs/nuscenes_data/bev_data/
cp -rv  bos挂载路径/apollo_nuscenes_bev_data/occ_gt_release_v1_0/* /mnt/pfs/nuscenes_data/bev_data/
```

数据集解压与整理 NuScenes 数据集处理

```
cd /mnt/pfs/nuscenes_data/bev_data/nuscenes
##### 运行指令解压数据集，也可以开多个终端依次解压数据集
bash tar_files.sh
```

nuscenes数据集解压后目录结构：

```
/mnt/pfs/nuscenes_data/bev_data# tree nuscenes/ -L 1
nuscenes/
├── maps //地图
├── samples //数据集
├── sweeps //数据集
├── v1.0-trainval//数据集
├── tar_files.sh //解压指令
├── nuscenes_infos_temporal_train_occ_gt.pkl //目标检测和occupancy的pkl文件
├── nuscenes_infos_temporal_val_occ_gt.pkl  //目标检测和occupancy的pkl文件
├── maps.tar.gz //地图压缩包
├── v1.0-trainval01_blobs.tgz //数据集压缩包
├── v1.0-trainval02_blobs.tgz //数据集压缩包
├── v1.0-trainval03_blobs.tgz //数据集压缩包
├── v1.0-trainval04_blobs.tgz //数据集压缩包
├── v1.0-trainval05_blobs.tgz //数据集压缩包
├── v1.0-trainval06_blobs.tgz //数据集压缩包
├── v1.0-trainval07_blobs.tgz //数据集压缩包
├── v1.0-trainval08_blobs.tgz //数据集压缩包
├── v1.0-trainval09_blobs.tgz //数据集压缩包
└── v1.0-trainval10_blobs.tgz //数据集压缩包
```

occ_gt_release_v1_0数据集处理：

解压occ_gt_release_v1_0压缩包

```
cd /mnt/pfs/nuscenes_data/bev_data
tar -zxvf occ_gt_release_v1_0.tar.gz
```

解压后

```
occ_gt_release_v1_0
├── occ_gt_train.json
├── occ_gt_val.json
├── train
└── val
```

复制目标检测和occupancy的pkl文件到occ_gt_release_v1_0

```
cp /mnt/pfs/nuscenes_data/bev_data/nuscenes/*.pkl /mnt/pfs/nuscenes_data/bev_data/occ_gt_release_v1_0/
```

最终目录结构

```
/mnt/pfs/nuscenes_data/bev_data# tree occ_gt_release_v1_0 -L 1
occ_gt_release_v1_0
├── nuscenes_infos_temporal_train_occ_gt.pkl
├── nuscenes_infos_temporal_val_occ_gt.pkl
├── occ_gt_train.json
├── occ_gt_val.json
├── train
└── val
```

CAN-Bus 数据处理

```
cd /mnt/pfs/nuscenes_data/bev_data
tar -zxvf can_bus.tar.gz
```

数据集结构规范

```
├── data/
│   ├── can_bus/
│   ├── nuscenes/
│   │   ├── maps/
│   │   ├── samples/
│   │   ├── sweeps/
│   │   ├── v1.0-test
│   │   ├── v1.0-trainval
│   │   ├── nuscenes_infos_temporal_train.pkl
│   │   ├── nuscenes_infos_temporal_val.pkl
│   ├── occ_gt_release_v1_0/
│   │   ├── train/
│   │   ├── val/
│   │   ├── occ_gt_train.json
│   │   ├── occ_gt_val.json
│   │   ├── nuscenes_infos_temporal_train_occ_gt.pkl
│   │   ├── nuscenes_infos_temporal_val_occ_gt.pkl
```

数据集挂载 开发机内数据集挂载：

```
cd ~/Apollo-Vision-Net/data

##### 创建符号链接（注意使用正确的 -s 参数）
ln -s /mnt/pfs/nuscenes_data/bev_data/nuscenes nuscenes
ln -s /mnt/pfs/nuscenes_data/bev_data/occ_gt_release_v1_0 occ_gt_release_v1_0
ln -s /mnt/pfs/nuscenes_data/bev_data/can_bus can_bus
```

最终项目结构：

```
~/Apollo-Vision-Net/data/
├── can_bus -> /mnt/pfs/nuscenes_data/bev_data/can_bus
├── nuscenes -> /mnt/pfs/nuscenes_data/bev_data/nuscenes
└── occ_gt_release_v1_0 -> /mnt/pfs/nuscenes_data/bev_data/occ_gt_release_v1_0
```

训练模型

```
##### 进入工作目录
cd ~/Apollo-Vision-Net/
##### 训练模型
./tools/dist_train.sh \                              # 训练脚本
  ./projects/configs/bevformer/bev_tiny_det_occ_apollo.py \ # 模型配置文件路径
  8                                          # 使用8个GPU进行分布式训练
```

训练暂停后继续训练指令

```
##### 进入工作目录
cd ~/Apollo-Vision-Net/

##### 训练模型
./tools/dist_train.sh \                               # 训练脚本
  ./projects/configs/bevformer/bev_tiny_det_occ_apollo.py \   # 模型配置文件路径
  8 \                                         # 使用8个GPU进行分布式训练
  --resume-from work_dirs/bev_tiny_det_occ_apollo/epoch_4.pth # 从第4epoch的检查点恢复训练
```

模型评测

```
##### 进入工作目录
cd ~/Apollo-Vision-Net/

##### 启动评测脚本
./tools/dist_test.sh \
  ./projects/configs/bevformer/bev_tiny_det_occ_apollo.py \ # 模型配置文件路径
  work_dirs/bev_tiny_det_occ_apollo/epoch_4.pth \         # 训练好的模型权重文件
  4                                          # 使用*个GPU进行分布式测试
```

将训练得到的 .pth 模型文件在模型导出所使用的开发机上执行模型导出功能，将该.pth文件转换为符合 Apollo 要求的 ONNX 模型文件，以用于后续apollo系统部署验证模型效果。

附：Apollo 11.0 BEV+OCC 模型训练使用

流程	使用方式	备注
Apollo-BEV-Train	百舸平台	训练 Apollo BEV+OCC模型
Apollo-BEV-Model-Export	百舸平台	将训练好的BEV+OCC模型导出成onnx文件
Apollo-Model-Deployment	百舸平台或本地验证	apollo环境镜像，验证bev模型效果以及仿真等apollo工具

Isaac工作流--动作数据合成&扩增、模仿学习

NVIDIA Isaac Sim 是构建于 NVIDIA Omniverse 框架之上的高性能机器人仿真环境，它利用 GPU 加速的物理引擎实现物理级精确的仿真，并具备大规模、多传感器 RTX 渲染能力

。该平台集成了从合成数据生成、强化学习训练、ROS 系统对接到数字孪生应用在内的完整端到端开发工作流。本文将具体演示如何在百舸 AI 计算平台上，运用 Isaac Lab 组件提供的数据合成与扩增功能，以及其模仿学习算法，并结合 Cosmos 模型的视觉增强技术，来生成大规模、高质量的机器人演示数据集，进而训练出能够有效应对视觉场景变化的模仿学习策略

。所述工作流基于 Isaac Lab 2.2、Isaac Sim 5.0.0 和 Cosmos-Transfer1 环境构建并完成测试验证。

使用说明

整个流程包含以下四个主要阶段：

1. **生成演示数据**：使用 Isaac Lab Mimic 生成初始的机器人操作演示。您可以直接在开发机运行仿真任务，或通过分布式训练进行采集。我们提供了一份示例数据集，您也可参照官方指南采集自己的数据。
2. **视觉数据增强**：通过部署并调用 Cosmos 在线服务，对演示中的视觉观测进行增强，以提升数据的多样性和质量。
3. **训练策略模型**：利用 Isaac Lab 支持的 robomimic 框架，训练一个 BC-RNN 策略模型。此步骤支持在开发机上进行本地训练，也可提交分布式训练任务进行离线训练。

4. **评估模型鲁棒性**：对训练完成的策略模型进行系统性评估，测试其在不同条件下的鲁棒性与性能。

一、环境配置

	部署要求	最佳实践
CPU	8核	建议按表单默认值及以上
内存	64G	建议按表单默认值及以上
GPU	Isaac Lab 2.2.0：L20 / 支持RT Core的GPU卡 ×1	视觉仿真渲染场景显卡必须含有RTCore；基于物理仿真的强化学习场景可以使用A10/A100/V100等其他不含RTCore的显卡
CDS	按需	建议存储资源>500G, 渲染时会涉及到大量3D资产及中间文件存储
其它	无	无

注意：部署该实例的集群需要具备访问公网的能力，配置方式见：https://cloud.baidu.com/doc/CCE/s/plmk196bx

> Isaac Lab 2.2.0 依赖于 Isaac Sim 5.0，该仿真环境在渲染时使用了 DLSS（超采样技术，Deep Learning Super Sampling），因此推荐使用支持 RT Core 的 GPU 卡。**服务网卡配置** 由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置 HTTP & HTTPS 网络代理支持。服务网卡配置参考

二、数据准备

2.1 原始数据下载 使用 BOS 下载 Nvdia 开源原始数据集

```
bcecmd bos cp bos:/aihc-rdw-bj/cosmos_data/dataset.hdf5 /mnt/data/isaac_tmp/dataset/dataset.hdf5
```

2.2 数据标注 使用 isaaclab 脚本进行自动标注，同时您可以通过调整scripts/imitation_learning/isaaclab_mimic/annotate_demos.py中的`num_envs`参数，以增加或减少并行环境数量，充分利用显存。

```
 /workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/imitation_learning/isaaclab_mimic/annotate_demos.py \
--enable_cameras --task Isaac-Stack-Cube-Franka-IK-Rel-Visuomotor-Mimic-v0 --auto \
--input_file /mnt/data/isaac_tmp/dataset/dataset.hdf5 --output_file /mnt/data/isaac_tmp/dataset/annotated_dataset.hdf5 --headless
```

运行结束后文件夹中出现annotated_dataset.hdf5 文件

2.3 数据增广 使用以下脚本进行数据增广，可以通过调整`num_envs`参数，以增加或减少并行环境数量来充分利用显存，也可以通过--device cpu来限制计算运行设备，本处使用 GPU 因此没有添加该参数，同时通过调整`generation_num_trials`来调整增广数目（此处耗时巨大，若要体验可以将数字调整为 10 来进行体验）

```
 ##### 完整生成代码
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/imitation_learning/isaaclab_mimic/generate_dataset.py \
--enable_cameras --headless --num_envs 8 --generation_num_trials 1000 \
--input_file /mnt/data/isaac_tmp/dataset/annotated_dataset.hdf5 --output_file mnt/data/isaac_tmp/dataset/mimic_dataset_1k.hdf5 \
--task Isaac-Stack-Cube-Franka-IK-Rel-Visuomotor-Cosmos-Mimic-v0

##### 体验代码
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/imitation_learning/isaaclab_mimic/generate_dataset.py \
--enable_cameras --headless --num_envs 8 --generation_num_trials 10 \
--input_file /mnt/data/isaac_tmp/dataset/annotated_dataset.hdf5 --output_file mnt/data/isaac_tmp/dataset/mimic_dataset_10.hdf5 \
--task Isaac-Stack-Cube-Franka-IK-Rel-Visuomotor-Cosmos-Mimic-v0
```

同时可以通过下载 bos 预置数据直接进行体验：

```
bcecmd bos cp bos:/aihc-rdw-bj/cosmos_data/mimic_dataset_1k.hdf5 /mnt/data/isaac_tmp/dataset/mimic_dataset_1k.hdf5
```

三、使用 Cosmos 进行数据增强

3.1 格式转换 hdf5->mp4 通过以下代码将hdf5 格式文件转换为 mp4 格式：

```
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/tools/hdf5_to_mp4.py \
  --input_file /mnt/data/isaac_tmp/dataset/mimic_dataset_1k.hdf5 \
  --output_dir /mnt/data/isaac_tmp/dataset/mimic_dataset_1k_mp4 \
  --input_keys table_cam table_cam_depth table_cam_shaded_segmentation
```

3.2 Cosmos增强提示词生成 使用以下脚本生成提示词

```
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/tools/cosmos/cosmos_prompt_gen.py \
  --templates_path /workspace/isaaclab/scripts/tools/cosmos/transfer1_templates.json \
  --num_prompts 10 \
  --output_path /mnt/data/isaac_tmp/dataset/cosmos_prompts.txt
```

您也可以自己编写一些提示词，以下是 Nvidia 官方给出的提示词编写法则：

1. 尽量详细地保留prompts。最好对生成过程中每个可见对象/感兴趣区域都有一些指导。例如，我们提供的prompts涵盖了桌子、灯光、背景、机械臂、立方体和整体布景的细节。
2. 尽量使增强指令尽可能地真实和连贯。模型在保留输入控制视频的关键特征方面做得越不真实或不传统，它的表现就越糟糕。
3. 保持每个方面的增强指令同步。我们的意思是，所有对象/感兴趣区域的增强应该在彼此之间是连贯和传统的。例如，最好是有一个prompts，例如 "The table is of old dark wood with faded polish and food stains and the background consists of a suburban home" ，而不是像 "The table is of old dark wood with faded polish and food stains and the background consists of a spaceship hurtling through space" 。
4. 在输入控制视频的关键方面，必须包括应保留或保持不变的细节。在我们的prompts中，我们非常明确地提到立方体的颜色应保持不变，使得底部立方体是蓝色，中间是红色，顶部是绿色。请注意，我们不仅提到了应该保持不变的内容，还详细说明了该方面当前的形式。

3.3 Cosmos 视觉增强 3.3.1 在本地部署cosmos-transfer1服务进行数据处理****

- 克隆 cosmos-transfer1 源代码（如果网络状况不良可以使用镜像站）

```
 cd /root
git clone git@github.com:nvidia-cosmos/cosmos-transfer1.git
cd cosmos-transfer1
git submodule update --init --recursive
```

- 以下命令将创建 cosmos-transfer1 的 conda 环境（Cosmos 要求 Python 版本为 3.12.x），并安装用于推理的依赖项：

```
##### 创建 cosmos-transfer1 conda 环境
conda env create --file cosmos-transfer1.yaml

##### 激活 cosmos-transfer1 conda 环境
conda activate cosmos-transfer1

##### 安装依赖
pip install -r requirements.txt

##### 安装 vllm
pip install https://download.pytorch.org/whl/cu128/flashinfer/flashinfer_python-0.2.5%2Bcu128torch2.7-cp38-abi3-linux_x86_64.whl
export VLLM_ATTENTION_BACKEND=FLASHINFER
pip install vllm==0.9.0

##### 安装 decord
pip install decord==0.6.0

##### 修复 conda 环境中 Transformer Engine 的链接问题
ln -sf $CONDA_PREFIX/lib/python3.12/site-packages/nvidia/*/include/* $CONDA_PREFIX/include/
ln -sf $CONDA_PREFIX/lib/python3.12/site-packages/nvidia/*/include/* $CONDA_PREFIX/include/python3.12

##### 安装 Transformer Engine
pip install transformer-engine[pytorch]
```

- 运行以下命令来测试环境配置是否正确：

```
python scripts/test_environment.py
```

- 下载模型权重文件（注意，该文件约300G，下载需要较长时间与充足磁盘空间）
- 生成一个 Hugging Face 访问令牌，并将访问权限设置为 **"Read"**（默认是 **"Fine-grained"**）。
- 设置 Hugging Face 镜像

```
export HF_ENDPOINT=https://hf-mirror.com
```

- 使用访问令牌登录 Hugging Face

```
huggingface-cli login
```

- 进入以下每个 Hugging Face 模型的使用条款页面，滚动到最底部并选择同意

```
- https://huggingface.co/nvidia/Cosmos-Transfer1-7B
- https://huggingface.co/nvidia/Cosmos-Tokenize1-CV8x8x8-720p
- https://huggingface.co/nvidia/Cosmos-Guardrail1
- https://huggingface.co/meta-llama/Llama-Guard-3-8B
```

- 从 Hugging Face 下载 Cosmos 模型权重

```
python scripts/download_checkpoints.py --output_dir checkpoints/
```

下载后的文件结构应如下所示：

```
checkpoints/
├── nvidia
│   │
│   ├── Cosmos-Guardrail1
│   │   ├── README.md
│   │   ├── blocklist/...
│   │   ├── face_blur_filter/...
│   │   └── video_content_safety_filter/...
│   │
│   ├── Cosmos-Transfer1-7B
│   │   ├── base_model.pt
│   │   ├── vis_control.pt
│   │   ├── edge_control.pt
│   │   ├── seg_control.pt
│   │   ├── depth_control.pt
│   │   ├── 4kupscaler_control.pt
│   │   └── config.json
│   │
│   ├── Cosmos-Transfer1-7B-Sample-AV/
│   │   ├── base_model.pt
│   │   ├── hdmap_control.pt
│   │   └── lidar_control.pt
│   │
│   ├── Cosmos-Tokenize1-CV8x8x8-720p
│   │   ├── decoder.jit
│   │   ├── encoder.jit
│   │   ├── autoencoder.jit
│   │   └── mean_std.pt
│   │
│   └── Cosmos-UpsamplePrompt1-12B-Transfer
│       ├── depth
│       │   ├── consolidated.safetensors
│       │   ├── params.json
│       │   └── tekken.json
│       ├── README.md
│       ├── segmentation
│       │   ├── consolidated.safetensors
│       │   ├── params.json
│       │   └── tekken.json
│       ├── seg_upsampler_example.png
│       └── viscontrol
│           ├── consolidated.safetensors
│           ├── params.json
│           └── tekken.json
│
├── depth-anything/...
├── facebook/...
├── google-t5/...
├── IDEA-Research/...
└── meta-llama/...
```

3.3.2 在开发机调用在线模型服务 下载调用脚本：

```
bcecmd bos cp bos:/aihc-rdw-bj/cosmos_data/run_cosmos_local.py /mnt/data/isaac_tmp/dataset/run_cosmos_local.py
```

运行脚本：

```
python /mnt/data/isaac_tmp/dataset/run_cosmos_local.py --mp4_output_dir /mnt/data/isaac_tmp/dataset/mimic_dataset_1k_mp4 \
  --cosmos_output_dir /mnt/data/isaac_tmp/dataset/cosmos_dataset_1k_mp4 \
  --prompts_file /mnt/data/isaac_tmp/dataset/cosmos_prompts.txt  \
  --checkpoint_dir /root/cosmos-transfer1/checkpoints
```

3.4 格式转换 hdf5<-mp4 通过以下代码将hdf5 格式文件转换为 mp4 格式：

```
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/tools/mp4_to_hdf5.py \
  --input_file /mnt/data/isaac_tmp/dataset/mimic_dataset_1k.hdf5 \
  --videos_dir /mnt/data/isaac_tmp/dataset/cosmos_dataset_1k_mp4 \
  --output_file /mnt/data/isaac_tmp/dataset/cosmos_dataset_1k.hdf5
```

3.5 直接下载增强后数据（可选） 用户可通过以下命令下载数据跳过该阶段

```
bcecmd bos cp bos:/aihc-rdw-bj/cosmos_data/cosmos_dataset_1k.hdf5 /mnt/data/isaac_tmp/dataset/cosmos_dataset_1k.hdf5
```

3.6 数据集合并 将增强前后数据通过以下代码进行合并

```
 /workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/tools/merge_hdf5_datasets.py --input_files /mnt/data/isaac_tmp/dataset/mimic_dataset_1k.hdf5
/mnt/data/isaac_tmp/dataset/cosmos_dataset_1k.hdf5 --output_file /mnt/data/isaac_tmp/dataset/mimic_cosmos_dataset.hdf5
```

四、模型训练

通过以下指令进行训练（dataset 可根据您实际预处理结果使用您所有可用的 hdf5 训练集）（模型默认保存路径为当前文件夹下 log 文件夹）：

```
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/imitation_learning/robomimic/train.py \
  --task Isaac-Stack-Cube-Franka-IK-Rel-Visuomotor-Cosmos-v0 --algo bc \
  --dataset /mnt/data/isaac_tmp/dataset/mimic_cosmos_dataset.hdf5 \
  --name bc_rnn_image_franka_stack_mimic_cosmos2
```

五、模型评估

您可以使用如下指令进行模型评估，其中 input_dir为模型目录，log_dir为输出目录

```
/workspace/isaaclab/isaaclab.sh -p /workspace/isaaclab/scripts/imitation_learning/robomimic/robust_eval.py \
  --task Isaac-Stack-Cube-Franka-IK-Rel-Visuomotor-Cosmos-v0 \
  --input_dir /root/workspace/logs/robomimic/Isaac-Stack-Cube-Franka-IK-Rel-Visuomotor-Cosmos-v0/bc_rnn_image_franka_stack_mimic_cosmos2/20260107022104/models/ \
  --log_dir /log/ \
  --log_file result \
  --enable_cameras \
  --livestream 1 \
  --seeds 0 \
  --num_rollouts 15 \
  --headless
```

Apollo-Model-Deployment

为加速 BEV感知模型从研发到验证的闭环，Apollo 自动驾驶平台提供了一套**开箱即用**的工具链，涵盖**模型训练**（Apollo-BEV-Train） → **模型导出**（Apollo-BEV-Model-Export） → **模型验证**（Apollo-Model-Deployment）三大核心阶段。用户可按此顺序依次使用三个标准化环境，高效完成 BEV 模型的开发与评估。

Apollo 自动驾驶开放平台介绍

Apollo (阿波罗)是一个开放的、完整的、安全的平台，将帮助汽车行业及自动驾驶领域的合作伙伴结合车辆和硬件系统，快速搭建一套 属于自己的自动驾驶系统。
开放能力、共享资源、加速创新、持续共赢是 Apollo 开放平台的口号。百度把自己所拥有的强大、成熟、安全的自动驾驶技术和数据开 放给业界，旨在建立一个以合作为中心的生态体系，发挥百度在人工智能领域的技术优势，为合作伙伴赋能，共同促进自动驾驶产业的发 展和创新。
Apollo 自动驾驶开放平台为开发者提供了丰富的车辆、硬件选择，强大的环境感知、高精定位、路径规划、车辆控制等自动驾驶软件能 力以及高精地图、仿真、数据流水线等自动驾驶云服务，帮助开发者从 0 到 1 快速搭建一套自动驾驶系统。

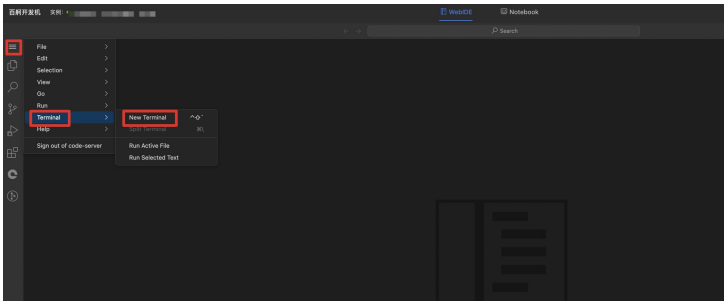
部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	按需	建议按表单默认值及以上
内存	按需	建议按表单默认值及以上
GPU	24G以上显存	建议24G以上显存
CDS	按需	按需

使用说明

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，并打开VScode中的terminal



- 代码保存路径：/apollo_workspace
- 数据默认挂载路径：/mnt/pfs/nuscenes_data/bev_data/nuscenes

Apollo环境中替换自己训练的ONNX文件

如果要替换为自己训练的onnx文件，则需要替换掉目录下的onnx文件

```
sudo rm -rf /apollo/modules/perception/data/models/apollo_bevnet_onnx/apollo_bevnet.onnx
sudo cp your-own-onnx.onnx /apollo/modules/perception/data/models/apollo_bevnet_onnx/apollo_bevnet.onnx
```

Apollo中运行纯视觉模型

准备record数据 通过下面链接可以下载nuscenes record：

id	link
6f83169d06734365825 1f72e1dd17dbc	http://apollo-perception.bj.bcebos.com/nuscenes_occ_records/6f83169d06734365825 1f72e1dd17dbc.record?authorization=bce-auth-v1%2FALTAKr8RyUUttStVHwGaOsvJyP%2F2024-12-02T08%3A12%3A56Z%2F-1%2Fhost%2F15500778f03ba45f19e6a7818b3af88952a739cbd53761f166d8bd542347821b
2fc3753772e241f2ab2 cd16a784cc680	http://apollo-perception.bj.bcebos.com/nuscenes_occ_records/2fc3753772e241f2ab2cd16a784cc680.record?authorization=bce-auth-v1%2FALTAKr8RyUUttStVHwGaOsvJyP%2F2024-12-02T08%3A11%3A30Z%2F-1%2Fhost%2Fb7b99869a804ed08c7f6804d8b724e15aebc3e1c13fde45b18b61e459b1b7ae5
bebf5f5b2a674631ab5 c88fd1aa9e87a	http://apollo-perception.bj.bcebos.com/nuscenes_occ_records/bebf5f5b2a674631ab5c88fd1aa9e87a.record?authorization=bce-auth-v1%2FALTAKr8RyUUttStVHwGaOsvJyP%2F2024-12-02T08%3A13%3A18Z%2F-1%2Fhost%2F1c25a91e913f512b8f188dc6e54bd07ffcbb301e1be2d92a3cc7b686e1456d80

选择车型环境使用,应用参数

```
aem profile use nuscenes_occ
```

打开dreamview_plus

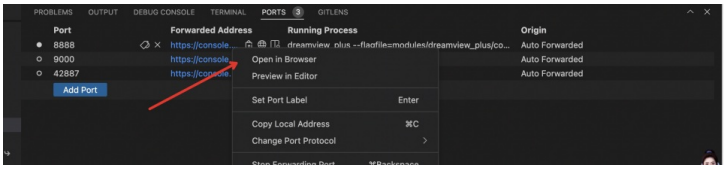
```
aem bootstrap start --plus
```

网页显示dreamview_plus方式：

访问方式（选择其一）：

方式1 - 百舸平台打开：
1. 执行脚本后,点击ports,选择8888端口的Fowarded Address的链接使用鼠标右键选择"Open in Browser"

方式2 - SSH端口转发
1. 在本地电脑执行: ssh -L 8888:localhost:8888 -p ssh映射端口 user@server_ip
2. 浏览器打开: http://localhost:8888

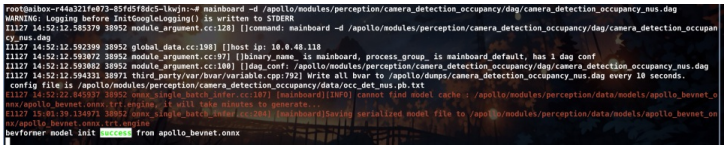


终端启动transform dag文件

```
mainboard -d /apollo/modules/transform/dag/static_transform.dag
```

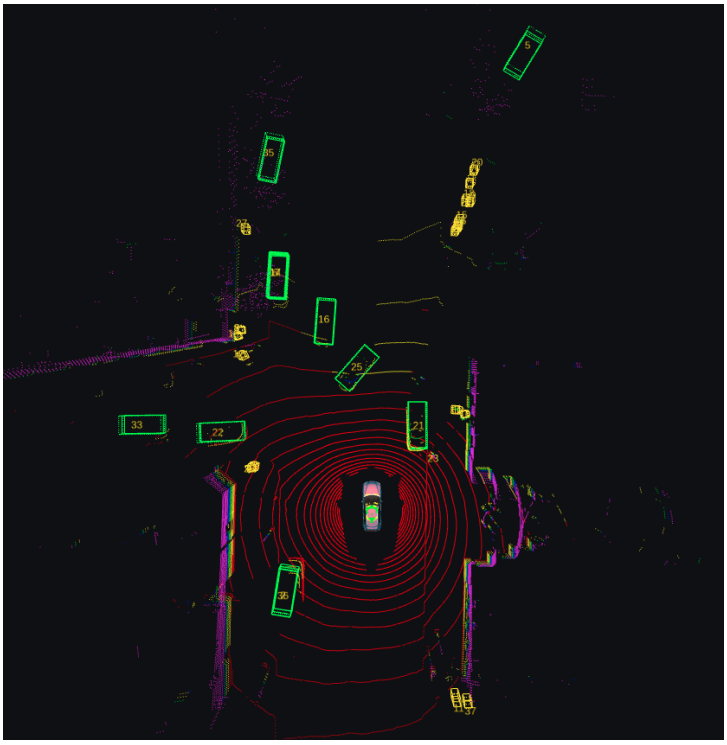
终端启动感知 dag文件

```
mainboard -d /apollo/modules/perception/camera_detection_occupancy/dag/camera_detection_occupancy_nus.dag
```

等待模型进行序列化，当终端给出如下日志时表示序列化完成，可进行下面的步骤  打开终端播放record包


```
cyber_recorder play -f 包名.record
```

随后就可以在dreamview_plus页面上看到目标检测结果



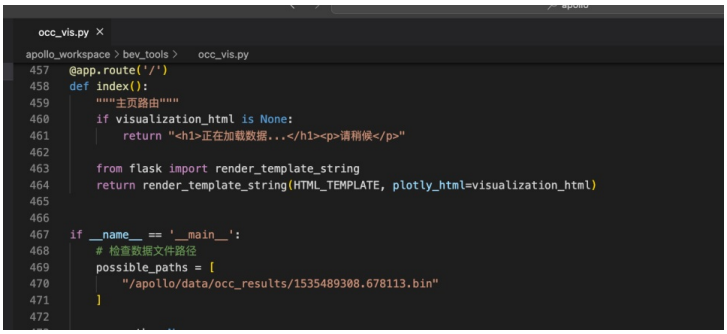
可视化occ结果 如果要查看occ结果，需要进行如下的步骤：
将配置occ_det_nus.pb.txt中save_occ_result设置为true，同时设置保存路径occ_save_path（默认路径为/apollo/data/occ_results）

Web端OCC可视化（支持远程访问）工具使用：

```
cd /apollo_workspace/bev_tools
python3 occ_vis.py
```

备注: 查看其他occ文件效果 修改occ_vis.py代码

```
#代码行数，470行
    # 检查数据文件路径
  possible_paths = [
      "/apollo/data/occ_results/1535489308.678113.bin"
  ]
```



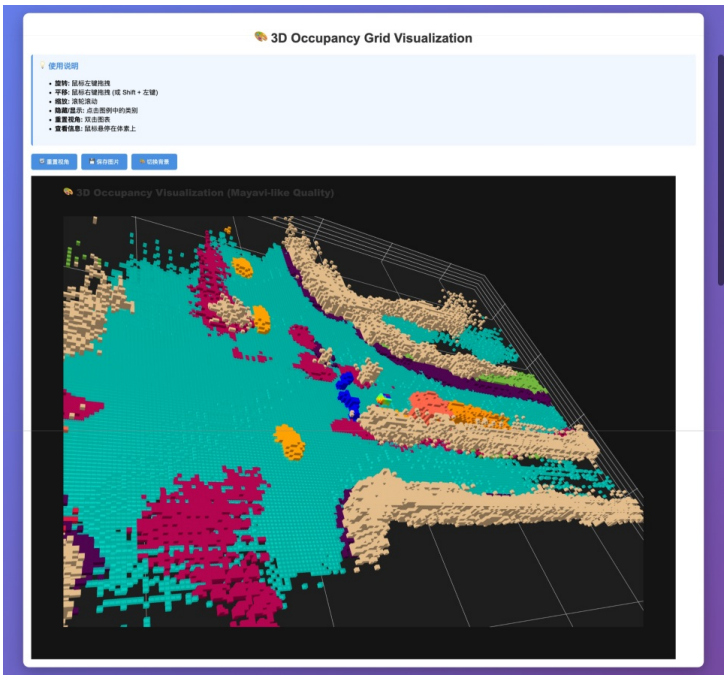
打开网页方式：

访问方式（选择其一）：

方式1 - 百舸平台打开：
1. 执行脚本后,点击ports,选择9000端口的Fowarded Address的链接使用鼠标右键选择“Open in Browser”

方式2 - SSH端口转发
1. 在本地电脑执行: ssh -L 9000:localhost:9000 -p ssh映射端口 user@server_ip
2. 浏览器打开: http://localhost:9000

Web端OCC可视化功能，支持远程实时访问与多维度场景分析。用户可通过浏览器访问服务端口（如localhost:9000），实现跨平台、跨网络的OCC渲染，无需依赖本地显示环境。该功能完整渲染**16类场景元素**（含车辆、行人、植被等），并为每类元素提供专属视觉标识。支持交互式视角操作（360°旋转/平移/缩放）、视角重置及截图保存（PNG格式），显著提升调试效率，实现“启动服务→实时验证模型效果”的闭环流程。



完整颜色映射表

序号	类别名称	颜色	用途说明
1	**Car** (轿车)	Orange	普通乘用车、小轿车
2	**Truck** (卡车)	Tomato	货车、厢式货车
3	**Trailer** (拖车)	Dark Orange	半挂车、拖挂车
4	**Bus** (公交车)	Orange Red	公交车、大巴
5	**Construction Vehicle** (工程车)	Dark Salmon	挖掘机、推土机等工程车辆
6	**Bicycle** (自行车)	Crimson	自行车
7	**Motorcycle** (摩托车)	Red	摩托车、电动车
8	**Pedestrian** (行人)	Blue	行人
9	**Traffic Cone** (交通锥)	Dark Slate Grey	交通锥、警示锥
10	**Barrier** (障碍物)	Slate Grey	护栏、路障
11	**Driveable Surface** (可行驶表面)	nuTonomy Green	道路、可行驶区域
12	**Other Flat** (其他平面)	Dark Magenta	其他平坦区域
13	**Sidewalk** (人行道)	Dark Purple	人行道、步道
14	**Terrain** (地形)	Yellow Green	草地、土地等自然地形
15	**Manmade** (人造物)	Burlywood	建筑物、墙体等人造结构
16	**Vegetation** (植被)	Green	树木、灌木等植被
-	**Ego Car** (自车)	Rainbow	自动驾驶车辆本身

附：Apollo 11.0 BEV+OCC 模型训练使用

流程	使用方式	备注
Apollo-BEV-Train	百舸平台	训练 Apollo BEV+OCC模型
Apollo-BEV-Model-Export	百舸平台	将训练好的BEV+OCC模型导出成onnx文件
Apollo-Model-Deployment	百舸平台或本地验证	apollo环境镜像，验证bev模型效果以及仿真等apollo工具

Apollo-BEV-Model-Export

为加速 BEV感知模型从研发到验证的闭环，Apollo 自动驾驶平台提供了一套**开箱即用**的工具链，涵盖**模型训练**（Apollo-BEV-Train） → **模型导出**（Apollo-BEV-Model-Export） → **模型验证**（Apollo-Model-Deployment）三大核心阶段。用户可按此顺序依次使用三个标准化环境，高效完成 BEV 模型的开发与评估。

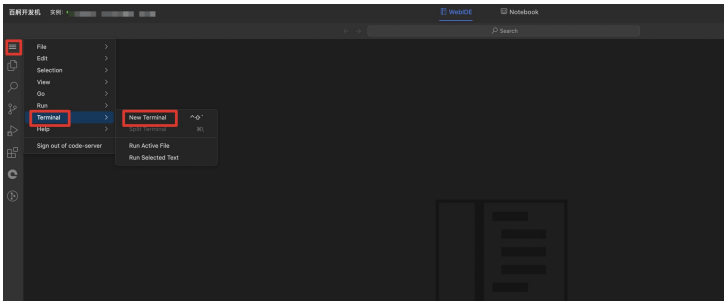
部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	按需	建议按表单默认值及以上
内存	按需	建议按表单默认值及以上
GPU	24G以上显存	建议24G以上显存
CDS	按需	按需

使用说明

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，并打开VScode中的terminal



- 代码保存路径：/root/Apollo-Vision-Net-Deployment
- 数据默认挂载路径：/mnt/pfs/nuscenes_data/bev_data/nuscenes

数据集挂载

开发机内数据集挂载(建议与Apollo-BEV-Train开发机pfs磁盘共用，不共用需要重新处理数据集)：

```
cd ~/Apollo-Vision-Net-Deployment/data

##### 创建符号链接（注意使用正确的 -s 参数）
ln -s /mnt/pfs/nuscenes_data/bev_data/nuscenes nuscenes
ln -s /mnt/pfs/nuscenes_data/bev_data/occ_gt_release_v1_0 occ_gt_release_v1_0
ln -s /mnt/pfs/nuscenes_data/bev_data/can_bus can_bus
```

模型导出

生成onnx文件 运行下面的命令，提供pth文件，生成onnx文件

```
cd ~/Apollo-Vision-Net-Deployment/
python tools/pth2onnx.py configs/apollo_bev/bev_tiny_det_occ_apollo_trt.py bos路径/epoch_*.pth --opset_version 13 --cuda
##### 使用4090卡导出onnx模型需要去掉--cuda参数
```

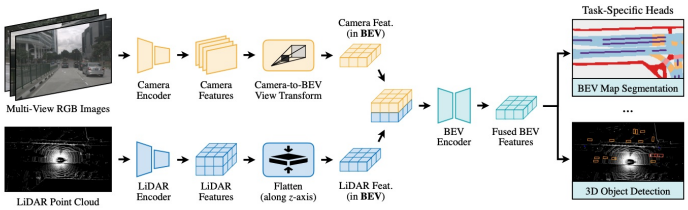
附：Apollo 11.0 BEV+OCC 模型训练使用

流程	使用方式	备注
Apollo-BEV-Train	百舸平台	训练 Apollo BEV+OCC模型
Apollo-BEV-Model-Export	百舸平台	将训练好的BEV+OCC模型导出成onnx文件
Apollo-Model-Deployment	百舸平台或本地验证	apollo环境镜像，验证bev模型效果以及仿真等apollo工具

BEVFormer-加速版

模型介绍

BEVFusion是一个高效且通用的多任务多传感器融合模型，用于自动驾驶的3D感知。通过统一的鸟瞰图表示，它保留了几何信息和语义信息，解决了点级融合方法的局限性。 **详细图文介绍**：



多传感器融合对于构建准确可靠的自动驾驶系统至关重要。近期的大部分方法是基于point-level的融合：使用图像特征增强LiDAR点云表示。然而，图像到LiDAR的映射舍弃了图像特征的语义密度，限制了这种方法的有效性，尤其是对于语义导向的任务（如3D场景分割）。BEVFusion提出一种高效且通用的多任务多传感器融合框架。它将多模态特征统一映射到鸟瞰视图（BEV）表示空间中，很好地保留了几何和语义信息。为实现这一目标，BEVFusion解决了视图转换中的关键效率瓶颈，通过优化的BEV pooling，将延迟降低了40倍以上。BEVFusion基本上是任务无感知的，几乎不需要进行架构改变就能无缝支持不同的3D感知任务。

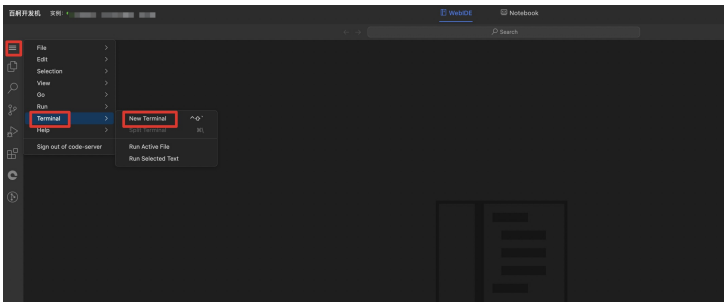
部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	按需	建议按表单默认值及以上
内存	按需	建议按表单默认值及以上
GPU	1~8卡 H20/RTX 5090	建议8卡 H20/RTX 5090
CDS	按需	按需
其它	如果更改了默认的数据集、存储挂载地址、代码地址，需要同时更改对应的脚本配置	若想获得最佳训练性能，建议将预置数据转储至PFS或CFS，转储方式见文末

使用说明

1. 创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，并打开VScode中的terminal



- 代码保存路径：/root/workspace/mmdetection3d
- 数据默认挂载路径：/mnt/dataset/autodrive
- 虚拟环境名：bevfusion

2. 启动训练

- 在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置
- 若想获得最佳训练性能，建议将预置数据转储至PFS或CFS，转储方式见文末

2.1 一键启动训练 可以通过脚本，一键启动（单机or多机）快速训练。

```
 ##### 设置数据路径，nuscenes数据默认路径/mnt/dataset/autodrive
export DATA_PATH=/mnt/dataset/autodrive
##### 设置训练配置文件，默认使用lidar+camera配置
export CONFIG=projects/BEVFusion/configs/bevfusion_lidar-cam_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d.py
##### 设置训练使用gpu卡数，默认使用全部gpu
export GPUS=$(nvidia-smi --query-gpu=gpu_name --format=csv,noheader | wc -l)
##### 设置训练结果保存路径，默认/root/workspace/mmdetection3d/work_dir
export WORK_DIR=/root/workspace/mmdetection3d/work_dir

source /root/env_bevfusion.sh

export PYTHONPATH=/root/workspace/mmdetection3d:$PYTHONPATH
cd /root/workspace/mmdetection3d && \
rm -rf ./data && ln -sf $DATA_PATH ./data

NNODES=${WORLD_SIZE:-1}
NODE_RANK=${RANK:-0}
PORT=${MASTER_PORT:-29600}
MASTER_ADDR=${MASTER_ADDR:-"127.0.0.1"}
##### 默认使用预训练模型进行训练
python3 -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --master_addr=$MASTER_ADDR \
    --nproc_per_node=$GPUS \
    --master_port=$PORT \
    tools/train.py \
    $CONFIG \
    --cfg-options load_from=pretrained/bevfusion_lidar_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d-2628f933.pth \
    --cfg-options model.img_backbone.init_cfg.checkpoint=pretrained/swint-nuimages-pretrained.pth \
    --launcher pytorch ${@:3} 2>&1 | tee output.log
```

2.2. 分步训练 分步进行训练任务，可以自定义训练相关参数信息。

2.2.1. 在环境变量中设置相关参数 若不修改默认配置，可跳过此步骤

```
 ##### 设置数据路径，nuscenes数据默认路径/mnt/dataset/autodrive
export DATA_PATH=/mnt/dataset/autodrive
##### 设置训练配置文件，默认使用lidar+camera配置
export CONFIG=projects/BEVFusion/configs/bevfusion_lidar-cam_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d.py
##### 设置训练使用gpu卡数，默认使用全部gpu
export GPUS=$(nvidia-smi --query-gpu=gpu_name --format=csv,noheader | wc -l)
##### 设置训练结果保存路径，默认/root/workspace/mmdetection3d/work_dir
export WORK_DIR=/root/workspace/mmdetection3d/work_dir
```

2.2.2. 激活模型运行虚拟环境

```
source /root/env_bevfusion.sh
```

2.2.3. 启动模型训练

```
 export PYTHONPATH=/root/workspace/mmdetection3d:$PYTHONPATH
cd /root/workspace/mmdetection3d && \
rm -rf ./data && ln -sf $DATA_PATH ./data

NNODES=${WORLD_SIZE:-1}
NODE_RANK=${RANK:-0}
PORT=${MASTER_PORT:-29600}
MASTER_ADDR=${MASTER_ADDR:-"127.0.0.1"}
##### 默认使用预训练模型进行训练
python3 -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --master_addr=$MASTER_ADDR \
    --nproc_per_node=$GPUS \
    --master_port=$PORT \
    tools/train.py \
    $CONFIG \
    --cfg-options load_from=pretrained/bevfusion_lidar_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d-2628f933.pth \
    --cfg-options model.img_backbone.init_cfg.checkpoint=pretrained/swint-nuimages-pretrained.pth \
    --launcher pytorch ${@:3} 2>&1 | tee output.log
```

3. 查看训练结果

训练结果保存在$WORK_DIR下，包含训练日志以及模型权重文件，使用其他保存路径可以修改环境变量$WORK_DIR。

```
 ##### 设置训练结果保存路径，默认/root/workspace/mmdetection3d/work_dir
export WORK_DIR=/root/workspace/mmdetection3d/work_dir
```

可在路径下查看：

- 完整训练日志： output.log
- 最终模型权重：latest.pth

附： BOS数据转储PFS方式

1. 挂载PFS或CFS

可以在创建或者更新开发机配置时，选择存储挂载，选择集群对应的PFS或者CFS，可自定义挂载路径



2. 将预置数据（BOS）拷贝到对应的PFS或CFS路径

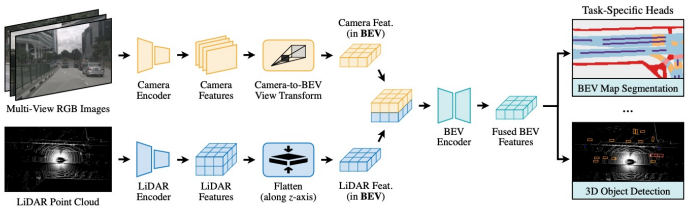
```
 ##### 命令格式  cp -rv  源路径 目标路径
cp -rv  bos路径  PFS或CFS路径
##### 命令示例  假设bos挂载地址为/mnt/dataset/autodrive   PFS挂载地址为/mnt/pfs/urvmsq
cp -rv  /mnt/dataset/autodrive/   /mnt/pfs/urvmsq/
```

3. 转储后，需要更改配置里的数据集变量DATA_PATH地址为/mnt/pfs/urvmsq/autodrive/文件夹

BEVFusion加速版

模型介绍

BEVFusion是一个高效且通用的多任务多传感器融合模型，用于自动驾驶的3D感知。通过统一的鸟瞰图表示，它保留了几何信息和语义信息，解决了点级融合方法的局限性。 **详细图文介绍**：



多传感器融合对于构建准确可靠的自动驾驶系统至关重要。近期的大部分方法是基于point-level的融合：使用图像特征增强LiDAR点云表示。然而，图像到LiDAR的映射舍弃了图像特征的语义密度，限制了这种方法的有效性，尤其是对于语义导向的任务（如3D场景分割）。BEVFusion提出一种高效且通用的多任务多传感器融合框架。它将多模态特征统一映射到鸟瞰视图（BEV）表示空间中，很好地保留了几何和语义信息。为实现这一目标，BEVFusion解决了视图转换中的关键效率瓶颈，通过优化的BEV pooling，将延迟降低了40倍以上。BEVFusion基本上是任务无感知的，几乎不需要进行架构改变就能无缝支持不同的3D感知任务。

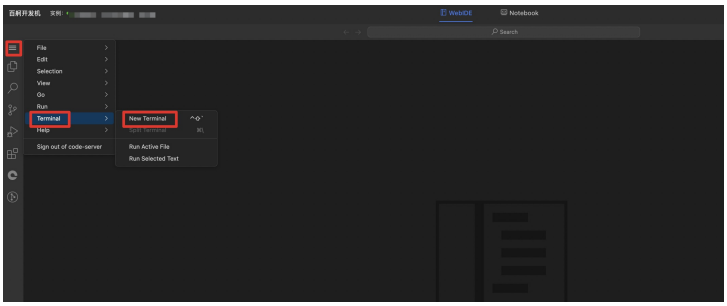
部署环境要求&最佳实践建议

	部署要求	最佳实践
CPU	按需	建议按表单默认值及以上
内存	按需	建议按表单默认值及以上
GPU	1~8卡 H20	建议4~8卡 H20
CDS	按需	按需
其它	如果更改了默认的数据集、存储挂载地址、代码地址，需要同时更改对应的脚本配置	若想获得最佳训练性能，建议将预置数据转储至PFS或CFS，转储方式见文末

使用说明

1. 创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机webIDE，并打开VScode中的terminal



- 代码保存路径：/root/workspace/mmdetection3d
- 数据默认挂载路径：/mnt/dataset/autodrive
- 虚拟环境名：bevfusion_env

2. 启动训练

- 在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置
- 若想获得最佳训练性能，建议将预置数据转储至PFS或CFS，转储方式见文末

2.1 一键启动训练 可以通过脚本，一键启动（单机or多机）快速训练。

```
 ##### 设置数据路径，nuscenes数据默认路径/mnt/dataset/autodrive
export DATA_PATH=/mnt/dataset/autodrive
##### 设置训练配置文件，默认使用lidar+camera配置
export CONFIG=projects/BEVFusion/configs/bevfusion_lidar-cam_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d.py
##### 设置训练使用gpu卡数，默认使用全部gpu
export GPUS=$(nvidia-smi --query-gpu=gpu_name --format=csv,noheader | wc -l)
##### 设置训练结果保存路径，默认/root/SparseDrive/work_dir
export WORK_DIR=/root/workspace/mmdetection3d/work_dir

export TORCH_CUDA_ARCH_LIST=9.0
export LD_LIBRARY_PATH=/root/anaconda3/envs/bevfusion_env/lib/python3.8/site-packages/torch/lib:$LD_LIBRARY_PATH
export PATH=/usr/local/cuda-12/bin/:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-12/lib64/:$LD_LIBRARY_PATH
export CUDA_HOME=/usr/local/cuda-12
source /root/anaconda3/etc/profile.d/conda.sh
conda activate bevfusion_env
export NCCL_IB_GID_INDEX

export PYTHONPATH=/root/workspace/mmdetection3d:$PYTHONPATH
cd /root/workspace/mmdetection3d && \
rm -rf ./data && ln -sf $DATA_PATH ./data

NNODES=${WORLD_SIZE:-1}
NODE_RANK=${RANK:-0}
PORT=${MASTER_PORT:-29600}
MASTER_ADDR=${MASTER_ADDR:-"127.0.0.1"}
##### 默认使用预训练模型进行训练
python3 -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --master_addr=$MASTER_ADDR \
    --nproc_per_node=$GPUS \
    --master_port=$PORT \
    tools/train.py \
    $CONFIG \
    --cfg-options load_from=pretrained/bevfusion_lidar_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d-2628f933.pth \
    --cfg-options model.img_backbone.init_cfg.checkpoint=pretrained/swint-nuimages-pretrained.pth \
    --launcher pytorch ${@:3} 2>&1 | tee output.log
```

2.2. 分步训练 分步进行训练任务，可以自定义训练相关参数信息。

2.2.1. 在环境变量中设置相关参数 若不修改默认配置，可跳过此步骤

```
 ##### 设置数据路径，nuscenes数据默认路径/mnt/dataset/autodrive
export DATA_PATH=/mnt/dataset/autodrive
##### 设置训练配置文件，默认使用lidar+camera配置
export CONFIG=projects/BEVFusion/configs/bevfusion_lidar-cam_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d.py
##### 设置训练使用gpu卡数，默认使用全部gpu
export GPUS=$(nvidia-smi --query-gpu=gpu_name --format=csv,noheader | wc -l)
##### 设置训练结果保存路径，默认/root/workspace/mmdetection3d/work_dir
export WORK_DIR=/root/workspace/mmdetection3d/work_dir
```

2.2.2. 激活模型运行虚拟环境

```
 export TORCH_CUDA_ARCH_LIST=9.0
export LD_LIBRARY_PATH=/root/anaconda3/envs/bevfusion_env/lib/python3.8/site-packages/torch/lib:$LD_LIBRARY_PATH
export PATH=/usr/local/cuda-12/bin/:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-12/lib64/:$LD_LIBRARY_PATH
export CUDA_HOME=/usr/local/cuda-12
source /root/anaconda3/etc/profile.d/conda.sh
conda activate bevfusion_env
export NCCL_IB_GID_INDEX=3
```

2.2.3. 启动模型训练

```
 export PYTHONPATH=/root/workspace/mmdetection3d:$PYTHONPATH
cd /root/workspace/mmdetection3d && \
rm -rf ./data && ln -sf $DATA_PATH ./data

NNODES=${WORLD_SIZE:-1}
NODE_RANK=${RANK:-0}
PORT=${MASTER_PORT:-29600}
MASTER_ADDR=${MASTER_ADDR:-"127.0.0.1"}
##### 默认使用预训练模型进行训练
python3 -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --master_addr=$MASTER_ADDR \
    --nproc_per_node=$GPUS \
    --master_port=$PORT \
    tools/train.py \
    $CONFIG \
    --cfg-options load_from=pretrained/bevfusion_lidar_voxel0075_second_secfpn_8xb4-cyclic-20e_nus-3d-2628f933.pth \
    --cfg-options model.img_backbone.init_cfg.checkpoint=pretrained/swint-nuimages-pretrained.pth \
    --launcher pytorch ${@:3} 2>&1 | tee output.log
```

3. 查看训练结果

训练结果保存在$WORK_DIR下，包含训练日志以及模型权重文件，使用其他保存路径可以修改环境变量$WORK_DIR。

```
 ##### 设置训练结果保存路径，默认/root/workspace/mmdetection3d/work_dir
export WORK_DIR=/root/workspace/mmdetection3d/work_dir
```

可在路径下查看：

- 完整训练日志： output.log
- 最终模型权重：latest.pth

附： BOS数据转储PFS方式

1. 挂载PFS或CFS

可以在创建或者更新开发机配置时，选择存储挂载，选择集群对应的PFS或者CFS，可自定义挂载路径



2. 将预置数据（BOS）拷贝到对应的PFS或CFS路径

```
 ##### 命令格式  cp -rv  源路径 目标路径
cp -rv  bos路径  PFS或CFS路径
##### 命令示例  假设bos挂载地址为/mnt/dataset/autodrive   PFS挂载地址为/mnt/pfs/urvmsq
cp -rv  /mnt/dataset/autodrive/   /mnt/pfs/urvmsq/
```

3. 转储后，需要更改配置里的数据集变量DATA_PATH地址为/mnt/pfs/urvmsq/autodrive/文件夹

Isaac工作流--使用 GR00T-Dreams 生成训练数据

GR00T-Dreams是NVIDIA推出的一套创新的机器人合成数据生成与神经仿真框架，它旨在通过生成式AI和世界模型，为机器人训练创造海量的、高质量的数据，从而解决机器人学习过程中面临的数据稀缺瓶颈。

使用说明 整个流程包含以下三个主要阶段：

- **世界模型后训练**: 使用训练数据对Cosmos-Predict2进行微调与推理
- **IDM 使用**: 使用 IDM 进行动作提取与数据转换
- **评分**: 通过开源 VLM 对模型进行评分

一、环境配置

	部署要求	最佳实践
CPU	32核	建议按表单默认值及以上
内存	384G	建议按表单默认值及以上
GPU	需要适合AI推理的多卡计算资源，如H20等，用于模型微调&IDM标注&评测等	本示例使用4卡H20开发机
存储	无	500G以上（需要下载模型及数据集）
其它	无	无

二、数据准备

2.1 原始数据下载 bos 工具下载：

```
 wget https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.9.zip   #下载
unzip linux-bcecmd-0.5.9.zip   #解压
apt install unzip
ln linux-bcecmd-0.5.9/bcecmd /usr/sbin/  #设置为全局使用
```

使用 BOS 下载 checkpoint（请保证存储设备有至少 500G 空间或者使用 PFS 等存储挂载）：

```
 bcecmd bos sync bos:/aihc-models-bj/Cosmos-Predict2/ /mnt/pfs/cosmos-predict2/
bcecmd bos sync bos:/aihc-rdw-bj/gr1-train/ /mnt/pfs/gr1-train/
```

三、Cosmos predict 微调与推理

3.1 环境验证

```
cd /workspace/cosmos-predict2
python ./scripts/test_environment.py
```

3.2 数据加载 将下载的 ckpt 放到指定文件夹：

```
rm -rf /workspace/cosmos-predict2/checkpoints
ln -s /mnt/pfs/cosmos-predict2/checkpoints /workspace/cosmos-predict2/checkpoints
```

将下载的数据集放置到指定文件夹：

```
rm -rf /workspace/cosmos-predict2/datasets/benchmark_train/gr1
mkdir -p /mnt/pfs/gr1/videos
ln -s /mnt/pfs/gr1 /workspace/cosmos-predict2/datasets/benchmark_train/gr1
cp /mnt/pfs/gr1-train/gr1/*mp4 /mnt/pfs/gr1/videos
cp /mnt/pfs/gr1-train/metadata.csv /mnt/pfs/gr1/
```

3.3 数据预处理

```
python -m scripts.get_t5_embeddings_from_groot_dataset --dataset_path datasets/benchmark_train/gr1/
```

处理后可查看 datasets/benchmark_train/gr1 目录，其中包含csv 及三个文件夹为处理正确

3.4 模型微调 通过以下指令进行模型微调：

```
torchrun --nproc_per_node=1 --master_port=12341 -m scripts.train --config=cosmos_predict2/configs/base/config.py -- experiment=predict2_video2world_training_2b_groot_gr1_480
```

3.5 模型推理 通过以下代码通过提示词生成视频（中英文均可）：

```
torchrun --nproc_per_node=4 --master_port=12341 -m examples.video2world_gr00t \
--model_size 14B --gr00t_variant gr1 \
--prompt "Use the right hand to pick up rubik's cube from from the bottom of the three-tiered wooden shelf to to the top of the three-tiered wooden shelf." \
--input_path assets/sample_gr00t_dreams_gr1/8_Use_the_right_hand_to_pick_up_rubik\'s_cube_from_from_the_bottom_of_the_three-tiered_wooden_shelf_to_to_the_top_of_the_three-tiered_wooden_shelf..png \
--num_gpus 4 --prompt_prefix "" --disable_guardrail \
--save_path output/generated_video_gr1.mp4

torchrun --nproc_per_node=4 --master_port=12341 -m examples.video2world_gr00t \
--model_size 14B --gr00t_variant gr1 \
--prompt "用右手将魔方从底层木架底层移至第三层" \
--input_path assets/sample_gr00t_dreams_gr1/8_Use_the_right_hand_to_pick_up_rubik\'s_cube_from_from_the_bottom_of_the_three-tiered_wooden_shelf_to_to_the_top_of_the_three-tiered_wooden_shelf..png \
--num_gpus 4 --prompt_prefix "" --disable_guardrail \
--save_path output/generated_video_gr1.mp4
```

3.5 使用拒绝采样提升生成视频评分 生成多个视频通过Cosmos-Reason 进行评分最后输出得分最高的视频：

```
torchrun --nproc_per_node=2 --master_port=12341 -m examples.video2world_bestofn \
--model_size 14B --gr00t_variant gr1 \
--prompt "Use the right hand to pick up rubik's cube from from the bottom of the three-tiered wooden shelf to to the top of the three-tiered wooden shelf." \
--input_path assets/sample_gr00t_dreams_gr1/8_Use_the_right_hand_to_pick_up_rubik\'s_cube_from_from_the_bottom_of_the_three-tiered_wooden_shelf_to_to_the_top_of_the_three-tiered_wooden_shelf..png \
--num_gpus 2 --num_generations 4 --prompt_prefix "" \
--disable_guardrail --save_path output/best-of-n-gr00t-gr1
```

四、IDM 动作提取与数据转换

下载 IDM 模型：

```
hf download seonghyeonye/IDM_gr1
cp -r /mnt/pfs/IDM_gr1/ /workspace/GR00T-Dreams/IDM_dump/data
cd /workspace/GR00T-Dreams/
```

相关脚本位于IDM_dump/scripts/preprocess目录下。每个脚本针对特定的**embodiment**进行设计。我们使用**gr1**（Fourier GR1 人形机器人）具身进行演示：

提取视频对应指令:

```
python IDM_dump/split_video_instruction.py \
  --source_dir "/workspace/cosmos-predict2/output/best-of-n-gr00t-gr1" \
  --output_dir "IDM_dump/data/gr1_data"
```

视频帧统一化处理:

```
python IDM_dump/preprocess_video.py \
  --src_dir "IDM_dump/data/gr1_data" \
  --dst_dir "IDM_dump/data/gr1_data_split" \
  --dataset gr1
```

转换为 LeRobot 格式:

```
python IDM_dump/raw_to_lerobot.py \
  --input_dir "IDM_dump/data/gr1_data_split" \
  --output_dir "IDM_dump/data/gr1_unified.data" \
  --cosmos_predict2
```

提取机器人动作序列：

```
 ##### 如果直接调用hf不通的话可以使用如下代码尝试镜像
##### export HF_ENDPOINT=https://hf-mirror.com

PYTHONPATH=. CUDA_VISIBLE_DEVICES=0,1 python IDM_dump/dump_idm_actions.py \
  --checkpoint "seonghyeonye/IDM_gr1" \
  --dataset "IDM_dump/data/gr1_unified.data" \
  --output_dir "IDM_dump/data/gr1_unified.data_idm" \
  --num_gpus 1 \
  --video_indices "0 8"
```

提取结果验证：

```
 uv pip install parquet-tools
parquet-tools csv IDM_dump/data/gr1_unified.data_idm/data/chunk-000/episode_000000.parquet
```

五、DreamGen Benchmark 评估

配置路径：

```
 video_dir="/workspace/cosmos-predict2/output/dream_gen_benchmark/cosmos_predict2_14b_gr1_object"
csv_path="/workspace/cosmos-predict2/output/dream_gen_benchmark/result.csv"
device="cuda:0"
```

以下调用方式为 HF 模型库直接下载调用，您也可以直接将模型下载至/root/.cache/huggingface/hub路径下直接运行

评分：

```
python -m dreamgenbench.eval_sr_qwen_whole \
  --video_dir "$video_dir" \
  --output_csv "$csv_path" \
  --device "$device"

##### 您可以通过添加--zeroshot true来声明未微调模型
```

如果您拥有 GPT API key，也可以使用 GPT 4o 进行评分：

```
python -m dreamgenbench.eval_sr_gpt4o_whole \
  --video_dir "$video_dir" \
  --output_csv "$csv_path" \
  --api_key "$api_key"
```

一致性测评 PA Score：

```
python -m dreamgenbench.eval_qwen_pa \
  --video_dir "$video_dir" \
  --output_csv "$csv_path" \
  --device "$device"
```

快速部署 OpenClaw（Clawdbot）

OpenClaw 是一个可以运行在终端设备上的个人AI助手，能在日常聊天应用（如 WhatsApp、Telegram、Slack、Mattermost（插件）、Discord、Google Chat、Signal、iMessage、WebChat）中进行回复，并在支持的平台上实现语音交互和实时画布，具备有状态的会话、记忆和工具功能。

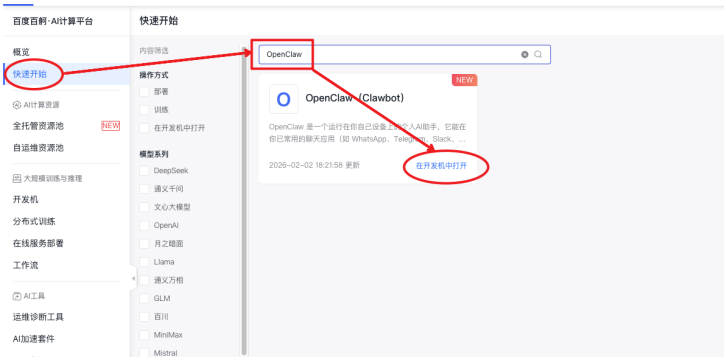
本文会介绍如何部署使用OpenClaw，在WebUI中OpenClaw进行对话。

使用需知：

1. OpenClaw需要至少搭配一个大模型服务，才能进行最基础的对话；
2. OpenClaw 既是一个产品，也是一项实验：接入真实的通讯平台和真实的工具。**不存在"绝对安全"的设置方案**。建议从能满足基本功能的最小权限开始，再逐步扩大其访问范围；
3. 部署OpenClaw的开发机实例和OpenClaw需要调用的大语言模型需要属于同一资源池。

如何连接聊天软件的部分，本教程暂不涉及，请参照 OpenClaw 官网： Chat Channels

一、部署OpenClaw



1. 在 **快速开始** 中查找 OpenClaw 点击 **在开发机中打开** ；
2. 填写创建开发机在开发机中打开 OpenClaw 所需的参数；
   - 不需要加速芯片，CPU和内存建议4核、16G
   - 使用百舸云盘/云磁盘
   - 开启启用SSH，填写公钥

- 选择支持公网访问的BLB（仅对话情况下带宽需求不高10Mbps即可，如需要图片传输，建议调大BLB带宽）

3. 填写完成创建部署 OpenClaw 的开发机，可在 **开发机** 中查看实例；

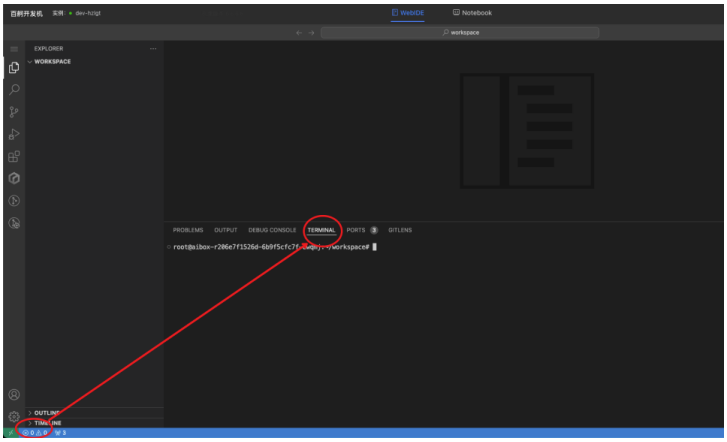
- /app：根目录下的app文件夹
- /home/node/.openclaw：存储openclaw配置文件
- /home/node/clawd：openclaw工作文件夹

二、初始化OpenClaw配置

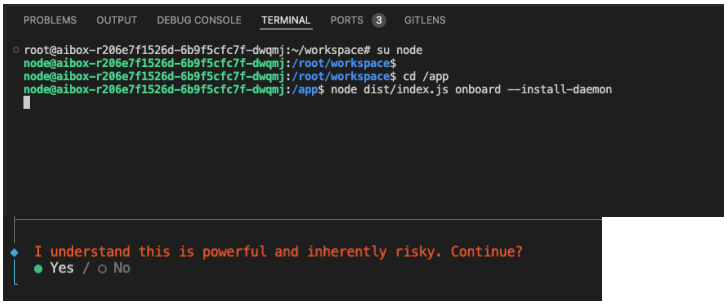
1. 开发机 “运行中” 后，可点击 **登录** 可进入开发机WebIDE；



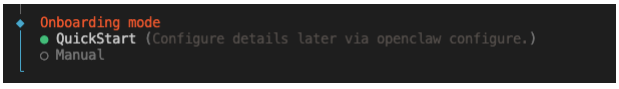
2. 在开发机的WebIDE中打开 Terminal界面内输入以下命令，对OpenClaw进行配置初始化；



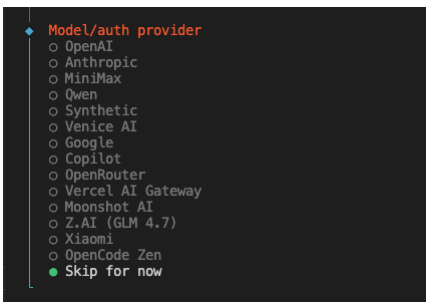
```
##### 从root用户切换到node，所有动作都要用node用户操作
su node
##### 进入OpenClaw根目录
cd /app
##### 开启设置界面
node dist/index.js onboard --install-daemon
```



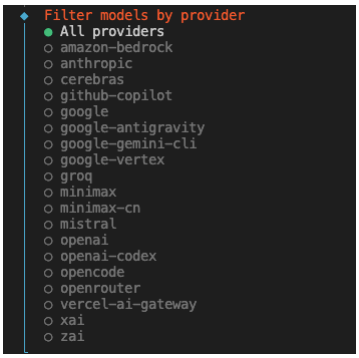
- 左右键切换为yes，回车下一步



- QuickStart，回车下一步



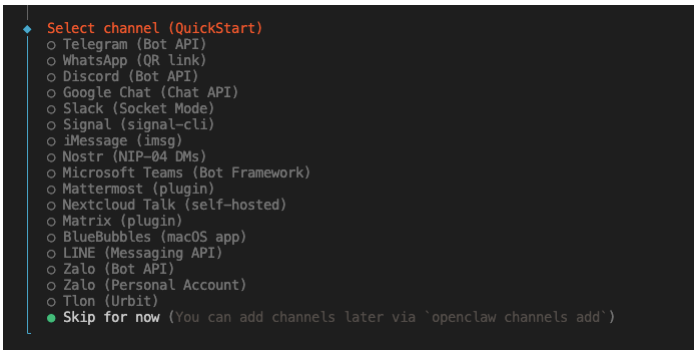
- 按一下上键，光标可直接选择Skip for now，回车下一步



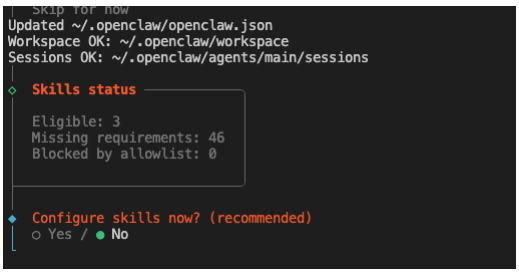
- 直接选择All providers回车



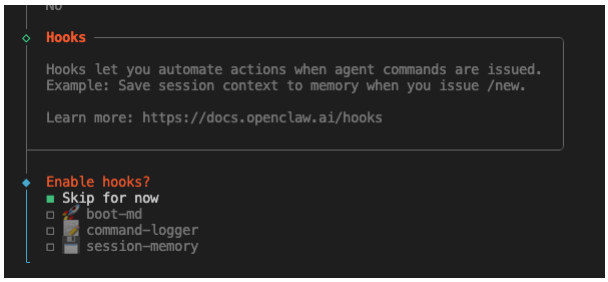
- 选择Keep current，回车下一步



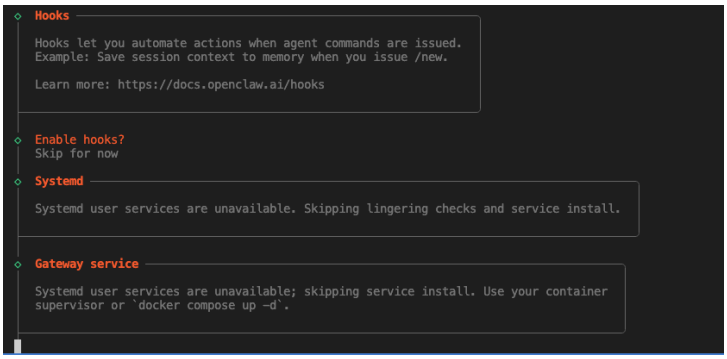
- 上键，光标跳转到Skip for now，回车下一步



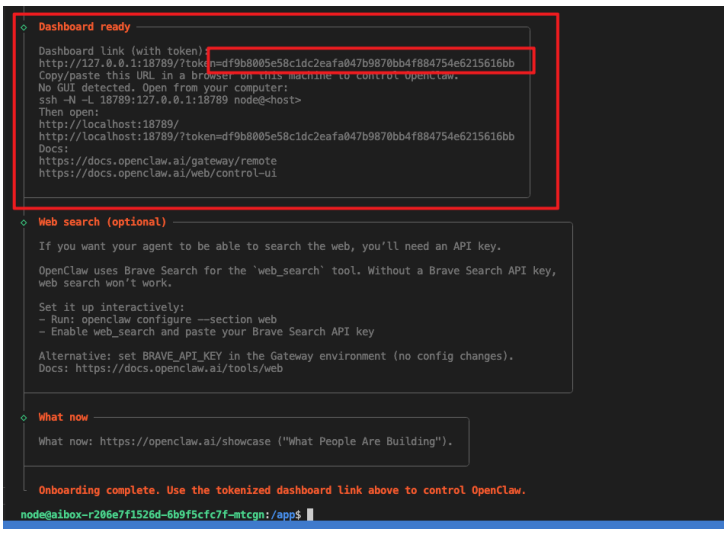
- 左右键选择No，回车选中



- 空格勾选上Skip for now，回车下一步



- 等待一段时间，出现配置流程



3. 启动Gateway

```
##### 从root用户切换到node，所有动作都要用node用户操作
su node # 已切换可忽略
##### 进入OpenClaw根目录
cd /app # 已进入可忽略
##### 前台开启Gateway，开启后，关掉这个bash窗口会导致Gateway停止
node dist/index.js gateway --port 18789 --verbose
```

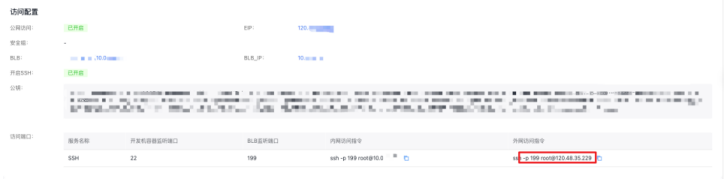


4. 验证OpenClaw是否已经正常启动

使用本地ssh将开发机的18789端口映射到本地，**本地**（Windows或Mac）命令行启动ssh作为转发通道（在使用vnc期间，命令行的ssh需要持续开启）：

```
##### 用本地环境的命令行执行这个ssh，-L是将开发机的5901监听连接到本地，后面的root、ip和-p部分根据实际开发机给出的公网访问命令执行
ssh -L 18789:127.0.0.1:18789 root@开发机公网IP地址 -p 开发机公网端口（可在开发机详情页复制）
```

开发机详情页公网IP和公网端口位置（在创建开发机时，选择开启ssh，并选择一个能连上公网的BLB，才能看到下面的部分）：



访问本地18789端口（带有token）



使用从这一步中提取出的url： http://127.0.0.1:18789/?token=df9b8005e58c1dc2eafa047b9870bb4f884754e6215616bb

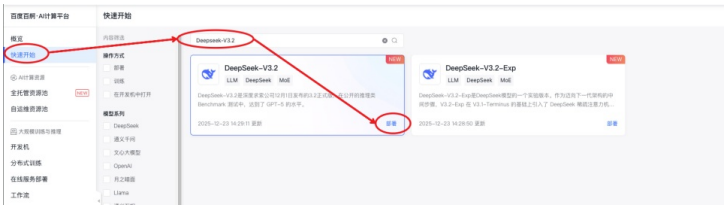


成功后，右上角会有【Health Ok】标识

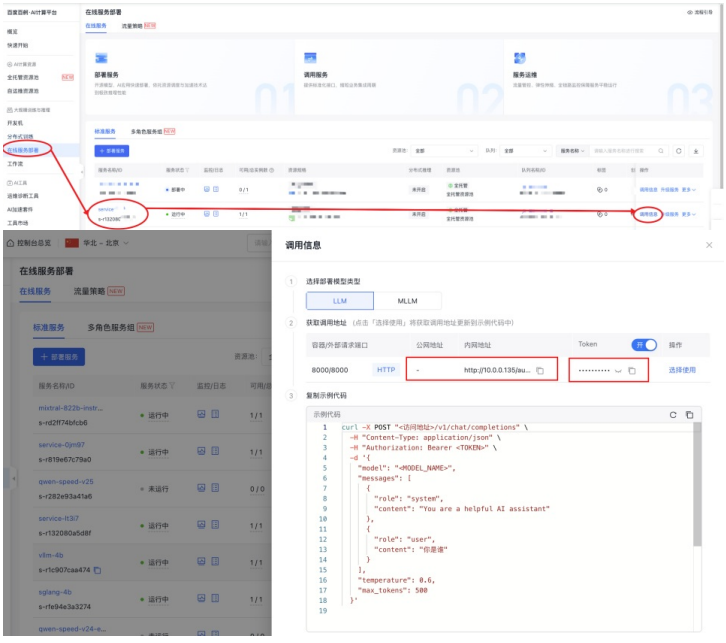
三、部署大语言模型

本教程使用会使用一个百舸创建的LLM模型服务，来提供OpenClaw所需要的模型支持

1. 在 **快速开始** 中搜索 Deepseek-V3.2模型，点击 **部署** 部署OpenClaw的开发机实例和OpenClaw需要调用的大语言模型需要属于同一资源池；



2. 在 **在线服务部署** 中查看已部署的模型，服务状态为"运行中" 即可调用，点击 **调用信息** 获取模型服务的调用地址、Token

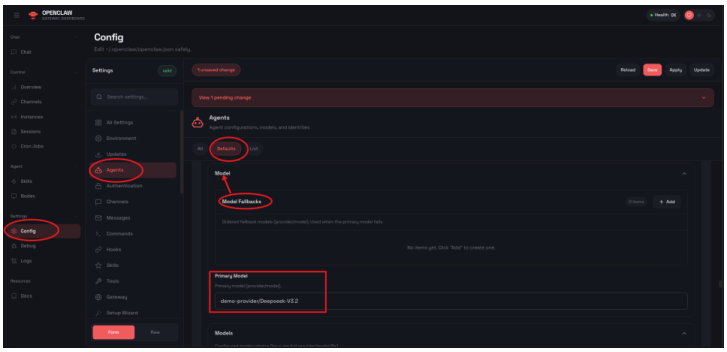


注意：访问地址需要自行拼接/v1，例：http://10.0.0.135/auth/s-rd2ff74bfcb6/8000/v1

3. 服务部署成功后，可以在开发机中请求服务，检查网络联通性。

四、配置 OpenClaw WebUI

1. **填写agent**

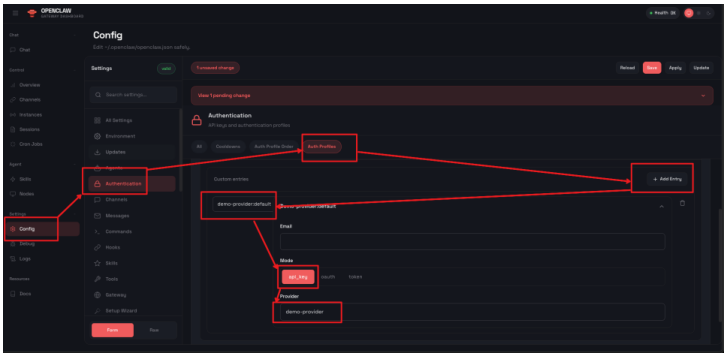


- 进入Config->Agents->Defaults；
- 通过网页搜索Model Fallbacks，找到Model区域（Model Fallbak还会搜到Image Model，跳过这个区域寻找Model区域）；
- Primary Model填写【provider名称 + model名称】，这两个名称都可以自定义，只要配置中填写的Agents、Authentication、Provider三个区域能够对应上即可，本教程中会使用【demo-provider】和【Deepseek-V3.2】



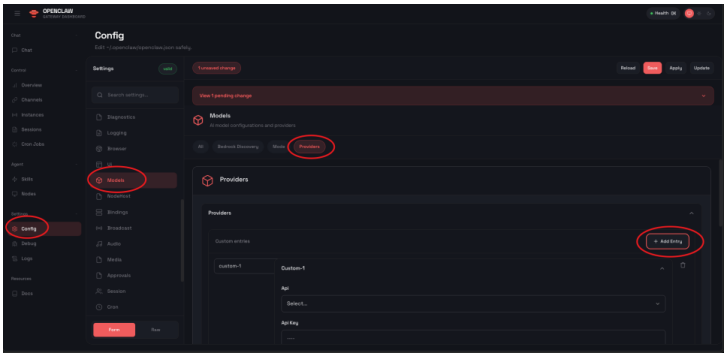
- 继续下拉到Models部分，需要点击右侧按钮将其展开
- 在Custom entries一栏点击【Add Entry】添加一条新的自定义Models
- 填入Primary model的值，【demo-provider/Deepseek-V3.2】
- 在alias中填入一个别名，我们这里使用【deepseek】这个alias
- 点击右上角的save按钮，保存修改

2. **填写Authentication**

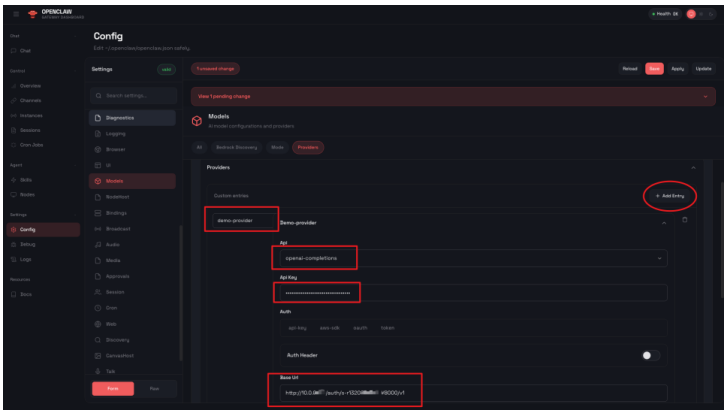


- 增加auth认证，进入config->Authentication->Auth Profiles界面，点击Add Entry按钮新增一项
- 填入标题为： demo-provider:default
- 选中 api_key 验证方式
- 填入Provider为： demo-provider
- 最后点击右上角的save

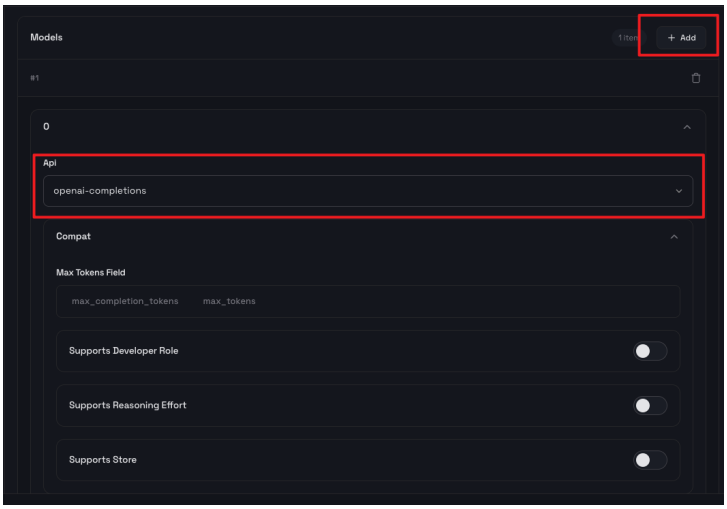
3. **填写Provider**



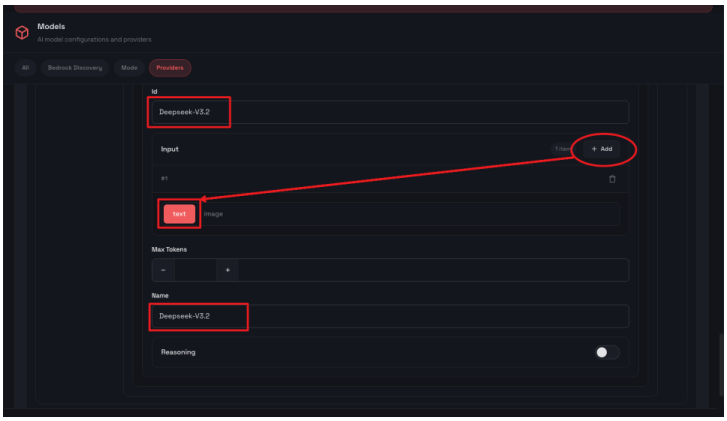
- 进入config->Models->Providers界面



- 点击Add Entry按钮新增一项
- 填入Provider名称为demo-provider
- 选择Api为【openai-completions】
- 将之前推理服务的Token填入这里的Api Key（不带Bearer）
- 将之前推理服务的url填入这里的Base Url（到v1为止）

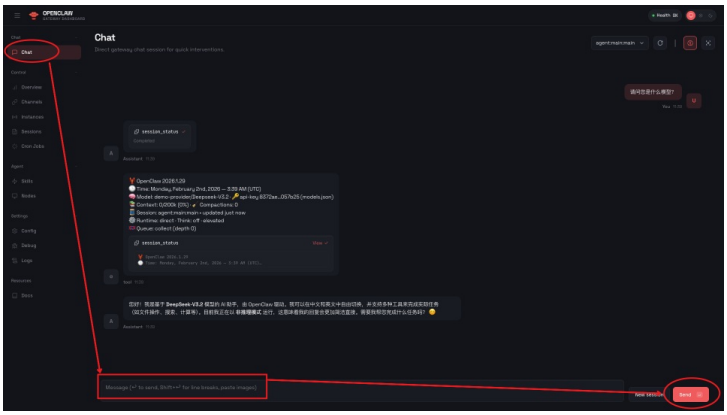


- 向下滚动到Models
- 点击Add按钮添加一项
- 选择Api为【openai-completions】



- 向下滚动到Models中填写id的部分，我们这里填入Deepseek-V3.2，也就是之前的Model名称
- 点击Add添加一个输入，选择text
- Name也填入Deepseek-V3.2
- 最后点击右上角的save

4. 配置完成，即可进行对话回到首页，进行大模型对话



RLinf：pi0.5模型的强化学习训练与评估

RLinf（Reinforcement Learning Infrastructure）是一个由清华大学、无问芯穹联合北京大学、伯克利等顶级机构联合开发的开源强化学习基础设施框架。它专为**大规模、高效率的强化学习（RL）后训练**而设计，旨在解决传统RL框架在训练具身智能体和大语言模型时面临的异构性、低效性和复杂性难题。

所述工作流基于 Rlinf 官方镜像环境针对 pi0.5 进行强化学习训练与评估。

硬件要求

以下是本快速开始使用的硬件配置：

组件	配置
GPU	8 块 A800
CPU	110 核
内存	900 GiB

运行环境

在官方镜像中，提供了三套环境对应不同的模型，即`openvla`、`openvla-oft` 和 `openpi`，可以通过以下指令进行切换：

```
 source switch_env <env_name>
##### source switch_env openvla
##### source switch_env openvla-oft
##### source switch_env openpi
```

项目下载

通过以下代码下载 Rlinf 项目：

```
 git clone https://github.com/RLinf/RLinf
##### 为提高国内下载速度，可以使用：
##### git clone https://ghfast.top/github.com/RLinf/RLinf.git
```

后续相关 example 都在该项目中。

模型下载

Rlinf 官方提供了不同版本的 Pi、Pi0.5 以及 VLA 模型，可以通过以下方式进行下载，此处以LIBERO空间物体目标任务 SFT 训练模型为例，如果需要下载其他模型可以参考附录选择需要的模型进行下载：

```
 pip install huggingface-hub
##### 此处可以调整自己的下载路径
##### 例如 pi05 模型
hf download RLinf/RLinf-Pi05-LIBERO-SFT --local-dir ./data_asserts/
##### pi0 模型
##### hf download RLinf/RLinf-Pi0-LIBERO-Spatial-Object-Goal-SFT --local-dir ./data_asserts/
```

运行参数配置

在项目下载中的 examples/embodiment/config 文件夹中可以找到相关运行参数文件，其中已经包含了针对部分任务的配置，对于 libero-10，对应 pi0 与 pi05 有如下示例配置文件：

- **π0+ PPO:** examples/embodiment/config/libero_10_ppo_openpi.yaml
- **π0+ GRPO:** examples/embodiment/config/libero_10_grpo_openpi.yaml
- **π0.5+ PPO:** examples/embodiment/config/libero_10_ppo_openpi_pi05.yaml
- **π0.5+ GRPO:** examples/embodiment/config/libero_10_grpo_openpi_pi05.yaml

以下以**libero_10_ppo_openpi_pi05**为例。

显卡运行参数配置 Rlinf 框架中有三个组件（env、rollout、actor），**Env提供规则和反馈，Rollout负责用当前策略去收集数据，Actor则利用这些数据来学习和改进策略本身**。三者形成一个闭环，驱动模型从零开始成长。

通过配置 cluster 参数可以灵活配置三个组件（env、rollout、actor）使用 GPU 数量，外加配置 rollout 中的 pipeline_stage_num:2来实现流水线重合提升效率，例如使用以下配置：

```
cluster:
  num_nodes: 1
  component_placement:
    env: 0-3
    rollout: 4-7
    actor: 0-7

rollout:
  pipeline_stage_num: 2
```

可以通过以下配置实现组件完全共享 GPU：

```
cluster:
 num_nodes: 1
 component_placement:
    env,rollout,actor: all
```

也可以通过以下配置实现 GPU 完全隔离，互不干扰：

```
cluster:
 num_nodes: 1
 component_placement:
    env: 0-1
    rollout: 2-5
    actor: 6-7
```

运行参数配置 2.1 模型参数

```
openpi:
 noise_level: 0.5
 action_chunk: ${actor.model.num_action_chunks}
 num_steps: ${actor.model.num_steps}
 train_expert_only: True
 action_env_dim: ${actor.model.action_dim}
 noise_method: "flow_sde"
 add_value_head: False
 pi05: False
 value_after_vlm: False
```

在 openpi 模块可以针对以上参数进行独立配置

2.2 LoRA设置 如果您想使用 LoRA 进行微调，请配置 is_lora: True并设置 lora_rank，当前不支持梯度检查点，请保证gradient_checkpointing: False：

```
model:
 is_lora: True
 lora_rank: 8
 gradient_checkpointing: False
```

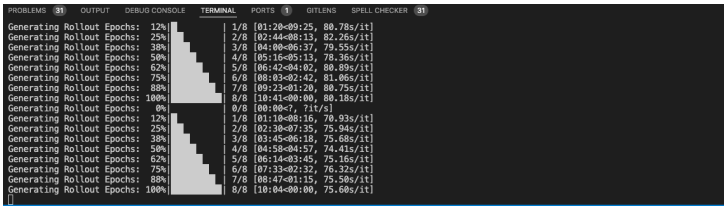
模型加载配置 通过修改以下配置来指向已下载的模型：

```
 ##### 使用模型下载中使用地址的绝对路径
model_dir: "./data_asserts/"
```

进行强化学习训练

通过以下命令进行训练：

```
 ##### 通过调整 CHOSEN_CONFIG 调用不同训练任务
##### bash examples/embodiment/run_embodiment.sh CHOSEN_CONFIG
bash examples/embodiment/run_embodiment.sh libero_10_ppo_openpi_pi05
```



运行如图

训练已默认配置 Tensorboard，可通过如下指令查看：

```
 ##### 可将 ./logs 改为日志地址
tensorboard --logdir ./logs --port 6006
```

训练输出文件默认位于./logs/任务时间-任务名文件夹中，输出文件如下所示，包含 checkpoint，tensorboard 以及训练输出 video：

```
.
├── libero_10_ppo_openpi_pi05
│   └── checkpoints
│       └── global_step_40
│           └── actor
│               ├── dcp_checkpoint
│               └── model_state_dict
├── tensorboard
└── video
    └── train
        ├── seed_0
        ├── seed_1
        ├── seed_2
        ├── seed_3
        ├── seed_4
        ├── seed_5
        ├── seed_6
        └── seed_7
```



模型评估

所有的评估辅助脚本位于 examples/embodiment/ 目录下，可以根据需要修改 checkpoint 路径、GPU ID 等配置即可，以下是使用 LIBERO 进行评测的脚本（eval_embodiment.sh）：

```
 ##### ! /bin/bash

export EMBODIED_PATH="$( cd "$(dirname "${BASH_SOURCE[0]}" )" && pwd )"
export REPO_PATH=$(dirname "$(dirname "$EMBODIED_PATH")")
export SRC_FILE="${EMBODIED_PATH}/eval_embodied_agent.py"

export MUJOCO_GL="osmesa"
export PYOPENGL_PLATFORM="osmesa"
export PYTHONPATH=${REPO_PATH}:$PYTHONPATH

##### LIBERO 仓库路径
export LIBERO_PATH="/opt/LIBERO"
export PYTHONPATH=${LIBERO_PATH}:$PYTHONPATH

export CUDA_LAUNCH_BLOCKING=1
export HYDRA_FULL_ERROR=1

CONFIG_NAME=${1:-libero_goal_grpo_openvlaoft.eval}

LOG_DIR="${REPO_PATH}/logs/$(date +'%Y%m%d-%H:%M:%S')"
MEGA_LOG_FILE="${LOG_DIR}/eval_embodiment.log"
mkdir -p "${LOG_DIR}"

CMD="python ${SRC_FILE} --config-path ${EMBODIED_PATH}/config/ \
    --config-name ${CONFIG_NAME} \
    runner.logger.log_path=${LOG_DIR}"

echo ${CMD}
${CMD} 2>&1 | tee "${MEGA_LOG_FILE}"
```

修改配置文件 修改examples/embodiment/config/libero_10_ppo_openpi.yaml 文件内相关配置

```
  runner:
  eval_policy_path: "/path/to/rl_ckpt.pt"    # 可选：.pt文件路径或直接填None。如果为None，将使用rollout.model.model_path中的检查点。
  only_eval: True    # 配置为 true 这样就不会再 eval 中间再跑 training 了
algorithm:
  eval_rollout_epoch: 1
rollout:
  model:
    model_path: "/path/to/sft_base_model/"
actor:
  model:
    model_path: "/path/to/sft_base_model/"
  tokenizer:
    tokenizer_model: "/path/to/sft_base_model/"
```

以下是一些关键配置字段：

字段名	作用
simulator_type	必须为 libero
task_suite_name	LIBERO 任务分支名（如 libero_goal）
max_episode_steps	每个 episode 的最大步数（默认 512）
seed	环境随机种子
num_envs	并行评估环境数量（例如 500）

运行评估脚本 通过以下命令运行评估脚本：

```
bash examples/embodiment/eval_embodiment.sh libero_10_ppo_openpi_pi05
```

请注意，如果您使用的显卡规格为NVIDIA A800，请修改评估脚本中的 7-8 行，改为使用 egl ，否则会导致产生异常 kill 的情况：

```
  ##### export MUJOCO_GL="osmesa"
 ##### export PYOPENGL_PLATFORM="osmesa"

export MUJOCO_GL="egl"
export PYOPENGL_PLATFORM="egl"
```

运行结束后，可以在./logs/时间 文件夹下找到对应的评估输出文件：

```
.
├── tensorboard
└── video
    └── eval
        ├── seed_0
        ├── seed_1
        ├── seed_2
        ├── seed_3
        ├── seed_4
        ├── seed_5
        ├── seed_6
        └── seed_7
```

评估矩阵 在其根目录下有对应评估报文，可以找到如下一行确认评估结果：

```
eval_metrics={
  'eval/success_once': array(0.9173387, dtype=float32),
  'eval/return': array(0.7520161, dtype=float32),
  'eval/episode_len': array(480., dtype=float32),
  'eval/reward': array(0.0015667, dtype=float32),
  'eval/success_at_end': array(0.7520161, dtype=float32)
 }
```

评估结果包含以下字段，如果启用了 TensorBoard，这些指标也会记录在TensorBoard 中：

1. **eval/success_once**：也就是成功率——“一个 episode 里只要曾经满足过一次成功条件”，然后对所有评估 episode 取平均。
2. **eval/success_at_end**：终态成功率——看 episode **结束时**是否处于成功状态的比例（通常会 ≤ success_once，因为可能中途成功过但最后失败/掉回去）。

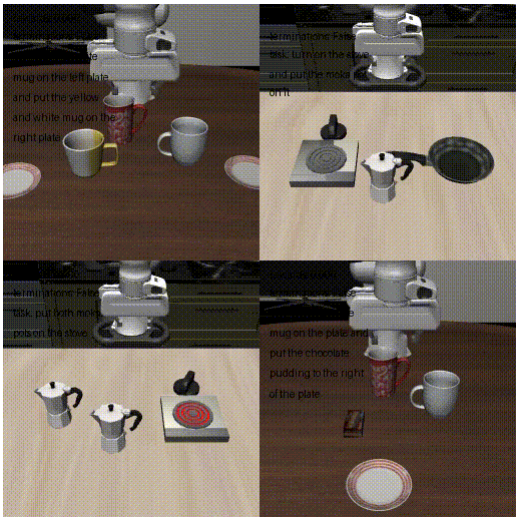
> **success_at_end** 其实现细节就是ignore_terminations即在max_step之前已经任务成功了但是还继续跑，直到截断。 这里两个的区别是：一个子任务是放置杯子，如果机械臂在操作过程中放置好了，但是离开的时候又把杯子碰倒了，那么success_once=1，但是success_at_end=0

3. **eval/episode_len**：平均 episode 长度（步数）。一般会 ≤ `max_episode_steps`（episode 的 horizon 上限）

> 1. `env.train.max_episode_steps`: Maximum number of steps per episode for training. 我们配置默认设置的是480，也就是说如果达到这么多步都没有结束就会被截断
> 2. `actor.model.num_action_chunks`：送入env中的最大截断action长度，actions_horizon是模型推理的最大长度，基于此截断送入env中交互
> 3. 模型推理的次数：`max_episode_steps` / `num_action_chunks` 就是当前配置下最大需要的推理次数

4. **eval/reward**：平均每步 reward（per-step mean）。也就是RL中的**单步奖励**
5. **eval/return**：每个 episode 的总回报（把该 episode 的每步 reward 累加起来），所以通常有 `return ≈ reward × episode_len`；官方示例里就是 1.0476562 ≈ 0.0130957 × 80。也就是RL中的T累积奖励

评估视频 同时在 video 文件夹中也可以找到生成的可视化视频（每个视频下格数为 total_num_envs / 显卡数 / pipeline_stage_num）：



每个视频都包含reward、terminated 以及 task三项说明

```
 reward: 0.0                  # 这一步环境给的即时奖励
terminated: false             # 当前task是否完成
task: pick up the book        # 子任务描述
and place it in the
back compartment of
the caddy
```

附录1： 官方模型权重

以下是截止 2026-02-06 时Rlinf 在 huggingface 所上传的所有模型，最新模型可直接访问：https://huggingface.co/RLinf/collections 或 https://hf-mirror.com/RLinf/collections 进行获取

模型名称	参数量	所属模型系列	说明	更新日期	HF 下载地址
GR00t-RL系列					
RLinf-GR00t-RL-Spatial-Step400	-	GR00t-RL系列	GR00T RL模型 - 空间推理任务	2025-12-16	https://huggingface.co/RLinf/RLinf-GR00t-RL-Spatial-Step400
RLinf-GR00t-RL-Goal-Step500	-	GR00t-RL系列	GR00T RL模型 - 目标导向任务	2025-12-11	https://huggingface.co/RLinf/RLinf-GR00t-RL-Goal-Step500

RLinf-Gr00t-RL-Long-Step300	-	Gr00t-RL系列	GR00T RL模型 - 长程任务	2025-12-11	https://huggingface.co/RLinf/RLinf-Gr00t-RL-Long-Step300
RLinf-Gr00t-RL-Object-Step400	-	Gr00t-RL系列	GR00T RL模型 - 物体操作任务	2025-12-11	https://huggingface.co/RLinf/RLinf-Gr00t-RL-Object-Step400
RLinf-Gr00t-RL-Stack-cube	-	Gr00t-RL系列	GR00T RL模型 - Stack cube任务	2026-02-06	https://huggingface.co/RLinf/RLinf-Gr00t-RL-Stack-cube
Gr00t-SFT系列					
RLinf-Gr00t-SFT-Object	3B	Gr00t-SFT系列	GR00T SFT模型 - 物体操作任务	2025-12-11	https://huggingface.co/RLinf/RLinf-Gr00t-SFT-Object
RLinf-Gr00t-SFT-Goal	3B	Gr00t-SFT系列	GR00T SFT模型 - 目标导向任务	2025-12-11	https://huggingface.co/RLinf/RLinf-Gr00t-SFT-Goal
RLinf-Gr00t-SFT-Long	3B	Gr00t-SFT系列	GR00T SFT模型 - 长程任务	2025-12-11	https://huggingface.co/RLinf/RLinf-Gr00t-SFT-Long
RLinf-Gr00t-SFT-Spatial	3B	Gr00t-SFT系列	GR00T SFT模型 - 空间推理任务	2025-11-14	https://huggingface.co/RLinf/RLinf-Gr00t-SFT-Spatial
RLinf-Gr00t-SFT-Stack-cube	3B	Gr00t-SFT系列	GR00T SFT模型 - Stack cube任务	2026-02-06	https://huggingface.co/RLinf/RLinf-Gr00t-SFT-Stack-cube
OpenSora-LIBERO系列					
RLinf-OpenSora-LIBERO-Object	1B	OpenSora-LIBERO系列	OpenSora模型 - LIBERO-Object场景任务	2026-01-23	https://huggingface.co/RLinf/RLinf-OpenSora-LIBERO-Object
RLinf-OpenSora-LIBERO-Spatial	1B	OpenSora-LIBERO系列	OpenSora模型 - LIBERO-Spatial空间推理任务	2026-01-23	https://huggingface.co/RLinf/RLinf-OpenSora-LIBERO-Spatial
OpenVLA系列					
RLinf-OpenVLA-GRPO-ManiSkill3-25ood	8B	OpenVLA系列	OpenVLA模型 - ManiSkill3 OOD任务 - GRPO训练	2025-10-10	https://huggingface.co/RLinf/RLinf-OpenVLA-GRPO-ManiSkill3-25ood
RLinf-OpenVLA-PPO-ManiSkill3-25ood	8B	OpenVLA系列	OpenVLA模型 - ManiSkill3 OOD任务 - PPO训练	2025-10-10	https://huggingface.co/RLinf/RLinf-OpenVLA-PPO-ManiSkill3-25ood
OpenVLAOFT系列					
RLinf-OpenVLAOFT-Behavior	8B	OpenVLAOFT系列	OpenVLAOFT通用行为模型	2025-10-29	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-Behavior
RLinf-OpenVLAOFT-GRPO-ManiSkill3-25ood	8B	OpenVLAOFT系列	OpenVLAOFT模型 - ManiSkill3 OOD任务 - GRPO训练	2025-10-10	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-GRPO-ManiSkill3-25ood
RLinf-OpenVLAOFT-PPO-ManiSkill3-25ood	8B	OpenVLAOFT系列	OpenVLAOFT模型 - ManiSkill3 OOD任务 - PPO训练	2025-10-10	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-PPO-ManiSkill3-25ood
RLinf-OpenVLAOFT-GRPO-LIBERO-object	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-object任务 - GRPO训练	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-GRPO-LIBERO-object
RLinf-OpenVLAOFT-GRPO-LIBERO-goal	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-goal任务 - GRPO训练	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-GRPO-LIBERO-goal
RLinf-OpenVLAOFT-GRPO-LIBERO-90	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-90任务 - GRPO训练	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-GRPO-LIBERO-90
RLinf-OpenVLAOFT-GRPO-LIBERO-spatial	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-spatial任务 - GRPO训练	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-GRPO-LIBERO-spatial
RLinf-OpenVLAOFT-GRPO-LIBERO-long	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-long任务 - GRPO训练	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-GRPO-LIBERO-long
RLinf-OpenVLAOFT-LIBERO-90-Base-Lora	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-90任务 - Base Lora	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora
RLinf-OpenVLAOFT-LIBERO-130-Base-Lora	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-130任务 - Base Lora	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora
RLinf-OpenVLAOFT-LIBERO-130	8B	OpenVLAOFT系列	OpenVLAOFT模型 - LIBERO-130任务	2025-12-21	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-LIBERO-130
RLinf-OpenVLAOFT-ManiSkill-Base-Lora	-	OpenVLAOFT系列	OpenVLAOFT模型 - ManiSkill任务 - Base Lora	2025-12-14	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-ManiSkill-Base-Lora
RLinf-OpenVLAOFT-ManiSkill-Base-Main	8B	OpenVLAOFT系列	OpenVLAOFT模型 - ManiSkill任务 - Base Main	2025-12-14	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-ManiSkill-Base-Main
OpenVLAOFT-RoboTwin系列					
RLinf-OpenVLAOFT-RoboTwin-SFT-beat_block_hammer	8B	OpenVLAOFT-RoboTwin系列	SFT模型 - 用锤子敲击方块	2026-01-10	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-SFT-beat_block_hammer
RLinf-OpenVLAOFT-RoboTwin-SFT-place_empty_cup	8B	OpenVLAOFT-RoboTwin系列	SFT模型 - 放置空杯子	2026-01-08	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-SFT-place_empty_cup
RLinf-OpenVLAOFT-RoboTwin-SFT-handover_block	8B	OpenVLAOFT-RoboTwin系列	SFT模型 - 交接方块	2026-01-31	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-SFT-handover_block
RLinf-OpenVLAOFT-RoboTwin-SFT-lift_pot	8B	OpenVLAOFT-RoboTwin系列	SFT模型 - 抬起锅	2026-01-31	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-SFT-lift_pot
RLinf-OpenVLAOFT-RoboTwin-SFT-move_can_pot	8B	OpenVLAOFT-RoboTwin系列	SFT模型 - 移动罐子到锅中	2026-01-31	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-SFT-move_can_pot
RLinf-OpenVLAOFT-RoboTwin-SFT-pick_dual_bottles	8B	OpenVLAOFT-RoboTwin系列	SFT模型 - 抬取两个瓶子	2026-01-10	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-SFT-pick_dual_bottles
RLinf-OpenVLAOFT-RoboTwin-RL-beat_block_hammer	-	OpenVLAOFT-RoboTwin系列	RL模型 - 用锤子敲击方块	2026-01-14	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-RL-beat_block_hammer
RLinf-OpenVLAOFT-RoboTwin-RL-handover_block	-	OpenVLAOFT-RoboTwin系列	RL模型 - 交接方块	2026-01-31	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-RL-handover_block

handover_block		RoboTwin系列		01-31	handover_block
RLinf-OpenVLAOFT-RoboTwin-RL-lift_pot	8B	OpenVLAOFT-RoboTwin系列	RL模型 - 抬起锅	2026-01-31	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-RL-lift_pot
RLinf-OpenVLAOFT-RoboTwin-RL-move_can_pot	8B	OpenVLAOFT-RoboTwin系列	RL模型 - 移动罐子到锅中	2026-01-31	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-RL-move_can_pot
RLinf-OpenVLAOFT-RoboTwin-RL-pick_dual_bottles	8B	OpenVLAOFT-RoboTwin系列	RL模型 - 拾取两个瓶子	2026-01-13	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-RL-pick_dual_bottles
RLinf-OpenVLAOFT-RoboTwin-RL-place_empty_cup	-	OpenVLAOFT-RoboTwin系列	RL模型 - 放置空杯子	2026-01-13	https://huggingface.co/RLinf/RLinf-OpenVLAOFT-RoboTwin-RL-place_empty_cup
Pi0系列					
RLinf-Pi0-Behavior	4B	Pi0系列	Pi0通用行为模型	2025-11-27	https://huggingface.co/RLinf/RLinf-Pi0-Behavior
RLinf-Pi0-CALVIN-ABC-D-RL-FlowNoise	-	Pi0系列	Pi0模型 - CALVIN ABC-D任务 - FlowNoise算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi0-CALVIN-ABC-D-RL-FlowNoise
RLinf-Pi0-CALVIN-ABC-D-RL-FlowSDE	-	Pi0系列	Pi0模型 - CALVIN ABC-D任务 - FlowSDE算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi0-CALVIN-ABC-D-RL-FlowSDE
RLinf-Pi0-CALVIN-ABC-D-SFT	4B	Pi0系列	Pi0模型 - CALVIN ABC-D任务 - SFT	2025-11-29	https://huggingface.co/RLinf/RLinf-Pi0-CALVIN-ABC-D-SFT
RLinf-Pi0-LIBERO-Long-SFT	4B	Pi0系列	Pi0模型 - LIBERO长程任务 - SFT	2025-10-16	https://huggingface.co/RLinf/RLinf-Pi0-LIBERO-Long-SFT
RLinf-Pi0-LIBERO-Spatial-Object-Goal-SFT	4B	Pi0系列	Pi0模型 - LIBERO空间物体目标 - SFT	2025-10-16	https://huggingface.co/RLinf/RLinf-Pi0-LIBERO-Spatial-Object-Goal-SFT
RLinf-Pi0-ManiSkill-25Main-RL-FlowNoise	-	Pi0系列	Pi0模型 - ManiSkill2 25个主要任务 - FlowNoise算法	2025-12-28	https://huggingface.co/RLinf/RLinf-Pi0-ManiSkill-25Main-RL-FlowNoise
RLinf-Pi0-ManiSkill-25Main-RL-FlowSDE	-	Pi0系列	Pi0模型 - ManiSkill2 25个主要任务 - FlowSDE算法	2025-12-28	https://huggingface.co/RLinf/RLinf-Pi0-ManiSkill-25Main-RL-FlowSDE
RLinf-Pi0-ManiSkill-25Main-SFT	4B	Pi0系列	Pi0模型 - ManiSkill2 25个主要任务 - SFT	2025-12-18	https://huggingface.co/RLinf/RLinf-Pi0-ManiSkill-25Main-SFT
RLinf-Pi0-MetaWorld-RL-FlowNoise	-	Pi0系列	Pi0模型 - MetaWorld任务 - FlowNoise算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi0-MetaWorld-RL-FlowNoise
RLinf-Pi0-MetaWorld-RL-FlowSDE	-	Pi0系列	Pi0模型 - MetaWorld任务 - FlowSDE算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi0-MetaWorld-RL-FlowSDE
RLinf-Pi0-MetaWorld-SFT	4B	Pi0系列	Pi0模型 - MetaWorld任务 - SFT	2025-12-19	https://huggingface.co/RLinf/RLinf-Pi0-MetaWorld-SFT
RLinf-Pi0-RoboCasa	4B	Pi0系列	Pi0模型 - RoboCasa场景任务	2025-12-17	https://huggingface.co/RLinf/RLinf-Pi0-RoboCasa
Pi05系列					
RLinf-Pi05-CALVIN-ABC-D-RL-FlowNoise	-	Pi05系列	Pi05模型 - CALVIN ABC-D任务 - FlowNoise算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi05-CALVIN-ABC-D-RL-FlowNoise
RLinf-Pi05-CALVIN-ABC-D-RL-FlowSDE	-	Pi05系列	Pi05模型 - CALVIN ABC-D任务 - FlowSDE算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi05-CALVIN-ABC-D-RL-FlowSDE
RLinf-Pi05-CALVIN-ABC-D-SFT	4B	Pi05系列	Pi05模型 - CALVIN ABC-D任务 - SFT	2025-11-29	https://huggingface.co/RLinf/RLinf-Pi05-CALVIN-ABC-D-SFT
RLinf-Pi05-GSEnv-PutCubeOnPlate-V0-SFT	4B	Pi05系列	Pi05模型 - GSEnv环境中将方块放置到盘子上	2026-02-02	https://huggingface.co/RLinf/RLinf-Pi05-GSEnv-PutCubeOnPlate-V0-SFT
RLinf-Pi05-LIBERO-130-fullshot-SFT	4B	Pi05系列	Pi05模型 - LIBERO-130任务 - fullshot SFT	2025-11-06	https://huggingface.co/RLinf/RLinf-Pi05-LIBERO-130-fullshot-SFT
RLinf-Pi05-LIBERO-SFT	4B	Pi05系列	Pi05模型 - LIBERO任务 - SFT	2025-11-04	https://huggingface.co/RLinf/RLinf-Pi05-LIBERO-SFT
RLinf-Pi05-ManiSkill-25Main-RL-FlowNoise	-	Pi05系列	Pi05模型 - ManiSkill2 25个主要任务 - FlowNoise算法	2025-12-28	https://huggingface.co/RLinf/RLinf-Pi05-ManiSkill-25Main-RL-FlowNoise
RLinf-Pi05-ManiSkill-25Main-RL-FlowSDE	-	Pi05系列	Pi05模型 - ManiSkill2 25个主要任务 - FlowSDE算法	2025-12-28	https://huggingface.co/RLinf/RLinf-Pi05-ManiSkill-25Main-RL-FlowSDE
RLinf-Pi05-ManiSkill-25Main-SFT	4B	Pi05系列	Pi05模型 - ManiSkill2 25个主要任务 - SFT	2025-12-18	https://huggingface.co/RLinf/RLinf-Pi05-ManiSkill-25Main-SFT
RLinf-Pi05-MetaWorld-RL-FlowNoise	-	Pi05系列	Pi05模型 - MetaWorld任务 - FlowNoise算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi05-MetaWorld-RL-FlowNoise
RLinf-Pi05-MetaWorld-RL-FlowSDE	-	Pi05系列	Pi05模型 - MetaWorld任务 - FlowSDE算法	2025-12-29	https://huggingface.co/RLinf/RLinf-Pi05-MetaWorld-RL-FlowSDE
RLinf-Pi05-MetaWorld-SFT	4B	Pi05系列	Pi05模型 - MetaWorld任务 - SFT	2025-11-20	https://huggingface.co/RLinf/RLinf-Pi05-MetaWorld-SFT
RLinf-Pi05-PPO-LIBERO-130	-	Pi05系列	Pi05模型 - LIBERO-130任务 - PPO训练	2025-11-13	https://huggingface.co/RLinf/RLinf-Pi05-PPO-LIBERO-130
Pi0-LIBERO-130-fullshot系列					
RLinf-Pi0-LIBERO-130-fullshot-SFT	4B	Pi0-LIBERO-130-fullshot系列	Pi0模型 - LIBERO-130任务 - fullshot SFT	2025-11-06	https://huggingface.co/RLinf/RLinf-Pi0-LIBERO-130-fullshot-SFT
ResNet10系列					
RLinf-ResNet10-pretrained	-	ResNet10系列	ResNet10预训练模型	2025-12-23	https://huggingface.co/RLinf/RLinf-ResNet10-pretrained
Math系列					
				2025-	

RLinf-math-1.5B	2B	Math系列	数学推理模型 - 1.5B参数版本	09-01	https://huggingface.co/RLinf/RLinf-math-1.5B
RLinf-math-7B	8B	Math系列	数学推理模型 - 7B参数版本	2025-10-10	https://huggingface.co/RLinf/RLinf-math-7B
WideSeek-R1系列					
WideSeek-R1-4b	4B	WideSeek-R1系列	宽Scaling多Agent信息检索模型，通过MARL训练实现lead-agent和subagent协同	2026-02-05	https://huggingface.co/RLinf/WideSeek-R1-4b

LingBot-Depth训推

> https://technology.robbyant.com/lingbot-depth 以掩码深度建模实现超越工业级相机的稠密深度感知



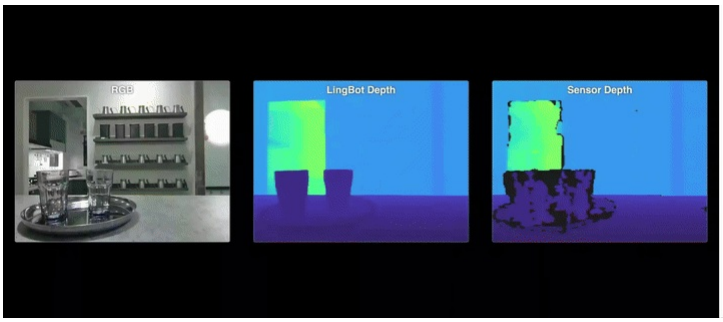
模型概述

LingBot-Depth 是一种面向真实场景的深度补全模型，旨在将不完整且受噪声干扰的深度传感器数据转化为高质量、具备真实尺度的三维测量结果。该模型针对机器人与自主系统中的关键挑战：消费级深度相机在玻璃、镜面及金属等强反光表面上容易失效，而这些场景恰恰对可靠深度感知提出了更高要求。

实验结果表明，本模型在深度精度与像素覆盖率两项核心指标上均超越业界顶级工业级深度相机。在多个基准测试中，LingBot-Depth 在深度补全、单目深度估计及双目匹配任务上均达到当前最优水平，并在无需显式时序建模的情况下保持视频级时间一致性。下游任务验证进一步表明，模型能够在 RGB 与深度两种模态之间学习到对齐的潜在空间表征，从而实现对透明及反光物体的稳定机器人抓取。

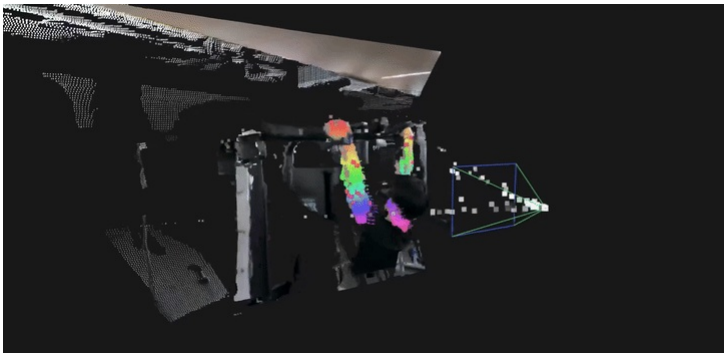
为推动空间感知研究的发展，蚂蚁灵波开源了模型，以及 200 万对高质量 RGB–深度配对数据。

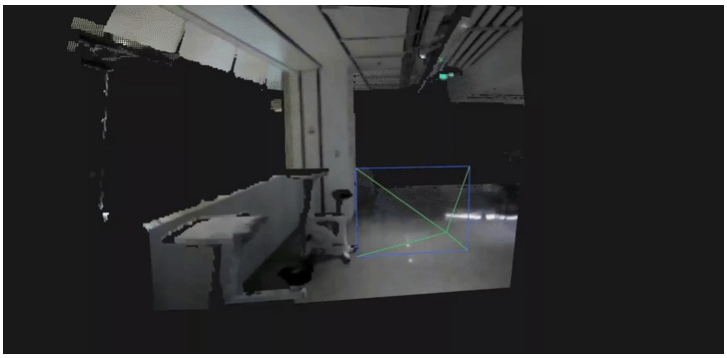
精准且稳定的相机深度感知

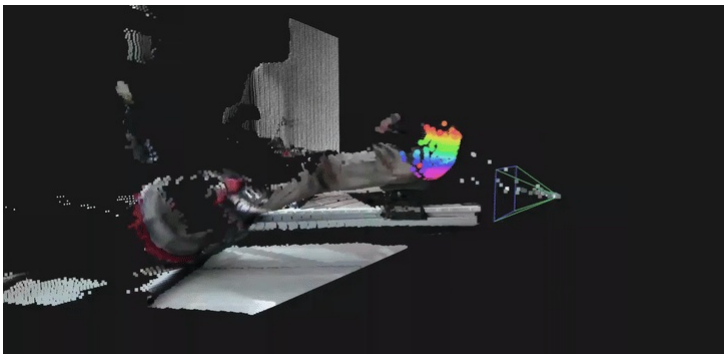


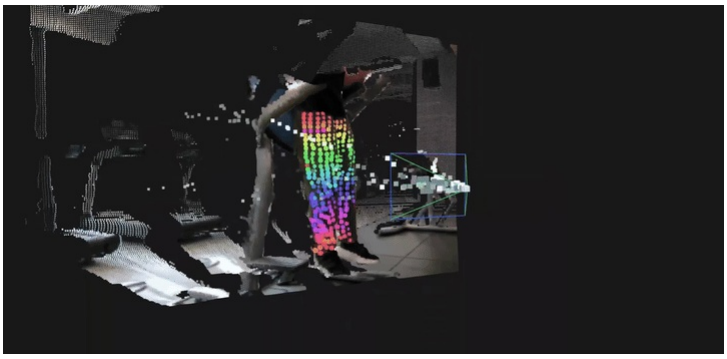
LingBot-Depth 在传统深度传感器易失效的复杂场景中，仍可输出具备真实尺度的高精度深度结果，包括透明物体、玻璃表面以及高反光材质等极具挑战性的环境。不同于依赖硬件改进的方案，本模型从视觉理解层面弥补传感器缺陷，实现对真实三维结构的可靠恢复。 除单帧精度优势外，LingBot-Depth 还表现出优异的时间一致性。在无需显式时序建模的情况下，模型即可为视频输入生成稳定、连贯的深度序列，有效避免闪烁与结构跳变问题，为机器人操作、AR/VR 以及动态场景感知等应用提供可靠的连续空间理解能力。

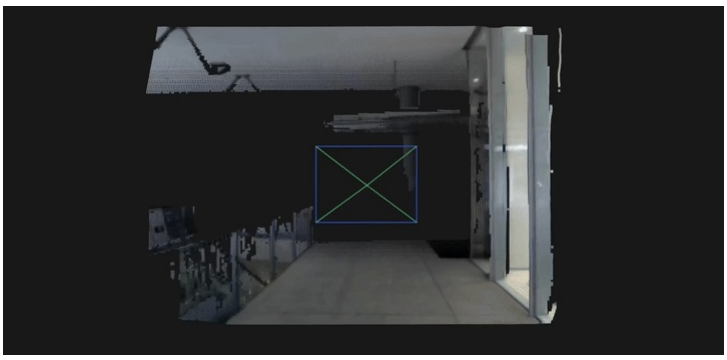
卓越的 3D 和 4D 环境感知能力

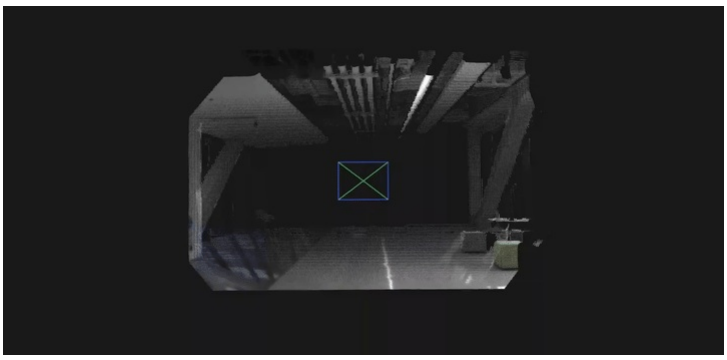












LingBot-Depth 为下游空间感知任务提供了坚实而通用的基础能力。通过将含噪且不完整的传感器深度优化为干净、稠密且具备真实尺度的三维测量结果，模型显著提升了多种高层视觉任务的稳定性与精度。具体而言，LingBot-Depth 支持：（1）更加准确的结构化室内场景建图，并有效提升相机位姿与运动轨迹估计的精度；（2）面向机器人学习的可靠 4D 点跟踪能力，在统一的真实尺度空间中同时刻画静态场景几何结构与动态物体运动。这使得系统能够在复杂真实环境中建立一致、连续且可用于决策与交互的空间理解表征。

灵巧抓取操作适用于透明与反光物体


1x speed, fully autonomous


1x speed, fully autonomous


1x speed, fully autonomous


1x speed, fully autonomous


1x speed, fully autonomous

通过在统一潜在空间中联合对齐 RGB 外观信息与深度几何结构，LingBot-Depth 使机器人在以往难以处理的复杂场景中实现稳定可靠的操作能力。基于模型优化后的高质量深度结果及跨模态对齐特征，我们进一步训练了一种基于扩散模型的抓取位姿生成策略，在透明杯、反光金属容器等具有挑战性的物体上取得了较高的抓取成功率，而在这些场景中原始传感器深度往往完全失效。

真机 Scaling Law

现有基于双目的深度相机工作方式类似人眼：两个镜头从略有差异的视角同时拍摄场景，系统通过匹配两幅图像中的对应点来计算深度。然而，这种方法存在先天缺陷——在纹理缺失区域、透明材质（如玻璃）或强反光表面（如镜子和金属）上，匹配算法往往失效，因为两路图像要么过于相似、要么发生严重畸变。结果就是：恰恰在最需要深度信息的地方，传感器反而输出空洞或错误数据。

我们的方法称为掩码深度建模（Masked Depth Modeling, MDM）。训练过程中，我们使用海量 RGB–深度图像对，但刻意遮挡其中一部分深度区域，让模型仅根据 RGB 图像去预测缺失的深度值。随着训练进行，模型逐渐学会建立“外观—几何”之间的对应关系，也就是从“物体看起来像什么”推断“它大概有多远”。

核心洞察在于：传感器失效并非随机噪声，而是在特定材质与光照条件下可预测地发生。因此，我们不再将这些区域简单视为需要滤除的坏数据，而是把它们当作有价值的学习信号。模型会逐渐掌握这样的规律：“当我看到这种类似玻璃的外观和反射模式时，对应的深度大致应落在这个范围。”

在涵盖家庭、办公环境、健身房及户外场景的上千万张图像数据上完成训练后，LingBot-Depth 已能够对受损深度图进行智能补全，填补空洞区域，并生成在几何上合理、物理上可信的深度结果。

1. **把传感器深度失效视为掩码：一种自然的掩码深度建模机制**

我们将原始深度图中的固有缺损不再视为噪声，而是重新诠释为揭示几何歧义的"自然掩码"信号。基于这一洞察，我们提出了一种新颖的RGB-D掩码自编码器（RGBD-MAE）：模型接收完整的RGB token与仅有的有效原始深度token，学习基于视觉上下文预测缺失区域的深度信息。

2. **针对MDM的大规模数据采集管线**

我们构建了一套可扩展的数据整理与筛选流程，结合真实采集与高逼真模拟数据，共收集 300 万对 RGB–深度图像（其中 200 万为真实数据，100 万为合成数据）。真实数据覆盖家庭、办公室、健身房、大堂及户外等多种环境，使用多款主流商用深度相机采集。合成数据流程则渲染具备照片级真实感的场景，并模拟双目匹配伪影，从而逼真复现真实传感器的典型失效模式。在监督信号构建方面，我们为真实数据通过双目红外图像对计算高精度深度，为合成数据直接采用无误差的渲染深度作为真值，从而在传感器通常失效的区域仍能提供干净、可靠的监督目标。

3. **显式几何与隐式特征表征的双重对齐**

LingBot-Depth 学习到一个统一的潜在空间，使 RGB 外观信息与深度几何结构实现紧密耦合。模型不仅能够输出具备真实尺度的高精度深度图（显式几何），还可生成在不同模态间对齐的特征表示（隐式表征）。这种双重对齐机制为多种下游应用提供了强大支撑：显式深度信息可直接服务于机器人操作与抓取等任务，而跨模态对齐的特征表示则有助于三维点跟踪、场景理解等更高层次的空间感知任务。

模型测评结果

LingBot-Depth 在多个深度感知benchmark上取得领先，也在机器人任务上展现出巨大潜力：

深度补全（Depth Completion）

- 在 iBims、NYUv2、DIODE、ETH3D 等标准基准上，相比现有最佳方法实现 40–50% 的误差降低
- 在稀疏 SfM 输入条件下，室内场景 RMSE 降低 47%，室外场景降低 38%
- 在从轻度到极端损坏的不同难度设置下均保持稳定领先表现

超越工业级硬件

- 在复杂场景中生成比 ZED 双目相机更完整且更精确的深度结果
- 在结构光和双目系统同时失效的区域（如玻璃幕墙、镜面、水族馆通道）成功补全深度空洞
- 在无需显式时序建模的情况下保持视频帧间的时间一致性

机器人操作能力

- 使以往无法抓取的物体成为可能——透明收纳箱抓取成功率从 0% 提升至 50%
- 在多种反光和透明物体上提升 30–78% 的抓取成功率
- 钢杯：65% → 85%，玻璃杯：60% → 80%，玩具车：45% → 80%

基础模型能力（Foundation Model）

- 作为单目深度估计的优质预训练骨干，在 10 项基准上优于 DINOv2
- 加速双目匹配模型训练——FoundationStereo 在采用本模型初始化后收敛更快
- 对齐的 RGB–深度特征支持下游 3D/4D 跟踪任务，且无需针对具体任务进行微调

应用场景

基于 LingBot-Depth 的RGBD视频三维点追踪 LingBot-Depth 通过为现有三维点跟踪方法提供完整且具备真实尺度的深度信息，实现了更加稳健的 3D 点跟踪能力。模型补全了由传感器失效导致的深度空洞，从而显著提升了相机位姿轨迹估计的准确性，并支持在包含玻璃、镜面及高反光表面的复杂环境中对动态物体进行稳定跟踪。

基于 RGB-D 相机的灵巧手抓取 LingBot-Depth 改变了机器人对透明和反光物体的操作能力——这些场景以往因传感器失效而几乎无法处理。通过从受损深度中重建准确的几何结构，模型为基于扩散的抓取策略提供可靠输入，从而生成精确的抓取位姿，使机器人能够在玻璃杯、金属容器及半透明收纳盒等物体上实现稳定抓取。

在百舸平台上快速开始

部署环境要求

	部署要求	最佳实践
CPU	4核	建议按表单默认值及以上
内存	16G	建议按表单默认值及以上
GPU	按需	A10/A100/V100/L20等至少一张
CDS	按需	建议存储资源>50G
其它	无	无

镜像

请使用百舸官方镜像：`ccr-2hpgw1bh-pub.cnc.bj.baidubce.com/aihc-private/lingbot-depth:v0.5`

本镜像使用基础镜像：`registry.baidubce.com/inference/aibox-cuda:v2.0-cu12.9-cudnn9-ubuntu22.04`进行封装

如果不使用本示例提供的`lingbot-depth:v0.5`镜像，安装方法参考下文【依赖配置过程参考】

依赖配置过程参考（镜像中已完成配置）

```
 git clone https://github.com/robbyant/lingbot-depth
cd lingbot-depth
##### 安装包（使用 'python -m pip' 确保使用正确环境）
conda create -n lingbot-depth python=3.9
conda activate lingbot-depth
python -m pip install -e .
```

快速开始 推理示例：

```
 import torch
import cv2
import numpy as np
from mdm.model.v2 import MDMModel

##### 加载模型
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = MDMModel.from_pretrained('robbyant/lingbot-depth-pretrain-vitl-14-v0.5').to(device)

##### 加载并准备输入
image = cv2.cvtColor(cv2.imread('examples/0/rgb.png'), cv2.COLOR_BGR2RGB)
h, w = image.shape[:2]
image = torch.tensor(image / 255, dtype=torch.float32, device=device).permute(2, 0, 1)[None]
depth = cv2.imread('examples/0/raw_depth.png', cv2.IMREAD_UNCHANGED).astype(np.float32) / 1000.0
depth = torch.tensor(depth, dtype=torch.float32, device=device)[None]
intrinsics = np.loadtxt('examples/0/intrinsics.txt')
intrinsics[0] /= w  # 按宽度归一化 fx 和 cx
intrinsics[1] /= h  # 按高度归一化 fy 和 cy
intrinsics = torch.tensor(intrinsics, dtype=torch.float32, device=device)[None]

##### 执行推理
output = model.infer(image, depth_in=depth, intrinsics=intrinsics)
depth_pred = output['depth']  # 精化后的深度图
points = output['points']     # 三维点云
```

运行示例：模型首次运行时将自动从 Hugging Face 下载（无需手动下载）：

```
 ##### 基本用法 - 处理示例 0
python example.py
##### 使用不同的示例（可选 0-7）
python example.py --example 1
##### 使用深度补全优化模型
python example.py --model robbyant/lingbot-depth-postrain-dc-vitl14-v0.5
##### 自定义输出目录
python example.py --output my_results
##### 查看所有选项
python example.py --help
```

处理结果包含：

rgb.png（输入图像）、depth_input.npy（输入深度，float32，米）、depth_refined.npy（精化深度，float32，米）、depth_input.png（输入深度可视化）、depth_refined.png（精化深度可视化）、depth_comparison.png（对比图）、point_cloud.ply（三维点云）。

`examples/` 目录中内置 8 组示例场景（0-7）。

处理结果将被保存到/results文件夹下（或者其他用户指定的文件夹）

```
 result/
├── rgb.png              # Input RGB image
├── depth_input.npy      # Input depth (float32, meters)
├── depth_refined.npy    # Refined depth (float32, meters)
├── depth_input.png      # Input depth visualization
├── depth_refined.png    # Refined depth visualization
├── depth_comparison.png # Side-by-side comparison
└── point_cloud.ply      # 3D point cloud
```

模型详情

架构

- **编码器**：带 RGB-D 融合的视觉 Transformer（Large）
- **解码器**：多尺度特征金字塔，包含专用预测头
- **预测头**：深度回归
- **训练方式**：带重建目标的遮蔽深度建模

输入格式

RGB 图像

- **形状**：`[B, 3, H, W]`，归一化至 `[0, 1]`
- **格式**：PyTorch 张量，float32

深度图

- **形状**：[B, H, W]
- **单位**：米（可通过 `scale` 参数配置）
- **无效区域**：值为 `0` 或 `NaN`

相机内参

- **形状**：[B, 3, 3]
- **格式**：归一化格式，fx'=fx/W，fy'=fy/H，cx'=cx/W，cy'=cy/H

示例：

```
[[fx/W,   0,   cx/W],
[  0,  fy/H,  cy/H],
[  0,   0,    1  ]]
```

输出格式

模型返回一个字典：

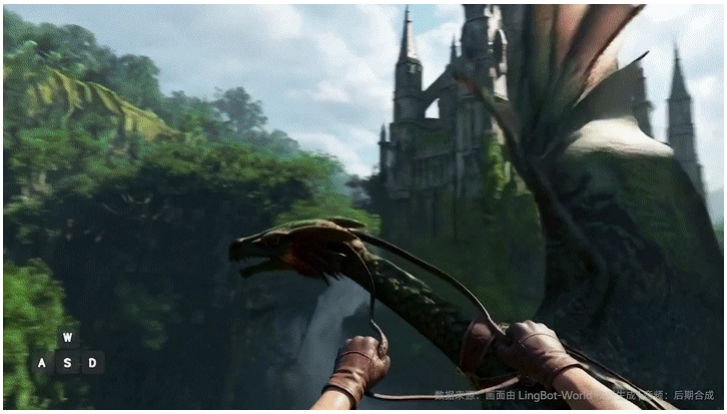
```
{
  'depth': torch.Tensor,   # 精化后的深度图 [B, H, W]
  'points': torch.Tensor,  # 点云 [B, H, W, 3]，相机坐标系
}
```

推理参数

```
model.infer(
  image,          # RGB 张量 [B, 3, H, W]
  depth_in=None,  # 输入深度图 [B, H, W]
  use_fp16=True,  # 混合精度推理
  intrinsics=None # 相机内参 [B, 3, 3]
)
```

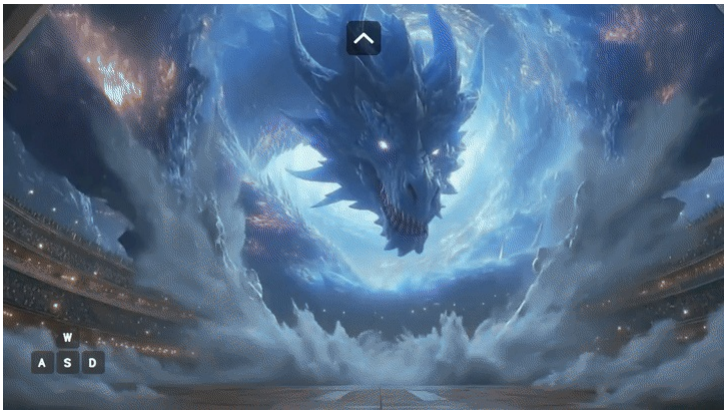
LingBot-World训推

> https://technology.robbyant.com/lingbot-world 世界模型的开放前沿



LingBot-World 是一个专为交互式世界模型设计的开源框架。其核心 **LingBot-World-Base** 致力于提供高保真、可控制且逻辑一致的模拟环境。该模型由一个可扩展数据引擎（Scalable Data Engine）驱动，通过从大规模游戏环境中学习物理规律与因果关系，超越了传统的被动式视频合成，实现了与生成世界的交互。

高保真模拟与精准控制







LingBot-World 告别了随机的"幻觉"式生成。它支持精细化的、由动作驱动的生成（action-conditioned generation），能够精确响应用户指令，渲染出高质量且符合物理真实感的动态场景。

长时序一致性与记忆







凭借增强的上下文记忆能力，LingBot-World 能够在长达数分钟的轨迹中，持续保持结构的完整性、物体恒存性（object permanence）以及叙事逻辑的连贯性。

统一物理世界与游戏世界的建模







团队利用专有的可扩展数据引擎，将游戏引擎视为无限的数据生成器。该模型统一了物理世界与游戏世界的内在逻辑，使其能够从合成数据中获得强大的泛化能力，并成功应用于真实世界场景。

涌现能力

随着世界模型规模不断扩展，团队观察到一系列超越简单视频生成的复杂行为开始涌现，这表明模型对空间逻辑、时间持续性和物理约束具备了真正的理解。

动态的屏外记忆



模型不仅实现了简单的物体恒存性，还能对离开画面的智能体（例如视频中的猫）保持持续的记忆，并推断其在屏外的行为。这确保了当视角切回时，世界状态是自然演进的，而非静止冻结。

探索生成边界

团队的模型突破了时间连贯性的限制，现已能够生成超长时序的视频，并维持稳定、高保真的环境质量而不发生衰减。

基于物理的真实约束



模型能够执行真实的碰撞动力学，防止智能体出现"穿模"或无视固体障碍等不符合物理规律的行为。这种对空间逻辑的严格遵循，确保了所有运动都具备物理真实感，而非单纯的视觉幻觉。

> 更多特性说明请参考蚂蚁灵波官网：https://technology.robbyant.com/lingbot-world

局限性与未来方向

尽管 LingBot-World 模型展现了巨大的潜力，但仍面临若干技术瓶颈。首先，高昂的推理成本目前需要企业级 GPU 支持，这使得该技术难以在消费级硬件上普及。其次，由于模型的记忆能力是从上下文窗口中涌现的，而非来自一个显式的存储模块，模拟缺乏长期稳定性，常常导致"环境漂移"（environmental drifting）——即场景在长时间运行后逐渐失去结构完整性。此外，当前的控制能力仅限于基础的导航，尚不具备复杂交互或特定物体操控所需的精细控制精度。最后，通过因果蒸馏（causal distillation）来实现实时性能，目前不得不在一定程度上牺牲视觉保真度。

展望未来，团队的技术路线图将优先扩展动作空间和物理引擎，以支持更多样化、更复杂的交互。为了确保长期稳定性，团队计划引入显式的记忆模块，而不是依赖于涌现的上下文。此外，团队正致力于消除"生成漂移"（generation drift），为实现稳健的、可无限运行的游戏体验和更强大的模拟功能铺平道路。

在百舸平台上快速开始

部署环境要求

	部署要求	最佳实践
CPU	32核	64核
内存	320G	建议按表单默认值及以上
GPU	A800 * 4	本示例中使用8卡A800
CDS	按需	建议挂载云盘大小 100G 以上
其它	无	无

镜像

请使用百舸官方镜像：ccr-2hpgw1bh-pub.cnc.bj.baidubce.com/aihc-private/lingbot-world:v0.1

本镜像使用基础镜像：registry.baidubce.com/inference/aibox-cuda:v2.0-cu12.8-cudnn9-ubuntu22.04

数据准备与系统依赖配置

```
!set -e && \
wget -q https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.10.zip && \
unzip -q linux-bcecmd-0.5.10.zip && \
chmod +x linux-bcecmd-0.5.10/bcecmd && \
mv linux-bcecmd-0.5.10/bcecmd /usr/local/bin/bcecmd && \
rm -rf linux-bcecmd-0.5.10 linux-bcecmd-0.5.10.zip

!bcecmd -v
```

```
 apt-get update && apt-get install -y build-essential cmake

cd lingbot-world
source .venv/bin/activate

bcecmd bos cp -r bos:/aibox-private/LingBot/LingBot-World/lingbot-world-base-cam ./lingbot-world-base-cam
bcecmd bos cp -r bos:/aibox-private/LingBot/LingBot-World/lingbot-world-base-act ./lingbot-world-base-act
```

推理命令

LingBot-World-Base (Cam) - 480P

```
torchrun --nproc_per_node=8 generate.py \
  --task i2v-A14B \
  --size 480*832 \
  --ckpt_dir ./lingbot-world-base-cam \
  --image examples/00/image.jpg \
  --action_path examples/00 \
  --dit_fsdp \
  --t5_fsdp \
  --ulysses_size 8 \
  --frame_num 161 \
  --prompt "The video presents a soaring journey through a fantasy jungle."
```

LingBot-World-Base (Cam) - 720P

```
torchrun --nproc_per_node=8 generate.py \
  --task i2v-A14B \
  --size 720*1280 \
  --ckpt_dir ./lingbot-world-base-cam \
  --image examples/00/image.jpg \
  --action_path examples/00 \
  --dit_fsdp \
  --t5_fsdp \
  --ulysses_size 8 \
  --frame_num 161 \
  --prompt "The video presents a soaring journey through a fantasy jungle."
```

无控制信号运行

```
torchrun --nproc_per_node=8 generate.py \
  --task i2v-A14B \
  --size 480*832 \
  --ckpt_dir ./lingbot-world-base-cam \
  --image examples/00/image.jpg \
  --dit_fsdp \
  --t5_fsdp \
  --ulysses_size 8 \
  --frame_num 161 \
  --prompt "The video presents a soaring journey through a fantasy jungle."
```

LingBot-World-Base (Act)

```
torchrun --nproc_per_node=8 generate.py \
  --task i2v-A14B \
  --size 480*832 \
  --ckpt_dir ./lingbot-world-base-act \
  --image examples/00/image.jpg \
  --action_path examples/00 \
  --dit_fsdp \
  --t5_fsdp \
  --ulysses_size 8 \
  --frame_num 161 \
  --prompt "Your prompt here"
```

参数说明

参数	说明
--task	任务类型，使用 i2v-A14B
--size	输出分辨率，如 480*832 或 720*1280
--ckpt_dir	模型权重目录
--image	输入图像路径
--action_path	控制信号路径（包含 intrinsics.npy, poses.npy）
--dit_fsdp	DiT 模型使用 FSDP 分布
--t5_fsdp	T5 编码器使用 FSDP 分布
--ulysses_size	Ulysses 并行度，通常等于 GPU 数量
--frame_num	生成帧数，161帧约10秒（16 FPS）
--prompt	文本提示
--t5_cpu	可选，将 T5 放到 CPU 以节省显存

生成更长时间视频

如果显存充足，可以增加 frame_num 生成1分钟视频：

```
--frame_num 961  # 1分钟视频 @ 16FPS
```

如果显存不足，使用 `--t5_cpu` 减少显存占用。

输入格式

控制信号（可选）

- `intrinsics.npy`: Shape `[num_frames, 4]`，格式 `[fx, fy, cx, cy]`
- `poses.npy`: Shape `[num_frames, 4, 4]`，OpenCV 坐标系下的变换矩阵

控制信号可以使用 ViPE 从视频中提取。

模型版本

模型	控制信号	分辨率	下载链接
LingBot-World-Base (Cam)	相机姿态	480P & 720P	HuggingFace / ModelScope
LingBot-World-Base (Act)	动作	480P & 720P	HuggingFace
LingBot-World-Fast	-	-	*待发布*

量化模型

对于显存有限的用户，可以使用 4-bit 量化版本：

下载链接：https://huggingface.co/cahlen/lingbot-world-base-cam-nf4

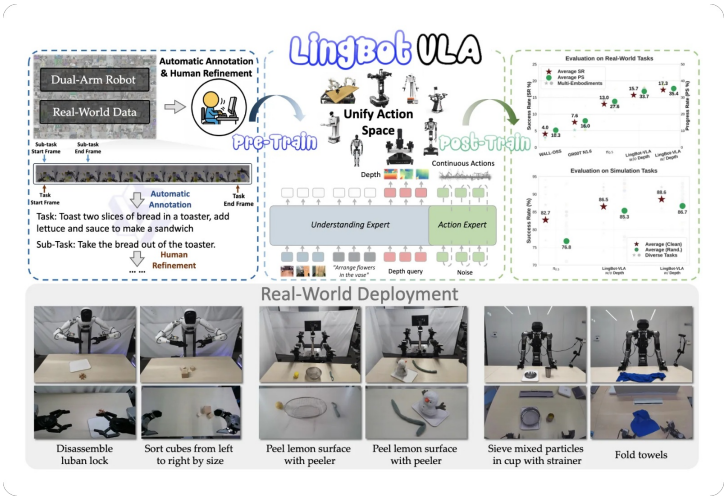
> ⚠ 注意：量化模型仅用于推理，视觉质量和时间一致性可能略有下降。

LingBot-VLA训推

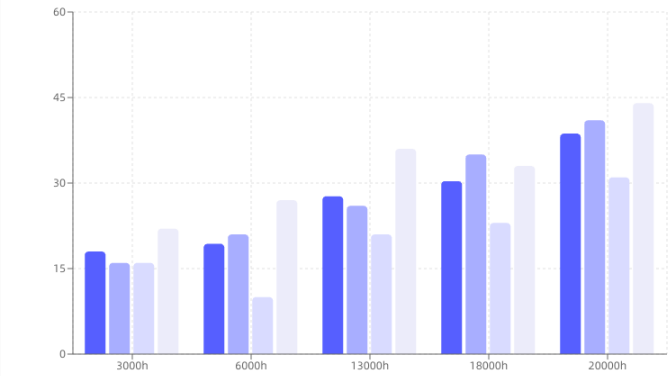
> https://technology.robbyant.com/lingbot-vla 一个实用的 VLA 基础模型

模型概要

蚂蚁灵波团队开发了 LingBot-VLA 具身智能基座模型，其训练数据涵盖 9 种主流双臂机器人配置、总计约 20,000 小时的真实世界操作数据。团队在 3 种机器人构型上进行了系统性评估，每种构型设置 100 项任务，每项任务使用 130 条后训练数据进行适配。实验结果表明，LingBot-VLA 相较于现有方案具有显著优势，充分验证了其卓越的性能表现与广泛的泛化能力。与此同时，团队构建了一套高效的后训练工具链，在 8 卡 GPU 配置下实现了每秒 261 个样本的吞吐量，较现有 VLA 代码库提升了 1.5 至 2.8 倍（具体取决于所依赖的 VLM 基座模型）。上述特性使该模型能够充分满足实机部署需求。为推动机器人学习领域的持续发展，团队开源了全部代码、基座模型及评测基准数据，旨在为更具挑战性的任务研究提供支撑，并促进领域内评估标准的规范化建设。

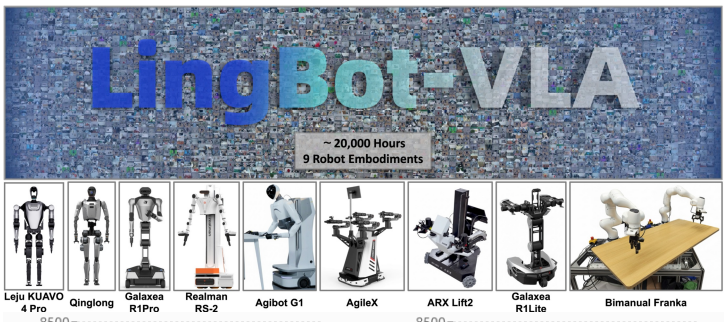


真机 Scaling Law



基于在海量的真实世界数据上的预训练，团队第一次系统研究了 VLA 模型在真实机器人任务性能上随着数据规模增长时的 scaling law。该项目发现随着预训练数据规模从 3,000 小时扩展到 6,000、13,000、18,000，最终至20,000 小时，模型在下游任务的成功率获得持续且显著的提升。值得注意的是，预训练数据量达到 20,000 小时时，模型性能仍呈现上升趋势，表明 VLA 的性能仍然能够随着数据量的增加而提升。这些实验结果证明了 VLA 模型在用真实数据预训练时呈现了良好的可扩展性，为未来的 VLA 开发和大规模数据挖掘提供了重要启示。

大规模数据整理和高效训练代码库



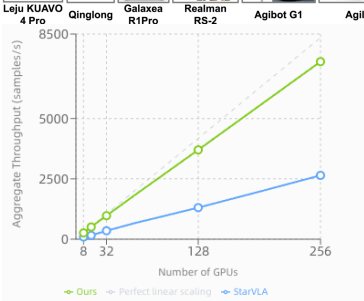


Figure 5. Training throughput analysis of the Qwen2.5-VL-3B-π model.

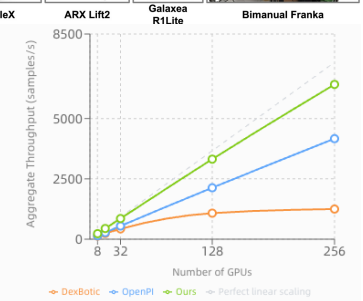


Figure 6. Training throughput analysis of the PaliGemma-3B-pt-224-π model.

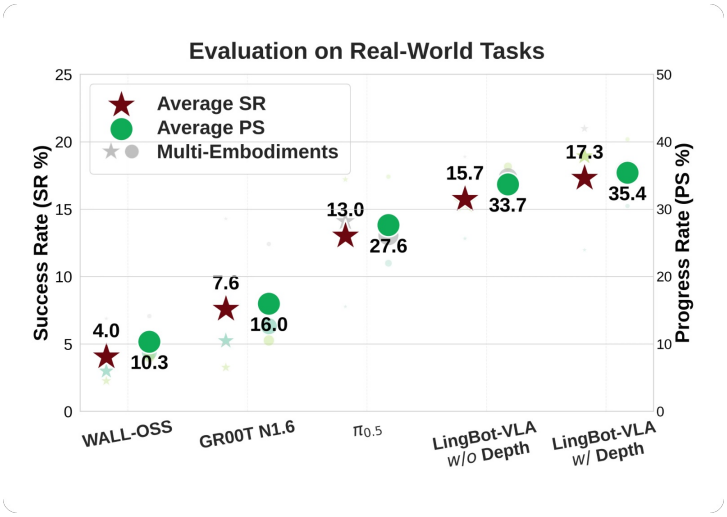
团队仔细构造了 20,000 小时的真实机器人训练数据，涵盖了 9 种主流的双臂机器人构型。为了进行精确的数据标注，数据里的视频由人工标注者按原子动作进行切分，并用大模型标注视频对应任务和子任务。在 codebase 的开发中，适配了 Fully Sharded Data Parallel (FSDP) 分布式、混合精度、算子融合等优化。高质量的海量数据和高效的训练 codebase 是 LingBot-VLA 优越性能的基础。

大规模多构型真机测试



团队在多构型机器人（Agibot G1, AgileX, Galaxea R1Pro）上进行了大规模真机实验验证：

每个模型在各构型100个任务上进行评测，各任务分别采集130条轨迹用于模型训练。实验结果表明LingBot-VLA模型对比现有VLA模型表现出明显的优势，展现了LingBot-VLA强大的准确性和泛化性。



深度信息辅助的机器人操作

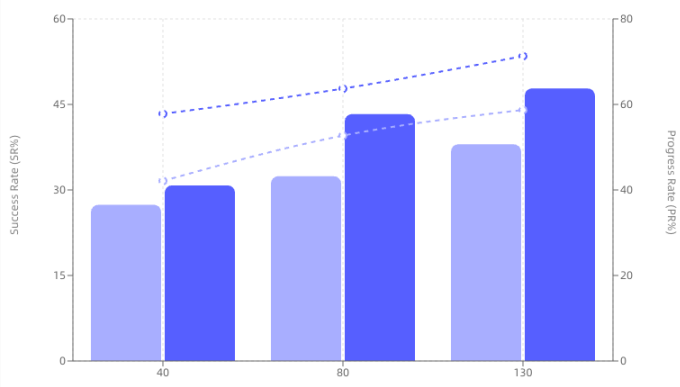
真机实验结果 |Platform|π0.5||Ours w/o depth||Ours w/ depth|| |-|-|-|-|-|-| ||SR|PS|SR|PS|SR|PS| |Agibot G1|7.77%|21.98%|12.82%|30.04%|11.98%|30.47%| |AgileX|17.20%|34.82%|15.50%|36.31%|18.93%|40.36%| |Galaxea R1Pro|14.10%|26.14%|18.89%|34.71%|20.98%|35.40%| |Average|13.02%|27.65%|15.74%|33.69%|17.30%|35.41%|

仿真实验结果 (a). Clean Scenes ||π0.5|Ours w/o depth|Ours w/ depth| |-|-|-| |Average SR|82.74%|86.50%|88.56%|

(b). Randomized Scenes ||π0.5|Ours w/o depth|Ours w/ depth| |-|-|-| |Average SR|76.76%|85.34%|86.68%|

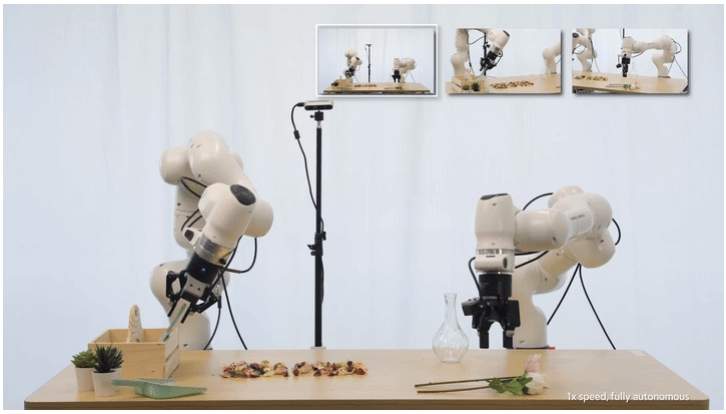
为了显式捕捉操控环境中的空间感知能力，并进一步提升机器人执行的鲁棒性，团队采用了一种基于查询向量（query）的深度蒸馏方法。具体而言，团队引入了与三视角操作图像相对应的可学习 queries，这些 queries 经 VLM 处理后，与 LingBot-Depth 输出的 depth embeddings 进行对齐。这种对齐机制在维持模型训练与推理的效率的同时，有效将深度信息集成到 LingBot-VLA 中。在真实机器人平台和仿真环境下进行的广泛实验证明，深度信息的融入提升了 LingBot-VLA 的操控性能。

机器人操作任务的高效迁移



得益于涵盖主流构型和详尽任务的大规模预训练，LingBot-VLA 具备强大的通用操控能力，并且能够将其高效迁移到多样的下游机器人任务中。实验表明，LingBot-VLA 在下游任务中能够使用更少的数据，达到超越 π0.5 的性能；并且性能优势会随着数据量的增加而持续扩大。

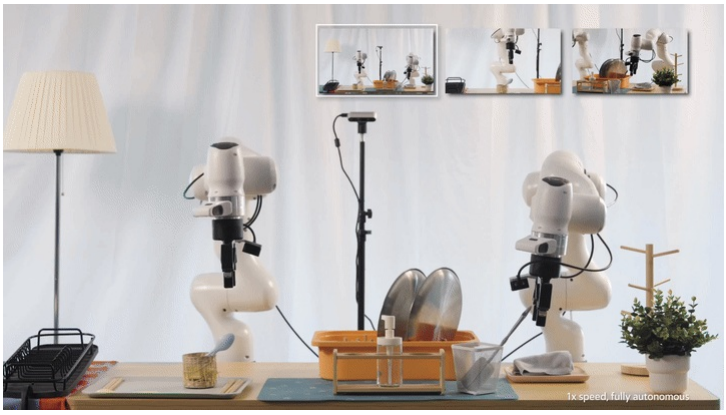
与透明物体交互

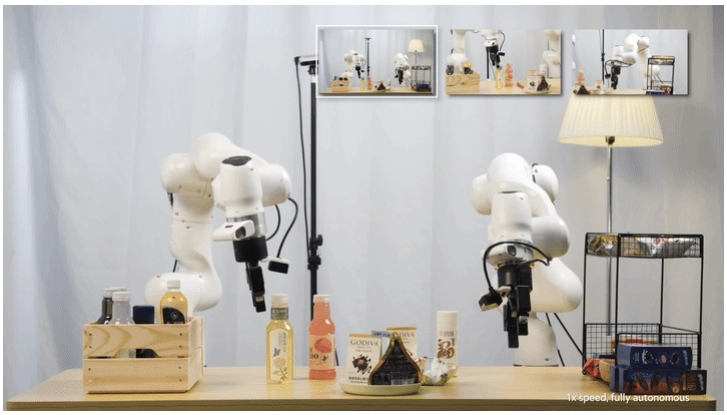


引入深度信息后，团队的模型可以更好得感知场景内透明物体，如玻璃花瓶，并依照指令“插花”，完成指定动作。

复杂精细化操作







经过快速微调后，团队的模型可以完成相对复杂，包含较多精细化操作的指令，比如清洁餐具，并将餐具摆放到指定位置

在百舸平台上快速开始

部署环境要求

	部署要求	最佳实践
CPU	32核	64核
内存	320G	建议按表单默认值及以上
GPU	A800 * 4	本示例中使用8卡A800
CDS	按需	按需
其它	无	无

镜像

请使用百舸官方镜像：ccr-2hpgw1bh-pub.cnc.bj.baidubce.com/aihc-private/lingbot-vla:v0.1-0312-main-16ac002

本镜像使用基础镜像：registry.baidubce.com/inference/aibox-cuda:v2.0-cu12.8-cudnn9-ubuntu22.04

如果不使用本示例提供的lingbot-vla:v0.1-0312-main-16ac002镜像，安装方法参考下文【依赖配置过程参考】

依赖配置过程参考（镜像中已完成配置）

```
 ##### 安装 PyTorch 2.8.0
pip install torch==2.8.0 torchvision==0.23.0 torchaudio==2.8.0 --index-url https://download.pytorch.org/whl/cu128

##### 克隆 LeRobot
GIT_LFS_SKIP_SMUDGE=1 git clone https://github.com/huggingface/lerobot.git
cd lerobot
git checkout 0cf864870cf29f4738d3ade893e6fd13fbd7cdb5
pip install -e .

##### 安装 flash attention（需提前下载 wheel 文件）
pip install /path/to/flash_attn-2.8.3+cu12torch2.8cxx11abiTRUE-cp312-cp312-linux_x86_64.whl

##### 克隆 lingbot-vla
git clone https://github.com/robbyant/lingbot-vla.git
cd lingbot-vla/
git submodule update --remote --recursive
pip install -e .
pip install -r requirements.txt

##### 安装 LingBot-Depth 依赖
cd ./lingbotvla/models/vla/vision_models/lingbot-depth/
pip install -e . --no-deps
cd ../MoGe
pip install -e .
```

获取基座模型checkpoint

【方式1】从百度BOS获取

拷贝模型

```
 !set -e && \
wget -q https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.10.zip && \
unzip -q linux-bcecmd-0.5.10.zip && \
chmod +x linux-bcecmd-0.5.10/bcecmd && \
mv linux-bcecmd-0.5.10/bcecmd /usr/local/bin/bcecmd && \
rm -rf linux-bcecmd-0.5.10 linux-bcecmd-0.5.10.zip

!bcecmd -v
```

```
 bcecmd bos cp -r bos:/aibox-private/LingBot/LingBot-VLA/Qwen2.5-VL-3B-Instruct/ ~/workspace/lingbot-vla/Qwen2.5-VL-3B-Instruct
bcecmd bos cp -r bos:/aibox-private/LingBot/LingBot-VLA/lingbot-vla-4b-depth/ ~/workspace/lingbot-vla/lingbot-vla-4b-depth
bcecmd bos cp -r bos:/aibox-private/LingBot/LingBot-VLA/lingbot-vla-4b/ ~/workspace/lingbot-vla/lingbot-vla-4b

cd ~/workspace/lingbot-va
```

【方式2】从HuggingFace获取

```
##### 配置网络代理
export HF_ENDPOINT=https://hf-mirror.com
export http_proxy=http://<你使用的网络代理>
export https_proxy=http://<你使用的网络代理>

##### 下载 VLA 模型
python3 scripts/download_hf_model.py --repo_id robbyant/lingbot-vla-4b --local_dir .

##### 下载带深度的 VLA 模型
python3 scripts/download_hf_model.py --repo_id robbyant/lingbot-vla-4b-depth --local_dir .

##### 下载 Qwen2.5-VL-3B-Instruct（Tokenizer 需要）
python3 scripts/download_hf_model.py --repo_id Qwen/Qwen2.5-VL-3B-Instruct --local_dir .
```

训练命令

无深度版本

```
cd ~/workspace/lingbot-vla

##### 配置网络代理（可选）
export http_proxy=http://10.0.7.40:18000
export https_proxy=http://10.0.7.40:18000
export HF_ENDPOINT=https://hf-mirror.com

##### 运行训练
bash train.sh tasks/vla/train_lingbotvla.py ./configs/vla/robotwin_load20000h.yaml \
 --model.model_path ./lingbot-vla-4b \
 --data.train_path lerobot/aloha_static_fork_pick_up \
 --train.output_dir /root/workspace/output/lingbot_robotwin5tasks \
 --model.tokenizer_path ./Qwen2.5-VL-3B-Instruct \
 --train.micro_batch_size 1 \
 --train.global_batch_size 8
```

有深度版本

```
cd ~/workspace/lingbot-vla

##### 配置网络代理（可选）
export http_proxy=http://10.0.7.40:18000
export https_proxy=http://10.0.7.40:18000
export HF_ENDPOINT=https://hf-mirror.com

##### 运行训练
bash train.sh tasks/vla/train_lingbotvla.py ./configs/vla/robotwin_load20000h_depth.yaml \
 --model.model_path /root/workspace/lingbot-vla/lingbot-vla-4b-depth \
 --data.train_path lerobot/aloha_static_fork_pick_up \
 --train.output_dir /root/workspace/output/lingbot_depth_robotwin5tasks \
 --model.tokenizer_path ./Qwen2.5-VL-3B-Instruct \
 --model.moge_path /root/workspace/lingbot-vla/lingbot-vla-4b-depth/moge2-vitb-normal.pt \
 --model.morgbd_path /root/workspace/lingbot-vla/lingbot-vla-4b-depth/LingBot-Depth \
 --train.micro_batch_size 1 \
 --train.global_batch_size 8
```

参数说明 |参数|说明|示例值| |-|-|-| |`--model.model_path`|预训练 VLA 模型路径|`/root/workspace/lingbot-vla/lingbot-vla-4b`| |`--model.tokenizer_path`|Qwen2.5-VL-3B-Instruct 路径|`/root/workspace/lingbot-vla/Qwen2.5-VL-3B-Instruct`| |`--data.train_path`|训练数据路径|`path/to/mixed_robotwin_5tasks`| |`--train.output_dir`|输出目录|`/root/workspace/output/lingbot_robotwin5tasks`| |`--train.micro_batch_size`|单卡 batch size|1| |`--train.global_batch_size`|全局 batch size|8（8卡）| |`--model.moge_path`|MoGe 模型路径（深度版本）|`.../lingbot-vla-4b-depth/moge2-vitb-normal.pt`| |`--model.morgbd_path`|LingBot-Depth 路径（深度版本）|`.../lingbot-vla-4b-depth/LingBot-Depth`|

评估

```
##### robotwin2.0 评估
export QWEN25_PATH=./Qwen2.5-VL-3B-Instruct
python -m deploy.lingbot_robotwin_policy \
 --model_path /root/workspace/output/lingbot_depth_robotwin5tasks \ # 填写你需要评估的模型的路径
 --use_length 50 \
 --port port
```

目录结构

```
lingbot-vla/
├── lingbot-vla-4b/          # VLA 模型（无深度）
├── lingbot-vla-4b-depth/    # VLA 模型（有深度）
├── Qwen2.5-VL-3B-Instruct/  # Tokenizer
├── configs/vla/             # 训练配置
│   ├── robotwin_load20000h.yaml
│   └── robotwin_load20000h_depth.yaml
├── tasks/vla/               # 训练脚本
│   └── train_lingbotvla.py
├── deploy/                  # 部署脚本
├── scripts/                 # 工具脚本
└── train.sh                 # 训练入口脚本
```

LingBot-VA训推

> https://technology.robbyant.com/lingbot-va lingbo-va——通用机器人控制的因果视频动作世界模型

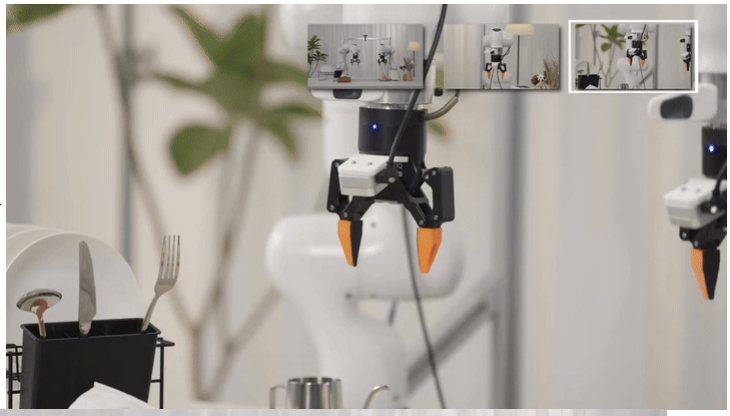
模型概述

LingBot-VA 是一个自回归框架，它将视频世界建模与策略学习统一起来，联合学习未来帧预测和动作执行。该模型在长程操作、数据高效的后训练以及对新配置的鲁棒泛化方面展现出显著潜力。

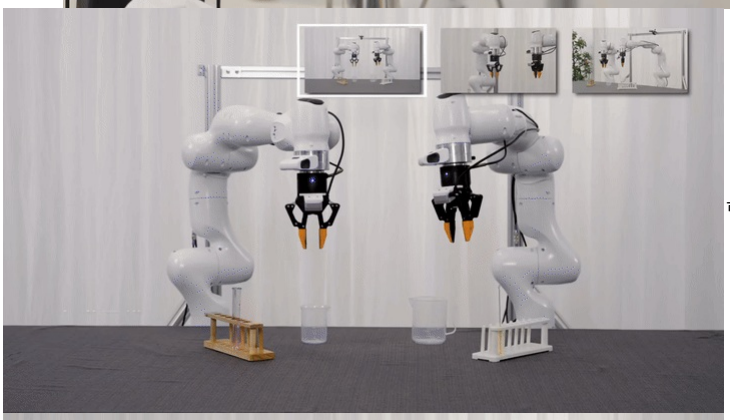
它是一个多功能的全能型模型，在各种场景下都表现出色——从长程任务到高精度控制，再到可变形物体和关节物体的操作。

任务示例 |类别|任务| |-|-| |**长程任务**|制作早餐、拆快递| |**高精度任务**|插入试管、拾取螺丝| |**可变形物体**|叠衣服、开抽屉|

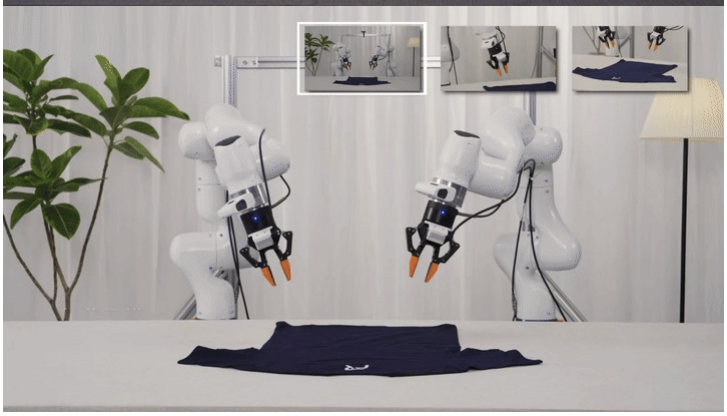
长程任务



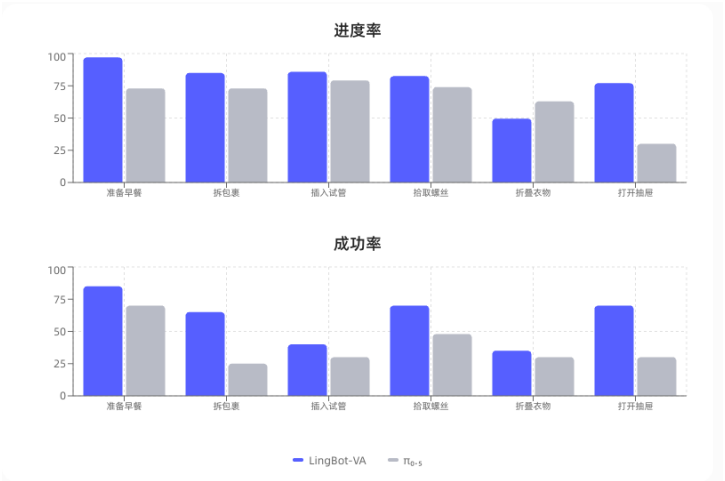
高精度任务



可变形物体任务

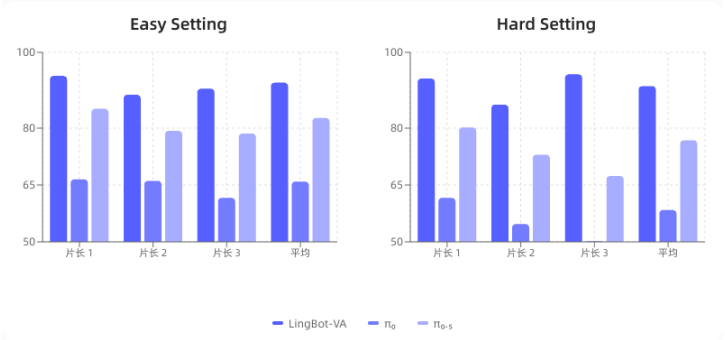


性能结果



仿真基准测试

LingBot-VA 在两个仿真基准测试上进行了评估：RoboTwin 2.0 和 LIBERO。这些测试涵盖多种机器人构型的各种操作任务。结果表明，该模型相较于现有最先进的方法具有一致的改进。



RoboTwin 2.0 简单设置 |视野|1|2|3|平均| |-|-|-|-| |LingBot-VA|50|65|80|100| |π_0|50|65|80|100| |$\pi_{0.5}$|50|65|80|100|

困难设置 |视野|1|2|3|平均| |-|-|-|-| |LingBot-VA|50|65|80|100| |π_0|50|65|80|100| |$\pi_{0.5}$|50|65|80|100|

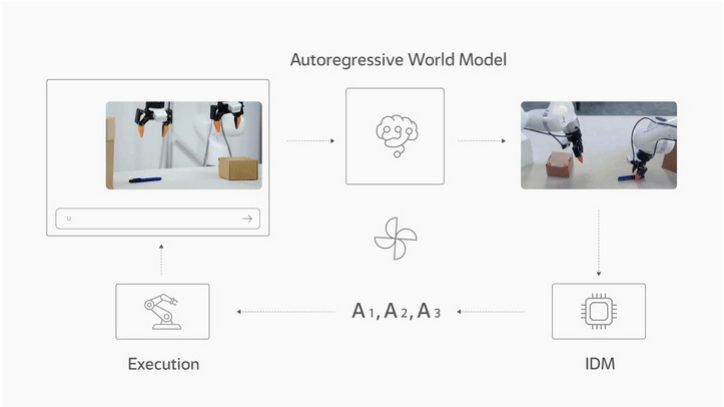
技术原理 LingBot-VA 是一个自回归扩散框架，在架构上将视觉动力学预测和动作推断统一在单个交错序列中。这使得机器人能够同时推理未来状态并执行精确的闭环控制。

大规模预训练 LingBot-VA 在大规模机器人视频动作数据集上进行预训练，学习丰富的视觉动力学，为理解物理世界的演变和在其中运行建立了坚实基础。



框架架构 该框架分为三个阶段运行：

1. **自回归视频生成**：根据当前观察和语言指令预测未来帧
2. **逆动力学模型 (IDM)**：从预测的视频中解码出动作
3. **闭环控制**：执行后，用真实观察替换视频 KV-cache，将视频动作模型锚定在实际结果中



逆动力学模型 逆动力学模型 (IDM) 能够准确地从预测视频中解码动作，并在不同的环境和机器人构型中表现出良好的泛化能力。

预测与现实对比 预测的视频与机器人实际执行过程中观察到的视频高度一致。执行解码后的动作产生的真实观察与预测帧相符。

在百舸平台上快速开始

环境要求

- Python 3.10.16
- PyTorch 2.9.0
- CUDA 12.6
- GPU（推理：单卡 RTX 4090 即可；训练：需要 8 卡 A800）

镜像：ccr-registry.baidubce.com/aihc/lingbot-va:v0.2.3

数据准备

拷贝模型

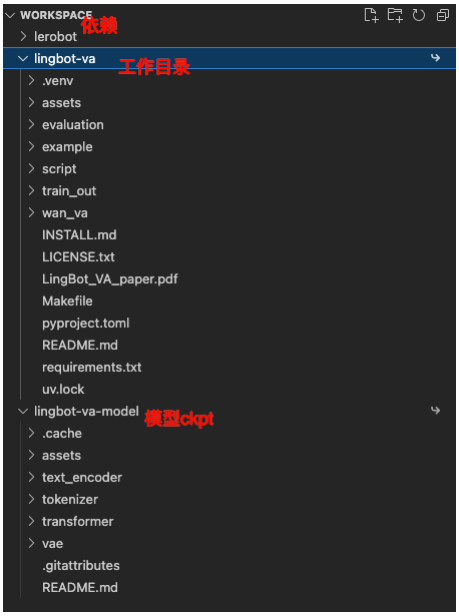
```
!set -e && \
wget -q https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.10.zip && \
unzip -q linux-bcecmd-0.5.10.zip && \
chmod +x linux-bcecmd-0.5.10/bcecmd && \
mv linux-bcecmd-0.5.10/bcecmd /usr/local/bin/bcecmd && \
rm -rf linux-bcecmd-0.5.10 linux-bcecmd-0.5.10.zip

!bcecmd -v
```

```
bcecmd bos cp -r bos:/aihc-models-bj/robbyant/lingbot-va-base ~/workspace/lingbot-va-model

cd ~/workspace/lingbot-va
```

工作目录结构如下：



启动Python虚拟环境

```
source /root/workspace/lingbot-va/.venv/bin/activate
```

环境配置

> 运行 LingBot-VA 所需的环境已在镜像中完成配置

模式	attn_mode 值	说明
训练	"flex"	训练必需
推理/评估	"torch" 或 "flashattn"	推理必需，flex 模式会报错

修改方法：编辑模型目录下的 transformer/config.json，找到 "attn_mode" 字段修改。

安装依赖

> 用户无需执行，仅做参考

```
##### 安装 uv
pip install uv

##### 创建虚拟环境
uv venv /root/workspace/lingbot-va/.venv
source /root/workspace/lingbot-va/.venv/bin/activate

##### 安装 PyTorch 2.9.0
uv pip install torch==2.9.0 torchvision==0.24.0 torchaudio==2.9.0 --index-url https://download.pytorch.org/whl/cu126

##### 安装依赖
uv pip install diffusers==0.36.0 transformers==4.55.2 accelerate einops easydict tqdm \
  'imageio[ffmpeg]' websockets msgpack opencv-python matplotlib ftfy safetensors Pillow


##### 安装 flash-attn（编译时间较长）
uv pip install wheel
uv pip install flash-attn --no-build-isolation
```

推理测试

1. 修改模型配置 模型需要设置为推理模式：

```
##### 编辑 transformer config
vim /root/workspace/lingbot-va-model/transformer/config.json

##### 将 "attn_mode": "flex" 改为 "attn_mode": "torch"
```

2. 修改配置文件 更新配置文件中的模型路径：

```
##### 编辑配置
vim /root/workspace/lingbot-va/wan_va/configs/va_demo_cfg.py

##### 修改模型路径
wan22_pretrained_model_name_or_path = "/root/workspace/lingbot-va-model"
```

3. 运行推理

```
 cd /root/workspace/lingbot-va
source .venv/bin/activate

##### 单 GPU 推理
python -m torch.distributed.run \
    --nproc_per_node=1 \
    --master_addr=127.0.0.1 \
    --master_port=29501 \
    -m wan_va.wan_va_server \
    --config-name demo_i2av
```

推理输出示例

```
 Loading checkpoint shards: 100%|■■■■■■■■■■■■| 3/3 [00:00<00:00, 109.97it/s]
USE I2AV mode
FRAME START ID: 0
 17%|■▌         | 1/6 [00:00<00:03,  1.37it/s]
...
100%|■■■■■■■■■■■■| 6/6 [00:01<00:00,  4.28it/s]
```

训练测试

配置wandb（可选） 如需使用 wandb 记录训练过程：

```
 export WANDB_API_KEY="your-key"
export WANDB_BASE_URL="https://api.wandb.ai"
export WANDB_TEAM_NAME="your-team"
export WANDB_PROJECT="your-project"
```

运行训练

```
 cd /root/workspace/lingbot-va
source .venv/bin/activate

##### 8 GPU 训练
python -m torch.distributed.run --nproc_per_node=8 --master_addr=127.0.0.1 --master_port=29501 -m wan_va.train --config-name robotwin_train
```

训练配置参数 |参数|说明|示例值| |-|-|-| |`--config-name`|配置名称|`robotwin_train`, `demo_train`| |NGPU|GPU 数量|8| |`dataset_path`|训练数据路径|`/path/to/dataset`| |`batch_size`|Batch size|1| |`num_steps`|训练步数|2000|

目录结构

```
/root/workspace/lingbot-va/
├── .venv/                  # 虚拟环境
├── lingbot-va-model/       # 模型文件（约 23GB）
│   ├── transformer/        # Transformer 模型
│   ├── vae/                # VAE 模型
│   ├── text_encoder/       # 文本编码器
│   └── tokenizer/          # Tokenizer
├── wan_va/                 # 源代码
│   ├── configs/            # 配置文件
│   ├── modules/            # 模型模块
│   ├── distributed/        # 分布式训练
│   ├── dataset/            # 数据集处理
│   └── train.py            # 训练入口
├── script/                 # 训练脚本
│   ├── run_va_posttrain.sh
│   └── run_launch_va_server_sync.sh
└── example/                # 示例数据
```

常见问题

1. FSDP dtype 不一致错误

```
AssertionError: FSDP expects uniform original parameter dtype but got {torch.float32, torch.bfloat16}
```

解决：确保使用 PyTorch 2.9.0 和 CUDA 12.6

2. 推理提示 attn_mode 错误

```
Unsupported attention mode: flex, only support torch and flashattn
```

解决：将模型 `transformer/config.json` 中的 `attn_mode` 改为 `"torch"`或`"flashattn"`

3. 单 GPU 训练 OOM 训练需要约 8 卡 A800 才能运行，单卡会 OOM。推理可以在单卡 RTX 4090 上运行。

参考资料

- 技术报告
- 代码仓库
- Hugging Face
- ModelScope

GEAR-SONIC在开发机中训练

一、产品简介

GEAR-SONIC 是一个基于强化学习的通用人形机器人全身运动控制框架，能够让人形机器人实现多种运动模式（行走、跑步、跳跃等）的统一控制。该方法利用大规模人体运动数据进行训练，通过 token 化的动作表

示和 Transformer 架构，使机器人能够泛化到多样化的运动技能。

二、快速开始

环境准备与开发机创建 推荐硬件配置 |配置项|最低要求|推荐配置| |-|-| |GPU|1 *L20 或更高规格的加速卡*|8 L20 或更高规格的加速卡| |内存|32GB|64GB+| |CPU|8 核|16 核+| |存储|50GB+|100GB+（含数据集）|

1.1 创建开发机 打开百舸控制台 → 进入开发机页面 → 点击【创建实例】，按以下信息配置：

基本信息：

配置项	填写说明
实例名称	自定义命名
资源池类型/资源池/队列	根据您已有的资源池类型选择
资源规格	按上方推荐硬件配置选择
云磁盘	推荐预留 50GiB，保证运行空间

1.2 登录开发机 创建成功后，在开发机列表中找到刚创建的实例，等待状态变为「运行中」后，点击【登录】进入终端，打开目录下预置 notebook。

> **提示**：如果开发机状态长时间处于「创建中」，请检查资源池是否有可用资源。 **1.3 开始操作** 首先进入项目目录：

```
cd /root/workspace/GR00T-WholeBodyControl
```

第一步：下载资源 使用 `bcecmd` 从 BOS 同步模型权重和数据：

```
bcecmd bos sync bos:/aihc-models-bj/nvidia/GEAR-SONIC ../GEAR-SONIC/
```

第二步：验证下载文件

```
ls ../GEAR-SONIC/
ls ../GEAR-SONIC/sonic_release/last.pt
```

第三步：启动训练（单卡） 使用单卡运行训练脚本，适用于快速验证和调试。

```
/workspace/isaaclab/isaaclab.sh -p \
  gear_sonic/train_agent_trl.py \
  +exp=manager/universal_token/all_modes/sonic_release \
  +checkpoint=/root/workspace/GEAR-SONIC/sonic_release/last.pt \
  num_envs=256 headless=True \
  ++manager_env.commands.motion.motion_lib_cfg.motion_file=/root/workspace/GEAR-SONIC/sample_data/robot_filtered \
  ++manager_env.commands.motion.motion_lib_cfg.smpl_motion_file=/root/workspace/GEAR-SONIC/data/smpl_filtered \
  algo.config.num_learning_iterations=100
```

单卡预期结果：

指标	预期值
Iteration time	~2.5s
steps/s	~2,500

第三步：启动训练（八卡） 使用 8 卡并行训练，通过 `accelerate` 启动多进程分布式训练，适用于大规模正式训练场景。相比单卡，环境数从 256 提升到 4096，训练迭代次数从 100 提升到 1000。

```
/workspace/isaaclab/isaaclab.sh -p -m accelerate.commands.launch --num_processes=8 \
  gear_sonic/train_agent_trl.py \
  +exp=manager/universal_token/all_modes/sonic_release \
  +checkpoint=/root/workspace/GEAR-SONIC/sonic_release/last.pt \
  num_envs=4096 headless=True \
  ++manager_env.commands.motion.motion_lib_cfg.motion_file=/root/workspace/GEAR-SONIC/sample_data/robot_filtered \
  ++manager_env.commands.motion.motion_lib_cfg.smpl_motion_file=/root/workspace/GEAR-SONIC/sample_data/smpl_filtered \
  algo.config.num_learning_iterations=1000
```

八卡预期结果：

指标	预期值
Iteration time	~4.9 - 5.0s
Computation (steps/s)	~158,000 - 161,000
Collection time	~2.6 - 2.7s
Learning time	~2.2s
GPU 显存	~14 - 15 GB / 24 GB
GPU 利用率	~80 - 85%
Mean rewards (iter 10)	~19（从 checkpoint finetune）
Mean episode length (iter 10)	~186 steps

第三步：启动训练（加速版） 对于适配了指定显卡的加速镜像，您可以通过以下脚本启动加速训练：

```
#!/usr/bin/env bash
##### 8-GPU single-node SONIC training under Isaac Sim 6.0 + Isaac Lab 3.0, NEWTON backend.
##### Mirrors train_sonic_8gpu_sim6.sh (PhysX) exactly except physics_backend=newton,
##### so Collection-phase throughput is directly comparable.
set -euo pipefail

export OMNI_KIT_ACCEPT_EULA=YES
export TRL_EXPERIMENTAL_SILENCE=1
export VIRTUAL_ENV=/opt/sim6-venv
export PATH=/opt/sim6-venv/bin:$PATH

cd /root/workspace/GR00T-WholeBodyControl

/opt/sim6-venv/bin/python -m accelerate.commands.launch --num_processes=8 \
  gear_sonic/train_agent_trl.py \
  +exp=manager/universal_token/all_modes/sonic_release \
  +checkpoint=/workspace/data/sonic_release/last.pt \
  num_envs=4096 headless=True \
  ++manager_env.config.physics_backend=newton \
  ++manager_env.commands.motion.motion_lib_cfg.motion_file=/workspace/data/sample_data/robot_filtered \
  ++manager_env.commands.motion.motion_lib_cfg.smpl_motion_file=/workspace/data/data/smpl_filtered \
  algo.config.num_learning_iterations=1000 \
  use_wandb=false
```

八卡预期结果：

指标	预期值
Iteration time	~1.92s
Collection time	~1.00s
Learning time	~0.92s
Steps/s	~410,000

附录：按需补充资源

A. Policy ONNX（仅部署推理时需要） HuggingFace 开源地址：https://huggingface.co/nvidia/GEAR-SONIC

```
pip install huggingface_hub[cli]

hf download nvidia/GEAR-SONIC \
  model_encoder.onnx \
  model_decoder.onnx \
  observation_config.yaml \
  --local-dir gear_sonic_deploy
```

B. Bones-SEED 完整训练数据集

1. 申请访问（gated，需同意 license）：https://huggingface.co/datasets/bones-studio/seed
2. 下载 g1.tar.gz（23.5 GB）
3. 转换：

```
python gear_sonic/data_process/convert_soma_csv_to_motion_lib.py \
  --input /path/to/bones_seed/g1/csv/ \
  --output data/motion_lib_bones_seed/robot \
  --fps 30 --fps_source 120 --individual --num_workers 16
```

4. 过滤：

```
python gear_sonic/data_process/filter_and_copy_bones_data.py \
  --source data/motion_lib_bones_seed/robot \
  --dest data/motion_lib_bones_seed/robot_filtered
```

5. 训练时替换 `motion_file` 路径为 `data/motion_lib_bones_seed/robot_filtered`

DreamZero在开发机中训练

一、产品简介

DreamZero 是一个基于视频生成模型（Wan2.1-I2V-14B-480P）的具身智能训练框架，支持机器人操作任务的视觉-语言-动作端到端训练。该框架通过预计算 T5 Prompt Embedding 缓存、`torch.compile` 全量编译和 DeepSpeed ZeRO-2 优化，实现高吞吐量的大规模分布式训练。DreamZero 支持 DROID 标准数据集与自定义 LeRobot 格式数据集，可直接用于 DROID 推理，也可在新本体（如 AgiBot）上进行微调。

二、快速开始

环境准备与开发机创建 推荐硬件配置 |配置项|最低要求|推荐配置| |-|-| |GPU 数量|4 卡|**8 卡**| |GPU 型号|拥有80G显存显卡 *4/拥有80G显存显卡* 8| |CPU|80 核|180 核| |主机内存|512 GB|1 TB+| |存储|建议不少于 500G|建议不少于 500G|

> **存储说明**：14B 模型首次加载约需 10–15 分钟，第二次命中 page cache 后可秒级加载。推荐将 `checkpoints/` 目录放在共享存储上。 **1.1 创建开发机** 打开百舸控制台 → 进入开发机页面 → 点击【创建实例】，按以下信息配置：

基本信息：

配置项	填写说明
实例名称	自定义命名
资源池类型 / 资源池 / 队列	根据已有资源池类型选择
资源规格	按上方推荐硬件配置选择（建议 8 卡）
云磁盘	推荐预留 500 GiB 以上，保证模型权重与数据集存储空间

1.2 登录开发机 创建成功后，在开发机列表中找到刚创建的实例，等待状态变为「运行中」后，点击【登录】进入终端。并在终端进入项目目录：

```
cd /workspace/dreamzero
```

> **提示**：如果开发机状态长时间处于「创建中」，请检查资源池是否有可用资源。

第一步：下载模型权重 创建 checkpoints 目录，使用 `bcecmd` 从 BOS 同步所有权重：

```
##### 创建目录
mkdir -p checkpoints

##### 下载 DreamZero-DROID（用于推理）
bcecmd bos sync bos://aihc-models-bj/GEAR-Dreams/DreamZero-DROID/ ./checkpoints/DreamZero-DROID/

##### 下载 DreamZero-AgiBot（用于在新本体上进行微调）
##### YAM 和 AgiBot 的训练脚本默认使用 pretrained_model_path=./checkpoints/DreamZero-AgiBot
bcecmd bos sync bos://aihc-models-bj/GEAR-Dreams/DreamZero-AgiBot/ ./checkpoints/DreamZero-AgiBot/

##### 下载预训练基础模型权重
##### DreamZero 建立在 Wan2.1-I2V-14B-480P 之上，并使用 umt5-xxl 分词器
bcecmd bos sync bos://aihc-models-bj/Wan-AI/Wan2.1-I2V-14B-480P/ ./checkpoints/Wan2.1-I2V-14B-480P/
bcecmd bos sync bos://aihc-models-bj/google/umt5-xxl/ ./checkpoints/umt5-xxl/
```

> **提示**：训练脚本在配置路径下未找到权重时会自动触发下载，但建议预先下载以避免启动时的额外延迟。

第二步：准备数据集 DreamZero 使用 LeRobot 格式数据集，推荐目录结构如下：

```
datasets/
└── myrobot/
    ├── data/        # 训练样本 parquet
    ├── videos/      # MP4 视频
    └── meta/
```

从 BOS 下载 DROID 标准数据集：

```
bcecmd bos sync bos:/aihc-rdw-bj/huggingface.co/datasets/GEAR-Dreams/DreamZero-DROID-Data  ./data/droid_lerobot/
```

下载完成后，目录结构应为：

```
./data/droid_lerobot/
├── data/                                # 训练样本 parquet
├── meta/
│   ├── info.json
│   ├── stats.json
│   ├── modality.json                    # 必需
│   ├── relative_stats.json
│   ├── relative_horizon_stats_dreamzero.json
│   └── tasks.jsonl
└── videos/                              # MP4 视频
```

第三步：构建 T5 Prompt Embedding 缓存（推荐） UMT5-XXL（5.6B）在训练时被冻结，但会常驻每张 GPU 约 11 GB 显存。**预先离线计算 Prompt Embedding 并存入 LMDB**，训练时直接查表替代 forward 推理，可腾出 11 GB 显存用于将 AdamW 优化器状态从 CPU 搬回 GPU，**提升约 20.5% 的步进吞吐量**。

一次性构建（单卡，约 3 分钟）：

```
python scripts/encode_t5_offline.py \
  --droid-data-root ./data/droid_lerobot \
  --t5-checkpoint ./checkpoints/Wan2.1-I2V-14B-480P/models_t5_umt5-xxl-enc-bf16.pth \
  --tokenizer-path ./checkpoints/umt5-xxl \
  --output-dir ./caches/t5_droid_full
```

构建完成后，目录结构如下：

```
./caches/t5_droid_full/
├── data/
│   ├── data.mdb       # ~2.32 GB
│   └── lock.mdb
└── meta.json
```

> **常见报错**：若出现 `lmdb.MapFullError: MDB_MAP_FULL`，追加参数 `--map-size-gb 1024` 重新执行。

第四步：启动训练 最小启动命令

```
export DREAMZERO_T5_CACHE_PATH=./caches/t5_droid_full
bash scripts/train/droid_training_full_finetune_optimization.sh
```

> 不设置 `DREAMZERO_T5_CACHE_PATH` 时，launcher 自动回退到 CPU offload + live T5 forward 安全路径，功能可用但吞吐量较低。**自定义配置（路径 / GPU 数 / Batch Size）**

```
export DROID_DATA_ROOT="./data/droid_lerobot"
export OUTPUT_DIR="./checkpoints/dreamzero_droid_full"
export NUM_GPUS=8            # 默认 8，已验证最优
export BS=2                  # BS=3/4 在 96 GiB 显卡上 OOM
export WAN_CKPT_DIR="./checkpoints/Wan2.1-I2V-14B-480P"
export TOKENIZER_DIR="./checkpoints/umt5-xxl"
export DREAMZERO_T5_CACHE_PATH=./caches/t5_droid_full

bash scripts/train/droid_training_full_finetune_optimization.sh
```

launcher 内置优化项说明 |环境变量|默认值|作用| |-|-|-| | OMP_NUM_THREADS|20|提升 DeepSpeed CPU Adam 多线程效率，节省约 2.5 s/step| |DREAMZERO_COMPILE|1|对 DiT 40 个 block 全量执行 torch.compile| |DREAMZERO_COMPILE_MODE|default|编译模式（不可使用 reduce-overhead）| |TORCHINDUCTOR_CACHE_DIR|./inductor_cache|持久化编译缓存，冷启动从 4 分钟降至 30 秒|

DeepSpeed 配置自动选择规则 |DREAMZERO_T5_CACHE_PATH|DS 配置|行为| |-|-|-| |已设置| zero2_no_offload_optimized.json|优化器状态保留在 GPU（吞吐量更高）| |未设置 / 为空|zero2_offload_optimized.json|优化器状态 offload 到 CPU（兜底模式）|

三、常见问题

Q：执行 nvidia-smi 报错 couldn't find libnvidia-ml.so ？

执行以下命令修复软链接：

```
rm -f /usr/lib/x86_64-linux-gnu/libnvidia-ml.so
rm -f /usr/lib/x86_64-linux-gnu/libnvidia-ml.so.1
ln -s /usr/lib/x86_64-linux-gnu/libnvidia-ml.so.580.126.20 /usr/lib/x86_64-linux-gnu/libnvidia-ml.so
ln -s /usr/lib/x86_64-linux-gnu/libnvidia-ml.so.580.126.20 /usr/lib/x86_64-linux-gnu/libnvidia-ml.so.1
ldconfig
```

Q：T5 缓存构建时报 MDB_MAP_FULL ？

追加 --map-size-gb 1024 参数后重新执行构建命令即可。

Q：训练启动后 inductor 编译耗时过长？

首次冷启动约需 4 分钟用于 torch.compile 编译，编译结果会缓存至 ./inductor_cache ，后续启动约 30 秒即可完成。

Q：使用 AgiBot 本体进行微调，应选哪个权重？

使用 DreamZero-AgiBot 权重（./checkpoints/DreamZero-AgiBot），YAM 和 AgiBot 训练脚本默认读取此路径，详见 pretrained_model_path 配置项。

Q：数据集中缺少 modality.json 文件训练报错？

modality.json 为必需文件，请确认数据集 meta/ 目录下包含此文件。如使用自定义数据集，可参考 docs/DROID_CONVERSION.md 进行格式转换。

WBC-AGILE在开发机中训练

一、产品简介

WBC-AGILE（AGILE: A Generic Isaac-Lab based Engine for humanoid loco-manipulation learning）是一个基于 NVIDIA Isaac Lab 的人形机器人全身控制（Whole-Body Control）强化学习框架，可用于训练人形机器人运动控制、移动操作（Loco-Manipulation）、全身协调控制与 Sim-to-Real 控制策略。

WBC-AGILE 支持 Booster T1、Unitree G1 等机器人形态，并提供 Teacher-Student 蒸馏、历史观测、RNN 策略、模型评估、HTML 报告等能力，适合在人形机器人运动控制、全身控制策略验证和仿真到真实迁移等场景中使用。

二、快速开始

环境准备与开发机创建 推荐硬件配置 |配置项|最低要求|推荐配置| |-|-|-| |GPU|1 *L20 或更高规格的加速卡*|8 L20 或更高规格的加速卡| |内存|32GB|64GB+| |CPU|8 核|16 核+| |存储|100GB+|200GB+（含数据集与 USD 资产）|

1.1 创建开发机 打开百舸控制台 → 进入开发机页面 → 点击【创建实例】，按以下信息配置：

基本信息：

配置项	填写说明
实例名称	自定义命名，例如 wbc-agile-dev
资源池类型 / 资源池 / 队列	根据您已有的资源池类型选择
资源规格	按上方推荐硬件配置选择
云磁盘	推荐预留 100GiB 以上，若需要完整数据集与资产，建议 200GiB+

> **提示**：WBC-AGILE 依赖 Isaac Sim / Isaac Lab 运行环境，建议优先选择百舸官方提供的预置镜像，避免用户在开发机内自行安装 Isaac Sim、Isaac Lab 和 CUDA 依赖。

1.2 登录开发机 创建成功后，在开发机列表中找到刚创建的实例，等待状态变为「运行中」后，点击【登录】进入终端。

> **提示**：如果开发机状态长时间处于「创建中」，请检查资源池是否有可用资源，或联系资源池管理员确认队列配额。

1.3 开始操作 进入 WBC-AGILE 项目目录：

```
cd /workspace/agile
```

第一步：下载资源 使用 bcecmd 从 BOS 同步训练所需数据集和 USD 资产。

1. 下载 AMASS_G1 数据集

```
mkdir -p /workspace/datasets

bcecmd bos sync \
  bos:/aihc-rdw-bj/huggingface.co/datasets/fleaven \
  /workspace/datasets/
```

2. 下载 Isaac Sim / Isaac Lab USD 资产

```
mkdir -p /workspace/nucleus_assets

bcecmd bos sync \
  bos:/aihc-rdw-bj/agile/nucleus_assets \
  /workspace/nucleus_assets/
```

USD 资产目录建议保持如下结构：

```
/workspace/nucleus_assets/
└── Isaac/
    ├── Environments/
    ├── Materials/
    ├── Props/
    ├── IsaacLab/
    └── Robots/
```

该目录通常包含：

- G1 Robot
- Booster T1
- Warehouse 场景
- YCB 道具
- DexCube
- Isaac Lab 运行所需资产

> **说明**：原始开源项目文档中同时提到 AMASS_G1 Dataset 和 Retargeted AMASS for Robotics。在快速开始场景中，用户只需要下载百舸提供的 AMASS_G1 数据集即可；Retargeted AMASS for Robotics 是其上游原始数据来源说明，不作为本文的独立下载步骤。百舸官方文档中统一改为 BOS 下载方式，后续实际发布时请以百舸提供的 BOS 路径为准。 **第二步：验证下载文件** 执行以下命令确认数据集与 USD 资产已正确下载：

```
ls /workspace/datasets/AMASS_G1/
ls /workspace/nucleus_assets/Isaac/
```

第三步：配置 Isaac Sim 资产路径 训练前执行以下命令，将 Isaac Sim 默认云端资产路径重定向到本地 BOS 下载的 USD 资产目录：

```
mkdir -p /root/.nvidia-omniverse/config

cat > /root/.nvidia-omniverse/config/user.toml <<'TOML'
[persistent.isaac.asset_root]
cloud = "/workspace/nucleus_assets"
TOML
```

配置完成后，WBC-AGILE 训练脚本会优先从本地目录加载机器人、场景和道具资产，减少外部网络依赖。

第四步：启动训练 WBC-AGILE 训练统一通过 Isaac Lab 启动脚本执行。

单卡快速验证 以下命令用于快速验证训练流程，适合功能验证和调试：

```
/workspace/isaaclab/isaaclab.sh -p scripts/train.py \
  --task Velocity-T1-v0 \
  --max_iterations 5 \
  --num_envs 16 \
  --headless
```

G1 机器人速度跟踪训练

```
/workspace/isaaclab/isaaclab.sh -p scripts/train.py \
  --task Velocity-G1-History-v0 \
  --max_iterations 100 \
  --num_envs 256 \
  --headless
```

Booster T1 速度跟踪训练

```
/workspace/isaaclab/isaaclab.sh -p scripts/train.py \
  --task Velocity-T1-v0 \
  --max_iterations 100 \
  --num_envs 256 \
  --headless
```

Booster T1 起身任务训练 首次运行起身任务时会自动生成 fallen state 缓存：

```
/workspace/isaaclab/isaaclab.sh -p scripts/train.py \
  --task StandUp-T1-v0 \
  --max_iterations 100 \
  --num_envs 256 \
  --headless \
  agent.fallen_state_dataset_cfg.num_spawns_per_level=1 \
  agent.fallen_state_dataset_cfg.fall_duration_s=0.1 \
  agent.fallen_state_dataset_cfg.cache_enabled=False
```

第五步：评估模型 训练完成后，可使用以下命令评估模型效果：

```
/workspace/isaaclab/isaaclab.sh -p scripts/eval.py \
  --task Velocity-T1-v0 \
  --num_envs 32 \
  --checkpoint <checkpoint_path>
```

其中 `<checkpoint_path>` 替换为训练输出目录下的模型权重路径。

三、常用任务列表

3.1 Velocity-G1-History-v0 G1 机器人速度跟踪任务，使用历史观测输入。

```
--task Velocity-G1-History-v0
```

3.2 Velocity-Height-G1-v0 G1 机器人速度 + 高度跟踪任务。

```
--task Velocity-Height-G1-v0
```

3.3 Velocity-Height-G1-Distillation-Recurrent-v0 Teacher-Student 蒸馏任务，Student 使用 RNN 策略。

```
--task Velocity-Height-G1-Distillation-Recurrent-v0
```

3.4 Velocity-Height-G1-Distillation-History-v0 Teacher-Student 蒸馏任务，Student 使用 History Observation。

```
--task Velocity-Height-G1-Distillation-History-v0
```

3.5 Velocity-T1-v0 Booster T1 速度跟踪任务。

```
--task Velocity-T1-v0
```

3.6 StandUp-T1-v0 Booster T1 起身任务。

```
--task StandUp-T1-v0
```

3.7 HeightTracking-G1-v0 G1 高度跟踪任务。

```
--task HeightTracking-G1-v0
```

3.8 G1-PickPlace-Tracking-v0 G1 抓取放置任务。

```
--task G1-PickPlace-Tracking-v0
```

该任务需要 YCB、DexCube 等资产，请确认 `/workspace/nucleus_assets` 中已包含相关 USD 资产。

四、Teacher-Student 蒸馏说明

WBC-AGILE 支持 Teacher-Student 蒸馏训练模式，适用于 Sim-to-Real 迁移和部署优化场景。

模式	说明
Teacher	训练阶段使用完整观测或 privileged observation
Student	部署阶段使用轻量观测输入，降低推理开销
History	使用历史状态作为观测输入
RNN	使用循环网络策略建模时序信息

推荐在需要真实机器人部署或推理性能优化时，优先使用蒸馏相关任务。

五、常见问题

5.1 启动失败 请检查以下配置：

- 开发机 GPU 是否可用
- 运行目录是否为`/workspace/agile`
- 是否选择了包含 Isaac Sim / Isaac Lab 的百舸官方镜像
- CUDA、PyTorch 与驱动版本是否匹配
- `/workspace/isaaclab/isaaclab.sh` 是否存在
- 当前目录是否为 WBC-AGILE 项目目录

5.2 USD 资产缺失 如果运行时报错提示找不到机器人、场景或材质资产，请检查：

```
ls /workspace/nucleus_assets/Isaac/
```

5.3 数据集缺失 如果训练时报错提示动作文件或数据集不存在，请检查：

```
ls /workspace/datasets/
```

如目录为空，请重新执行 BOS 同步命令。

5.4 显存不足 可尝试以下方式降低显存占用：

- 降低 `--num_envs`
- 减少训练 batch size
- 使用 `--headless` 模式
- 先使用单卡小规模配置验证流程，再切换到多卡或更大规格资源

六、推荐训练配置

场景	推荐 GPU
基础流程验证	1 * L20 或更高规格加速卡
常规训练	1-4 * L20 或更高规格加速卡
蒸馏训练	8 * L20 或更高规格加速卡
复杂移动操作任务	多机多卡或更高规格资源

FastWAM模型训推

概述

FastWAM 训练加速镜像面向具身智能模型训练场景，预置 FastWAM 训练所需的基础运行环境、依赖组件和性能优化配置，可在百舸 AIHC 平台上快速启动容器并完成数据集、模型权重准备及训练验证。

本文介绍如何使用 FastWAM 训练加速镜像在百舸环境中准备运行环境、下载 BOS 中已缓存的数据集与模型权重，并执行 FastWAM 训练任务。

模型训练

准备工作 使用前请确认已具备以下条件：

- 使用上述镜像创建GPU开发机，GPU 最小要求支持 RTX 的显卡* 8，创建开发机详见：https://cloud.baidu.com/doc/AIHC/s/Tm6db1z9p
- 开发机已挂载可读写的持久化存储目录（可挂载 PFS）
- 已根据平台环境准备好 BOS 下载工具，详见：https://cloud.baidu.com/doc/BOS/s/qjwvyqegc

目录规划 本文以 FastWAM 项目位于 `/root/workspace/FastWAM` 为例说明。建议将数据集与权重放置到如下目录：

```
/root/workspace/FastWAM
├── checkpoints/
│   ├── Wan-AI/
│   │   ├── Wan2.2-TI2V-5B/
│   │   └── Wan2.1-T2V-1.3B/
│   ├── DiffSynth-Studio/
│   │   └── Wan-Series-Converted-Safetensors/
│   ├── fastwam_release/
│   │   └── ActionDiT_linear_interp_Wan22_alphascale_1024hdim.pt
└── data/
    ├── libero_mujoco3.3.2/
    └── robotwin2.0/
```

准备数据集 FastWAM 使用的预处理数据集已从外网地址缓存至 BOS。使用时请从 BOS 下载到项目对应目录，避免在训练环境中直接访问外网。

```
##### 您可以先执行以下代码清理目录
rm -f -- /root/workspace/FastWAM/data
rm -f -- /root/workspace/FastWAM/checkpoints
```

下载 LIBERO 数据集

```
cd /root/workspace/FastWAM
mkdir -p data/libero_mujoco3.3.2

bcecmd bos sync \
  bos:/aihc-rdw-bj/huggingface.co/datasets/yuanty/LIBERO-fastwam/ \
  data/libero_mujoco3.3.2/

cd data/libero_mujoco3.3.2
for f in *.tar.gz; do
  tar -xzf "$f"
done
```

解压后的目录结构应包含：

```
data/libero_mujoco3.3.2/
├── libero_10_no_noops_lerobot/
├── libero_goal_no_noops_lerobot/
├── libero_object_no_noops_lerobot/
└── libero_spatial_no_noops_lerobot/
```

下载 RoboTwin 数据集

```
cd /root/workspace/FastWAM
mkdir -p data/robotwin2.0

bcecmd bos sync \
  bos:/aihc-rdw-bj/huggingface.co/datasets/yuanty/robotwin2.0-fastwam/ \
  data/robotwin2.0/

cd data/robotwin2.0
cat robotwin2.0.tar.gz.part-* | tar -xzf -
```

解压后的目录结构应包含：

```
data/robotwin2.0/
└── robotwin2.0/
    ├── data/
    ├── meta/
    └── videos/
```

如果根目录下保留了 `data/robotwin2.0/dataset_stats.json`，可直接作为当前仓库配置中的统计文件使用，也可以根据需要重新计算。

准备模型权重 训练和推理前均需要准备模型权重。相关外网权重已缓存至 BOS，请从 BOS 下载到 `checkpoints` 目录。

下载权重

```
cd /root/workspace/FastWAM
mkdir -p checkpoints/Wan-AI/Wan2.2-TI2V-5B \
      checkpoints/Wan-AI/Wan2.1-T2V-1.3B \
      checkpoints/DiffSynth-Studio/Wan-Series-Converted-Safetensors \
      checkpoints/fastwam_release

bcecmd bos sync \
  bos:/aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B/ \
  checkpoints/Wan-AI/Wan2.2-TI2V-5B/
bcecmd bos sync \
  bos:/aihc-models-bj/Wan-AI/Wan2.1-T2V-1.3B-Diffusers/ \
  checkpoints/Wan-AI/Wan2.1-T2V-1.3B/
bcecmd bos sync \
  bos:/aihc-models-bj/noodlepop/Wan-Series-Converted-Safetensors/ \
  checkpoints/DiffSynth-Studio/Wan-Series-Converted-Safetensors/
bcecmd bos sync \
  bos:/aihc-models-bj/yuanty/fastwam/ \
  checkpoints/fastwam_release/
```

FastWAM release 权重目录下载完成后应包含：

```
checkpoints/fastwam_release/
├── libero_uncond_2cam224.pt
├── libero_uncond_2cam224_dataset_stats.json
├── robotwin_uncond_3cam_384.pt
└── robotwin_uncond_3cam_384_dataset_stats.json
```

生成 ActionDiT 主干权重 在训练或推理前，需要预生成 ActionDiT 主干权重。

```
cd /root/workspace/FastWAM

mkdir -p checkpoints/Wan-AI/Wan2.1-T2V-1.3B/google

ln -s ../../Wan2.2-TI2V-5B/google/umt5-xxl \
  checkpoints/Wan-AI/Wan2.1-T2V-1.3B/google/umt5-xxl

export DIFFSYNTH_MODEL_BASE_PATH="$(pwd)/checkpoints"
export DIFFSYNTH_SKIP_DOWNLOAD=true

python scripts/preprocess_action_dit_backbone.py \
  --model-config configs/model/fastwam.yaml \
  --output checkpoints/ActionDiT_linear_interp_Wan22_alphascale_1024hdim.pt \
  --device cuda \
  --dtype bfloat16
```

生成后应存在以下文件：

```
/root/workspace/FastWAM/checkpoints/ActionDiT_linear_interp_Wan22_alphascale_1024hdim.pt
```

启动训练 在您开始首次训练前需要执行以下代码生成缓存：

```
cd /root/workspace/FastWAM

torchrun \
  --standalone \
  --nproc_per_node=8 \
  scripts/precompute_text_embeds.py \
  task=libero_uncond_2cam224_1e-4
```

完成数据集与权重准备后，可执行以下命令启动训练验证：

```
cd /root/workspace/FastWAM

export TORCHINDUCTOR_MIX_ORDER_REDUCTION=0
export FASTWAM_SDPA_BACKEND=cudnn
export FASTWAM_CUDNN_BENCHMARK=1

accelerate launch \
  --config_file scripts/accelerate_configs/accelerate_zero2_ds_bucket200m.yaml \
  --num_processes 8 \
  scripts/train.py \
  task=libero_uncond_2cam224_1e-4 batch_size=54 num_workers=12 \
  max_steps=50 log_every=1 eval_every=999999 save_every=999999 \
  seed=42 wandb.enabled=false \
  ++encode_overlap=true ++encode_prefetch_factor=4 \
  ++dataloader_persistent_workers=true \
  ++compile_mot=false ++compile_vae=true ++compile_vae_mode=default
```

训练正常启动后，日志中会出现模型组件加载、数据集解析、ZeRO 优化器初始化及训练 step 输出。例如：

```
 Loading Wan2.2-TI2V-5B components...
Finished loading Wan2.2-TI2V-5B components
Initialized MoT with experts: ['video', 'action']
Train/val dataset size: ...
Starting training with max_steps=50
step=1/50 loss=...
```

注意事项

- 本文中的 BOS 路径用于替代 Hugging Face、ModelScope 等外网下载地址，建议在百舸训练环境中优先通过 BOS 获取数据和权重。
- 示例中的 BOS 下载使用 `bcecmd bos sync`：先创建目标目录，再将 BOS 路径同步到对应本地目录。
- `Wan2.1-T2V-1.3B` 在 BOS 中的目录名为 `Wan2.1-T2V-1.3B-Diffusers`，下载到本地后可按项目配置放置到 `checkpoints/Wan-AI/Wan2.1-T2V-1.3B`。
- 训练命令中的 `--num_processes 8`、`batch_size=54`、`num_workers=12` 等参数与 GPU 数量、显存规格和数据盘性能相关，可按实际资源规格调整。
- 若训练时出现模型文件或数据文件找不到的问题，请优先检查本地目录结构是否与本文建议目录一致。

模型推理

准备工作 使用前请确认已具备以下条件：

- 使用上述镜像创建GPU开发机，GPU 最小要求支持 RTX 的显卡* 1 ，创建开发机详见：https://cloud.baidu.com/doc/AIHC/s/Tm6db1z9p
- 开发机已挂载可读写的持久化存储目录（可挂载 PFS）
- 已根据平台环境准备好 BOS 下载工具，详见：https://cloud.baidu.com/doc/BOS/s/qjwvyqegc

目录规划 本文以 FastWAM 项目位于 `/root/workspace/FastWAM` 为例说明。建议将数据集与权重放置到如下目录：

```
/root/workspace/
├── checkpoints/
│   ├── fastwam_release/
└── data/
    ├── libero_mujoco3.3.2/
    └── robotwin2.0/
```

准备数据集 FastWAM 使用的预处理数据集已从外网地址缓存至 BOS。使用时请从 BOS 下载到项目对应目录，避免在推理环境中直接访问外网。

```
cd /workspace/FastWAM
```

下载 LIBERO 数据集

```
mkdir -p data/libero_mujoco3.3.2

bcecmd bos sync \
  bos:/aihc-rdw-bj/huggingface.co/datasets/yuanty/LIBERO-fastwam/ \
  data/libero_mujoco3.3.2/

for f in *.tar.gz; do
  tar -xzf "$f"
done
```

解压后的目录结构应包含：

```
data/libero_mujoco3.3.2/
├── libero_10_no_noops_lerobot/
├── libero_goal_no_noops_lerobot/
├── libero_object_no_noops_lerobot/
└── libero_spatial_no_noops_lerobot/
```

下载 RoboTwin 数据集

```
mkdir -p data/robotwin2.0

bcecmd bos sync \
  bos:/aihc-rdw-bj/huggingface.co/datasets/yuanty/robotwin2.0-fastwam/ \
  data/robotwin2.0/

cat robotwin2.0.tar.gz.part-* | tar -xzf -
```

解压后的目录结构应包含：

```
data/robotwin2.0/
└── robotwin2.0/
    ├── data/
    ├── meta/
    └── videos/
```

如果根目录下保留了 data/robotwin2.0/dataset_stats.json，可直接作为当前仓库配置中的统计文件使用，也可以根据需要重新计算。

准备模型权重 推理和推理前均需要准备模型权重。相关外网权重已缓存至 BOS，请从 BOS 下载到 checkpoints 目录。

下载权重

```
mkdir -p checkpoints/fastwam_release
bcecmd bos sync \
  bos:/aihc-models-bj/yuanty/fastwam/ \
  checkpoints/fastwam_release/
```

FastWAM release 权重目录下载完成后应包含：

```
checkpoints/fastwam_release/
├── libero_uncond_2cam224.pt
├── libero_uncond_2cam224_dataset_stats.json
├── robotwin_uncond_3cam_384.pt
└── robotwin_uncond_3cam_384_dataset_stats.json
```

生成 ActionDiT 主干权重 在推理或推理前，需要预生成 ActionDiT 主干权重。

```
cd /root/workspace/FastWAM

mkdir -p checkpoints/Wan-AI/Wan2.1-T2V-1.3B/google

ln -s ../../Wan2.2-TI2V-5B/google/umt5-xxl \
  checkpoints/Wan-AI/Wan2.1-T2V-1.3B/google/umt5-xxl

export DIFFSYNTH_MODEL_BASE_PATH="$(pwd)/checkpoints"
export DIFFSYNTH_SKIP_DOWNLOAD=true

python scripts/preprocess_action_dit_backbone.py \
  --model-config configs/model/fastwam.yaml \
  --output checkpoints/ActionDiT_linear_interp_Wan22_alphascale_1024hdim.pt \
  --device cuda \
  --dtype bfloat16
```

生成后应存在以下文件：

```
/root/workspace/FastWAM/checkpoints/ActionDiT_linear_interp_Wan22_alphascale_1024hdim.pt
```

启动推理 robotwin

```
python experiments/robotwin/run_robotwin_manager.py \
task=robotwin_uncond_3cam_384_1e-4 \
ckpt=checkpoints/fastwam_release/robotwin_uncond_3cam_384.pt \
EVALUATION.dataset_stats_path=checkpoints/fastwam_release/robotwin_uncond_3cam_384_dataset_stats.json \
MULTIRUN.num_gpus=1 \
MULTIRUN.max_tasks_per_gpu=1 \
EVALUATION.timing_enabled=true
```

libero

```
python experiments/libero/run_libero_manager.py \
task=libero_uncond_2cam224_1e-4 \
ckpt=checkpoints/fastwam_release/libero_uncond_2cam224.pt \
EVALUATION.dataset_stats_path=checkpoints/fastwam_release/libero_uncond_2cam224_dataset_stats.json \
MULTIRUN.num_gpus=1 \
MULTIRUN.max_tasks_per_gpu=1 \
+EVALUATION.timing_enabled=true
```

RLinf v0.3 基于 LIBERO 的训练

概述

Rlinf是一个面向 **Embodied AI（具身智能）** 和 **Agentic AI（智能体 AI）** 的开源强化学习基础框架。

通过百舸快速开始，您可以使用通过开发机调试部署 Rlinf v0.3最新版本，启动不同模型的训练任务，并通过 TensorBoard、Ray Dashboard、训练日志和 GPU 指标监控训练进展，本文将以 OpenVLA 与 π0.5 模型为例。

适用场景

- 使用 RLinf 框架训练不同仿真环境下的机器人任务策略模型。
- 使用不同基座模型继续进行 GRPO 强化学习训练。
- 上复现 libero_90_grpo_openvlaoft 或迁移到 libero_130_grpo_openvlaoft。
- 在单机开发机上排查调试 仿真环境、模型、robosuite、transformers、Ray 和 TensorBoard 相关训练问题。
- 对 rollout 阶段耗时、cgroup OOM、Ray worker 异常或 rollout 后 soft hang 进行定位。
- 在 /root/workspace/目录下已经为您准备了可以按顺序执行的 notebook，您可以参考运行。

训练规格参考 & 环境配置

推荐训练配置如下：

项目	推荐配置
GPU	单机 8 卡，单卡约 96-97GiB显存
CPU	建议 80 核以上
容器内存	建议 900GiB 以上；90/130 原始配置更稳妥给到 1TiB+
存储空间	如果您需要下载资源测试，建议不少于 600G

项目目录配置如下表所示：

项目	推荐配置
CUDA / 驱动	建议使用满足平台镜像要求的默认驱动环境
工作目录	`/root/workspace/RLinf`
Python 环境	`/root/workspace/RLinf/.venv`
OpenVLA-OFT	`/root/workspace/openvla-oft`
训练入口	`/root/workspace/RLinf/examples/embodiment/train_embodied_agent.py`
模型目录	`/root/workspace/models`
日志目录	`/root/workspace/logs`

预期效果参考

统一模型在 LIBERO 多个任务组和 90/130 综合任务集上的评测结果（以 OpenVLA 为例）：

模型	Object	Spatial	Goal	Long	90	130
huggingface OpenVLA-OFT (LoRA-base)	50.20%	51.61%	49.40%	11.90%	42.67%	42.09%
huggingface OpenVLA-OFT (RLinf-GRPO)	99.60%	98.69%	98.09%	93.45%	98.02%	97.85%
效果提升	+49.40%	+47.08%	+48.69%	+81.55%	+55.35%	+55.76%

准备工作

如果您所在网络环境访问 HuggingFace 较慢，建议提前配置 HuggingFace 镜像源，或配置代理加速。

huggingface 镜像配置：

```
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
```

百舸平台已提供数据集和模型权重的 BOS 地址，您也可以直接从 BOS 下载到训练任务的工作目录，避免依赖外部网络。

环境说明

> 说明：本文中的路径均以百舸训练任务内的 `/root/workspace` 为例。实际使用时，可根据任务挂载目录调整。

当前实跑使用以下目录：

```
/root/workspace/RLinf
/root/workspace/RLinf/.venv
/root/workspace/openvla-oft
/root/workspace/models
/root/workspace/logs
```

步骤一：准备模型权重

训练需要准备 RLinf OpenVLA-OFT LIBERO Base-Lora 权重，90 任务和 130 任务分别使用对应模型目录。

方式一：通过 HuggingFace 下载模型权重

```
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
mkdir -p /root/workspace/models

huggingface-cli download RLinf/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora \
  --repo-type model \
  --revision v3.0 \
  --local-dir /root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora
huggingface-cli download RLinf/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora \
  --repo-type model \
  --revision v3.0 \
  --local-dir /root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora
```

方式二：通过 BOS 下载 已知 BOS 模型路径（目录路径必须带结尾 `/`）：

```
bos:/aihc-models-bj/RLinf/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora/
bos:/aihc-models-bj/RLinf/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora/
```

建议下载到：

```
/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora
/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora
```

从 BOS 下载，建议放入 tmux 并持续检查结果：

```
mkdir -p /root/workspace/models /root/workspace/logs
SESSION=bos_download_models
LOG=/root/workspace/logs/bos_download_models.log

cat > /tmp/download_rlinf_models.sh <<'EOF'
##### !/usr/bin/env bash
set -euo pipefail
{
  date
  echo "download 90 start"
  bcecmd bos cp -r \
    bos:/aihc-models-bj/RLinf/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora/ \
    /root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora/
  echo "download 130 start"
  bcecmd bos cp -r \
    bos:/aihc-models-bj/RLinf/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora/ \
    /root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora/
  echo "download done"
  date
} 2>&1 | tee -a /root/workspace/logs/bos_download_models.log
EOF
chmod +x /tmp/download_rlinf_models.sh

tmux new-session -d -s "$SESSION" /tmp/download_rlinf_models.sh
```

查看下载进度：

```
tmux ls 2>/dev/null | grep bos_download_models || true
tail -100 /root/workspace/logs/bos_download_models.log
du -sh /root/workspace/models/* 2>/dev/null || true
find /root/workspace/models -maxdepth 3 -type f | wc -l
```

如果 bcecmd 报 Object don't exist，先执行以下命令确认真实目录和结尾 / ：

```
bcecmd bos ls bos:/aihc-models-bj/RLinf/
```

验证模型目录：

```
for MODEL in \
 /root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora \
 /root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora
do
 echo "===== $MODEL ====="
 ls "$MODEL"
 find "$MODEL" -maxdepth 2 -type f | head -50
done
```

步骤二：确认训练配置

以 libero_90_grpo_openvlaoft 为例，核心配置如下：

```
cluster:
 num_nodes: 1
 component_placement:
  actor,env,rollout: all

algorithm:
 group_size: 8
 adv_type: grpo
 loss_type: actor
 reward_type: action_level
 logprob_type: token_level
 entropy_type: token_level
 loss_agg_func: token-mean
 kl_beta: 0.0
 clip_ratio_high: 0.28
 clip_ratio_low: 0.2
 reward_coef: 5.0

env:
 train:
  rollout_epoch: 64
  total_num_envs: 64
  group_size: 8
 eval:
  total_num_envs: 500

actor:
 global_batch_size: 16384
 micro_batch_size: 32

rollout:
 pipeline_stage_num: 1
```

原始 YAML 中模型路径通常仍是占位符：

```
/path/to/model/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora/
```

实际启动时请通过 Hydra CLI override 注入真实模型路径：

```
rollout.model.model_path=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora
actor.model.model_path=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora
```

如果使用官方 `run_embodiment.sh`，需要确认脚本是否支持透传额外 Hydra override。当前实跑环境中该脚本不透传额外参数，因此使用直接 Python 入口启动。

推荐参数分档 在 8 卡 显存 96 GB 显卡和 1.5TB 容器内存环境中，原始 `rollout_epoch=64` + `actor.global_batch_size=16384` 可以完成 rollout，但可能在 rollout 后长时间无主日志、无 TensorBoard scalar，表现为疑似 soft hang。建议按以下档位逐步启动：

档位	参数	用途
稳定首跑	`env.train.rollout_epoch=16 actor.global_batch_size=8192`	90/130 任务推荐首跑配置
130 降 KL 稳定档	`env.train.rollout_epoch=16 actor.global_batch_size=8192 actor.optim.lr=1.0e-5`	130 任务如 step=0 `approx_kl` 明显偏高，优先降学习率后重跑
提升吞吐	`env.train.rollout_epoch=32 actor.global_batch_size=8192 actor.optim.lr=1.0e-5`	130 降 KL 后连续多个 step 稳定，再扩大采样量
原始复现	`env.train.rollout_epoch=64 actor.global_batch_size=16384`	仅在稳定首跑确认无 soft hang 且 KL 稳定后使用

90/130 稳定首跑推荐 override：

```
 env.train.rollout_epoch=16 \
env.train.total_num_envs=64 \
algorithm.group_size=8 \
actor.micro_batch_size=32 \
actor.global_batch_size=8192
```

130 降 KL 稳定档推荐 override：

```
 env.train.rollout_epoch=16 \
env.train.total_num_envs=64 \
algorithm.group_size=8 \
actor.micro_batch_size=32 \
actor.global_batch_size=8192 \
actor.optim.lr=1.0e-5
```

实测 130 默认学习率 `2.0e-5` 时，step=0 `train/actor/approx_kl≈0.884` 偏高；加入 `actor.optim.lr=1.0e-5` 后可稳定推进到 Global Step 17/1000，TensorBoard 连续写入 17 个 step，step=16 `train/actor/approx_kl≈0.0103`，KL 明显回落。130 任务建议默认采用降 KL 稳定档。

步骤三：启动训练

建议使用 tmux 启动训练，避免 SSH 断开导致任务中止。

以下命令使用稳定首跑配置启动。先按任务选择变量：90 任务使用 `libero_90_grpo_openvlaoft`，130 任务使用 `libero_130_grpo_openvlaoft`。若需要原始复现，可将 `env.train.rollout_epoch=16` 改为 `64`、`actor.global_batch_size=8192` 改为 `16384`，但应优先确认稳定首跑无 soft hang。

```
 cd /root/workspace/RLinf

##### 90 任务
SESSION=libero90_train
CONFIG=libero_90_grpo_openvlaoft
MODEL=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-90-Base-Lora
START_SCRIPT=/tmp/start_libero90_train.sh

##### 130 任务改用以下变量
##### SESSION=libero130_train
##### CONFIG=libero_130_grpo_openvlaoft
##### MODEL=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora
##### START_SCRIPT=/tmp/start_libero130_train.sh

LOG_DIR=/root/workspace/logs/$(date +%Y%m%d-%H:%M:%S)-${CONFIG}_rollout16_gbs8192
LOG_FILE=${LOG_DIR}/run_embodiment.log
mkdir -p "$LOG_DIR"

cat > "$START_SCRIPT" <<EOF
##### !/usr/bin/env bash
set -e
cd /root/workspace/RLinf
export PATH=/root/workspace/RLinf/.venv/bin:\$PATH
export EMBODIED_PATH=/root/workspace/RLinf/examples/embodiment
export REPO_PATH=/root/workspace/RLinf
export SRC_FILE=/root/workspace/RLinf/examples/embodiment/train_embodied_agent.py
export MUJOCO_GL=egl
export PYOPENGL_PLATFORM=egl
export ROBOTWIN_PATH=/path/to/RoboTwin
export PYTHONPATH=/root/workspace/RLinf:/root/workspace/openvla-oft:/path/to/RoboTwin:\$PYTHONPATH
export OMNIGIBSON_NO_OMNI_LOGS=1
export OMNIGIBSON_DEBUG=0
export ROBOT_PLATFORM=LIBERO
export RAY_memory_usage_threshold=0.98
export TOKENIZERS_PARALLELISM=false
CMD="python /root/workspace/RLinf/examples/embodiment/train_embodied_agent.py --config-path /root/workspace/RLinf/examples/embodiment/config/ --config-name ${CONFIG}
runner.logger.log_path=${LOG_DIR} rollout.model.model_path=${MODEL} actor.model.model_path=${MODEL} env.train.rollout_epoch=16 env.train.total_num_envs=64 algorithm.group_size=8
actor.micro_batch_size=32 actor.global_batch_size=8192"
echo "\$CMD" > ${LOG_FILE}
echo "LOG_DIR=${LOG_DIR}" >> ${LOG_FILE}
echo "MODEL=${MODEL}" >> ${LOG_FILE}
\$CMD 2>&1 | tee -a ${LOG_FILE}
EOF

chmod +x "$START_SCRIPT"

if tmux has-session -t "$SESSION" 2>/dev/null; then
  tmux kill-session -t "$SESSION"
fi

tmux new-session -d -s "$SESSION" "$START_SCRIPT"
printf "session=%s\nlog=%s\n" "$SESSION" "$LOG_FILE"
```

稳定首跑已验证：`rollout_epoch=16`、`actor.global_batch_size=8192` 可以越过原始 run 的 rollout 后 soft hang 点，并写出 TensorBoard scalar。90/130 任务默认先使用该档；如需扩大采样量，再升到 `rollout_epoch=32` 或原始复现档。

从 90 任务切换到 130 任务 如果需要停止当前 90 任务并重新运行 130 任务，优先通过 tmux 停止，不使用 `pkill -f train_embodied_agent.py`：

```
if tmux has-session -t libero90_train 2>/dev/null; then
  tmux kill-session -t libero90_train
fi
sleep 8

ps -eo pid,stat,pcpu,pmem,rss,args --sort=-pcpu | \
  grep -E 'train_embodied_agent.py|ray::EmbodiedFSDPActor.run_training|ray::MultiStepRolloutWorker|ray::CollectiveManager' | grep -v grep || true
```

确认 130 模型目录存在且非空：

```
MODEL=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora
if [ ! -d "$MODEL" ] || [ -z "$(find "$MODEL" -maxdepth 2 -type f 2>/dev/null | head -1)" ]; then
  echo "ERROR: 130 model missing or empty: $MODEL" >&2
  exit 2
fi
```

130 降 KL 稳定档启动命令已验证：

```
python /root/workspace/RLinf/examples/embodiment/train_embodied_agent.py \
--config-path /root/workspace/RLinf/examples/embodiment/config/ \
--config-name libero_130_grpo_openvlaoft \
runner.logger.log_path=${LOG_DIR} \
rollout.model.model_path=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora \
actor.model.model_path=/root/workspace/models/RLinf-OpenVLAOFT-LIBERO-130-Base-Lora \
env.train.rollout_epoch=16 \
env.train.total_num_envs=64 \
algorithm.group_size=8 \
actor.micro_batch_size=32 \
actor.global_batch_size=8192 \
actor.optim.lr=1.0e-5
```

启动后验证点：

- `tmux ls` 中存在 `libero130_train`。
- 日志中 `--config-name libero_130_grpo_openvlaoft`。
- Hydra 配置中 `task_suite_name: libero_130`。
- Ray 启动并识别 1 node / 8 accelerators。
- 无 `Traceback`、`RayOutOfMemory`、`worker died`。

训练启动后建议立即打开 TensorBoard 和 Ray Dashboard：

```
ssh -p <port> -L 6006:127.0.0.1:6006 -L 18265:127.0.0.1:8265 root@<host>
```

重点查看：

- TensorBoard：`env/success_once`、`env/reward`、`rollout/rewards`、`train/actor/*`、`time/actor/*`
- Ray Dashboard：actor/rollout worker 状态、CPU/GPU 使用、异常 worker
- 主日志：rollout 进度、`global_step`、异常栈和 OOM 信息

步骤四：监控训练

查看主日志：

```
tail -f /root/workspace/logs/<run-dir>/run_embodiment.log
```

提取关键进展：

```
grep -nE 'Generating Rollout Epochs|global_step|success_once|reward=|actor/|Traceback|Exception|RayOutOfMemory|worker died' \
/root/workspace/logs/<run-dir>/run_embodiment.log | tail -100
```

查看 GPU：

```
watch -n 2 'nvidia-smi --query-gpu=index,utilization.gpu,memory.used,memory.total --format=csv,noheader,nounits'
```

查看 cgroup 内存和 OOM：

```
watch -n 2 'cat /sys/fs/cgroup/memory.current; cat /sys/fs/cgroup/memory.max; cat /sys/fs/cgroup/memory.events'
```

查看 Ray worker：

```
ps -eo pid,stat,pcpu,pmem,rss,args --sort=-pcpu | \
grep -E 'train_embodied_agent.py|ray::EmbodiedFSDPActor.run_training|ray::MultiStepRolloutWorker|ray::CollectiveManager' | head -30
```

步骤五：启动 TensorBoard 和 Ray Dashboard（可选）

启动 TensorBoard：

```
cd /root/workspace/RLinf
export PATH=/root/workspace/RLinf/.venv/bin:$PATH
tmux new-session -d -s tensorboard_rlinf \
  'cd /root/workspace/RLinf && tensorboard --logdir /root/workspace/logs --port 6006 --host 127.0.0.1'
```

本地访问：

```
ssh -p 5259 -L 6006:127.0.0.1:6006 root@xxx.xxx.xxx.xxx
```

浏览器打开：

```
http://127.0.0.1:6006
```

Ray Dashboard 远端默认监听：

```
127.0.0.1:8265
```

本地访问：

```
ssh -p 5259 -L 18265:127.0.0.1:8265 root@xxx.xxx.xxx.xxx
```

浏览器打开：

```
http://127.0.0.1:18265
```

远端验证 Ray Dashboard：

```
curl -s http://127.0.0.1:8265/api/version
```

步骤六：解析 TensorBoard 指标

使用脚本读取 TensorBoard scalar：

```
cd /root/workspace/RLinf
export PATH=/root/workspace/RLinf/.venv/bin:$PATH
export TF_CPP_MIN_LOG_LEVEL=3

python - <<'PY'
from pathlib import Path
from tensorboard.backend.event_processing.event_accumulator import EventAccumulator
run = Path('/root/workspace/logs/<run-dir>')
for event in sorted((run / 'tensorboard').glob('events.out.tfevents.*')):
    print(f'EVENT {event.name} size={event.stat().st_size}')
    ea = EventAccumulator(str(event), size_guidance={'scalars': 0})
    ea.Reload()
    for tag in sorted(ea.Tags().get('scalars', [])):
        vals = ea.Scalars(tag)
        if vals:
            v = vals[-1]
            print(f'{tag}\tcount={len(vals)}\tstep={v.step}\tvalue={v.value}')
PY
```

重点关注：

- `env/success_once`
- `env/reward`
- `env/return`
- `rollout/rewards`
- `train/actor/policy_loss`
- `train/actor/approx_kl`
- `train/actor/grad_norm`
- `time/actor/run_training`
- `time/actor/actor/recv_traj`
- `time/actor/actor/sync_model_to_rollout`

90 稳定首跑已观测关键指标：

指标	值
env/success_once	0.7763671875
env/reward	0.0315093882
env/return	3.8818359375
env/num_trajectories	1024
rollout/rewards	0.0106468508
time/actor/run_training	886.1267
time/actor/actor/recv_traj	862.7485
time/actor/actor/sync_model_to_rollout	23.0961
train/actor/approx_kl	0.0723108
train/actor/grad_norm	0.335577
train/actor/lr	2.0e-05
train/actor/policy_loss	0.00241828

130 降 KL 稳定档已观测关键指标（actor.optim.lr=1.0e-5）：

指标	值
global_step	17/1000
env/success_once	0.158203125
env/reward	0.0062359129
env/return	0.791015625
env/num_trajectories	1024
rollout/rewards	0.0043383949
time/actor/run_training	885.4130
time/generate_rollouts	854.4069
time/step	1755.9901
train/actor/approx_kl	0.0102794
train/actor/grad_norm	0.4322200
train/actor/policy_loss	5.23655e-05
train/actor/total_loss	1.63642e-06

性能参考：

指标	当前已观测参考
rollout 64 epoch	约 56 分钟，约 52.7 秒/epoch
rollout 16 epoch	约 13-15 分钟
rollout 32 epoch	预计约 28-30 分钟
稳定首跑 actor run_training	约 886 秒
稳定首跑 actor sync_model_to_rollout	约 23 秒
稳定首跑 actor recv_traj	约 863 秒

常见问题处理

问题	解决方案
模型路径仍是 /path/to/model/...	启动命令必须追加 rollout.model.model_path=$MODEL actor.model.model_path=$MODEL
Ray worker 被 memory killer 杀	检查 /sys/fs/cgroup/memory.max 和 /sys/fs/cgroup/memory.events，优先调大容器内存到 900GiB+
rollout 后主日志停在 100%	先看 TensorBoard、GPU、worker CPU、actor worker 日志；不要只凭主日志判断停止
rollout 后疑似 soft hang	表现为 rollout 100% 后长时间无主日志、TensorBoard event 仅几十字节且无 scalar，但 GPU/worker 仍活跃；先抓栈保留现场，再用稳定首跑配置 env.train.rollout_epoch=16 actor.global_batch_size=8192 重启验证

具身智能-Isaac Lab 3.0

工具介绍



> Isaac Lab 是由 NVIDIA 推出的开源机器人学习框架，基于 GPU 加速实现大规模并行仿真，统一强化学习、模仿学习和运动规划等机器人研究工作流程，推动机器人 sim-to-real（仿真到现实）的高效迁移。

Isaac Lab 3.0 的主要特点：

1. **多物理后端：**新增 Newton（MJWarp）物理后端，可脱离 Isaac Sim 运行（kit-less），无需安装 Isaac Sim、无需接受 Isaac Sim EULA；PhysX 后端仍可选（需 Isaac Sim）；
2. **统一命令行：**isaaclab train / play / benchmark / list_envs 统一入口，通过 --rl_library 切换 RSL-RL、RL Games、SKRL、SB3 等 RL 库，通过 physics= / renderer= / presets= 切换后端与配置；
3. **官方预训练权重：**--checkpoint pretrained 自动下载并加载官方权重，便于与自训模型对照；
4. **基准评测：**isaaclab benchmark play / training 输出 reward、episode 长度、成功率、FPS 等结构化结果；
5. **多种可视化：**--viz newton_gl / rerun / viser / kit，其中 viser 可通过浏览器远程查看。

对比项	Isaac Lab 2.x	Isaac Lab 3.0
依赖	必须安装 Isaac Sim	Newton 后端无需 Isaac Sim
启动方式	./isaaclab.sh -p scripts/reinforcement_learning/<库>/train.py	isaaclab train --rl_library <库>
后端选择	仅 PhysX	physics=newton_mjwarp / physics=isaacsim_physx
无界面运行	--headless	默认无界面，--viz 按需开启
加载官方权重	--use_pretrained_checkpoint	--checkpoint pretrained

镜像选择：Isaac Lab 与 Isaac Lab kitless

Isaac Lab 3.0 官方在 NGC 同一镜像仓库发布两类镜像，以 tag 后缀区分：

对比项	Isaac Lab（标准镜像）	Isaac Lab kitless
镜像地址	ccr-registry.baidubce.com/aihc/nvcr.io/nvidia/isaac-lab:3.0.0	ccr-registry.baidubce.com/aihc/nvcr.io/nvidia/isaac-lab:3.0.0-kitless
基础镜像	Isaac Sim 6.2.0 镜像（nvcr.io/nvidia/isaac-sim:6.2.0）	Ubuntu 24.04 + Python 3.12，不含 Isaac Sim
镜像大小（amd64，压缩后）	约 16.7 GB	约 5.9 GB
物理后端	Isaac Sim PhysX（physics=isaacsim_physx）+ Newton + OVPhysX	Newton（physics=newton_mjwarp）+ OVPhysX（physics=ovphysx），无 Isaac Sim PhysX
渲染	Omniverse Kit RTX 渲染，另含 OVRTX	OVRTX 渲染（--viz newton_rtx）与 OpenGL 光栅化（--viz newton_gl），均可离屏
可视化	包含 --viz kit（Isaac Sim 图形界面）	newton_gl / newton_rtx / viser / rerun，不支持 --viz kit
RL 库	RSL-RL、RL Games、SKRL、SB3	同左（四个均内置）
许可	需接受 Isaac Sim EULA（-e ACCEPT_EULA=Y）	无 Isaac Sim EULA
容器挂载	需挂载 Isaac Sim 的 kit/ov/GL 等多个缓存目录	仅需挂载 logs 等工作目录，自带 uv 与 Warp 缓存
适用场景	需要 Isaac Sim 图形界面、Isaac Sim PhysX 训练、PhysX↔Newton 跨后端验证或 Kit RTX 传感器	用 Newton 训练强化学习策略，追求更小的镜像、无 EULA

部署环境要求&最佳实践建议

	部署要求
CPU	建议按表单默认值
内存	建议按表单默认值
GPU	**aihc.lgn7ib ×1**（单卡显存约 96-97 GiB，本文实测机型）
驱动	实测 580.126.20 / CUDA 13.0 可正常训练与评测
CDS	按需（单次复现产物约 50 MB）
其它	无

使用说明

本文以 **快速验证环境** 和 **ANYmal-D 四足机器人速度跟踪训练、评测、导出** 两个案例，介绍 Isaac Lab 3.0 的使用过程。

创建与登录开发机

根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机 webIDE，打开 Terminal 界面；也可在开发机详情页开启 SSH 后通过本地终端登录。

主要路径介绍

- Isaac Lab 源码路径：/workspace/isaaclab
- Python 虚拟环境：/opt/isaaclab-venv（已加入 PATH，可直接执行 isaaclab 命令）
- 训练日志与模型：/workspace/isaaclab/logs

环境检查

执行如下命令，确认版本、GPU 与关键依赖：

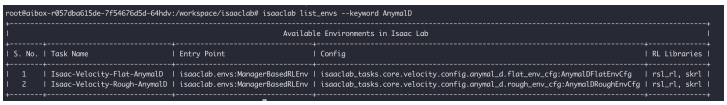
```
cd /workspace/isaaclab
cat VERSION
nvidia-smi --query-gpu=name,memory.total,driver_version --format=csv,noheader
uv pip list --python /opt/isaaclab-venv/bin/python | grep -iE '^(isaaclab |newton |warp-lang|torch |rsl-rl-lib)'
```

期望输出：VERSION 为 3.0.0，newton 1.6.1，rsl-rl-lib 5.5.1，torch 2.11.0+cu130，warp-lang 1.17.0。

快速启动训练任务

执行如下命令，展示 Isaac Lab 中可用的任务，以及每个任务支持的 RL 库和后端：

```
cd /workspace/isaaclab
isaaclab list_envs --show_presets
##### 按关键字过滤，例如只看 ANYmal-D
isaaclab list_envs --keyword AnymalD
```



以倒立摆任务 Isaac-Cartpole-Direct 为例，执行如下命令启动训练。这里设置的 num_envs 和 max_iterations 仅为了快速验证环境，实测约 90 秒完成：

```
cd /workspace/isaaclab
isaaclab train --rl_library rsl_rl --task Isaac-Cartpole-Direct \
  --num_envs 16 presets=newton_mjwarp --max_iterations 5
```

训练输出包含仿真场景配置、观测空间、奖励函数、模型定义以及日志保存路径等信息，命令正常退出即说明 GPU、驱动与 Isaac Lab 环境可用。

> 首次运行会编译 Warp 内核并下载仿真资产，耗时略长，后续运行会复用缓存。

输出示例：

```
 | S. No. | Task Name                | Entry Point                    | RL Libraries |
|  1   | Isaac-Velocity-Flat-AnymalD  | isaaclab.envs:ManagerBasedRLEnv | rsl_rl, skrl |
|  2   | Isaac-Velocity-Rough-AnymalD | isaaclab.envs:ManagerBasedRLEnv | rsl_rl, skrl |
```

举例：训练 ANYmal-D 四足机器人跟踪速度指令

我们使用 RSL-RL PPO 在 Newton MJWarp 后端上训练 ANYmal-D 四足机器人跟踪线速度与角速度指令，并与官方预训练权重做同口径对照。官方任务提供两个难度：

项	平地（Flat）	崎岖地形（Rough）
任务名	Isaac-Velocity-Flat-AnymalD	Isaac-Velocity-Rough-AnymalD
并行环境数	4096	4096
训练迭代	300	2000
网络结构（actor/critic）	[128,128,128]	[512,256,128]
观测维度 → 动作维度	48 → 12	235 → 12（含高度扫描）
实测训练耗时	约 3.5 分钟	约 26 分钟
用途	快速验证链路（推荐首跑）	完整验证

启动训练

执行如下命令启动训练。训练时间较长，建议使用 nohup 后台运行，避免终端断开导致训练中断：

```
cd /workspace/isaaclab && mkdir -p logs/repro
export TASK=Isaac-Velocity-Flat-AnymalD     # 崎岖地形改为 Isaac-Velocity-Rough-AnymalD
nohup isaaclab train --rl_library rsl_rl --task $TASK physics=newton_mjwarp --seed 42 \
    > logs/repro/train_${TASK}.log 2>&1 &
echo $! > logs/repro/train_${TASK}.pid
```

训练日志：

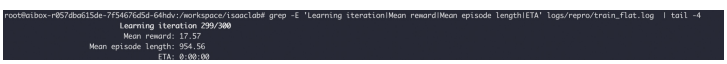
```
################################################################################
                      Learning iteration 299/300

                       Total steps: 29491200
                Steps per second: 167726
                  Collection time: 0.509s
                    Learning time: 0.077s
                  Mean value loss: 0.0061
              Mean surrogate loss: -0.0046
                Mean entropy loss: 3.1088
                      Mean reward: 17.57
              Mean episode length: 954.56
                 Mean action std: 0.32
    Episode_Reward/track_lin_vel_xy_exp: 0.8407
     Episode_Reward/track_ang_vel_z_exp: 0.4317
            Episode_Reward/lin_vel_z_l2: -0.0263
         Episode_Reward/ang_vel_xy_l2: -0.0374
        Episode_Reward/dof_torques_l2: -0.1284
            Episode_Reward/dof_acc_l2: -0.0832
        Episode_Reward/action_rate_l2: -0.0445
         Episode_Reward/feet_air_time: -0.0385
      Episode_Reward/undesired_contacts: -0.0006
     Episode_Reward/flat_orientation_l2: -0.0248
          Episode_Reward/dof_pos_limits: 0.0000
                Metrics/success_rate: 0.9792
    Metrics/base_velocity/error_vel_xy: 0.1698
   Metrics/base_velocity/error_vel_yaw: 0.1392
          Episode_Termination/time_out: 0.9229
      Episode_Termination/base_contact: 0.0771
--------------------------------------------------------------------------------
                       Iteration time: 0.59s
                         Time elapsed: 0:03:26
                                  ETA: 0:00:00

Training time: 208.66 seconds
```

查看训练进展：

```
cd /workspace/isaaclab
grep -E 'Learning iteration|Mean reward|Mean episode length|ETA' logs/repro/train_${TASK}.log | tail -4
```



训练过程中 Mean reward 持续上升、Mean episode length 接近 1000 即说明机器人逐渐学会稳定行走。训练结束时日志末尾输出 `Training time: <秒> seconds`，模型每 50 次迭代保存一次：

- 平地：`logs/rsl_rl/anymal_d_flat/<时间戳>/model_299.pt`
- 崎岖地形：`logs/rsl_rl/anymal_d_rough/<时间戳>/model_1999.pt`

也可使用 TensorBoard 查看训练曲线：

```
 cd /workspace/isaaclab
tensorboard --logdir logs/rsl_rl --port 6006 --bind_all
```

回放与导出策略

执行如下命令加载训练好的模型进行回放。`play` 加载 checkpoint 后会自动将策略导出为 TorchScript 和 ONNX，保存在运行目录的 `exported/` 下，可直接用于部署：

```
 cd /workspace/isaaclab
export RUN_DIR=logs/rsl_rl/anymal_d_flat/<时间戳>      # 替换为实际训练目录
isaaclab play --rl_library rsl_rl --task $TASK \
   --num_envs 16 physics=newton_mjwarp --checkpoint $RUN_DIR/model_299.pt --viz none
```



`play` 会持续回放不会自动退出，看到导出文件后按 `Ctrl+C` 结束即可。导出产物：

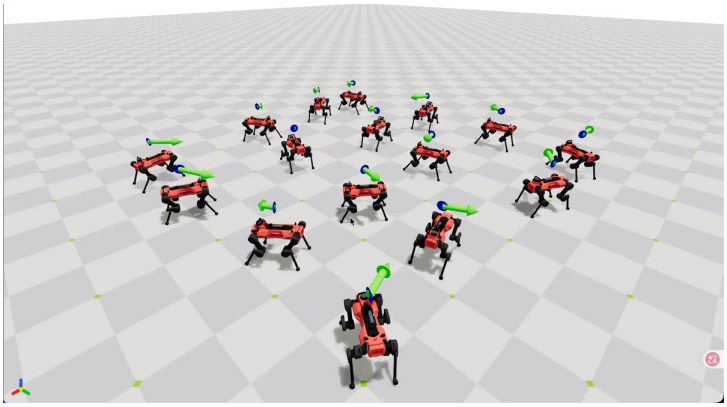
```
<RUN_DIR>/exported/
├── policy.pt        # TorchScript
├── policy.onnx      # ONNX
└── policy.onnx.data
```

如需在浏览器中查看机器人行走效果，将 `--viz none` 改为 `--viz viser`，并在本地建立端口转发后，按日志中输出的地址在浏览器打开：

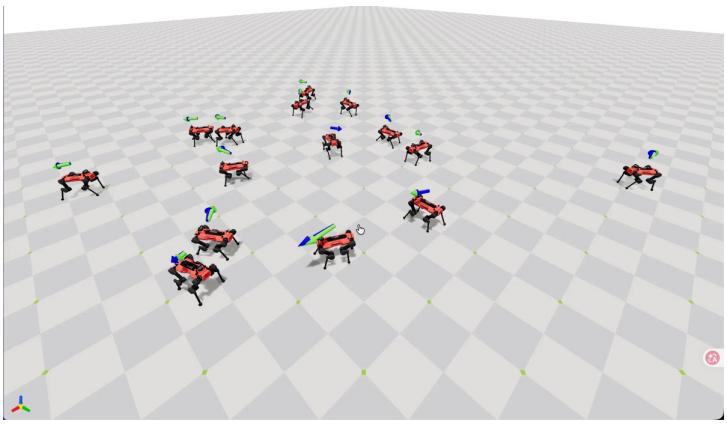
```
 ##### 本地终端执行，端口以日志输出为准
ssh -p <port> -L 8080:127.0.0.1:8080 root@<开发机地址>
```

如需直接查看官方预训练模型的效果，将 `--checkpoint` 改为 `pretrained` 即可自动下载并加载官方权重。

训练前：



训练后：



评测：自训模型 vs 官方预训练模型

Isaac Lab 3.0 内置 `isaaclab benchmark play`，可在相同参数下分别评测自训模型与官方模型。首先下载官方预训练权重（平地 0.94 MB，崎岖地形 6.56 MB）：

```
 cd /workspace/isaaclab
B=https://omniverse-content-production.s3-us-west-2.amazonaws.com/Assets/Isaac/6.2/Isaac/IsaacLab/PretrainedCheckpoints/rsl_rl
curl -fsSL -o logs/repro/official_${TASK}.pt "$B/${TASK}_newtonmjwarp_none_rsl_rl.pt"
```

两次评测参数完全一致，只替换 `--checkpoint` 与 `--output_path`，单次约 1 分钟：

```
cd /workspace/isaaclab
for TAG in own official; do
  CKPT=$([ $TAG = own ] && echo "$RUN_DIR/model_299.pt" || echo "logs/repro/official_${TASK}.pt")
  isaaclab benchmark play --rl_library rsl_rl --task $TASK \
      --num_envs 4096 --num_steps 1000 --warmup_steps 50 \
      --checkpoint $CKPT --seed 42 \
      --benchmark_formatter schema,summary \
      --output_path logs/benchmark_results/play_${TAG} \
      physics=newton_mjwarp
done
```

评测结果保存在 `logs/benchmark_results/play_{own,official}/` 下的 JSON 文件中，主要字段为 `reward`（平均回合奖励）、`ep_length`（平均回合长度，最大 1000）、`success_rate`（任务上报的成功率），三项均只统计评测窗口内已结束的回合。实测结果如下：

任务	模型	reward	ep_length	success_rate
平地	自训 model_299	22.58	980.6	99.54%
平地	官方预训练	21.94	977.0	97.91%
崎岖地形	自训 model_1999	23.06	979.6	99.37%
崎岖地形	官方预训练	17.32	884.1	92.54%

两个任务的自训模型各项指标均达到或超过官方预训练模型。

> 评测须显式指定 checkpoint 文件。`--checkpoint latest` 会选择最新的运行目录，如果之后执行过吞吐基准（会新建一个 10 次迭代的运行目录），`latest` 将指向该目录，导致评测或导出了错误的模型。

训练吞吐基准

执行如下命令测试训练吞吐，参数与官方性能基准一致（8192 环境、10 次迭代）。测试期间请确保 GPU 上无其他训练任务，否则 FPS 不具参考性：

```
cd /workspace/isaaclab
isaaclab benchmark training --rl_library rsl_rl --task $TASK \
  --num_envs 8192 --max_iterations 10 --warmup_steps 50 --seed 42 \
  --benchmark_formatter schema,summary \
  --output_path logs/benchmark_results/training \
  physics=newton_mjwarp
```

实测与官方性能基准对比：

任务	实测 total FPS	官方基准 total FPS	比例
平地	212,487	233,348	91%
崎岖地形	132,603	138,258	96%

仿真资产说明

ANYmal-D 机器人模型、地面等仿真资产在首次运行时自动从 NVIDIA 官方资产库下载，无需手动准备：

- 下载地址：`https://omniverse-content-production.s3-us-west-2.amazonaws.com/Assets/Isaac/6.2`
- 实际下载内容：`anymal_d.usd`（12 KB）、`instanceable_meshes.usd`（1.06 MB）、`default_ground_plane.usda`（3.4 KB），合计约 1.1 MB
- 缓存位置：`/tmp/isaaclab_usd_<随机ID>/`，每次启动重新校验

如开发机无法访问上述地址，可二选一：

```
##### 方式一：使用中国区资产镜像
unset ISAACSIM_ASSET_ROOT
export ISAACSIM_ASSET_REGION_PROFILE=china

##### 方式二：使用自建离线资产目录（目录下需包含 Isaac/ 与 NVIDIA/）
export ISAACSIM_ASSET_ROOT=/path/to/Assets/Isaac/6.2
```

> 如默认地址可访问，请保持默认配置，无需设置上述环境变量。

常见问题

问题	原因与处理
首次启动较慢	首次运行需编译 Warp 内核并下载资产，约 1 分钟；后续复用缓存 `~/.cache/warp/`
play 命令不退出	play 导出策略后会持续回放，按 Ctrl+C 结束；脚本中可使用 `timeout 150 isaaclab play ...`，退出码 124 属正常
评测或导出用错了模型	日志中 `Loading model checkpoint from:` 显示实际加载路径；请显式指定 `model_<N>.pt`，不要使用 `latest`
吞吐 FPS 偏低	检查是否有其他训练同时占用 GPU（`nvidia-smi`）；吞吐测试需独占 GPU
评测结果 `reward` 为 `null`	评测窗口内没有回合结束，不代表为 0，可增大 `--num_steps`
产物丢失	训练产物默认保存在开发机本地盘，释放开发机前请复制到 PFS 等持久化存储

参考

- Isaac Lab 仓库：`https://github.com/isaac-sim/IsaacLab`（tag `v3.0.0`）
- 快速入门：https://isaac-sim.github.io/IsaacLab/main/source/isaaclab/quickstart.html
- 入门教程（Getting started tutorial）：https://isaac-sim.github.io/IsaacLab/main/source/setup/tutorial.html
- 操作指南（How-to guides）：https://isaac-sim.github.io/IsaacLab/main/source/how-to/index.html

- 旧版本（2.1.0 / 2.2.0）使用说明见同目录《具身智能-Isaac Lab.md》

在百舸上使用lingbot-va加速镜像训练视觉-动作模型

在百舸上使用 lingbot-va 加速镜像训练视觉-动作模型

概述

lingbot-va 是基于 Wan2.2（TI2V-5B 系）的视觉-动作（Vision-Action，VA）模型，在 fixed176（LIBERO pick-n-place 系）数据集上做监督微调（SFT）训练。

本镜像面向多机多卡训练场景进行了针对性优化，通过 FSDP 重分片（resharding）、block 级分片粒度、HSDP 二维 mesh 与数据加载调优，消除跨节点 FSDP AllGather 瓶颈，在单机 8 卡下训练吞吐较基线提升约 24.9%，双机 16 卡时吞吐几乎线性翻倍（多机扩展效率约 98.9%）。

适用场景

- 基于 Wan2.2 系列基座模型进行视觉-动作模型微调训练。
- 需要多机多卡分布式训练、对训练扩展效率有要求的场景。
- 使用 fixed176（LIBERO pick-n-place 系）等标准数据集开箱训练，或替换为自有数据集。

环境与资源要求

项目	推荐配置
GPU	支持规格：aihc.lgn7ib；单机 8 卡（推荐），支持双机 16 卡
工作目录	/root/workspace/lingbot-va
运行时环境	镜像内置系统 Python，无需 venv/conda

镜像内置软件栈：

组件	版本
Python	3.12.8
PyTorch	2.8.0+cu128
transformers	4.51.3
accelerate	1.14.0
lerobot	0.3.3
flash_attn	2.8.3
triton	3.4.0

准备工作

创建训练任务

1. 登录百舸平台，进入训练任务创建页。
2. 选择推荐 GPU 资源规格（单机 8 卡或双机 16 卡）。
3. 工作目录设置为 /root/workspace/lingbot-va。
4. 启动命令配置见下文「启动训练」。

准备模型权重 训练权重为 lingbot-va-base（Wan2.2 系），从百舸 BOS 同步到工作目录：

```
bcecmd bos sync "bos:/aihc-models-bj/robbyant/lingbot-va-base/" \
"/root/workspace/models/lingbot-va-base/" --concurrency 16
```

> **路径可改**：train.sh 中 BASE_MODEL_PATH 与 DATASET_PATH 均为 ${VAR:-默认值} 形式，权重与数据集可以放到任意路径（含挂载盘），只需在启动前覆盖对应环境变量，例如：

```
export BASE_MODEL_PATH=/root/workspace/models/lingbot-va-base
export DATASET_PATH=/root/workspace/datasets/pick-n-place-sq-lerobot-v21-fixed176
bcecmd bos sync "bos:/aihc-models-bj/robbyant/lingbot-va-base/" "${BASE_MODEL_PATH}/" --concurrency 16
bash /root/workspace/lingbot-va/train.sh
```

准备数据集 数据集 fixed176（pick-n-place-sq-lerobot-v21-fixed176，约 794M）已随镜像内置到 /root/workspace/datasets/pick-n-place-sq-lerobot-v21-fixed176，无需额外下载。

如需换用自有数据集，通过 DATASET_PATH 环境变量指向实际路径即可，内置副本不会被读取。

启动训练

train.sh 已内置全部优化配置与默认参数，直接执行即可：

```
##### 单机 8 卡
bash /root/workspace/lingbot-va/train.sh

##### 双机 16 卡：两节点各执行一次，拓扑由平台注入环境变量决定
bash /root/workspace/lingbot-va/train.sh
```

单机与多机使用同一套脚本，集群拓扑由平台注入的环境变量决定（NPROC_PER_NODE / WORLD_SIZE=节点数 / RANK=节点序号 / MASTER_ADDR / MASTER_PORT）；单机时 WORLD_SIZE 缺省为 1。

启动脚本 train.sh 完整内容如下（可直接复制为训练任务启动命令）：

```
 ##### !/bin/bash
set -o pipefail

CODE_DIR=${CODE_DIR:-"/root/workspace/lingbot-va"}
DATASET_PATH=${DATASET_PATH:-"/root/workspace/datasets/pick-n-place-sq-lerobot-v21-fixed176"}
BASE_MODEL_PATH=${BASE_MODEL_PATH:-"/root/workspace/models/lingbot-va-base"}
LOG_DIR=${LOG_DIR:-"/root/workspace/logs"}
RUN_TAG=${RUN_TAG:-"opt_66edf60"}

GPUS_PER_NODE=${NPROC_PER_NODE:-8}
NNODES=${WORLD_SIZE:-1}
NODE_RANK=${RANK:-0}
MASTER_ADDR=${MASTER_ADDR:-"127.0.0.1"}
MASTER_PORT=${MASTER_PORT:-"29700"}

export LINGBOT_FSDP_GRANULARITY=block
export LINGBOT_FSDP_RESHARD_AFTER_FORWARD=0
if (( NNODES > 1 )); then
  export LINGBOT_FSDP_SHARDING=hsdp
  export LOCAL_WORLD_SIZE="${GPUS_PER_NODE}"
fi
export PYTHONPATH="${CODE_DIR}:${PYTHONPATH:-}"
export WANDB_DISABLED=true
export WANDB_MODE=disabled
export HF_HUB_OFFLINE=1
export TRANSFORMERS_OFFLINE=1
export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True
export OMP_NUM_THREADS=8
export DATASET_PATH BASE_MODEL_PATH

LOG_FILE="${LOG_DIR}/${RUN_TAG}_$(hostname).log"
mkdir -p "${LOG_DIR}"
cd "${CODE_DIR}" || exit 1

echo "[lingbot-va][topology] nnodes=${NNODES} node_rank=${NODE_RANK} gpus_per_node=${GPUS_PER_NODE} master=${MASTER_ADDR}:${MASTER_PORT} sharding=${LINGBOT_FSDP_SHARDING:-fsdp}"

python -m torch.distributed.run \
  --nnodes="${NNODES}" \
  --node_rank="${NODE_RANK}" \
  --master_addr="${MASTER_ADDR}" \
  --master_port="${MASTER_PORT}" \
  --nproc_per_node="${GPUS_PER_NODE}" \
  -m wan_va.train --config-name=libero_train \
  2>&1 | tee "${LOG_FILE}"
```

训练参数说明 |参数|默认值|说明| |-|-|-| |batch_size|1|训练将 batch 与 frame 展平为统一序列，batch_size 需保持为 1| |gradient_accumulation_steps|10|通过梯度累积等效放大 batch，是调节训练批次的唯一旋钮| |num_steps|50|训练步数，正式训练请按需调整| |load_worker|16|DataLoader 工作进程数| |pin_memory / persistent_workers / prefetch_factor|true / true / 1|数据加载与计算重叠（已由脚本内置）|

> 所有训练参数均通过环境变量传入，**不支持命令行传参**（如 save_interval=20 会被 argparse 直接拒绝并退出）。

性能参考

以下数据为特定测试环境下的参考值（8 卡单机 / 16 卡双机，num_steps=50，batch_size=1、gradient_accumulation_steps=10）。实际性能受资源规格、数据读取方式、网络环境、任务参数等因素影响。

配置	拓扑	step_time	samples/s	峰值显存/GPU
优化版	单机 8 卡	15.018 s	5.327	45507 MiB
优化版	双机 16 卡	15.188 s	10.53	—
基线	单机 8 卡	18.750 s	4.267	42307 MiB

- 加速比（单机 8 卡，优化版 vs 基线）：**1.249x（+19.9%）**。
- 多机扩展效率（双机 vs 单机，均值窗口 step 10-49）：**98.9%**，跨机扩展近乎无损。
- 训练中 loss 正常收敛。

梯度累积步数（GA）标定参考 由于 batch_size 恒为 1，训练批次仅能通过 gradient_accumulation_steps 调节。6 个档位实测流量（参考值）：

GA	step_time (mean)	峰值显存/GPU
1	1.5932 s	26015 MiB
2	3.0878 s	45541 MiB
3	4.5548 s	45545 MiB
4	6.0504 s	45541 MiB
5	7.5408 s	45545 MiB
10	15.0189 s	45525 MiB

注：GA=1 与 GA≥2 之间存在约 19 GiB 的峰值显存台阶，该差异与 GA 取值无关（GA≥2 后显存不再随 GA 增长）；显存敏感场景可优先考虑 GA=1，追求训练效率建议 GA≥2。

常见问题

Q：如何调整训练步数、保存间隔等参数？ 通过环境变量传入，例如：SAVE_INTERVAL / NUM_STEPS / GRADIENT_ACCUMULATION_STEPS / SAVE_ROOT / RESUME_FROM。命令行传参会被直接拒绝（报 unrecognized arguments，训练不启动）。

Q：如何断点续训？ 通过 RESUME_FROM 环境变量指向已保存的 checkpoint 目录（含 transformer/ 与 training_state.pt），续训从断点步数继续、lr 不回 warmup。

```
 ##### 首训：每 20 步保存一次
SAVE_INTERVAL=20 NUM_STEPS=20 SAVE_ROOT=<空目录> bash /root/workspace/lingbot-va/train.sh

##### 续训：从 checkpoint 继续
RESUME_FROM=$SAVE_ROOT/checkpoints/checkpoint_step_20 NUM_STEPS=23 SAVE_ROOT=<另一个目录> bash /root/workspace/lingbot-va/train.sh
```

当前版本为 step 级断点续训：恢复模型权重、全局步数与学习率调度器，**不恢复优化器动量**（优化器从零重建）。

Q：权重/数据集可以放在非默认目录吗？ 可以。train.sh 中 BASE_MODEL_PATH、DATASET_PATH 均为环境变量（含默认值），启动前用环境变量覆盖即可，支持任意挂载路径（含 PFS 等）。

Q：多机训练需要注意什么？ 两节点各执行一次 train.sh 即可。脚本将平台注入的 WORLD_SIZE 当作节点数使用（NNODES=${WORLD_SIZE:-1}），多机时自动开启 HSDP 分片，无需额外配置。

Q：显存不足如何优化？ 确认脚本已生效的 PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True；必要时可通过降低 gradient_accumulation_steps 调节（注意 GA=1 与 GA≥2 之间存在约 19 GiB 显存台阶）。

后续操作

1. 训练日志输出到 ${LOG_DIR}/${RUN_TAG}_$(hostname).log，可通过日志观察训练进度与 STEP_TIMING 指标。
2. 训练完成前确认 $SAVE_ROOT/checkpoints/ 下已正常落盘 transformer/ 权重与 training_state.pt（后者应远小于 1 MB）。
3. 如需替换数据集或权重，通过 DATASET_PATH / BASE_MODEL_PATH 环境变量重新指向后重新启动训练。

在百舸上使用 openpi (JAX) 加速镜像训练 pi0.5 模型

在百舸上使用 openpi (JAX) 加速镜像训练 pi0.5 模型

一、概述

本文介绍如何在百舸平台使用 openpi（JAX 版）pi0.5 训练加速镜像，完成 pi0.5 模型在 LIBERO 数据集上的监督微调（SFT）训练。该加速镜像在保持训练效果与收敛稳定性（等价性）的前提下，通过数据供给优化与计算侧优化，大幅缩短单轮迭代耗时、提升训练吞吐与 GPU 利用率，并已打通单机 8 卡与多机（2 节点）数据并行训练全链路。

openpi 训练框架为 JAX 0.11 + flax nnx 的单控制器 SPMD 数据并行（非 PyTorch DDP / torchrun），由jax.jit + NamedSharding 驱动多卡、XLA 自动插入集合通信，**每节点启动 1 个训练进程**。

二、适用场景

- 在 lgn7ib GPU 资源上训练 / 微调 pi0.5（π0.5）VLA 模型。
- 基于 LIBERO 数据集进行 pi05_libero SFT 训练。
- 需要单机 8 卡高吞吐训练，或多机 2 节点数据并行横向扩展。
- 希望开箱即用、免环境搭建的 JAX 0.11 训练环境。

三、模型简介

- **pi0.5（π0.5）** 是 Physical Intelligence 开源的 VLA（Vision-Language-Action）模型，openpi 为其官方实现（含 JAX / PyTorch 两条路径，本镜像为**JAX 路径**）。
- 结构：PaliGemma VLM（SigLIP 视觉塔 + Gemma LLM）+ action expert，以 flow-matching 生成动作 chunk；gemma 主干 18 层以nn.scan（→ XLA while loop）堆叠。
- 本任务 pi05_libero：在 LIBERO（40 个桌面操作任务）数据上做 SFT，动作 chunk 长度 action_horizon=10。
- 官方仓库：https://github.com/Physical-Intelligence/openpi

四、环境与资源要求

项	说明
GPU 资源	lgn7ib，单卡约 96GB，推荐单机 8 卡；多机场景推荐 2 节点 × 8 卡
代码目录	/root/workspace/openpi
运行时环境	镜像自带 JAX 0.11 栈，开箱即用，无需挂载 venv / conda

五、准备工作

本镜像的训练过程包含进入终端、配置环境变量、软链数据、按脚本启动训练、查看日志等交互式操作，建议在百舸开发机中执行：

1. 在百舸平台创建 / 进入开发机，选择 lgn7ib GPU 资源规格（推荐 8 卡）与上述训练镜像。
2. 打开开发机终端（或 WebIDE 终端），进入代码目录 /root/workspace/openpi。
3. 按后续章节顺序准备数据集、准备模型权重、配置训练环境变量、启动训练。
4. 在开发机中查看训练日志与产物。

六、准备数据集

LIBERO 数据集（lerobot 格式，repo_id=physical-intelligence/libero，1693 episode / 27.3 万帧）。

方式一：从 BOS 下载（推荐）

```
mkdir -p /root/workspace/datasets/libero
bosutil cp -r bos://aihc-rdw-bj/huggingface.co/datasets/physical-intelligence/libero/ /root/workspace/datasets/libero/
```

方式二：从 HuggingFace 下载

```
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
export HF_HOME=/root/workspace/models/huggingface
export HF_HUB_CACHE=$HF_HOME/hub
export TRANSFORMERS_CACHE=$HF_HOME/transformers
export HF_DATASETS_CACHE=$HF_HOME/datasets

huggingface-cli download physical-intelligence/libero --repo-type dataset --local-dir /root/workspace/datasets/libero
```

七、准备模型权重

pi05_base 基座权重（JAX / Orbax 格式）。

方式一：从 BOS 下载（推荐）

```
mkdir -p /root/workspace/models/pi05_base/params
bosutil cp -r bos:/aihc-models-bj/gs/openpi-assets/pi05_base/params/ /root/workspace/models/pi05_base/params/
```

方式二：从 Google Cloud Storage 下载（需联网）

```
##### GCS Orbax 源：gs://openpi-assets/checkpoints/pi05_base/params
```

八、配置训练环境变量

数据与权重不入镜像，运行时通过标准环境变量指定。各集群挂载点不同，用你的实际路径覆盖对应变量即可：

```
export DATA_PATH="${DATA_PATH:-/root/workspace/datasets/libero}"
export BASE_CHECKPOINT_PATH="${BASE_CHECKPOINT_PATH:-/root/workspace/models/pi05_base/params}"

if [[ -z "${HF_HOME:-}" || "${HF_HOME}" == /app/* ]]; then
    export HF_HOME=/root/workspace/openpi/.cache/hf
fi
export HF_LEROBOT_HOME="${HF_LEROBOT_HOME:-${HF_HOME}/lerobot}"
mkdir -p "$HF_LEROBOT_HOME/physical-intelligence"

ln -sfn "$DATA_PATH" "$HF_LEROBOT_HOME/physical-intelligence/libero"
```

数据目录按 lerobot 格式组织：`physical-intelligence/libero/{data,meta,videos}`。norm stats 已随镜像提供于 `assets/`；若更换数据集需重算：

```
python scripts/compute_norm_stats.py --config-name pi05_libero --num-workers 48
```

启动训练前，请确认 GPU 空闲、数据 / 权重已就位、磁盘可用空间充足，并为本次实验使用独立输出目录，避免覆盖历史实验。

九、启动训练

9.1 关键参数说明 |参数|值|说明| |-|-|-| |`--batch-size`|256（默认）|**全局** batch（跨 8 卡）| |`--num-workers`|8（优化）/ 2（默认）|DataLoader worker 数，本负载主瓶颈| |`--fsdp-devices`|1（默认，纯 DP）|>1 走 ZeRO-3 分片，省显存慢 6–8%；追吞吐峰值 / 多机弱扩展时使用| |`--model.max-token-len`|28（优化）/ 200（默认）|libero 实测 max=21，降到 28 可省 FFN 计算| |`--weight-loader.params-path`|`${BASE_CHECKPOINT_PATH}`|base 权重路径| |`--num-train-steps`|30000|总步数|

9.2 优化开关 以下开关默认关闭（等于上游行为），`run_opt.sh` 已默认开启全部优化：

```
export OPENPI_SKIP_VIDEO_DECODE=1            # 跳过未消费的视频解码（占 71% 数据成本）
export OPENPI_SKIP_IMAGES=right_wrist_0_rgb  # 跳过恒 masked 的右腕图 token
export OPENPI_FLASH_ATTN=1 OPENPI_FLASH_IMPL=xla   # flash attention（XLA 实现）
export OPENPI_WORKER_CPU=1                   # 数据 worker 赶到 CPU，不抢 GPU
##### CLI：--num-workers 8 --model.max-token-len 28
##### config-B XLA flag（大 batch 下 +2.9%；OPENPI_STRICT_EQUIV=1 可关这组 flag）：
export JAX_OPTIMIZATION_LEVEL=01
export XLA_FLAGS="--xla_gpu_triton_gemm_any=false --xla_gpu_enable_latency_hiding_scheduler=true --xla_gpu_enable_pipelined_all_gather=true"
```

9.3 启动脚本 容器默认工作目录 `/root/workspace/openpi`，三个脚本均在此目录：

```
##### 基线（上游默认：nw2 / tok200 / 无优化）
bash /root/workspace/openpi/run_baseline.sh
##### 优化（全优化）
bash /root/workspace/openpi/run_opt.sh
##### 追吞吐峰值（大 batch，需 fsdp 分片腾显存）
bash /root/workspace/openpi/run_opt.sh --fsdp-devices 8 --batch-size 2048
```

脚本支持 `EXP_NAME=xxx`、`RESUME=1`、`SAVE_INTERVAL=n` 等环境变量，以及任意 `train.py` 参数透传。

9.4 断点续训

- 设置 `RESUME=1` 时脚本走 `--resume`，由 Orbax CheckpointManager 从 `<checkpoint-base-dir>/pi05_libero/<EXP_NAME>` 恢复完整 `train_state`；未设置时走 `--overwrite`。`--resume` 与 `--overwrite` 互斥。
- checkpoint 路径模板：`<checkpoint-base-dir>/pi05_libero/<EXP_NAME>/<loop_index>`；单个 checkpoint（params + optimizer + EMA）约 **42GB**，`max_to_keep=1` 会清理同一实验中较旧的 checkpoint。
- 存盘语义：`SAVE_INTERVAL=n` 表示每 n 步存一次；训练最后一步必定会存一次 checkpoint。checkpoint 根目录用 `--checkpoint-base-dir <共享存储路径>` 指定，**必须落共享文件系统**。

```
##### 首次训练（存 ckpt）
EXP_NAME=myrun SAVE_INTERVAL=1000 bash /root/workspace/openpi/run_opt.sh --num-train-steps 30000
##### 中断后续训（同名 EXP_NAME + RESUME=1）
EXP_NAME=myrun RESUME=1 bash /root/workspace/openpi/run_opt.sh --num-train-steps 30000
```

十、多机训练（2 节点数据并行）

openpi JAX 为 SPMD 单控制器，**每节点启动 1 个进程**执行 `python scripts/train.py`，该进程经 jax.jit + NamedSharding 驱动本节点全部 8 卡（不是每卡 1 进程）。各节点运行同一条命令，平台注入以下变量：

平台 env	含义	JAX 映射
`MASTER_ADDR` / `MASTER_PORT`	coordinator(rank0) 地址:端口	`coordinator_address`
`RANK`	节点 index（0=coordinator）	`process_id`
`WORLD_SIZE`	**节点数**（非卡数）	`num_processes`
`NPROC_PER_NODE`	每节点卡数	仅信息用，JAX 自动探测本地卡

启动脚本 `run_multinode.sh` 已封装上述变量读取、NCCL / IB 环境与优化配方。

容器 RDMA / RoCE 要求（关键）：多机需容器能做 RDMA 内存注册，创建容器时必须具备以下能力：

```
--network host --ipc host --gpus all \
--device /dev/infiniband --cap-add=IPC_LOCK --ulimit memlock=-1 --ulimit stack=67108864
```

- 缺 `IPC_LOCK` / `memlock=-1` 会报 `ibv_reg_mr_iova2 failed: Cannot allocate memory` 或 `NET/IB: No device found`。
- 若网卡为 RoCE（link_layer=Ethernet），需按需设置 `NCCL_IB_GID_INDEX=3`。
- 集合通信依赖 IB / RoCE，`NCCL_IB_DISABLE=1` 强制走 socket 会导致 rendezvous 超时。

十一、性能参考

> 以下数据为特定测试环境下的参考值。实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

单机 8 卡加速比（bs256 全局 batch，fsdp1）：

配置	step time (s)	samples/s
baseline	20.10	12.73
优化	**1.57**	**163.01**
加速比	**12.8×**	**12.8×**

> baseline 吞吐偏低的根因是上游默认 `num_workers=2` 导致数据供给严重不足（GPU 几乎空等）；优化后转为 compute-bound。优化收益主力来自数据供给（worker 数提升 + worker CPU 隔离 + 跳过无用视频解码），计算侧优化（跳过 masked 图 token / flash-xla / 降低 max-token-len）为锦上添花。

单机吞吐峰值：

配置	samples/s
优化 fsdp1 bs256（默认）	163.01
优化 fsdp8 bs2048（追峰值）	**228.88**

双机 2×8 弱扩展：

配置	samples/s	扩展比
单机 fsdp8 bs2048	228.88	1×（基准）
双机 fsdp16 bs4096	**443.41**	**1.94× / 96.9%**

等价性说明：优化版的 `--model.max-token-len 28` 与 `OPENPI_SKIP_IMAGES=right_wrist_0_rgb` 改变了输入张量形状，带来约 1e-4 的浮点漂移；两者数学上只影响被 mask 的无效 token，收敛终态一致，step0 两者仅差约 1e-4。数据顺序由 `seed=42` 固定、可复现。因此优化前后 loss 收敛曲线一致，训练结果等价。

十二、常见问题

Q1：多机训练 checkpoint 存盘卡在 Waiting for Save Finalize？

checkpoint 目录必须落在所有节点同一路径可见的共享文件系统上。orbax 多进程 finalize 需跨进程协调 rename，若各节点写各自本地盘，finalize 屏障永不达成。请通过 `--checkpoint-base-dir <共享存储路径>` 指定共享存储目录。

Q2：多机启动报 `ibv_reg_mr_iova2 failed` 或 `NET/IB: No device found`？

容器缺少 RDMA 内存注册能力，请按「十、多机训练」中的要求加上 `--device /dev/infiniband --cap-add=IPC_LOCK --ulimit memlock=-1`。

Q3：显存不足（OOM）？

优先使用 `--fsdp-devices >1` 走 ZeRO-3 分片省显存（会慢 6–8%），或适当降低 `--batch-size`。

Q4：离线环境下 lerobot 访问 HuggingFace Hub 报 `Network unreachable`？

设置离线开关并确保数据软链已建立：

```
export HF_HUB_OFFLINE=1
export TRANSFORMERS_OFFLINE=1
```

十三、后续操作

- 训练完成后，checkpoint 落于 `--checkpoint-base-dir` 指定的共享存储目录，可用于后续评测或继续训练。
- 如需长训，建议配置合理的 `SAVE_INTERVAL` 并使用独立的 `EXP_NAME`，便于断点续训与版本管理。
- 更换数据集时，请重新计算 norm stats（见第八章）。

在百舸上使用 Motus 加速镜像训练 VLA 模型

概述

Motus（`thu-ml/Motus`）是统一潜动作世界模型（VLA），模型规模约 8B，由视频生成模块（Wan2.2-5B）、视觉语言理解模块（Qwen3-VL-2B）以及动作专家与理解专家组成。本加速镜像面向 Motus 模型训练场景，已内置完整训练栈与优化后的训练脚本，适配 lgn7ib 芯片，支持单机多卡与多机扩展训练。

本示例以 RoboTwin2.0 仿真数据集（action/state dim 14，video 8 帧 384×320）为例，介绍在百舸平台上完成训练全流程。

适用场景

- 在百舸平台训练 Motus VLA 模型，用于机器人操作仿真（RoboTwin 等）任务。
- 需要单机多卡训练，或双机扩展训练以获得更高吞吐的场景。
- 需要断点续训、性能参考与常见问题指引的客户。

环境与资源要求

项	内容
适配芯片	支持规格：aihc.lgn7ib
源码	已烘入镜像 `/root/workspace/Motus`
镜像环境	自带训练栈（无需 venv / conda）
工作目录	`/root/workspace/Motus`
数据集	RoboTwin2.0（`TianxingChen/RoboTwin2.0`，需转 Motus 格式）
模型权重	Motus(Stage2) + Wan2.2-TI2V-5B + Qwen3-VL-2B-Instruct

在百舸平台准备训练环境

1. 登录百舸平台，创建或进入开发机，选择 lgn7ib 资源规格与上述加速镜像。
2. 打开开发机终端（或 WebIDE 终端），进入工作目录 `/root/workspace/Motus`。
3. 按本章节顺序准备数据与权重，然后启动训练。

准备权重

下载 HuggingFace 资产（配置镜像站） 镜像内不包含权重与数据，训练前需下载以下资产。使用 HuggingFace 下载前先设置镜像站：

```
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
export HF_HOME=/root/workspace/models/huggingface
export HF_HUB_CACHE=$HF_HOME/hub
export TRANSFORMERS_CACHE=$HF_HOME/transformers
export HF_DATASETS_CACHE=$HF_HOME/datasets
```

权重资产清单 |资产|HuggingFace 来源|BOS 路径| |-|-| |权重 Motus（Stage2 finetune）|motus-robotics/Motus|bos://aihc-models-bj/motus-robotics/Motus/| |权重 Motus_Wan2_2_5B_pretrain（Stage1 VGM）|motus-robotics/Motus_Wan2_2_5B_pretrain|待补齐（详见下方说明）| |基础 WAN 视频模型|Wan-AI/Wan2.2-TI2V-5B|bos://aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B/| |基础 VLM|Qwen/Qwen3-VL-2B-Instruct|bos://aihc-models-bj/Qwen/Qwen3-VL-2B-Instruct/|

方式一：HuggingFace 下载

```
huggingface-cli download motus-robotics/Motus --local-dir /root/workspace/models/Motus
huggingface-cli download motus-robotics/Motus_Wan2_2_5B_pretrain --local-dir /root/workspace/models/Motus_Wan2_2_5B_pretrain
huggingface-cli download Wan-AI/Wan2.2-TI2V-5B --local-dir /root/workspace/models/Wan2.2-TI2V-5B
huggingface-cli download Qwen/Qwen3-VL-2B-Instruct --local-dir /root/workspace/models/Qwen3-VL-2B-Instruct
```

方式二：BOS 下载

```
mkdir -p /root/workspace/models/Motus
bosutil cp -r bos://aihc-models-bj/motus-robotics/Motus/ /root/workspace/models/Motus/

mkdir -p /root/workspace/models/Wan2.2-TI2V-5B
bosutil cp -r bos://aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B/ /root/workspace/models/Wan2.2-TI2V-5B/

mkdir -p /root/workspace/models/Qwen3-VL-2B-Instruct
bosutil cp -r bos://aihc-models-bj/Qwen/Qwen3-VL-2B-Instruct/ /root/workspace/models/Qwen3-VL-2B-Instruct/
```

> 说明：`Motus_Wan2_2_5B_pretrain` 权重尚未上传至 BOS，如需离线/内网使用请联系百舸相关人员协助上传。也可以直接通过 HuggingFace 镜像站下载。

准备数据集

数据集 BOS 上是原始 zip，需转成 Motus 格式（videos / qpos / metas / umt5_wan），转换约 3 分钟、使用 1 张卡。转换前需先确保 Wan2.2 权重已就位（见上文权重下载）。

下载数据集

```
##### 方式一：HuggingFace（镜像站）
huggingface-cli download TianxingChen/RoboTwin2.0 --repo-type dataset --local-dir /root/workspace/data/raw/RoboTwin2.0

##### 方式二：BOS
mkdir -p /root/workspace/data/raw/RoboTwin2.0
bosutil cp -r bos://aihc-rdw-bj/huggingface.co/datasets/TianxingChen/RoboTwin2.0/ /root/workspace/data/raw/RoboTwin2.0/
```

下载完成后，原始 zip 位于 `/root/workspace/data/raw/RoboTwin2.0` 目录下，下文以 `/root/workspace/data/raw/RoboTwin2.0` 为原始数据目录。

转换数据集

```
##### 1. 解压成 {split}/{task} 布局
S=/root/workspace/data/staging; mkdir -p $S/clean
unzip -q -o /root/workspace/data/raw/RoboTwin2.0/aloha-agilex_clean_50.zip -d $S/clean
mv $S/clean/aloha-agilex_clean_50 $S/clean/adjust_bottle

##### 2. 转换 + 生成 T5 嵌入（使用 converter 自带的按行 T5）
export CUDA_VISIBLE_DEVICES=0
rm -rf /root/workspace/data/robotwin2/clean/adjust_bottle
cat > /tmp/convert.yml <<EOF
source_root: "/root/workspace/data/staging"
target_root: "/root/workspace/data/robotwin2"
fps: 30
target_width: 320
target_height: 360
max_workers: 16
skip_existing: true
enable_t5_embeddings: true
wan_repo_path: "/root/workspace/models/Wan2.2-TI2V-5B"
t5_max_length: 512
cuda_devices: ["0"]
log_level: "INFO"
save_debug_frames: false
EOF
cd /root/workspace/Motus/data/robotwin2/robotwin_data_convert
python robotwin_converter.py --config /tmp/convert.yml
```

校验转换结果

```
D=/root/workspace/data/robotwin2/clean/adjust_bottle
for d in metas qpos umt5_wan videos; do echo "$d $(ls $D/$d | wc -l)"; done
```

各目录条数应为 49 / 49 / 49 / 50（episode 8 被 converter 跳过，属正常现象）。

配置训练环境变量

镜像内置启动脚本 `/root/workspace/Motus/run_motus_optimized.sh`（accelerate + FSDP），参数均通过环境变量传入。

变量	不设时	说明
DATA_PATH	/root/workspace/data/robotwin2	转换后的数据集
BASE_CHECKPOINT_PATH	/root/workspace/models/Wan2.2-TI2V-5B	Wan2.2 权重
TOKENIZER_PATH	/root/workspace/models/Qwen3-VL-2B-Instruct	VLM 权重
CHECKPOINT_PATH	/root/workspace/models/Motus	Stage2 finetune 权重
CKPT_DIR	/root/workspace/checkpoints	ckpt 目录，多机须为共享盘
BATCH_SIZE	robotwin.yaml（8）	每卡 batch
MAX_STEPS	robotwin.yaml	总步数
SAVE_INTERVAL / SAVE_AT_END	5000 / 1	存盘间隔 / 末步是否存盘
VAL_INTERVAL	500	验证间隔
RESUME_CHECKPOINT_PATH	robotwin.yaml	续训 ckpt 目录
WORLD_SIZE / RANK / MASTER_ADDR	1 / 0 / 无	多机：节点数 / 节点序号 / rank0 IP

启动训练

单机训练（8 卡）

```
export DATA_PATH=/root/workspace/data/robotwin2
export BASE_CHECKPOINT_PATH=/root/workspace/models/Wan2.2-TI2V-5B
export TOKENIZER_PATH=/root/workspace/models/Qwen3-VL-2B-Instruct
export CHECKPOINT_PATH=/root/workspace/models/Motus
export CKPT_DIR=/root/workspace/checkpoints
bash /root/workspace/Motus/run_motus_optimized.sh
```

多机训练（双机 16 卡示例） 除上述变量外，两台节点分别配置分布式环境变量：

```
##### rank0 节点
export WORLD_SIZE=2 RANK=0 MASTER_ADDR=<rank0 IP>

##### rank1 节点
export WORLD_SIZE=2 RANK=1 MASTER_ADDR=<rank0 IP>
```

再各自执行 `bash /root/workspace/Motus/run_motus_optimized.sh` 启动。双机两端需在 15 分钟内都启动；若通过百舸训练任务提交，`WORLD_SIZE` / `RANK` / `MASTER_ADDR` 会自动注入。

NCCL 参数脚本中已有默认值（`mlx5_0` / GID 3 / QPS 8），可用环境变量覆盖调优；多机训练时 `CKPT_DIR` 须为共享盘路径。

性能参考

以下数据为 v1_std-261103.01 镜像（FSDP）实测结果：

场景	samples/s	结果
单机 8 卡，每卡 bs8	47.94	基准 48.30
单机 8 卡，每卡 bs12	53.93	1×
双机 16 卡，每卡 bs12	101.05	扩展比 0.937
长稳 1000 步，每卡 bs8	约 48.4	loss 1.99 → 0.95，无 NaN

> 以上数据为特定测试环境下的参考值。实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

常见问题

数据集转换后各目录条数不是完全一致 转换后各目录条数为 49 / 49 / 49 / 50，其中 episode 8 被 converter 跳过，属正常现象。

双机训练未正常开始 双机两端需在 15 分钟内都启动，否则训练无法正常开始。可通过查看日志确认各节点启动状态。

训练中断后想要续训 参考「续训」章节，设置 `RESUME_CHECKPOINT_PATH` 与 `MAX_STEPS`（总步数）后重新启动即可，会恢复模型、优化器、LR 调度器和随机数状态，从下个 step 继续。

后续操作

保存检查点 `SAVE_INTERVAL=N` 表示每 N 步存一次检查点（`SAVE_AT_END=1` 末步也存），落在 `$CKPT_DIR/robotwin/robotwin/checkpoint_step_<N>/`，单份约 40 GB。

断点续训 其它变量与首次训练相同，添加 `RESUME_CHECKPOINT_PATH`，并将 `MAX_STEPS` 设为总步数：

```
export RESUME_CHECKPOINT_PATH=/root/workspace/checkpoints/robotwin/robotwin/checkpoint_step_1000
export MAX_STEPS=2000
bash /root/workspace/Motus/run_motus_optimized.sh
```

续训从 Step N+1 接着跑。校验方法：日志首条应为 `Step N+1/...`，LR 与不中断训练一致。

> 说明：续训的 loss 不会与不中断训练逐位一致（dataloader 位置不恢复），属正常现象。

参考

- Motus 官方仓库：https://github.com/thu-ml/Motus

在百舸上使用 WSA 加速镜像训练 VLA 模型

概述

WSA（TbotSA VLA）是视觉-语言-动作（VLA）模型：Qwen3-VL-2B 主干 + Cosmos tokenizer + DA3 depth teacher（WSA-Large 版另加 Wan2.2 / Wan2.1），在 RoboTwin-LeRobot 数据集上微调。提供**WSA-Base**、**WSA-Large** 两个规模。本加速镜像针对 RTX Pro6000（sm_120）芯片做训练加速，已内置完整训练栈与优化后的训练脚本，支持单机多卡与多机扩展训练。

本示例以 RoboTwin-LeRobot-v3.0 仿真数据集为例，介绍在百舸平台上完成训练全流程。

适用场景

- 在百舸平台训练 WSA VLA 模型，用于机器人操作仿真（RoboTwin 等）任务
- 需要单机多卡训练，或双机扩展训练以获得更高吞吐的场景
- 需要断点续训、性能参考与常见问题指引的客户

环境与资源要求

项	内容
适配芯片	支持规格：aihc.lgn7ib
源码	已烘入镜像 `/root/workspace/WSA`
镜像环境	自带训练栈（无需 venv / conda）
工作目录	/root/workspace/WSA
数据集	RoboTwin-LeRobot-v3.0
模型权重	WSA-Base、WSA-Large、Qwen3-VL-2B-Instruct、Cosmos-0.1-Tokenizer-CI8x8、DA3-LARGE-1.1、Wan2.2-TI2V-5B、Wan2.1-T2V-1.3B

同一镜像通过开关切换即得 baseline / optimized：内置 `run_wsa_base_optimized.sh`（优化全开）/ `run_wsa_base_baseline.sh`（对照）+ 续训参数透传。

在百舸平台准备训练环境

1. 登录百舸平台，创建或进入开发机，选择 RTX Pro6000 资源规格与上述加速镜像
2. 打开开发机终端（或 WebIDE 终端），进入工作目录 `/root/workspace/WSA`
3. 按本章节顺序准备数据与权重，然后启动训练

准备数据与权重

数据集与权重**不打包进镜像**，训练前需下载/挂载到 `/root/workspace` 下（默认部署路径见下表，均可用环境变量整体覆盖到 PFS，见「启动训练 · 参数说明」）。HF 为公开参考地址，实际下载以 BOS 为准。

资产清单 |物料|HuggingFace 地址|BOS 地址|部署路径（默认，可 env 覆盖）| |-|-|-| |RoboTwin-LeRobot-v3.0（数据集）|hxma/RoboTwin-LeRobot-v3.0|bos://aihc-rdw-bj/huggingface.co/datasets/hxma/RoboTwin-LeRobot-v3.0/|/root/workspace/data/RoboTwin-LeRobot-v3.0（DATA_PATH）| |WSA-Base（主权重）|zaleni/WSA-Base|bos://aihc-models-bj/zaleni/WSA-Base/|/root/workspace/models/WSA-Base（CHECKPOINT_PATH）| |WSA-Large（主权重）|zaleni/WSA-Large|bos://aihc-models-bj/zaleni/WSA-Large/|/root/workspace/models/WSA-Large| |Qwen3-VL-2B-Instruct|Qwen/Qwen3-VL-2B-Instruct|bos://aihc-models-bj/Qwen/Qwen3-VL-2B-Instruct/|/root/workspace/models/Qwen3-VL-2B-Instruct（QWEN_VL_PATH）| |Cosmos-0.1-Tokenizer-CI8x8|nvidia/Cosmos-0.1-Tokenizer-CI8x8|bos://aihc-models-bj/nvidia/Cosmos-0.1-Tokenizer-CI8x8/|/root/workspace/models/Cosmos-0.1-Tokenizer-CI8x8（COSMOS_TOKENIZER_PATH）| |DA3-LARGE-1.1|depth-anything/DA3-LARGE-1.1|bos://aihc-models-bj/depth-anything/DA3-LARGE-1.1/|/root/workspace/models/DA3-LARGE-1.1（DA3_PATH）| |Wan2.2-TI2V-5B（WSA-Large 用）|Wan-AI/Wan2.2-TI2V-5B|bos://aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B/|/root/workspace/models/Wan2.2-TI2V-5B| |Wan2.1-T2V-1.3B（WSA-Large 用）|Wan-AI/Wan2.1-T2V-1.3B|bos://aihc-models-bj/Wan-AI/Wan2.1-T2V-1.3B/|/root/workspace/models/Wan2.1-T2V-1.3B|

方式一：HuggingFace 下载 使用 HuggingFace 下载前先设置镜像站：

```
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
export HF_HOME=/root/workspace/models/huggingface
```

```
##### 数据集
huggingface-cli download hxma/RoboTwin-LeRobot-v3.0 --repo-type dataset --local-dir /root/workspace/data/RoboTwin-LeRobot-v3.0
##### 权重
huggingface-cli download zaleni/WSA-Base --local-dir /root/workspace/models/WSA-Base
huggingface-cli download zaleni/WSA-Large --local-dir /root/workspace/models/WSA-Large
##### ...（Qwen3-VL-2B-Instruct / Cosmos-0.1-Tokenizer-CI8x8 / DA3-LARGE-1.1 / Wan2.2-TI2V-5B / Wan2.1-T2V-1.3B 同理）
```

方式二：BOS 下载

```
mkdir -p /root/workspace/data/RoboTwin-LeRobot-v3.0
bosutil cp -r bos://aihc-rdw-bj/huggingface.co/datasets/hxma/RoboTwin-LeRobot-v3.0/ /root/workspace/data/RoboTwin-LeRobot-v3.0/

mkdir -p /root/workspace/models/WSA-Base
bosutil cp -r bos://aihc-models-bj/zaleni/WSA-Base/ /root/workspace/models/WSA-Base/

##### ...（其余权重同理，路径见上方资产清单）
```

训练前预处理（生成 repo 列表 + norm_stats，每套只需一次）

在 `/root/workspace/WSA` 下执行。

WSA-Base

```
##### !/bin/bash
DATA_PATH="${DATA_PATH:-/root/workspace/data/RoboTwin-LeRobot-v3.0}"
find -L "${DATA_PATH}" -path "*/meta/info.json" -print \
  | while read -r info; do dirname "$(dirname "$info")"; done \
  | sort -u > robotwin_repo_ids.txt
python tools/compute_norm_stats_multi.py \
  --repo_id_file robotwin_repo_ids.txt \
  --action_mode delta --chunk_size 50 --num_workers 8 \
  --output_path ./norm_stats/aloha/delta/stats.json
```

> 产物落在相对路径 ./norm_stats/aloha/delta/stats.json，与 NORM_STATS_ROOT 默认值对应。若把 NORM_STATS_ROOT 改到别处（如 PFS），须先把该文件复制到 ${NORM_STATS_ROOT}/aloha/delta/stats.json，否则训练在 make_dataset 报 use_external_stats=True but no external stats file was found。

WSA-Large（另加 text_embeds 预计算）

```
##### !/bin/bash
DATA_PATH="${DATA_PATH:-/root/workspace/data/RoboTwin-LeRobot-v3.0}"
WAN_MODEL_PATH="${WAN_MODEL_PATH:-/root/workspace/models/Wan2.2-TI2V-5B}"
WAN_TOKENIZER_PATH="${WAN_TOKENIZER_PATH:-/root/workspace/models/Wan2.1-T2V-1.3B}"
python tools/discover_robotwin_repos.py \
  --robotwin-root ${DATA_PATH} \
  --output-file robotwin_repo_ids_large.txt --require-three-cameras true
python tools/precompute_text_embeds.py \
  --model-id ${WAN_MODEL_PATH} --tokenizer-model-id ${WAN_TOKENIZER_PATH} \
  --repo-id-file robotwin_repo_ids_large.txt \
  --text-embedding-cache-dir outputs/WSA_Large/text_embeds/robotwin \
  --redirect-common-files false --device cuda
python tools/compute_norm_stats_multi.py \
  --repo_id_file robotwin_repo_ids_large.txt \
  --action_mode delta --chunk_size 32 --num_workers 8 \
  --output_path ./norm_stats_large/aloha/delta/stats.json
```

实测总耗时约 25 分钟（text embeds 约 8.6 min、产物 24G；norm stats 约 16.6 min）。产物 `outputs/WSA_Large/text_embeds/`、`norm_stats_large/` 为相对 `/root/workspace/WSA` 的路径、无环境变量入口，要落 PFS 需把这两个目录 bind mount 进容器。Large 的 stats（chunk_size 32）与 Base（chunk_size 50）不可混用。

启动训练

参数说明 优化版 launcher 已把数据/权重/输出/命名全部做成**环境变量**，默认落 `/root/workspace`，可整体覆盖到 PFS。

环境变量	默认值	说明
DATA_PATH	/root/workspace/data/RoboTwin-LeRobot-v3.0	RoboTwin 数据集（底层 ROBOTWIN_ROOT）
CHECKPOINT_PATH	/root/workspace/models/WSA-Base	WSA-Base 主权重（底层 POLICY_INIT_PATH）
QWEN_VL_PATH	/root/workspace/models/Qwen3-VL-2B-Instruct	Qwen3-VL 主干
COSMOS_TOKENIZER_PATH	/root/workspace/models/Cosmos-0.1-Tokenizer-CI8x8	Cosmos tokenizer
DA3_PATH	/root/workspace/models/DA3-LARGE-1.1	DA3 depth teacher
OUTPUT_DIR	/root/workspace/WSA/outputs	输出根（含 checkpoints）
NORM_STATS_ROOT	/root/workspace/WSA/norm_stats	norm_stats 目录

优化开关（run_wsa_base_optimized.sh 默认全开；run_wsa_base_baseline.sh 全关，用于对照/复现加速比）：COMPILE_MODEL / COMPILE_DA3_TEACHER_BACKBONE / OPTIMIZER_FUSED / FREEZE_PERMANENTLY_UNUSED_PARAMS（=true）、ATTENTION_BACKEND=sdpa、LEROBOT_DDP_BUCKET_CAP_MB=600（baseline=25）。多机由 NODE_COUNT / NODE_RANK / MASTER_ADDR / MASTER_PORT 注入。

覆盖到 PFS 示例：

```
DATA_PATH=/pfs/pfs-0qLB2M/dataset/RoboTwin-LeRobot-v3.0 CHECKPOINT_PATH=/pfs/pfs-0qLB2M/models/zaleni/WSA-Base OUTPUT_DIR=/pfs/<你的目录>/outputs bash run_wsa_base_optimized.sh
```

WSA-Base 训练（单机 / 多机自适应） 镜像内置 `/root/workspace/WSA/run_wsa_base_optimized.sh`（优化全开）与 `run_wsa_base_baseline.sh`（全关，对照）：

```
##### !/usr/bin/env bash
##### WSA-Base 优化版 launcher（单机/多机自适应）—— 优化项全开
##### 打开：COMPILE_MODEL / COMPILE_DA3_TEACHER_BACKBONE / OPTIMIZER_FUSED /
##### FREEZE_PERMANENTLY_UNUSED_PARAMS / LEROBOT_DDP_BUCKET_CAP_MB=600（sdpa 同 baseline）。
##### 覆盖到 PFS 例：DATA_PATH=/pfs/... CHECKPOINT_PATH=/pfs/... ... bash run_wsa_base_optimized.sh
##### 续训（须 FREEZE=false）：FREEZE_PERMANENTLY_UNUSED_PARAMS=false bash run_wsa_base_optimized.sh \
##### --resume=true --config_path=<OUTPUT_DIR>/.../checkpoints/last/pretrained_model/train_config.json
set -euo pipefail
cd "$(dirname "${BASH_SOURCE[0]}")"

DATA_PATH="${DATA_PATH:-/root/workspace/data/RoboTwin-LeRobot-v3.0}"
CHECKPOINT_PATH="${CHECKPOINT_PATH:-/root/workspace/models/WSA-Base}"
QWEN_VL_PATH="${QWEN_VL_PATH:-/root/workspace/models/Qwen3-VL-2B-Instruct}"
COSMOS_TOKENIZER_PATH="${COSMOS_TOKENIZER_PATH:-/root/workspace/models/Cosmos-0.1-Tokenizer-CI8x8}"
DA3_PATH="${DA3_PATH:-/root/workspace/models/DA3-LARGE-1.1}"
OUTPUT_DIR="${OUTPUT_DIR:-/root/workspace/WSA/outputs}"
NORM_STATS_ROOT="${NORM_STATS_ROOT:-/root/workspace/WSA/norm_stats}"

export PROC_PER_NODE="${PROC_PER_NODE:-$(nvidia-smi -L | wc -l)}"
export NODE_COUNT="${NODE_COUNT:-1}"
export NODE_RANK="${NODE_RANK:-0}"
export MASTER_ADDR="${MASTER_ADDR:-127.0.0.1}"
export MASTER_PORT="${MASTER_PORT:-6379}"
export HF_HUB_OFFLINE="${HF_HUB_OFFLINE:-1}"
export LEROBOT_DDP_BUCKET_CAP_MB="${LEROBOT_DDP_BUCKET_CAP_MB:-600}"

USE_DIST_LOADING="${USE_DIST_LOADING:-true}" \
BATCH_SIZE="${BATCH_SIZE:-8}" \
GRADIENT_ACCUMULATION_STEPS="${GRADIENT_ACCUMULATION_STEPS:-4}" \
COMPILE_MODEL="${COMPILE_MODEL:-true}" \
COMPILE_MODE="${COMPILE_MODE:-default}" \
COMPILE_DA3_TEACHER_BACKBONE="${COMPILE_DA3_TEACHER_BACKBONE:-true}" \
COMPILE_REGIONAL_MODULES="${COMPILE_REGIONAL_MODULES:-false}" \
OPTIMIZER_FUSED="${OPTIMIZER_FUSED:-true}" \
FREEZE_PERMANENTLY_UNUSED_PARAMS="${FREEZE_PERMANENTLY_UNUSED_PARAMS:-true}" \
ATTENTION_BACKEND="${ATTENTION_BACKEND:-sdpa}" \
STEPS="${STEPS:-200000}" \
WARMUP_STEPS="${WARMUP_STEPS:-20}" \
LOG_FREQ="${LOG_FREQ:-10}" \
SAVE_FREQ="${SAVE_FREQ:-10000}" \
POLICY_INIT_PATH="${CHECKPOINT_PATH}" \
QWEN3_VL_PRETRAINED_PATH="${QWEN_VL_PATH}" \
COSMOS_TOKENIZER_PATH_OR_NAME="${COSMOS_TOKENIZER_PATH}" \
DA3_MODEL_PATH_OR_NAME="${DA3_PATH}" \
ROBOTWIN_ROOT="${DATA_PATH}" \
BASE_OUTPUT_DIR="${OUTPUT_DIR}" \
ACTION_TYPE=delta \
USE_EXTERNAL_STATS=true \
DATASET_EXTERNAL_STATS_ROOT="${NORM_STATS_ROOT}" \
LOG_DA3_TEACHER_TIMING=true \
bash launch/wsa_base_finetune_robotwin.sh "$@"
```

baseline 版（run_wsa_base_baseline.sh）结构相同，仅 6 处开关不同：LEROBOT_DDP_BUCKET_CAP_MB=25、COMPILE_MODEL=false、COMPILE_DA3_TEACHER_BACKBONE=false、OPTIMIZER_FUSED=false、FREEZE_PERMANENTLY_UNUSED_PARAMS=false（ATTENTION_BACKEND=sdpa 与路径/多机段一致）。

调用：

```
bash run_wsa_base_optimized.sh                                          # 单机 8 卡
NODE_COUNT=2 NODE_RANK=0 MASTER_ADDR=<rank0内网IP> MASTER_PORT=6379 bash run_wsa_base_optimized.sh  # 双机 rank0
NODE_COUNT=2 NODE_RANK=1 MASTER_ADDR=<rank0内网IP> MASTER_PORT=6379 bash run_wsa_base_optimized.sh  # 双机 rank1（先起 rank1）
```

WSA-Large 训练 训练脚本：

```
##### !/bin/bash

##### ==================== 路径配置 ====================
DATA_PATH="${DATA_PATH:-/root/workspace/data/RoboTwin-LeRobot-v3.0}"
CHECKPOINT_PATH="${CHECKPOINT_PATH:-/root/workspace/models/WSA-Large}"
QWEN_VL_PATH="${QWEN_VL_PATH:-/root/workspace/models/Qwen3-VL-2B-Instruct}"
COSMOS_TOKENIZER_PATH="${COSMOS_TOKENIZER_PATH:-/root/workspace/models/Cosmos-0.1-Tokenizer-CI8x8}"
DA3_PATH="${DA3_PATH:-/root/workspace/models/DA3-LARGE-1.1}"
WAN_MODEL_PATH="${WAN_MODEL_PATH:-/root/workspace/models/Wan2.2-TI2V-5B}"
WAN_TOKENIZER_PATH="${WAN_TOKENIZER_PATH:-/root/workspace/models/Wan2.1-T2V-1.3B}"

##### ==================== 训练启动 ====================
##### 数据加载
NUM_WORKERS="${NUM_WORKERS:-16}" \
USE_DIST_LOADING="${USE_DIST_LOADING:-true}" \
BATCH_SIZE="${BATCH_SIZE:-6}" \
GRAD_ACCUM_STEPS="${GRAD_ACCUM_STEPS:-2}" \
\
COMPILE_MODEL="${COMPILE_MODEL:-false}" \
COMPILE_DA3_TEACHER_BACKBONE="${COMPILE_DA3_TEACHER_BACKBONE:-true}" \
COMPILE_REGIONAL_MODULES="${COMPILE_REGIONAL_MODULES:-true}" \
WSA_MOT_CKPT_LAYERS="${WSA_MOT_CKPT_LAYERS:-12}" \
WSA_GC_FREEZE="${WSA_GC_FREEZE:-1}" \
WSA_TS9="${WSA_TS9:-1}" \
WSA_MOT_BLOCK_SPLIT_ATTN="${WSA_MOT_BLOCK_SPLIT_ATTN:-1}" \
OPTIMIZER_FUSED="${OPTIMIZER_FUSED:-true}" \
WSA_VAE_COMPILE_MODE="${WSA_VAE_COMPILE_MODE:-max-autotune-no-cudagraphs}" \
ATTENTION_BACKEND="${ATTENTION_BACKEND:-sdpa}" \
WARMUP_STEPS="${WARMUP_STEPS:-40}" \
\
POLICY_INIT_PATH="${CHECKPOINT_PATH}" \
QWEN3_VL_PRETRAINED_PATH="${QWEN_VL_PATH}" \
COSMOS_TOKENIZER_PATH_OR_NAME="${COSMOS_TOKENIZER_PATH}" \
DA3_MODEL_PATH_OR_NAME="${DA3_PATH}" \
ROBOTWIN_ROOT="${DATA_PATH}" \
WAN_MODEL_ID="${WAN_MODEL_PATH}" \
WAN_TOKENIZER_MODEL_ID="${WAN_TOKENIZER_PATH}" \
\
ACTION_TYPE=delta \
NORMALIZATION_STATS_PATH=./norm_stats_large/aloha/delta/stats.json \
TEXT_EMBED_CACHE_DIR=outputs/WSA_Large/text_embeds/robotwin \
LOAD_TEXT_ENCODER=false \
SKIP_DIT_LOAD_FROM_PRETRAIN=true \
LOG_DA3_TEACHER_TIMING=true \
bash launch/wsa_large_finetune_robotwin.sh
```

注意：

- 步数用 `TRAIN_MAX_STEPS=n` 控制。Large 下 `STEPS` 不生效，会被 `NUM_EPOCHS=5` 改写（实际约 31.6 万步）
- 脚本中的 `QWEN_VL_PATH` / `COSMOS_TOKENIZER_PATH` 底层 launcher 未引用，Large 不需要准备这两份权重

性能参考

以下数据为 RTX Pro6000 ×8 实测（STEPS=60，取 step20–60 稳态；同一镜像开关关闭 = baseline、打开 = optimized）。

WSA-Base 单机加速比 |配置|稳态 update|吞吐（全局）|加速比|da3 teacher| |-|-|-|-| |baseline（全关，sdpa）|~3.18s|80.5 samples/s|1.00×|0.132s| |optimized（全开）|~2.61s|≈98.0 samples/s|≈**1.22×**|0.102s|

优化版峰值显存 ≈**94.5 GB**（近 97.9 GB 上限，主要来自 torch.compile inductor autotuning 首步瞬时占用）。

小显存卡/更大 batch 时建议关 `COMPILE_MODEL` 或设 `PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True`。（sdpa 为两版共用功能基线，其收益不计入此 1.22×。）

WSA-Base 双机 2×8 扩展 双机 2×8（16 卡，effective batch = 8 × 16 × 4 = 512，取 step30–60 稳态）：稳态 ~2.96s/update，聚合吞吐**172.9 samples/s**（约 10.8/卡），相对单机 optimized（98.0 samples/s）**8→16 扩展比 1.76×、并行效率 88%**。

每卡 batch 16 档（GA=4 不变，其余开关同 optimized，单机 batch 扫描 10/12/14/16 均不 OOM，batch 16 吞吐最高；多机训练建议 `BATCH_SIZE=16`）：

配置	effective batch	稳态 update	吞吐（全局）	峰值显存
单机 8 卡	8 × 16 × 4 = 512	4.733s	108.2 samples/s	91.3 GiB/卡
双机 2×8	16 × 16 × 4 = 1024	5.000s	204.8 samples/s	91.4 GiB/卡

8→16 扩展比 **1.89×、并行效率 94.6%**。

WSA-Large 单机 |配置|稳态 update|吞吐（全局）|峰值显存|loss（step10→60）| |-|-|-|-|-| |`train_large.sh` 默认（8×6×2=96）|1.700s|56.47 samples/s|86.3 GiB/卡|4.464 → 2.807|

> 以上数据为特定测试环境下的参考值。实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

续训

> `FREEZE_PERMANENTLY_UNUSED_PARAMS=true`（优化版默认）会硬拒 resume，续训须置为 `false`。

WSA-Base 续训

```
##### 1) 出 checkpoint（FREEZE=false）
FREEZE_PERMANENTLY_UNUSED_PARAMS=false bash run_wsa_base_optimized.sh
##### 2) 从断点续训
FREEZE_PERMANENTLY_UNUSED_PARAMS=false bash run_wsa_base_optimized.sh --resume=true --config_path=<OUTPUT_DIR>/<POLICY>/<JOB>/checkpoints/last/pretrained_model/train_config.json
```

续训时总步数需与首训一致：学习率调度按首训步数保存，调大步数后 lr 会停在调度终点。`FREEZE=false` 稳态 ≈91 samples/s（≈1.13×），比 `FREEZE=true`（98）降约 7%。

WSA-Large 续训

```
RESUME=true RESUME_CHECKPOINT_DIR=<OUTPUT_DIR>/<JOB>/checkpoints/<step> TRAIN_MAX_STEPS=<与首训相同> bash train_large.sh
```

续训时**总步数须与首训一致**：调大会使学习率调度沿原 cosine 回爬。

参考

- WSA 官方仓库：https://github.com/zaleni/WSA#framework

在百舸上使用 Mind2Act 评测镜像评测视觉-动作模型

在百舸上使用 Mind2Act 评测镜像评测视觉-动作模型

概述

Mind2Act 是面向「想&做」协同的跨架构诊断 Benchmark，提供 30 个双臂操作任务、90 种任务配置与 4,500 条专家示范。平台分别检验模型的「想」（Mind）与「做」（Act），再检验二者能否协同（Mind2Act），并为每个被测模型生成一张诊断卡，说明短板所在、失败发生在哪一步、下一步该补什么。

本镜像打包了社区版评测代码与完整运行环境：任务在 Isaac Sim 5.1 中由双臂 ARX X5 执行，分 old15、new15 两套 release，每个任务有 easy/medium/hard 三档配置，clean 模式每档 50 个冻结实例。镜像内置一键评测脚本，在百舸上申请多卡即可并行跑评测。

详见官网：https://mind2act-world.com/index.html

适用场景

- 对视觉-动作（VLA / 具身智能）模型做标准化、可复现的仿真评测与诊断。
- 需要多卡、多任务并行跑完整 Benchmark（4,500 实例）的场景。
- 通过 adapter 接入自有模型（端到端关节控制接口），在统一预算下与他人结果对齐比较。

环境与资源要求

项目	说明
镜像地址	ccr-registry.baidubce.com/aihc/mind2act:torch270_cuda128_sim5-1_20261110
代码目录	/root/workspace/mind2act-eval
评测脚本目录	/root/workspace/mind2act-scripts（一键入口 run.sh，另有 run_batch.sh、list_cases.py、summary.py）
运行时环境	Isaac Sim 5.1.0，入口 /root/isaac-sim/python.sh
GPU	支持图像渲染的卡型

镜像内置软件栈：

组件	版本
Python	3.11.13
PyTorch	2.7.0+cu128（CUDA 12.8）
torch 编译架构	sm_75 / 80 / 86 / 90 / 100 / 120
ffmpeg	6.1.1
其他	Vulkan / GL 运行库；/root/venvs/hf-dl 内置 huggingface_hub 2.4.0（用于下载资产）

> 镜像内不含任何凭证（HF token、模型 API key 等），运行时通过任务环境变量传入。镜像已清空着色器与 Warp 缓存，每种 GPU 首次启动 Isaac 会重新编译，首局偏慢属正常现象。 镜像登录 shell 会自动加载 /etc/profile.d/mind2act.sh，预置以下环境变量（一般无需改动）：

```
export ISAAC_PYTHON=/root/isaac-sim/python.sh
export M2A_CODE_ROOT=/root/workspace/mind2act-eval
export M2A_PYTHON_OVERLAY=/root/workspace/mind2act-overlay
export MIND2ACT_ASSET_ROOT=/mnt/pfs/datasets/Mind2act/mind2act-assets
export PYTHONNOUSERSITE=1
export PYTHONPATH="$M2A_CODE_ROOT/src:$M2A_PYTHON_OVERLAY${PYTHONPATH:+:$PYTHONPATH}"
```

准备工作

下载评测资产

评测资产约 70 GB，不打进镜像，需先下载到 PFS，再把 PFS 挂到评测环境的 /mnt/pfs。默认路径 /mnt/pfs/datasets/Mind2act/mind2act-assets，放在其他位置时修改环境变量 MIND2ACT_ASSET_ROOT。

从 BOS 下载（推荐）：百舸 BOS 上有一份同步自 HuggingFace 的镜像，且无需 HF token：

```
##### !/usr/bin/env bash
set -eo pipefail
##### 需已配置 bcecmd 凭证（bcecmd -c 或 ~/.go-bcecli/credentials）
A=/mnt/pfs/datasets/Mind2act/mind2act-assets
mkdir -p "$A"
bcecmd bos cp bos:/aihc-rdw-bj/huggingface.co/datasets/Mind2act/mind2act-assets/ "$A/"
  --recursive --yes --disable-bar
```

从 HuggingFace 下载（Mind2act/mind2act-assets 为私有仓库，需单独申请读权限）：

```
##### !/usr/bin/env bash
set -eo pipefail
##### HF_TOKEN 需对 Mind2act/mind2act-assets 有读权限，从环境变量传入，不要写进脚本
: "${HF_TOKEN:?export HF_TOKEN first}"
A=/mnt/pfs/datasets/Mind2act/mind2act-assets
mkdir -p "$A"
HF_HUB_DISABLE_PROGRESS_BARS=1 /root/venvs/hf-dl/bin/hf download Mind2act/mind2act-assets
  --repo-type dataset --revision v1.0.4
  --local-dir "$A" --max-workers 16
```

> 单 token 并发过高会被 HF 限流（429），下载中断时重跑同一条命令即可续传。 **资产自检**

下载完成后，先做资产完整性与「代码—资产」配对校验，两项都通过才可使用：

```
##### !/usr/bin/env bash
set -eo pipefail
export PYTHONPATH="${PYTHONPATH:-}"
. /etc/profile.d/mind2act.sh
A=$MIND2ACT_ASSET_ROOT; K=${OUTPUT_PATH:-/root/workspace/outputs}/checks; mkdir -p $K

##### 1. 资产完整性：逐文件比对 MANIFEST.json 的大小和 SHA-256
$ISAAC_PYTHON $A/tools/verify_manifest.py > $K/verify_manifest.log 2>&1
tail -c 300 $K/verify_manifest.log   # 期望 "status": "passed", "findings": []

##### 2. 代码与资产配对
cd $M2A_CODE_ROOT
$ISAAC_PYTHON tools/verify_release_pair.py --asset-root $A > $K/verify_release_pair.log 2>&1
tail -c 300 $K/verify_release_pair.log   # 期望 "paired_files_verified": true

##### 3. 实例输入门检查（CPU，不启动物理）
$ISAAC_PYTHON $A/tools/check_native_input_gates.py $K/native-input-gates.json > $K/gates.log 2>&1
tail -c 300 $K/gates.log   # 期望 "cases": 900, "passed": 900
```

GPU 渲染自检

不依赖资产，确认 Isaac、CUDA、RTX 渲染与物理均可用。换新型号 GPU（如 3090/4090）时建议先跑一次：

```
. /etc/profile.d/mind2act.sh
$ISAAC_PYTHON -c "import torch;print(torch.cuda.get_device_name(0),torch.cuda.get_device_capability(0))"
##### 随后用镜像内自检脚本渲染一帧，期望输出非空图像、方块正常落地
```

> 冷缓存下首次运行可能拿到空帧，重跑一次即可。 **开通 AgentOS（使用 duclaw adapter 时）**

若用镜像内置的 duclaw adapter 接入被测模型，需先在百度智能云 AgentOS 文档「API 管理」 开通服务，获取 BaseURL、创建 API Key。开通后系统自动创建一个默认 API Key（不可禁用、不可更换），也可在 AgentOS 助手页面按用途另建 Key。

- **API Key**：设置到百舸任务的环境变量 `MIND2ACT_API_KEY`，不要写进配置文件或启动命令。
- **BaseURL**：与默认地址 `https://duclaw-gateway.baidubce.com/star/openai/v1` 不同时，写入 adapter 配置的 `base_url`，详见「接入被测模型」。

开始评测

统一用一键脚本 `/root/workspace/mind2act-scripts/run.sh` 启动。它会先检查环境（输出目录可写、资产与镜像代码配对一致、所选 GPU 全部空闲），再用评测框架在指定卡上并行跑实例，最后汇总 `infra_valid`、`termination` 计数与缺失实例，明细写到 `<OUTPUT_PATH>/summary.csv`。

> 百舸任务中把 PFS 挂到 `/mnt/pfs`，用 `OUTPUT_PATH` 指定 PFS 上的输出目录（如 `/mnt/pfs/<your-dir>/m2a`）。无需挂载 `/shared`：环境中没有 `/shared` 时 `run.sh` 自动建软链 `/shared → OUTPUT_PATH`；若环境中已有 `/shared`，则 `OUTPUT_PATH` 必须设在其下。 **常用变量**

以下变量均可不设，取默认值即可跑通：

变量	默认值	说明
POLICY_KIND	空	空表示原始预算，仅用于环境检查；被测模型用 e2e 或 coding_agent
ADAPTER	mind2act_eval.adapters.hold:create	被测模型的 adapter（module:create）。hold 让机器人保持不动，仅用于环境检查；duclaw 用 mind2act_llm_e2e.duclaw:create
ADAPTER_CONFIG	空	adapter 的 JSON 配置文件路径，运行时按 sha256 校验
CASE_ID	空	只跑这些实例，逗号分隔
TASKS / LEVELS	空（全部）	逗号分隔，如 PR1,DR2、easy,medium
GPUS	0	本 Pod 内用哪些卡，逗号分隔（如 0,1,2,3）。每张卡同时跑 1 局，最多并行 6 局
SHARD / NSHARD	0 / 1	多个百舸任务分片，见「多任务多卡」
OUTPUT_PATH	/root/workspace/outputs	输出根。默认值在容器本地、任务结束即丢失，百舸任务中应设为 PFS 目录
ASSET_ROOT	空	资产不在默认路径时设置

> 模型 API key 等凭证用任务环境变量传入，不写进脚本。不同 `POLICY_KIND` 的结果不能混在一起比较，换协议时换一个 `OUTPUT_PATH`。 **单任务单卡**

```
##### 环境检查：hold，原始预算，只跑一局
OUTPUT_PATH=/mnt/pfs/<your-dir>/m2a-hold CASE_ID=PR1_easy_0c7134bad55b7c6c
 /root/workspace/mind2act-scripts/run.sh

##### duclaw，e2e 预算；MIND2ACT_API_KEY 已在任务环境变量中设置
OUTPUT_PATH=/mnt/pfs/<your-dir>/m2a-duclaw CASE_ID=PR1_easy_0c7134bad55b7c6c GPUS=0
 POLICY_KIND=e2e ADAPTER=mind2act_llm_e2e.duclaw:create ADAPTER_CONFIG=/mnt/pfs/<your-dir>/duclaw.json
 /root/workspace/mind2act-scripts/run.sh
tail -f /mnt/pfs/<your-dir>/m2a-duclaw/logs/worker.log
```

单任务多卡

一个任务×难度的全部实例分到多张卡并行，每张卡同时跑 1 局（单局不跨卡）：

```
OUTPUT_PATH=/mnt/pfs/<your-dir>/m2a-duclaw TASKS=PR1 LEVELS=easy GPUS=0,1,2,3
 POLICY_KIND=e2e ADAPTER=mind2act_llm_e2e.duclaw:create ADAPTER_CONFIG=/mnt/pfs/<your-dir>/duclaw.json
 /root/workspace/mind2act-scripts/run.sh
```

- 上一局写出 `ready.json` 后才启动下一局，两次启动至少间隔 60 s，错开 Isaac 冷启动。

- 并行局数等于 GPUS 卡数，超过 6 张时只并行 6 局。

多任务多卡

在一个 Pod 内跑多个任务和难度：

```
OUTPUT_PATH=/mnt/pfs/<your-dir>/m2a-duclaw TASKS=PR1,DR2,CP01 LEVELS=easy,medium GPUS=0,1,2,3,4,5
POLICY_KIND=e2e ADAPTER=mind2act_llm_e2e.duclaw:create ADAPTER_CONFIG=/mnt/pfs/<your-dir>/duclaw.json
/root/workspace/mind2act-scripts/run.sh
```

实例更多时拆成 n 个百舸任务，镜像和挂载相同。第 i 个任务（i = 0…n-1）：

```
SHARD=<i> NSHARD=<n> GPUS=0,1 OUTPUT_PATH=/mnt/pfs/<your-dir>/m2a-duclaw
POLICY_KIND=e2e ADAPTER=mind2act_llm_e2e.duclaw:create ADAPTER_CONFIG=/mnt/pfs/<your-dir>/duclaw.json
/root/workspace/mind2act-scripts/run.sh
##### 查看进度
tail -n 3 /mnt/pfs/<your-dir>/m2a-duclaw/w*/logs/worker.log
```

- n 个任务用同一个 OUTPUT_PATH，各分片输出到 $OUTPUT_PATH/w<i>、锁目录互不相同，各自汇总一次。全部结束后汇总所有分片：/root/isaac-sim/kit/python/bin/python3 -I /root/workspace/mind2act-scripts/summary.py /mnt/pfs/<your-dir>/m2a-duclaw。
- n 个任务间隔 60 s 提交，错开 Isaac 冷启动。
- 耗时按 hold 实测粗估：每局 430–630 s，4500 局 8 卡并行约 86 小时，模型推理时间另计。

内存上限

框架在每局启动前和运行中统计同一 UID 所有进程的 RSS 合计，超限的局判为 resource_limit。单局约 14 GiB。

- 只用 1 张卡时上限固定为 36 GiB，够用。
- 多卡并行且容器有 cgroup 内存上限时，脚本自动生成 --resource-config，RSS 合计上限默认 16 GiB × 并行局数 + 8 GiB，最高 128 GiB、并行 6 局。百舸任务的内存配额按此预留（如 6 卡并行建议设 256 GiB 以上）。
- 容器没有 cgroup 内存上限时，上限只能是 36 GiB，脚本自动把并行降到 2 局；需要更多并行时，在百舸任务中设置内存配额。

接入被测模型（duclaw e2e adapter）

镜像内置一个 duclaw 关节空间端到端（e2e）adapter（mind2act_llm_e2e.duclaw:create）：每次决策把任务指令、当前关节状态、剩余控制行数和 3 路相机 RGB 发给模型，模型返回若干绝对关节路点（两臂各 6 个关节角加夹爪开合），adapter 按单行步长上限线性展开成 30 Hz 控制行执行，执行完再问下一次。

- **开通**：使用 duclaw adapter 需先开通百度智能云 AgentOS 服务并创建 API Key，详见准备工作「开通 AgentOS」。
- **接口**：OpenAI Responses 兼容接口，默认地址 https://duclaw-gateway.baidubce.com/star/openai/v1/responses，默认模型 A6A。AgentOS 控制台给出的 BaseURL 与默认地址不同时，写入配置的 base_url（到 /v1 为止，adapter 自动拼接 /responses）。
- **凭证**：API Key 从环境变量 MIND2ACT_API_KEY 读取，不进日志、不进结果。在百舸任务的环境变量中设置，不要写进配置文件或启动命令。
- 只支持 POLICY_KIND=e2e（控制频率 30 Hz）。

配置文件（JSON）至少包含以下 3 项，未知字段会被拒绝：

```
{
 "log_dir": "/mnt/pfs/<your-dir>/m2a-duclaw/requests",
 "ledger": "/mnt/pfs/<your-dir>/m2a-duclaw/ledger.json",
 "ledger_limit": 200,
 "reasoning_effort": "medium"
}
```

字段	默认值	说明
log_dir	必填	每局一个子目录，每次决策写一个 decision-NNNNN.json（请求骨架、模型回复、HTTP 状态、耗时、token 用量；图像只记 sha256）
ledger	必填	请求账本，记录每次 HTTP 请求（含失败和重试）。跨局、跨进程累计，并行局共用同一账本安全（文件锁），永不重置
ledger_limit	必填	账本总请求上限（正整数）。用满后该局报错终止。账本已存在时上限不能改，换上限要换 ledger 路径。建议按「实例数 × 每局决策次数 × 2」预留
max_decisions	不限	单局决策次数上限，用于控费；正式评测不建议设置
reasoning_effort	不传	透传给 Responses 接口的 reasoning.effort，如 medium
request_retries	2	单次请求失败（408/429/5xx、网络错误）的重试次数
parse_retries	1	回复不是合法 JSON 时带错误说明重问的次数
max_rows_per_decision	150	单次决策最多执行的控制行数（30 Hz 下 5 s）
max_step_rad / max_step_gripper	0.05 / 0.25	每控制行关节、夹爪的最大变化量
timeout_s	180	单次 HTTP 请求超时
model	A6A	模型名
base_url	https://duclaw-gateway.baidubce.com/star/openai/v1	接口前缀，adapter 在其后拼接 /responses

> 镜像内的 /root/workspace/mind2act-llm-e2e/examples/duclaw.json 只是示例，其 log_dir、ledger 在容器本地、任务结束即丢失，百舸任务中应复制一份改到 PFS 上。不占用 GPU 也能自检 adapter：$ISAAC_PYTHON /root/workspace/mind2act-llm-e2e/offline_check.py，期望最后一行 tail truncation ok。

查看结果

- 单局结果在 <OUTPUT_PATH>/results/batch-<id>/episode-<NNNN>-attempt-01/result.json，关键字段为 termination、infra_valid、ticks、error，同目录有录像（结束后另存 .tar.gz）。
- 同一 batch 目录下有 summary.json（各局结果）和 resource-receipt.json（内存上限、并行数、每局分到的卡）。
- 日志在 <OUTPUT_PATH>/logs/：worker.log（每次启动一段）、batch-<时间>.log（框架输出）、manifest-<时间>.json（本次提交的实例与参数）。

- run.sh 汇总只是运行层面的统计；模型成绩与诊断卡以项目组按统一预算的判定为准。

常见问题

Q：评测资产放在非默认目录可以吗？ 可以。通过环境变量 MIND2ACT_ASSET_ROOT（或 ASSET_ROOT）指向实际路径即可。

Q：中断后如何续跑？ 重跑同一条启动命令即可。框架会跳过已有 infra_valid=true 结果的实例，只补余下的。

Q：为什么首局特别慢？ 镜像已清空着色器与 Warp 缓存，每种 GPU 首次启动 Isaac 会重新编译，属正常现象，后续局会恢复正常速度。

Q：多卡并行局数如何确定？ 并行局数等于 GPUS 卡数，单局不跨卡，最多并行 6 局。并行受内存上限约束，详见「内存上限」。

Q：不同协议的结果能一起比较吗？ 不能。不传 POLICY_KIND（原始预算）仅用于 hold 等环境检查；被测模型按类型选 e2e 或 coding_agent，不同协议的结果不能混在一起比较，换协议时换一个 OUTPUT_PATH。

后续操作

1. 评测日志输出到 <OUTPUT_PATH>/logs/，可通过 worker.log 观察进度。
2. 全部实例跑完后用 summary.py 汇总，核对 summary.csv 中的 infra_valid、termination 与缺失实例。
3. 需要换被测模型时，调整 ADAPTER / ADAPTER_CONFIG 并换一个 OUTPUT_PATH 后重新启动。

在百舸上使用 SparseBEV++ 加速镜像训练 3D 目标检测模型

概述

SparseBEV 是一种纯视觉（camera-only）3D 目标检测方法：以 ResNet-50 + FPN 提取 6 路相机图像的多尺度特征，通过多尺度多视图可变形采样（MSMV sampling）在 3D 参考点上做稀疏采样，再由稀疏查询解码器（self-attention + temporally-aware attention + adaptive mixing）迭代回归 3D 框。SparseBEV++ 为 SparseBEV 在 nuScenes 数据集上的训练加速优化版本，优化以 **RocketX**（零侵入训练加速库，以 wheel 形式安装进镜像、通过环境变量开关生效）承载；SparseBEV 源码侧仅保留少量保证源码可跑通的正确性/兼容性修复及多机扩展所需的最小改动。**不设任何开关时，训练行为与上游基线一致。**

本加速镜像已内置完整训练栈与优化后的训练脚本，适配 RTX Pro 6000（Blackwell，sm_120），单机多卡与多机扩展训练均支持（多机经 RoCE v2 / RDMA 实测拉通至 2 节点 × 8 卡）。本示例以 nuScenes 数据集为例，介绍在百舸平台上完成训练全流程。

适用场景

- 在百舸平台训练 SparseBEV 类纯视觉 3D 目标检测模型，用于自动驾驶感知（nuScenes 等）任务
- 需要单机多卡训练，或多机扩展训练以获得更高吞吐的场景
- 需要断点续训、性能参考与常见问题指引的客户

环境与资源要求

项	内容
适配芯片	支持规格：aihc.lgn7ib
源码	已内嵌于镜像 /root/workspace/SparseBEV（含正确性修复 + 多机改动）
镜像环境	自带训练栈（python 3.12 + mmcv 1.7.2 / mmdet 2.28.2 / mmdet3d 1.0.0rc6），开箱即用，无需 venv / conda
工作目录	/root/workspace/SparseBEV
数据集	nuScenes（samples/、sweeps/、maps/、sparsebev/ 元信息 pkl），不打入镜像，运行时挂载
预训练权重	Cascade Mask R-CNN R50-FPN（nuImages 20e）骨干权重，不打入镜像，运行时挂载

在百舸平台准备训练环境

1. 登录百舸平台，创建或进入开发机，选择 RTX Pro 6000 资源规格与上述加速镜像
2. 打开开发机终端（或 WebIDE 终端），进入工作目录 /root/workspace/SparseBEV
3. 按本章节顺序准备数据集与权重，然后启动训练

准备数据集与权重

数据集与预训练权重**不打入镜像**，运行时需挂载到容器内或通过环境变量指定路径。

数据集 nuScenes 数据集需包含以下目录结构（训练/验证元信息 pkl 为含时序的版本）：

```
data/nuscenes/
├── samples/          # 6 路相机图像（CAM_FRONT / CAM_FRONT_LEFT / CAM_FRONT_RIGHT / CAM_BACK / CAM_BACK_LEFT / CAM_BACK_RIGHT）
├── sweeps/           # 历史帧图像（时序输入）
├── maps/             # 地图（BEV map 分支用）
└── sparsebev/
    ├── nuscenes_infos_train_sweep_temporal.pkl   # 训练元信息（含时序）
    └── nuscenes_infos_val_sweep_temporal.pkl     # 验证元信息（含时序）
```

- 可通过 BOS 获取：bos://ai-model-yq/tianyaolin/sparsebev/datasets/nuscenes/（示例路径，实际以交付信息为准）
- 百舸训练任务等不能 -v 挂载的场景，设置 DATA_PATH=<数据集根目录> 即可（见「启动训练」）
- 若使用自定义数据，需按同名字段生成对应 pkl

预训练权重 load_from 指向 COCO 预训练的 Cascade Mask R-CNN R50-FPN（nuImages 20e）骨干权重：

```
cascade_mask_rcnn_r50_fpn_coco-20e_20e_nuim_20201009_124951-40963960.pth
```

- 可通过 BOS 获取：bos://ai-model-yq/tianyaolin/sparsebev/pretrained_weights/cascade_mask_rcnn_r50_fpn_coco-20e_20e_nuim_20201009_124951-40963960.pth（示例路径，实际以交付信息为准）
- 配置中的 revise_keys = [('backbone', 'img_backbone')] 负责把预训练权重里的 backbone.* 映射到本模型的 img_backbone.*

Docker 启动（挂载方式） 镜像内提供 run_docker.sh 运行脚本，自动挂载数据集与预训练权重：

```
##### 进入交互 shell
./run_docker.sh

##### 直接启动训练（以单机 8 卡、每卡 bs35、60 step 为例）
./run_docker.sh STEPS=60 GPUS=8 PER_GPU_BATCH=35 bash run_rocketx_verify.sh
```

训练产物默认写在容器内 `outputs/`（随容器销毁）。需要持久化或做多机共享时，额外挂载卷并设置 `OUTPUT_ROOT`，例如：

```
DOCKER_ARGS="-v /pfs/.../outputs:/verify/outputs" OUTPUT_ROOT=/verify/outputs ./run_docker.sh ...
```

参数说明

训练参数全部通过环境变量传入启动脚本 `run_rocketx_verify.sh`，**无需修改源码或配置文件**。

拓扑与启动参数（torchrun）

参数（环境变量）	默认值	说明
`NODE_RANK`	0	兼容平台注入的 RANK）当前节点编号，从 0 到 NNODES-1，多机每台必须唯一
`NNODES`	1	兼容平台注入的 WORLD_SIZE）参与训练的节点数；所有节点必须使用一致配置，并正确设置 MASTER_ADDR
`GPUS_PER_NODE`	自动探测	nvidia-smi -L 行数|每个节点启动的进程数，一进程一卡
`MASTER_ADDR`	127.0.0.1	rank 0 节点地址。NNODES>1 且为 127.0.0.1 时脚本直接报错退出，避免静默死锁
`MASTER_PORT`	29500	torchrun rendezvous 端口，所有节点必须一致
`TIMEOUT_S`	18000	rendezvous 预算（秒），多机冷启动 + 数据加载不易超时

训练超参

参数（环境变量）	默认值	说明
`PER_GPU_BATCH`	35	每张 GPU 的 micro-batch。显存是主要约束：95.6 GiB 卡上为接近上限的配置，调大需关注显存
`NUM_EPOCHS`	1	训练 epoch 数，总步数 = NUM_EPOCHS × STEPS
`STEPS`	28130 // GLOBAL_BATCH	每个 epoch 的 iteration 数，随全局 batch 自动重算
`CONFIG`	configs/r50_nuimg_704x256_memory_8x256.py	训练配置文件（输入 704×256，解码器 8 层、256 维、900 查询、含 DN 分组）
`RUN_NAME`	rocketx_<月日_时分秒>	本次运行的名称，决定权重与日志落盘目录
`WORKERS_PER_GPU`	8	每个训练进程的 DataLoader worker 数
`LOG_INTERVAL`	1	日志打印间隔，调大可减少日志刷新开销（适合性能测试）
`OUTPUT_ROOT`	outputs	训练产物输出根目录。**多机必须指向所有节点同路径可见的共享 FS**
`DATA_PATH`	空	数据集根目录。非空时脚本把 data/nuscenes 软链到该目录（用于不能 -v 挂载的场景）

全局 batch size = `NNODES × GPUS_PER_NODE × PER_GPU_BATCH × 1`（本配置梯度累积步数为 1）。

权重保存与断点续训参数

参数（环境变量）	默认值	说明
`CKPT_SAVE_INTERVAL`	STEPS × 10	按 iteration）权重保存间隔，设 100000 等价于关闭中间保存（最后一步仍会保存）
`CKPT_MAX_KEEP`	1	保留的权重个数（-1 = 全部保留）
`EVAL_INTERVAL`	STEPS × 10	按 iteration）验证间隔，设 0 关闭训练中评测
`RESUME_FROM`	空	断点续训入口，从该权重继续训练，work_dir 取该权重所在目录
`LOAD_FROM`	空	覆盖 config 中的 load_from（只加载模型权重，不恢复 optimizer/iter）

> 启动脚本会显式覆盖配置文件中的 `checkpoint_config.interval` / `max_keep_ckpts` / `eval_config.interval`（按 config 内置 batch_size 16 在配置解析期算出，`--override batch_size` 不会重算），避免短跑时「一个中间权重都不存」。

优化配置接口

优化项通过环境变量开关启用（1）/关闭（0）。**总开关是 `OPT`**：`OPT=1`（默认）打开全部性能优化，`OPT=0` 关闭全部优化（纯基线，用于 A/B 对照）。所有优化项开关**默认关**，不设开关时训练行为与基线一致。

优化开关一览（`run_rocketx_verify.sh` 中每项均为 `${VAR:-1}`，显式传入的值优先，支持「只关其中一项、其余全开」）：

```
export OPT=1                              # 1 = 全栈优化（默认）；0 = 纯基线（A/B 对照）
export ROCKETX_SKIP_VALIDATION=1          # 只跳过云实例 license 校验，不影响任何优化项
export ROCKETX_VERBOSE=1                  # 打印各项命中状态表（首次接入建议打开）

##### P0：DN 分桶（去噪查询按 known_num 分桶，削 pad 极值）
export ENABLE_ROCKETX_SPARSEBEV_DN_BUCKET=1
export ROCKETX_SPARSEBEV_DN_BUCKET_K=4    # 桶数，默认 4

##### P5-a：通用优化旋钮（8 项）
export ENABLE_ROCKETX_FUSED_OPTIMIZER=1   # AdamW → fused AdamW
export ENABLE_ROCKETX_GRADIENT_AS_BUCKET_VIEW=1 # DDP 梯度用 bucket view，省一次 grad copy
export ENABLE_ROCKETX_NUMA_BINDING=1      # 首次 set_device 时绑核，降跨 NUMA 访存
export ENABLE_ROCKETX_TF32=1              # matmul/cudnn 允许 TF32（引入可接受的数值差异）
export ENABLE_ROCKETX_TRANSPARENT_HUGEPAGE=1  # 透明大页（需内核 THP 权限）
export ENABLE_ROCKETX_GC_OPT=1            # gc.disable()，规避周期性 GC 停顿
export ENABLE_ROCKETX_TENSORBOARD=1       # TensorBoard 记录旁路
export ENABLE_ROCKETX_PIN_MEMORY=1        # pin_memory：H2D 搬运移出主线程（注意：会改变数据序列，见「常见问题」）

##### P5-b：MSMV sampling 反向 CUDA 算子（warp-shuffle 归约）
export ENABLE_ROCKETX_MSMV_SAMPLING=1

##### P6-a / P6-b：按模块关闭激活重计算（显存换反向重算时间）
export ENABLE_ROCKETX_SPARSEBEV_DISABLE_BACKBONE_CP=1  # 关闭 img_backbone 激活重计算
export ENABLE_ROCKETX_SPARSEBEV_DECODER_CP=1      # 按模块关闭 decoder 激活重计算
export SPARSEBEV_CP_OFF=sa,ss                     # 关闭的 decoder 子模块键 ∈ {sa,ta,ss,am}

##### P7：匈牙利匹配向量化 + scipy 并行
export ENABLE_ROCKETX_SPARSEBEV_HUNGARIAN=1
export SPARSEBEV_HUNGARIAN_PARALLEL=8     # scipy 并行线程数，设 0 则只向量化不并行

export WANDB_MODE=offline                 # 默认离线，避免无外网时 wandb 阻塞
```

实现说明：`OPT` 是启动脚本自身的 shell 分支开关（`OPT=1` 时逐项 `export` 各 `ENABLE_ROCKETX_*`，`OPT=0` 时全部不设），RocketX 的 `is_enabled()` 据此判断是否装补丁；变量只向下传给子进程，不影响登录 shell 或其它任务。

启动训练

启动脚本 `run_rocketx_verify.sh` **已适配单机与多机场景**，单机/多机共用同一脚本，仅通过环境变量区分。

单机训练（8 卡）

```
##### 基线（全关，A/B 对照）
OPT=0 bash run_rocketx_verify.sh

##### 优化（默认 OPT=1，全栈优化）
bash run_rocketx_verify.sh

##### 修改单卡 batch 为 16
OPT=0 PER_GPU_BATCH=16 bash run_rocketx_verify.sh

##### 跑一小段用于性能测试（120 步）
OPT=0 STEPS=120 NUM_EPOCHS=1 bash run_rocketx_verify.sh
```

多机训练（2 节点示例） 两节点使用同一脚本，仅 NODE_RANK / MASTER_ADDR 不同（node0 先起，承载 rendezvous store）：

```
##### node0（rank0）
MASTER_ADDR=<node0_ip> NODE_RANK=0 NNODES=2 GPUS_PER_NODE=8 OUTPUT_ROOT=<共享FS>/outputs bash run_rocketx_verify.sh

##### node1
MASTER_ADDR=<node0_ip> NODE_RANK=1 NNODES=2 GPUS_PER_NODE=8 OUTPUT_ROOT=<共享FS>/outputs bash run_rocketx_verify.sh
```

> 百舸平台会注入 MASTER_ADDR / MASTER_PORT / RANK（节点号）/ WORLD_SIZE（节点数），脚本对这四个变量做兼容读取，**百舸训练任务无需额外传参**；显式拼写的 NODE_RANK / NNODES 优先于注入值。 容器要求：每个节点 --network host、--ipc=host、--privileged、--shm-size 足够大；若用 --gpus "device=..." 限定可见 GPU，必须同时给 -e CUDA_VISIBLE_DEVICES=<容器内局部编号>。

多机正确性保障：采样器基于全局 rank / world_size 切分样本；只有 node 0 的 local rank 0 创建work_dir、写 train.log、备份代码（避免多节点在共享 FS 上互相干扰）；RDMA 设备做硬前置校验（缺失直接报错，避免静默退化成 socket 拉低吞吐）；非法拓扑（NODE_RANK ≥ NNODES / GPUS_PER_NODE < 1 / 多机却用 127.0.0.1）直接报错退出。

性能参考

以下数据为 RTX Pro 6000（95.6 GiB/卡）实测，PER_GPU_BATCH=35，取训练稳态数据。

单机 8 卡（GBS = 8 × 35 = 280）|指标|基线 OPT=0|优化 OPT=1|提升| |-|-|-| |单轮次迭代速度（第 100 step）|3.807 s/it|**2.261 s/it**|**1.68×（−40.6%）**| |单轮次迭代速度（稳态中位数，iter 6–120）|3.920 s/it|**2.254 s/it**|1.74×（−42.5%）| |吞吐（样本级，按稳态中位数换算）|71.4 samples/s|**124.2 samples/s**|+73.9%| |峰值显存（nvidia-smi 实测，8 卡取最大）|93.3 G / 95.6 G（97.6%）|**91.7 G / 95.6 G（95.9%）**|−1.6 G| |data_load 中位数|20 ms|6 ms|−70%| |单个权重大小|496.6 MB|496.6 MB|—|

双机 16 卡（GBS = 2 × 8 × 35 = 560，RoCE v2 / RDMA）|指标|基线 OPT=0|优化 OPT=1| |-|-|-| |单步耗时（稳态中位）|4.006 s/it|**2.295 s/it**| |吞吐（样本级）|139.8 samples/s|**244.0 samples/s**| |峰值显存（nvidia-smi 实测，16 卡取最大）|93.3 GiB|91.2 GiB| |扩展比（单机单卡单步耗时 / 双机单卡单步耗时，阈值 0.9）|0.98|**0.98**|

> 扩展比 0.98 说明从单机 8 卡扩到双机 16 卡（GBS 随卡数翻倍），单卡单步耗时基本不变，横向扩展几乎无损。 以上数据为特定测试环境下的参考值。实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

loss 收敛对齐 优化前后 loss 收敛曲线不受影响（同一次 120-iter 对比）：

对比项	基线 OPT=0	优化 OPT=1	差异
loss @ iter 120	41.0784	39.5046	−3.8%（同一水平）
收敛趋势	74.7 → 41.1	81.9 → 39.5	同趋势、同水平

iter 20 之后两者进入同一水平带（约 41），Δ 在 ±3% 内正负抖动，说明**优化不改变收敛性**。

断点续训

- **保存位置**：<OUTPUT_ROOT>/<model.type>/<RUN_NAME>/（默认 outputs/SparseBEV/<RUN_NAME>/），文件命名 iter_<iter>.pth，同时维护软链 latest.pth
- **保存频次**：每 CKPT_SAVE_INTERVAL 个 iteration 存一次，训练最后一步必定保存；CKPT_MAX_KEEP 控制保留个数
- **单机续训**：

```
##### 示例：每 1000 iter 存一次，保留 3 个
CKPT_SAVE_INTERVAL=1000 CKPT_MAX_KEEP=3 bash run_rocketx_verify.sh

##### 从断点继续训练
RESUME_FROM=outputs/SparseBEV/myrun/iter_1000.pth bash run_rocketx_verify.sh
```

RESUME_FROM 非空时脚本会校验文件存在，并令 work_dir = dirname(RESUME_FROM)（续训沿用原目录）；train.py 走 runner.resume()，恢复**模型权重 + 完整 AdamW optimizer state + iteration 计数 + fp16 loss scaler + 学习率调度位置**。多机续训时 OUTPUT_ROOT 必须是所有节点同路径可见的共享 FS。

常见问题

2026-11-05 前后版本说明

- **构建必须使用 legacy builder**（DOCKER_BUILDKIT=0 docker build）：RocketX 的 setup.py 用 torch.cuda.is_available() 决定是否编译 CUDA 扩展，BuildKit 的 RUN 沙箱看不到 GPU 会静默产出不含 _rocketx_ops 的 wheel（性能回退约 +7%）。镜像已内置构建期前置检查，拿不到 GPU / 算子缺失时直接 fail；
- run_rocketx_verify.sh 中 OPT=1 分支默认 ENABLE_ROCKETX_X=1（显式传入值优先），因此 OPT=1 ENABLE_ROCKETX_TF32=0 bash run_rocketx_verify.sh 可以正常只关 TF32。

日志里 mem_peak 和实际显存对不上？ 训练日志里的 mem_peak 是 torch.cuda.max_memory_allocated()（只统计 rank0 卡上仍存活的 tensor 高水位），与 nvidia-smi 实测值差约 27 G，**不要用日志** mem_peak **判断显存余量**。显存吃紧时给基线加 PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True 可显著降低碎片（实测全局峰值 93.3 G → 78.3 G，该变量需在 docker run 时注入）。

优化后 loss 与基线不一致，是不是有 bug？ 前 20 iter 的逐 iter 数值差异（如 iter 1：74.67 → 81.85）来自 ENABLE_ROCKETX_PIN_MEMORY：pin_memory 补丁探测 collate 返回类型时会真实消费一个 batch，且 DataLoader 初始化会从全局 torch RNG 取一次 base_seed，导致「同一分布、不同随机序列」——属于正常的 shuffle/增广随机性，不是数值偏差。如果严格逐位对照，设置 ENABLE_ROCKETX_PIN_MEMORY=0 即可（此时与基线逐位一致）。其他优化项：P6-a/P6-b（关闭激活重计算）与基线逐位一致；P5-b（MSMV 反向算子）近似中性（Δ≈0.004%）；TF32 与匈牙利匹配向量化对 iter 1 无影响；P0 DN 分桶使 iter 1 变化约 −0.38%（DN 是训练期辅助监督，推理时整组丢弃，不改变主损失与优化目标）。

多机启动超时/未正常开始

- 多机两端需正确设置 MASTER_ADDR / NODE_RANK / NNODES，MASTER_PORT 两端一致；node0 先启动。

本版本未覆盖项

- 节点规模验证到 2 节点 × 8 卡，更大规模（≥3 节点）的横向扩展性未实测。

参考

- SparseBEV 论文（纯视觉 3D 目标检测）：arXiv:2308.09244
- SparseBEV 官方仓库：https://github.com/MCG-NJU/SparseBEV
- RocketX 为镜像内置的零侵入训练加速库，以 wheel 形式安装、通过环境变量开关生效，无需修改训练代码

在百舸上使用 Wall-X 加速镜像训练 VLA 模型

概述

Wall-X 是视觉-语言-动作（VLA）模型：基座 Qwen2.5-VL-3B-Instruct，在 LIBERO single-arm 数据集上微调（wall-x v1.1.0 OSS + libero*all*）。*本加速镜像针对 RTX Pro 6000（P6k，单卡 ~97.9 GB）芯片做训练加速，已内置完整训练栈与优化后的训练脚本（一组 `WALLX_PERF*` 环境变量开关，工程等价为 torch.compile 重排 bf16 累加顺序，loss 曲线与 eager 一致）。***不设任何开关时，镜像等价于原始版本（vanilla），优化必须显式开启开关***；支持单机多卡与多机扩展训练。

本示例以 LIBERO（libero_all / single-arm）数据集为例，介绍在百舸平台上完成训练全流程。

适用场景

- 在百舸平台训练 Wall-X VLA 模型，用于机器人操作任务（LIBERO 等）
- 需要单机多卡训练，或多机扩展训练以获得更高吞吐的场景
- 需要断点续训、性能参考与常见问题指引的客户

环境与资源要求

项	内容
适配芯片	支持规格：aihc.lgn7ib
源码	已内嵌于镜像 /root/workspace/wall-x（容器入口 WORKDIR=/root/workspace）
软件栈	NGC PyTorch 26.07-py3：Python 3.12.3 / torch 2.13.0a0（nv26.07 构建）/ CUDA 13.3，开箱即用
工作目录	`/root/workspace/wall-x`
数据集	LIBERO（`libero_all` / `single-arm`），不打入镜像
模型权重	Qwen2.5-VL-3B-Instruct + wall-x v1.1.0 OSS（model.safetensors），不打入镜像

在百舸平台准备训练环境

1. 登录百舸平台，创建或进入开发机，选择 P6k 资源规格与上述加速镜像
2. 打开开发机终端（或 WebIDE 终端），进入工作目录 `/root/workspace/wall-x`
3. 按本章节顺序下载数据与权重到统一根 `/root/workspace/data`（交付 config 已默认指向此目录，无需改路径），然后启动训练

准备数据集与权重

镜像不含数据/权重，部署时下载到统一根 `/root/workspace/data`（交付 config 已默认指向此，无需改路径）。

物料	HuggingFace 地址	BOS 地址
LIBERO（libero_all / single-arm，数据集）	lerobot/libero	—
Qwen2.5-VL-3B-Instruct	Qwen/Qwen2.5-VL-3B-Instruct	bos://aihc-models-bj/Qwen/Qwen2.5-VL-3B-Instruct/
wall-x v1.1.0 OSS（wall-oss-0.5）	X-Square-Robot/wall-oss-0.5	—

> HuggingFace 为公开参考地址（内网可能不可达），实际下载以 BOS 为准。

参数说明

参数	值	说明
`batch_size_per_gpu`	26（交付默认）	优化版靠 `WALLX_PERF_ALLOC` 可稳定跑 bs26（~92 GB）；baseline 对照无 ALLOC，需降到 bs≤20，否则 OOM
`num_workers`	16	DataLoader worker 数
`gradient_accumulation_steps`	1	不开梯度累积
`use_fsdp`	false（DDP）	交付口径走 DDP；`WALLX_PERF_*` 全开关仅在 DDP 路径生效
`num_training_steps`	30（warmup 15）	计时口径（正式训练按需调大）；稳态取 step 15–29

优化配置接口

优化项由一组 `WALLX_PERF_*` 环境变量开关控制（=1 开启，不设置即默认关闭，不设任何开关时镜像等价于原始版本）：

开关	说明
`WALLX_PERF_ATTN`	强制 `attn_deterministic=False`（非确定 flash-attn）
`WALLX_PERF_ALLOC`	`PYTORCH_CUDA_ALLOC_CONF=expandable_segments`（减碎片，是本组开关省显存的主要来源）
`WALLX_PERF_DDP` + `WALLX_PERF_BUCKET`	重写 DDPStrategy.wrap，启用 static_graph + gradient_as_bucket_view + bucket_cap_mb（默认 100）
`WALLX_PERF_COMPILE` / `WALLX_PERF_COMPILE_VISUAL`	torch.compile 编译主干与视觉塔（配 `WALLX_PERF_COMPILE_RECOMP_LIMIT=64`）
`WALLX_PERF_NO_TIMER_BARRIER`	将 timer 的 barrier 置为 no-op（影响计时开销，不影响峰值显存）
`WALLX_PERF_LOG`	打印每个生效补丁一行日志，用于确认开关生效

推荐配置（优化开关，环境变量）：

```
export WALLX_PERF_ATTN=1
export WALLX_PERF_ALLOC=1
export WALLX_PERF_DDP=1
export WALLX_PERF_BUCKET=100
export WALLX_PERF_COMPILE=1
export WALLX_PERF_COMPILE_VISUAL=1
export WALLX_PERF_COMPILE_RECOMP_LIMIT=64
export WALLX_PERF_NO_TIMER_BARRIER=1
export WALLX_PERF_LOG=1  # 打印每个生效补丁一行日志，用于确认开关生效
```

> 注意：`COMPILE` / `COMPILE_VISUAL` 与 `DDP` / `BUCKET` 均在 DDP 路径（`use_fsdp=false`）内生效；FSDP 走另一 strategy、不经过该 wrap，compile 不会生效，故交付口径统一走 DDP。

启动训练

镜像内置两个启动脚本（位于 /root/workspace/wall-x/，仅差 WALLX_PERF_* 开关）：run_wallx_optimized.sh（8 开关全开，交付优化口径）、run_wallx_baseline.sh（关闭全部开关，vanilla 对照）。二者自动识别卡数、支持多机（NNODES / NODE_RANK / MASTER_ADDR），共用 workspace/example/libero_local_perf.yml。

单机训练（8 卡）

```
##### 优化版（交付优化口径）
bash run_wallx_optimized.sh

##### baseline（vanilla 对照）
bash run_wallx_baseline.sh
```

多机训练（2 节点示例） 每台节点各执行一次并注入环境变量（rank0 先起）：

```
##### node0（rank0）
NNODES=2 NODE_RANK=0 MASTER_ADDR=<rank0内网IP> bash run_wallx_optimized.sh

##### node1
NNODES=2 NODE_RANK=1 MASTER_ADDR=<rank0内网IP> bash run_wallx_optimized.sh
```

性能参考

以下数据为单机 8 卡 P6k（单卡 ~97.9 GB）实测，DDP，num_training_steps=30，稳态取 step 15–29。baseline = 同一镜像不设任何 WALLX_PERF_*（为真 vanilla），与优化版仅差「开关」这一个变量。

实测汇总（bs=26 交付口径，8 卡/机） |配置|bs|稳态步时 (s)|吞吐 (samples/s)|单卡峰值显存|loss (iter1→5→10→20→29)| |-|-|-|-|-| |单机 baseline（最大可行）|20|0.877|182.5|93.1 GB|0.299→0.245→0.177→0.123→0.104| |单机 optimized|26|0.859|242.1|91.5 GB|0.301→0.248→0.173→0.116→0.103| |双机 optimized 2×8|26|0.949|438.2|91.5 GB|0.301→0.245→0.173→0.101→0.082|

- **单机有效吞吐比 = 优化版@bs26 ÷ baseline@最大可行 bs20 = 242.1 / 182.5 ≈ 1.33×**（优化版靠 WALLX_PERF_ALLOC 吃下 bs26；baseline 无显存优化，bs22 起 OOM、最大只能 bs20）
- **双机扩展比 = 438.2 / 242.1 ≈ 1.81×（并行效率 90.5%）**
- 同为各自最大批时纯步时 ≈1.02×（bs26 计算 bound，优化收益主要来自"省显存 → 上更大 batch"）
- loss 三者同批可比、收敛正常（双机 global batch 更大 26×16=416，末步 loss 更低）

batch size 扫描（优化版单机，DDP） |bs/gpu|global batch|稳态步时 (s)|吞吐 (samples/s)|单卡峰值显存|结果| |-|-|-|-|-|-| |16|128|0.682|188|71.6 GB|OK| |20|160|0.730|219|79.6 GB|OK| |24|192|0.797|241|87.5 GB|OK（留 ~7 GB 余量）| |26|208|0.859|242|91.5 GB|**交付默认（最大可稳定跑）**| |28|224|—|—|~93.4 GB 后 OOM|OOM（iter1）|

优化分解（bs=16，实验环境计时） |里程碑|单步 (s)|相对 baseline| |-|-|-| |baseline（原始版本）|0.7677|—| |Phase 1（ATTN+ALLOC+DDP static_graph+BUCKET=100）|0.7335|−4.5%| |+COMPILE（backbone）|0.6960|−9.3%| |+COMPILE_VISUAL（最终版）|0.6722|−12.4%|

> 以上数据为特定测试环境下的参考值。实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

断点续训

支持断点续训（config 驱动、脚本不变）。在 workspace/example/libero_local_perf.yml 设置 checkpoint.resume_from，再照常启动脚本即可：

```
checkpoint:
resume_from: /root/workspace/outputs/<训练输出>/<step目录>          # 接续训练：指向 checkpoint 目录
# resume_from: /root/workspace/data/wall-oss-0.5/model.safetensors  # 冷启动：只加载权重、从头计步
```

```
bash run_wallx_optimized.sh  # 与首训同一脚本，读到 resume_from 即自动续训（baseline 换 run_wallx_baseline.sh）
```

续训时状态（optimizer / scheduler / RNG / global_step）均可正确恢复；已修复上游「词表扩容 vs resume 顺序」崩溃（resume 时先按 checkpoint 词表 resize embedding，再建 optimizer / wrap DDP，compile-on / compile-off 两种崩溃模式均已消除）。

> 注意：续训与「不中断连续训练」非逐位一致（对齐位置 loss 差约 0.8–7%），源于 3 处上游保真度缺口（np_rng 未随 checkpoint 保存、global_step off-by-one、lerobot 无 mid-epoch 数据态），不影响正常接续收敛。

可复现性

镜像内置数据侧随机种子修复（DataCollator 的 np_rng + DataLoader worker 播种，带 rank 偏移）。vanilla（不开 compile / attn 非确定）下同参数多次训练，loss 与 grad_norm 逐位一致（rtol=1e-3 判据 PASS），可用于回归对齐；开优化开关时收敛一致、但不保证逐位。

参考

- 数据集 LIBERO：lerobot/libero
- Wall-X 开源权重（wall-oss-0.5）：X-Square-Robot/wall-oss-0.5
- 基座模型：Qwen/Qwen2.5-VL-3B-Instruct

全托管资源池

创建全托管资源池

概述

全托管资源池是百舸提供的免运维资源池。在全托管资源池下，用户无需关注资源池和算力资源的运维管理，由平台保障运维稳定性。通常情况下，建议为企业内部的独立团队或项目分别创建独立的资源池，以实现团队间或项目间的资源隔离。本文档介绍如何创建全托管资源池。

前提条件

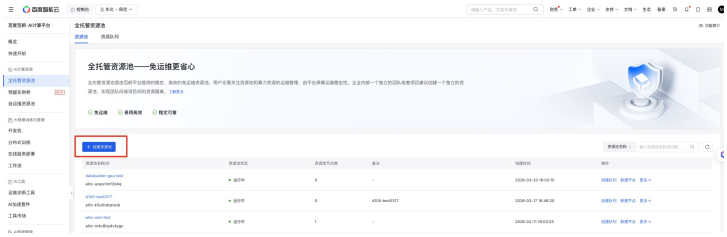
- 已登录 百舸AI计算平台AIHC控制台。
- 账号具备创建全托管资源池的权限，并已准备目标地域和可用区下可选的 VPC 网络、节点子网、容器子网和安全组；如无可用资源，可在创建页按界面提示新建。
- 提交订单前，请确认账户余额满足页面校验要求；若页面提示 **账户余额** 待确认，请先按提示完成充值或补足可用金额。

操作步骤

导航路径：**AIHC 控制台->全托管资源池->资源池**

步骤 1：进入创建全托管资源池页面

在 **【全托管资源池】** 页面确认当前为 **【资源池】** 页签后，单击 **【创建资源池】** ，进入 **创建全托管资源池** 页面。



步骤 2：填写资源池配置

在 **创建全托管资源池** 页面，根据实际业务需求完成基础配置、网络配置、节点配置和关联服务配置。

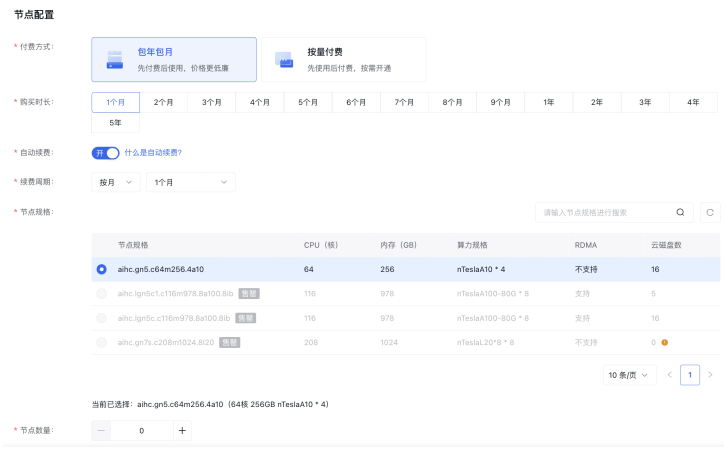
基础配置

参数	必填	说明
资源池名称	是	托管资源池名称。支持大小写字母、数字、中文及 -、_、/、. 等字符，必须以字母开头，长度为 1-65 个字符。
备注	否	按需填写资源池备注信息。
地域	是	按需选择资源池所属地域。
可用区	是	按需选择资源池所属可用区。



节点配置

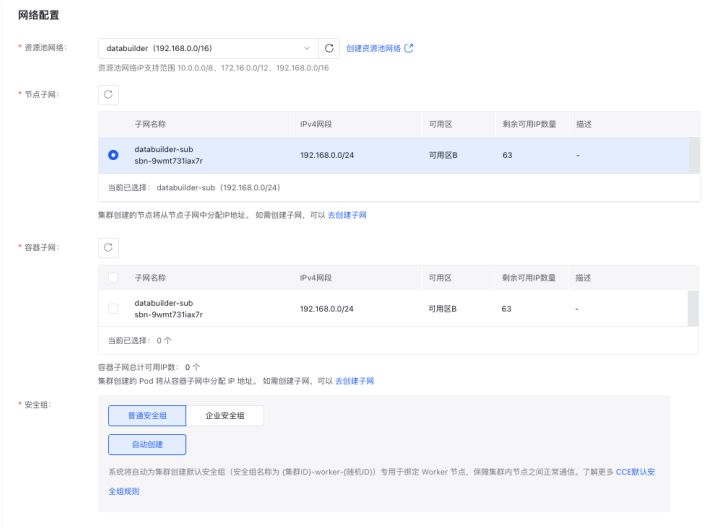
参数	必填	说明
付费方式	是	支持包年包月和按量付费，按需选择。
节点规格	是	选择资源池中需要创建的节点规格。
节点数量	是	填写资源池中需要创建的节点数量。
购买时长	否	选择包年包月时配置购买时长。
自动续费	否	选择包年包月时可配置自动续费。开通后，系统会在实例到期前按规则自动扣款续费，并通过短信和邮件提醒续费结果。
续费周期	否	开通自动续费后，按需选择按年或按月的续费周期及周期数量。



网络配置

参数	必填	说明
资源池网络	是	选择资源池中节点使用的VPC网络，您可以在当前可用区新建VPC。
节点子网	是	资源池创建的节点将从节点子网中分配 IP 地址，您可以新建子网。
容器子网	是	资源池创建的Pod将从容器子网中分配 IP 地址，支持多选，您可以新建子网。
安全组	是	资源池节点使用的安全组，可选择普通安全组或企业安全组。

注：平台预留网段为11.0.0.0/12，资源池网络和容器子网不能与该网段冲突。



关联服务

参数	必填	说明
存储配置	否	选择需要挂载的 PFS 实例或挂载服务。创建资源池后，系统会自动将所选 PFS 挂载到资源池节点。
监控配置（CPROM）	否	选择资源池监控指标推送的 CPROM 实例，用于保存监控数据并展示可观测大盘。

如无额外需求，**存储配置** 和 **监控配置（CPROM）** 可保持未选择状态，创建完资源池后有需求再关联绑定即可。



步骤 3：确认订单并提交

> **注意：** 提交后将根据所选节点规格、数量和付费方式产生费用，请在提交前确认配置信息。

单击 **【确认订单】**，核对节点规格、数量、地域、可用区和费用信息。页面会展示与所选配置对应的订单摘要和参考价格。

在 **前置检查** 区域等待检查完成；若 **账户余额** 项显示待确认，请先确保账户可用金额大于 ￥100，或按页面提示完成充值。检查满足提交条件后，勾选 **《百度智能云线上订购协议》**，单击 **【提交】**。

创建成功后，您可以在 **【全托管资源池】** 页面查看已创建的资源池状态。

管理全托管资源池

概述

本文介绍如何查看全托管资源池详情、节点管理、资源观测，有助于您更为全面地了解全托管资源池的信息，并进行操作。

资源池详情

1. 登录百舸AI计算平台AIHC控制台；
2. 在左侧菜单栏选择**全托管资源池**，进入全托管资源池页面，点击要查看的资源池名称，进入并查看资源池详情。



3. 对于在创建资源池中未绑定PFS的资源池，可在详情页面进行绑定



资源管理

节点管理

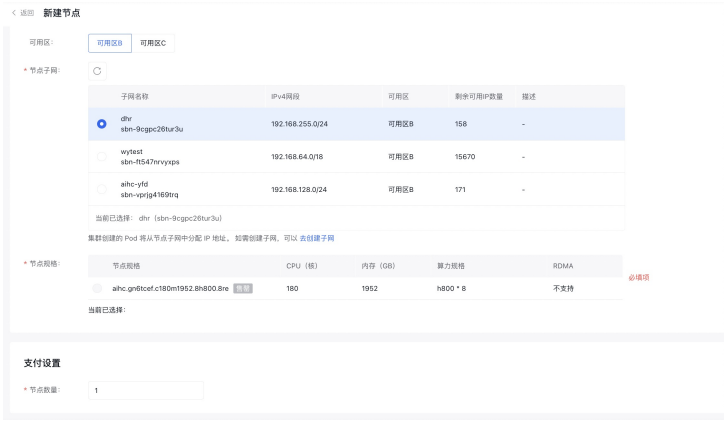
新建节点

1. 进入资源池详情后，选择**节点管理**，点击**新建节点**。



2. 在新建节点页面，根据实际需求完成相关参数配置。

配置项	必填/选填	配置说明
计费方式	必填	按需选择按量**付费**和**包年包月**
可用区	必填	购买节点所在的可用区，目前仅支持新建节点与当前资源池中存量节点同一可用区，如需添加其他可用区节点请新建资源池或释放所有节点后重新创建节点。
节点子网	必填	资源池创建的节点将从节点子网中分配 IP 地址。
节点规格	必填	选择要购买的百舸节点规格，不同规格对应不同的CPU、内存、算力，可参考表格中的信息选购。
节点数量	必填	本次购买的节点数量。



4. 确认配置信息，然后单击**下一步**提交订单。创建成功后，您可以在资源池详情**节点管理**页面查看已创建的节点列表。

封锁节点

1. 进入资源池详情后，选择**节点管理**，在节点列表中，找到要封锁的节点，并点击操作栏中的**封锁设置**按钮。



2. 在封锁设置弹窗中，可按需对节点进行封锁和解除封锁操作。封锁节点后，节点处于不可调节状态，新状态的Pod将不会调节到该节点上。



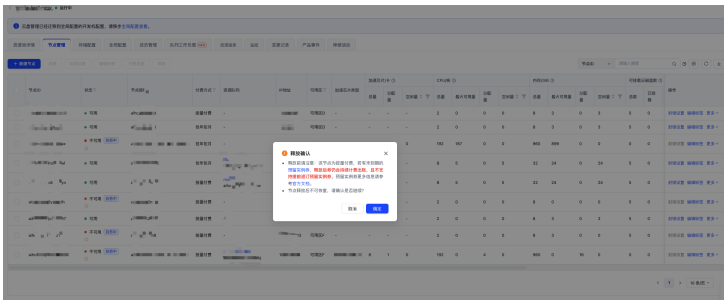
释放节点

1. 进入资源池详情后，选择**节点管理**，在节点列表中，找到要释放的节点，并点击操作栏中的**释放**按钮。目前仅支持释放按量付费和已到期的包年包月节点。

2. 在释放弹窗中点击确认，即可完成节点释放操作。确认释放的节点会立即结束使用并不再计费。

3. 从资源池释放节点前，请确保节点已经从队列移出，即该节点不属于任何一个队列。

4. 如果是后付费节点且购买了预留实例券，请确保预留实例券即将到期。在预留实例券到期前释放节点，预留实例券会正常出账计费。



云盘管理

进入资源池详情后，选择**云盘管理**，显示资源池内云盘情况。



云盘扩缩容

1.进入资源池详情后，选择**云盘管理**，点击操作栏中的**扩缩容**按钮。



2.在基本配置页面，根据实际需求完成**云盘容量**参数配置。

> 云盘最大容量100000GiB，最小扩容量20GiB，可以缩容至0GiB。



3.确认配置信息后，单击**确认订单**进入确认订单页面。



4.，确认信息后，勾选协议后，单击**提交**提交订单。操作成功后，您可以在**云盘管理**页面查看云盘容量。

全局配置

通用配置

资源使用配置

1.数据挂载配置

支持配置挂载数据集或直接挂载存储，确保计算节点能够快速访问所需数据。



2.CCR 镜像免密拉取



配置后，创建工作负载时无需手动输入 CCR（百度智能云容器镜像服务）的用户名密码。

- 自动创建访问凭据： 系统使用当前操作用户身份自动创建。
- 指定访问凭据： 使用指定用户的凭据，需确保该用户持续有效。



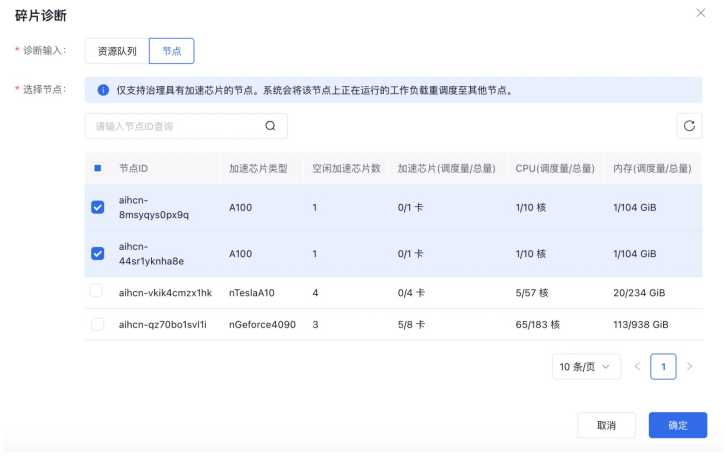
3.碎片治理

碎片治理功能专门用于解决分布式训练、小规模推理中出现的"非整机任务"导致的资源闲置问题。

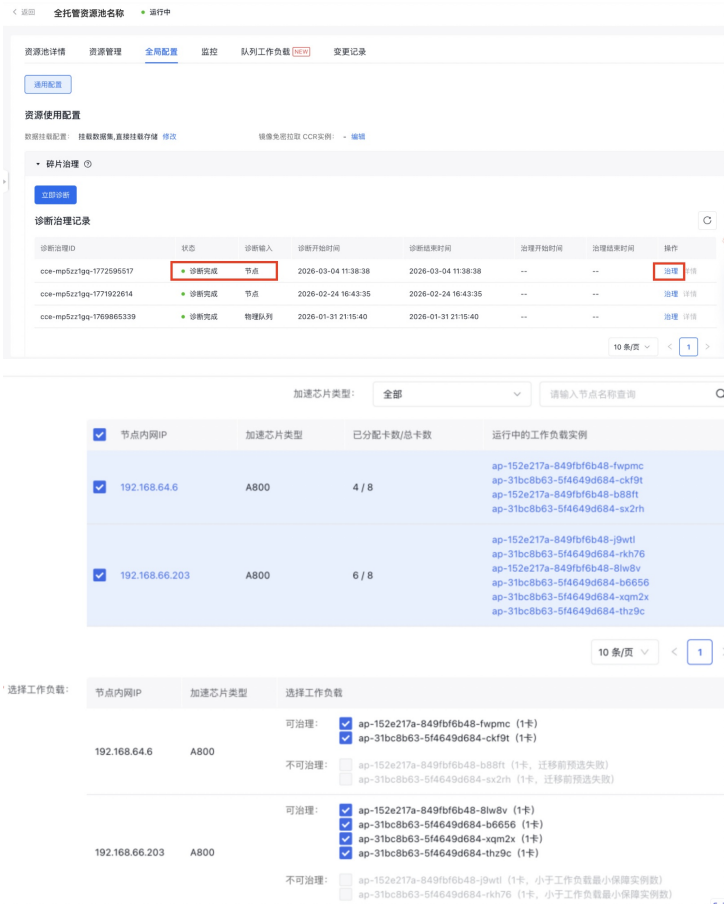
- 注意事项： 仅支持含加速芯片（GPU等）的队列和节点。如果任务处于预调度失败、已达最低副本数或手动关闭了重调度，将无法进行治理。

（1）点击**立即治理**，开始对指定队列或节点进行碎片诊断并治理。





（2）针对已完成的诊断，点击"治理"，在弹窗中选择需要治理的工作负载，开启碎片治理。



（3）治理完成后，点击“详情”，查看碎片治理结果详情信息。



开发机配置

通过预设开发机策略，您可以精细化管理团队的研发成本：

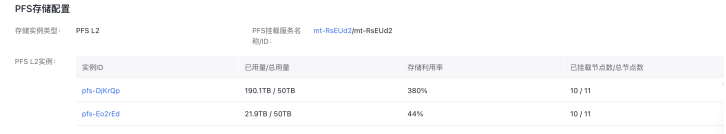
- 配额限制： 可针对单个成员设置最大创建实例数，并限制 GPU、CPU、内存及云磁盘的规格上限。
- 成本控制：
  - 低利用率关机： 自动关闭长时间低负载的实例。
  - 闲时关机： 在非工作时间段自动释放资源。
- 网络与权限： 统一控制开发机是否开启公网访问，以及是否允许用户自行制作镜像。



PFS 存储配置

资源池支持挂载高性能并行文件存储（PFS）：

- 实例管理： 实时监控 PFS 实例的已用量/总容量及存储利用率
- 自动挂载： 在此配置后，资源池内的新建节点将自动挂载关联的 PFS 实例，无需手动操作。



在线服务部署配置

针对推理场景，您可以全局控制在线服务部署公网访问的开关。开启后，部署的在线推理服务将具备公网接入能力；关闭则仅限内网通信，提升安全性。



创建和管理资源队列

概述

资源队列是全托管资源池中进行算力分配与任务调度的核心单元。所有提交至全托管资源池的工作负载（如开发机、训练任务、推理服务）均需归属于特定的资源队列，方可获得算力资源的调度与执行。

通过创建资源队列，您可以为不同的团队或项目划分专属的算力资源池，实现资源的逻辑隔离、配额管控与高效调度。

权限与访问

- **访问入口**

登录百度百舸·AI计算平台控制台 → 左侧导航栏选择 **【全托管资源池】** →查看**【资源队列】**Tab页面 → 单击 **【新建资源队列】** 按钮 → 进入创建队列页面。

- **权限说明**
  - **操作性：** 仅资源池管理员或者AIHCFullControlPolicy的用户可以创建和删除队列。
  - **可见性：** 队列管理员及队列开发者均可查看所属队列的列表和详情详情的所有信息。



创建队列参数详解

为确保您清晰理解各项配置的含义，以下提供详细的参数说明表：

配置分类	参数名称	选择类型	说明	注意事项
基本信息	**队列名称**	必填	支持大小写字母、数字、中文及 - 、 _ 、 / 特殊字符，长度不超过63字符。	
	备注	选填	用于记录队列用途、负责人等信息，长度不超过300字符。	便于后续的资源管理与识别。
资源信息	**来源类型**	固定	固定为“全托管资源池”。	由平台统一管理底层物理资源。
	资源池	必选	从下拉框选择队列所属的物理资源池。	需确保目标资源池状态正常且有可用配额。
	GPU虚拟化	可选	开启后支持按显存大小切分GPU卡（0-1之间小数分配）。	需满足节点和容器的操作系统、驱动及CUDA版本兼容性；**创建后不可修改**。
	节点规格	必选	首次创建仅支持添加同一可用区、同一芯片类型、同一弹性高性能计算集群（EHPC）下的一种规格套餐节点。	后续扩展支持分批添加不同CPU/内存配置的节点；特殊需求需提工单申请。
	负载类型	必选	选择队列支持的工作负载类型（如开发机、分布式训练、推理服务）。	**创建后不可删除**已配置的类型，需谨慎选择。
高级配置	**调度策略**	必选	提供“遍历策略（新）”、“遍历策略”和“FIFO策略”三种选项。1. **遍历策略（新）**：遍历未成功调度的任务，跳过暂时无法出队的任务并依次尝试后续任务。2. **遍历策略**：遍历排队阶段任务，跳过无法出队的任务，依次尝试后续任务。3. **FIFO策略**：严格按照任务提交顺序依次调度。	请根据业务对任务优先级的需求进行选择。
	超时重排	可选	开启后，排队超过设定时间（默认30分钟）的任务会自动重新排队。	可避免长尾任务阻塞后续任务，支持自定义超时时间。
	空闲资源出借	可选	开启后，空闲时段允许临时任务使用闲置资源。	当本队列有非闲时任务排队时，系统将立即回收所借出的资源，以提高整体资源利用率。
	队列状态	可选	默认为“开启”。	“关闭”后新提交任务将进入排队状态，已运行任务不受影响，可用于临时暂停队列接收新任务。



队列管理：编辑与删除

在队列创建完成后，您可以根据业务需求对队列配置进行调整，或在资源不再使用时安全地移除队列。

编辑资源队列 对于已有的资源队列，系统支持灵活调整其调度策略与运行状态，以适应动态变化的业务场景。点击操作栏中的"**编辑**"按钮，在弹出的"编辑资源队列"对话框中，您可以修改以下配置：

- **负载类型**：展示当前队列已配置的负载类型（如"分布式训练"）。**注意**：此选项在队列创建后锁定，不可删除已配置的负载类型，以确保运行中任务的稳定性。
- **调度策略**：支持在"遍历策略（新）"、"遍历策略"、"FIFO策略"之间切换。
- **超时重排**：可开启或关闭，并调整超时时间（默认30分钟）。开启后，排队超过设定时长的任务将自动移至队尾，防止长尾任务占用调度通道。
- **空闲资源出借**：可开启或关闭。开启后，在队列空闲时段，系统允许其他临时任务借用闲置资源；一旦本队列有新任务排队，借出资源将被立即回收。
- **队列状态**：可开启或关闭队列。关闭后，新提交的任务将进入排队状态，但已运行的任务不受影响，适用于临时维护或暂停接收新任务的场景。

确认修改无误后，点击"确定"保存配置。



删除资源队列 删除队列是一项不可逆的操作，请务必谨慎执行。在删除队列前，系统强制要求清理所有关联资源，以确保数据一致性与资源回收的完整性。

删除队列的前置条件：在执行删除操作前，您必须手动删除该队列下所有正在运行或已创建的工作负载，包括但不限于：开发机实例、训练任务、推理服务等。

删除后的资源去向：

- **计算节点释放**：队列删除后，其占用的所有计算节点将自动释放回所属的"全托管资源池"。
- **资源再利用**：释放回资源池的节点可被其他队列重新分配使用，或用于创建新的资源队列。
- **资源退订**：若这些节点不再需要，您也可以在资源池管理页面中对其进行退订操作，以停止计费。

查看与管理队列

队列创建成功后，您可以通过以下方式查看与管理：

1. **列表查看**：在【资源队列】页面，您可以查看已创建的队列列表及其基本信息。
2. **详情查看**：
   - 点击队列名称进入**队列详情页**。
   - 在**概览**面板查看队列的基本信息（如创建时间、调度策略）及资源水位（CPU/内存/加速芯片的总量与分配量）。
   - 通过**节点管理**、**负载列表**等Tab页进行具体的资源与任务治理。

混合调度能力

- 功能概述
  - 全托管资源池支持A100与A800混合调度服务。该功能允许您将A100与A800两种型号的GPU节点纳入同一队列进行统一管理。系统会将两种芯片的资源描述符进行抽象合并，实现跨型号的透明调度。您在提交训练任务时，无需关注底层具体的芯片型号，系统将自动分配可用资源，保障任务的弹性扩容与稳定运行。
- 网络环境要求
  - EHC一致性： A100与A800节点必须处于同一个EHC（弹性高性能计算集群）下，以确保节点间网络互通与低延迟通信。
- 队列配置说明
  - 开关展示逻辑： 当系统检测到当前资源池中同时存在A100和A800两种资源时，创建队列页面将自动展示【开启多芯】开关。
  - 不可逆特性： 一旦开启【队列多芯】，该队列将被锁定，仅支持添加A100和A800节点，不支持关闭或多芯类型变更。
  - 功能互斥： 若队列已开启【GPU虚拟化功能】，则无法开启【队列多芯】，两者互斥。
- 节点管理规则
  - 混合添加模式： 在已开启多芯的队列中，您可以分别添加A100和A800的套餐节点，系统将自动合并计算资源总量。
  - 单一限制模式： 若队列未开启多芯（或资源池不满足混合条件），则严格限制仅能添加单一规格/或同类规格的节点。

资源队列管理详情

概述

资源队列是百度百舸·AI计算平台资源分配与任务调度的核心单元。本指南详细介绍了队列详情页的资源定义、节点管理操作及高级配置策略，帮助管理员高效管理算力资源，优化任务调度效率。

权限与访问

- **访问入口**

登录百度百舸·AI计算平台控制台 → 查看【资源队列】Tab页面 → 点击目标【队列名称】 → 进入队列详情。

- **权限说明**
  - **可见性：** 队列管理员及队列开发者均可查看队列详情的所有信息。
  - **操作性：** 仅**队列管理员**拥有停止任务、运维节点等治理操作权限。

队列详情概览

进入队列详情页，您将看到队列的基本信息与资源概览面板。

- **基础信息**
  - **队列名称/ID**：队列的唯一标识。
  - **所属资源池**：该队列归属的物理或逻辑资源池。
  - **GPU虚拟化**：显示是否开启GPU虚拟化功能。
  - **调度策略**：当前生效的调度算法配置。
  - **超时重排策略**：任务排队超时后的处理策略。
  - **闲时资源出借**：显示是否配置了闲时资源出借给其他队列。



- **队列资源概览**

队列资源概览展示了资源队列的加速芯片（GPU、XPU等）、CPU及内存的实时状态。为了确保数据准确，各指标定义如下：

资源类型	指标名称	定义说明
加速芯片(GPU)	**总量**	队列中所有节点的加速芯片总数。
	最大可用量	队列中所有**可用节点**（已排除故障节点）的加速芯片总数。
	分配量	该队列中所有用户负载（开发机、分布式训练、在线服务）的已分配资源总和。
	空闲量	队列中所有**可用节点**（已排除故障节点、封锁节点）的空闲芯片总数。
CPU / 内存	**总量**	队列中所有节点的 CPU 核数或内存总和。
	最大可用量	队列中实际可调度和使用的资源上限，即所有可用节点（排除故障节点）的可用量总和。 注：队列中不同节点的可用量可能因系统预留不同而不一致。
	分配量	该队列中所有用户负载（开发机、分布式训练、在线服务）的已分配资源总和。
	空闲量	队列中所有**可用节点**（已排除故障节点、封锁节点）的空闲资源总和。

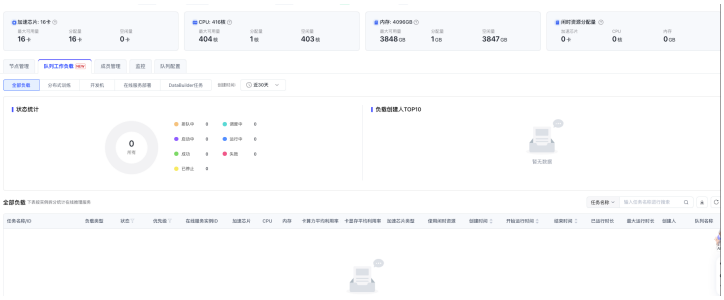
> **注意：** “调度中”的任务（尚未分配节点）不计入分配量。资源数据会随节点管理操作（如添加/移出节点）实时同步更新。

节点管理

在“节点管理”Tab页中，您可以查看队列内节点的基本信息，并进行资源的动态调整。支持以下节点操作：

- **添加节点**：将资源加入队列。
  - **首次添加限制**：仅支持添加同一可用区、同一芯片类型、同一弹性高性能计算集群（EHC）下的同一种规格套餐节点。
  - **后续扩展**：在满足上述前提条件下，支持分批次加入不同CPU/内存配置（即不同规格套餐）的节点。
  - *特殊需求*：如需跨上述限制添加节点，请提工单申请处理。
- **移出节点**：将该节点从队列释放，资源将返还给资源池，供其他队列使用。
- **封锁设置**：临时将节点设置为不可用状态，用于维护或隔离。
- **转让节点**：将节点从当前队列转移到另一个队列，注意节点转让规则如下：
  1. **基础环境一致**：仅支持在相同资源池、相同可用区、且节点规格相同的队列间进行转让。
  2. **网络要求**：若节点带有RDMA网卡，必须同属一个弹性高性能计算集群（EHC）。
  3. **状态要求**：**正在运行任务的节点不可转出**（需确保无进行中任务）。
  4. **虚拟化兼容**：若转入队列启用了GPU虚拟化，节点必须满足特定的驱动、内核及组件要求，否则无法转入。

队列工作负载



负载列表展示了创建时间近30天内的任务（支持时间范围筛选）。列表按**负载类型**分为三类：开发机、分布式训练、在线服务，支持下载负载列表。

- **调度状态体系**

本功能重点关注任务的**调度状态**（即资源分配进度），与业务状态有所区分：

调度状态	含义说明	典型场景
排队中	任务已提交，正在等待配额分配	资源不足，等待其他任务释放资源
调度中	配额已分配，正在向节点下发	寻找合适节点或K8s正在拉取镜像
启动中	Pod已初始化，正在启动容器	容器正在创建中
运行中	任务正常运行	训练/推理正在进行
已结束	任务被取消或正常结束	用户主动停止或任务完成
失败	任务执行异常	训练失败等

- **关键字段说明**

字段名称	适用类型	说明
名称/ID	全部	支持按名称和ID进行精确筛选
优先级	全部	展示任务提交时设定的优先级（高/中/低）
实例ID	在线服务	在线服务可能包含多实例，此处展示具体实例ID
资源规格	全部	展示GPU卡数、CPU核数、内存大小
利用率	全部	**卡算力平均利用率** & **卡显存平均利用率**
运行时长	全部	任务实际运行的时间跨度

- 资源利用率数据说明
  - **数据范围：** 平台默认存储并展示**最近15天**的资源利用率数据。
  - **计算逻辑：**
    - **运行中任务：** 从启动时间到当前时刻的平均值。
    - **已结束任务：** 从启动时间到结束时间的平均值。
  - **历史数据：** 若需查询超过15天的历史利用率数据，需联系管理员前往 Cprom 调整监控数据存储时长。

成员与权限管理



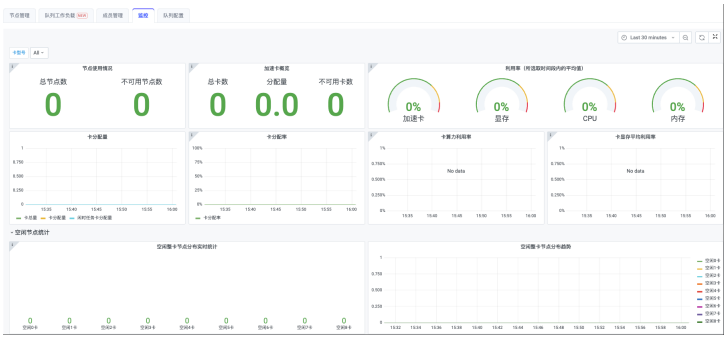
在"成员管理"Tab页中，您可以配置队列的访问控制列表，将平台用户添加至当前队列，并分配相应的角色权限。为了确保权限配置生效，您需要在 **IAM（身份与访问管理）** 中为相关用户绑定 `AIHCDevelopPolicy` 策略。

> **注意：**IAM 权限策略的更新通常需要 **3-5 分钟** 的同步时间才能在平台中生效。

队列支持两种核心角色，权限粒度如下表所示：

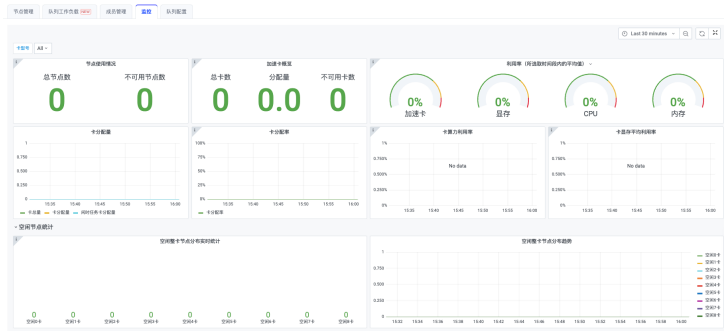
角色名称	权限范围	限制说明
队列管理员	拥有该队列的最高管理权限。包括：编辑队列调度策略、配置资源规格模板、查看队列全量监控信息、管理队列成员等。	**无平台级资源管理权**。无法创建或删除队列，也无法修改队列的底层物理配额。
队列内开发成员	拥有在该队列中进行AI开发工作的基础权限。包括：创建开发机环境、提交分布式训练任务、部署推理服务等。	**无管理权**。仅能操作属于自己的任务负载，无法操作使用其他用户的负载（共享任务除外）。

队列监控



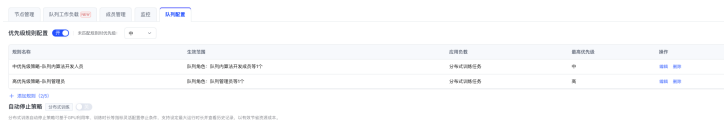
队列监控面板提供实时与历史维度的资源使用数据，帮助您掌握队列运行状态、识别资源瓶颈、优化任务调度策略。

- **节点使用情况（实时数据）**
  - **总节点数**：当前队列中包含的物理节点总数。
  - **不可用节点数**：因故障、维护或被封锁而不可调度的节点数量。该数值直接影响队列最大可用资源上限。
- **加速卡概览（实时数据）**
  - **总卡数**：队列内所有节点上加速卡（如GPU）的物理总数。
  - **分配量**：当前已被任务占用的加速卡数量（含部分分配），反映资源实际使用压力。
  - **不可用卡数**：因节点故障或硬件异常导致无法使用的加速卡数量。
- **资源利用率**（所选时间段内平均值）
  - **加速卡利用率**：加速卡计算核心的平均使用率，反映算力负载水平。
  - **显存利用率**：加速卡显存的平均占用比例，用于评估内存密集型任务的资源压力。
  - **CPU利用率**：节点CPU核心的平均使用率，辅助判断计算瓶颈是否在CPU侧。
  - **内存利用率**：节点系统内存的平均占用比例，用于评估任务内存需求与节点容量匹配度。
- **卡分配趋势**
  - **卡分配量**：随时间变化的加速卡分配数量趋势图，支持识别资源使用高峰与低谷。
  - **卡分配率**：已分配卡数占总卡数的比例趋势，反映资源饱和度变化。
- **卡算力与显存利用率趋势**
  - **卡算力利用率**：加速卡计算核心的利用率随时间变化曲线，用于分析任务资源使用波动。
  - **卡显存平均利用率**：加速卡显存占用比例随时间变化曲线，辅助识别显存泄漏或过度分配问题。
- **空闲节点统计**
  - **空闲整卡节点分布实时统计**：按空闲卡数（0卡至8卡）统计当前完全空闲的节点数量，用于评估可调度资源碎片化程度。
  - **空闲整卡节点分布趋势**：随时间变化的空闲节点分布趋势，帮助预测资源释放节奏与调度窗口。



队列高级配置策略

在“队列配置”Tab页中，支持对任务优先级和资源回收策略进行精细化管控。



优先级规则配置 支持对任务优先级进行灵活配置，以管控不同用户的提交权限，确保高优任务优先获得资源。

详情参考AIHC 优先级配置参考文档

自动停止策略 支持基于GPU利用率、训练时长等指标灵活配置停止条件，有效节省资源成本。

- **利用率阈值**：例如，当GPU利用率低于10%时触发。
- **持续时间**：例如，持续满足低利用率状态超过2小时。
- **历史记录**：支持查看策略触发的历史记录。

详情参考 AIHC 分布式训练自动停止策略文档

资源规格模板 支持管理员配置队列资源规格模板，保证队列资源充分利用。

- **两种使用模式**：支持自由创建模式（参考套用）和严格规定模式（强制选择），管理员可按需配置。
- **模板配置**：可为队列预设多组资源规格模板，覆盖不同负载场景。

详情参考 AIHC 资源规格模板文档

子用户资源配额管理

子用户资源配额管理功能允许队列管理员在队列维度下，为指定的子用户或用户组设置资源使用上限（加速芯片卡数、CPU 核数、内存容量），从而实现队列内部的精细化资源治理：

- 防止单个用户突发任务占满整个队列资源；
- 保障队列内多业务、多用户的公平共享；
- 在保留队列总体弹性的同时，控制每位成员的资源天花板。

详情参考 AIHC 子用户资源配额管理

CCR镜像免密拉取功能

概述

CCR镜像免密拉取是指在资源池中完成CCR实例免密拉取配置后，使用该资源池的算力资源创建工作负载时无需输入镜像用户名密码即可免密直接使用。

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**全托管资源池**，找到需要配置免密拉取功能的资源池名称，单击进入**资源池详情页**。
3. 单击【全局配置】-【通用配置】，点击镜像免密拉取右侧的【编辑】按钮。



4. 在“免密拉取配置”弹窗中配置CCR关联实例和免密配置的方式。支持2种免密配置方式：
   - 自动创建访问凭据：系统自动使用当前操作用户身份来创建访问凭据，请确保用户持续有效，否则无法拉取镜像；
   - 指定访问凭据：系统将使用指定用户的访问凭据，请确保用户及访问凭据持续有效，否则无法拉取镜像。



5. 确定完成即可。

碎片治理

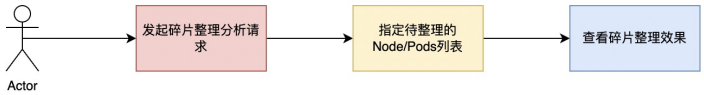
概述

为优化资源利用率并降低计算成本，您可以开启百舸碎片治理功能，有效解决分布式训练、小规模推理等场景下的非整机任务导致的资源碎片问题，实现资源高效回收与复用。

功能说明

碎片诊断

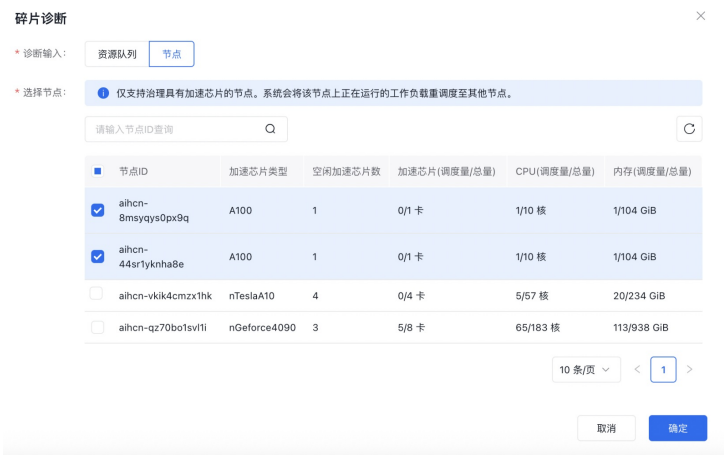
1. 诊断工作负载：开启了碎片迁移的推理服务（分布式推理不支持开启碎片迁移）。
2. 诊断输入：队列和节点，支持对指定队列或节点进行碎片诊断（仅支持治理含加速芯片的队列和节点），定位碎片资源分布情况。
3. 不可治理的原因： *关联分析任务已不存在* 任务关闭重调度 *预调度失败* 达到任务最低保障副本 *单负载的工作负载* 无工作负载管理的Pod *关联Pod已不存在* 错误的整理类型 *推理服务摘流失败* 推理服务终止失败 *该Pod已在重调度中，重复的重调度* Pod迁移失败 * 该工作负载已达到最大pod迁移数 **碎片治理** 诊断完成后，可以基于诊断结果进行碎片治理：



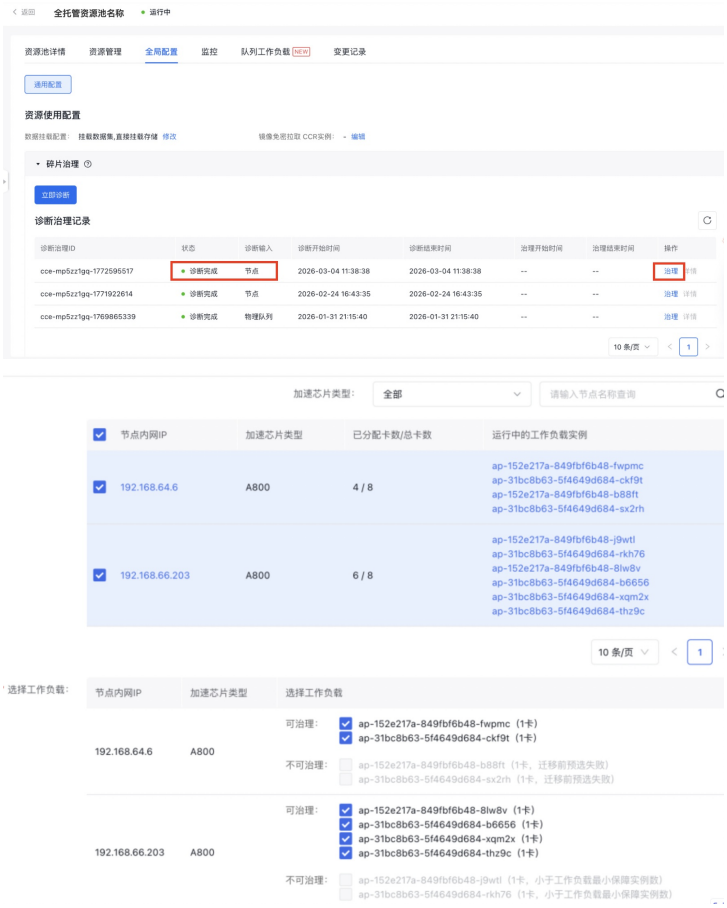
操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**全托管资源池**，找到您想要查看的资源池名称，单击ID。
3. 单击【全局配置】-【通用配置】。
4. 点击立即治理，开始对指定队列或节点进行碎片诊断并治理。





5. 针对已完成的诊断，点击“治理”，在弹窗中选择需要治理的工作负载，开启碎片治理。



6. 治理完成后，点击“详情”，查看碎片治理结果详情信息。



安全组规则

安全组是一种虚拟防火墙，为同一个私有网络内具有相同安全保护需求并相互信任的云服务器提供访问策略，具备状态检测和数据包过滤能力，用于在云端划分安全域，是重要的网络安全隔离手段。

您可以通过配置安全组规则控制安全组内云服务器的入流量和出流量。更多安全组介绍，请参见安全组。

使用须知

百舸计算AI平台对资源池使用安全组进行了优化，使用资源池时支持用户配置“自动创建默认安全组”和“使用已有安全组”，若无额外诉求，推荐使用默认安全组即可，无需关注规则细节，使用成本低；若有额外诉求，可参考默认安全组规则保障资源池内互通，降低因安全组规则使用问题影响资源池正常运行。

资源池默认安全组说明

资源池在创建时将会自动为 Master 节点和 Worker 节点分别创建一个安全组，其中 Master 节点的安全组名称是：**{集群ID}-master-{随机ID}**； Worker 节点的安全组名称是：**{集群ID}-worker-{随机ID}**。使用资源池时会额外创建一个 ENI 的安全组，名为**{集群ID}-eni-{随机ID}**。

Worker默认安全组规则

为保证资源池和节点本身功能的正常运作，确保资源池和节点之间的网络正常连通，百舸计算AI平台自动创建的 Worker 默认安全组规则为：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	容器网络 CIDR （VPC 网段）	1-65535	IPv4	允许	资源池内全部节点之间通信
入站	ALL	容器网络 CIDR	1-65535	IPv4	允许	全部 Pod 之间通信
入站	ALL	100.64.230.0/24	1-65535	IPv4	允许	允许百舸监控、日志等子网与节点通信
入站	TCP	ALL	30000-32767	IPv4	允许	开通资源池内全部 NodePort 类型 Service 之间基于 TCP 协议通信
入站	UDP	ALL	30000-32767	IPv4	允许	开通资源池内全部 NodePort 类型 Service 之间基于 TCP 协议通信
出站	ALL	ALL	1-65535	IPv4	允许	开通资源池对外全部通信
入站	TCP	ALL	22	IPv4	允许	开通公网 SSH 登录端口
入站	ICMP	ALL	不涉及	IPv4	允许	开通 Ping 操作

ENI默认安全组规则

为保证资源池节点上的Pod之间的网络正常连通，百舸AI计算平台自动创建的 ENI 默认安全组规则为：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	容器网络 CIDR （VPC 网段）	1-65535	IPv4	允许	资源池内全部节点之间通信
入站	ALL	容器网络 CIDR	1-65535	IPv4	允许	全部 Pod 之间通信
入站	ALL	100.64.230.0/24	1-65535	IPv4	允许	允许百舸监控、日志等子网与节点通信
入站	TCP	ALL	30000-32767	IPv4	允许	开通资源池内全部 NodePort 类型 Service 之间基于 TCP 协议通信
入站	UDP	ALL	30000-32767	IPv4	允许	开通资源池内全部 NodePort 类型 Service 之间基于 TCP 协议通信
出站	ALL	ALL	1-65535	IPv4	允许	开通资源池对外全部通信
入站	TCP	ALL	22	IPv4	允许	开通公网 SSH 登录端口
入站	ICMP	ALL	不涉及	IPv4	允许	开通 Ping 操作

Master默认安全组

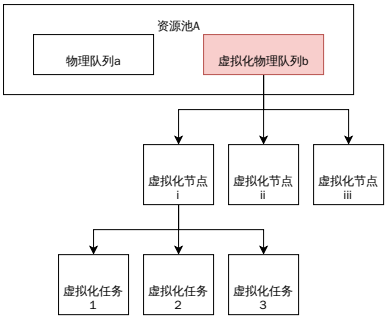
为保证资源池和节点本身功能的正常运作，确保资源池和节点之间的网络正常连通，百舸AI计算平台自动创建的 Master 默认安全组规则为：

方向	协议	网段	端口	IP 类型	策略	说明
入站	ALL	容器网络 CIDR （VPC 网段）	1-65535	IPv4	允许	资源池内全部节点之间通信
入站	ALL	容器网络 CIDR	1-65535	IPv4	允许	全部 Pod 之间通信
入站	ALL	100.64.230.0/24	1-65535	IPv4	允许	允许百舸监控、日志等子网与节点通信
入站	TCP	ALL	30000-32767	IPv4	允许	开通资源池内全部 NodePort 类型 Service 之间基于 TCP 协议通信
入站	UDP	ALL	30000-32767	IPv4	允许	开通资源池内全部 NodePort 类型 Service 之间基于 TCP 协议通信
出站	ALL	ALL	1-65535	IPv4	允许	开通资源池对外全部通信
入站	TCP	ALL	22	IPv4	允许	开通公网 SSH 登录端口
入站	ICMP	ALL	不涉及	IPv4	允许	开通 Ping 操作

GPU虚拟化和MIG虚拟化

产品概述

通过虚拟化技术将一张物理 GPU 卡切分为多个相互隔离的实例，支持开发机、训练任务、推理服务以 0.x 卡 的粒度申请资源，从而实现算力与显存的精细化分配。



核心价值

- 成本优化：支持单卡多任务并行，显著提升碎片化资源的利用率。
- 强隔离性：提供显存隔离与算力限制，确保任务间互不干扰。
- 场景覆盖：全面支持开发、训练、推理全流程业务。

虚拟化方案对比

百舸当前提供两种主流的虚拟化方案，您可以根据具体的芯片型号和隔离需求进行选择：

特性	sGPU 虚拟化	MIG 虚拟化
技术实现	百度自研内核态虚拟化方案	NVIDIA 硬件级多实例切分
主要功能	提供显存硬限制，算力时分复用	提供硬件级的显存与算力完全隔离
支持芯片	A100 等，更多芯片类型请咨询您的客户经理。	A100等，更多芯片类型请咨询您的客户经理。
切分粒度	百分比切分，支持提交0.X卡级别人物	固定比例切分（2/4/7 份）
核心优势	兼容性广，部署灵活	隔离性最强，任务间零干扰

使用说明

队列管理

注意：**同一个资源池内可以同时存在普通队列、sGPU 队列和 MIG 队列。

操作限制：

1. sGPU 虚拟化：

- 资源池相关组件达到一定版本，若开启该功能时提示组件版本低，请联系百舸技术人员升级相关组件。
- 指定物理队列开启虚拟化功能（控制台的GPU虚拟化）后，系统将自动筛选出支持虚拟化的可用节点。若所有节点不可用，请联系百舸技术人员升级相关组件。

2. MIG 虚拟化：

- 开启 MIG虚拟化 的队列只支持已开启MIG配置的节点的移入和移出，不支持节点转让操作。
- 开启 MIG虚拟化 的队列不支持碎片治理功能。
- 同一个队列中所有节点的所有卡，必须保持一致的切分比例。
- 指定物理队列开启MIG功能，仅支持添加开启了MIG配置的节点，加入开启了MIG的队列后节点不支持关闭MIG配置，仅将节点移出队列后再进行关闭。

任务限制

在提交工作负载时，请注意以下差异：

资源配额：

- sGPU 任务：支持输入0.X卡，也支持输入整数卡。
- MIG 任务：申请的“卡数”对应切分后的 GPU 实例数量（例如 1 份代表 1/4 卡）。

禁用功能：

- 使用 MIG 的任务暂不支持开启 RDMA。
- MIG 虚拟化仅支持开发机、单Pod训练、标准推理服务，暂不支持多角色服务组、分布式推理、PD 分离部署、分布式训练。

操作指南

sGPU虚拟化

创建虚拟化队列

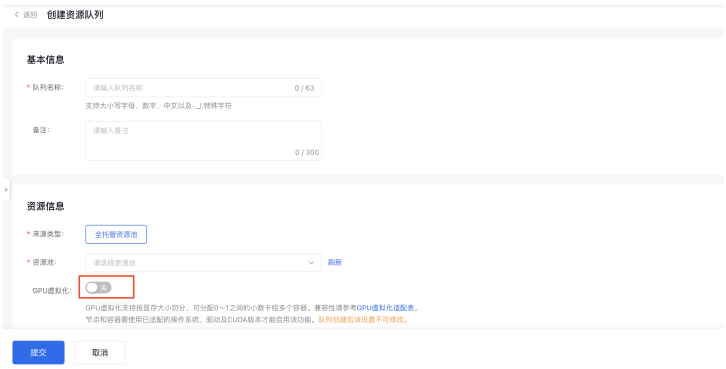
1.登录百舸AI计算平台AIHC控制台。

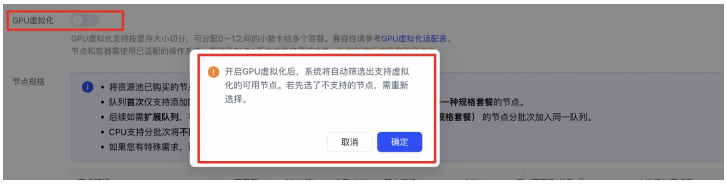
2.在左侧菜单栏选择全托管资源池，上方标签选择 “资源队列” 。

3.单击【创建资源队列】。



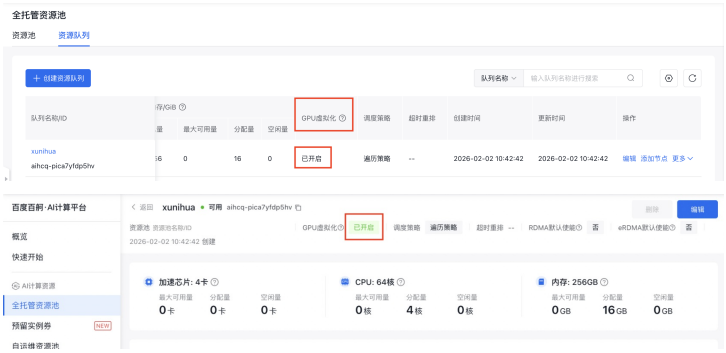
4.在【资源信息栏】-【GPU虚拟化】，点击开启，添加节点，即可创建GPU虚拟化队列，也可以在创建队列后再添加节点。







5.可以在队列列表中的 “GPU虚拟化” 列查看开启GPU虚拟化的队列。



创建虚拟化任务

以推理服务为例介绍创建虚拟化任务：

1.在左侧菜单栏选择在线服务部署，单击【+ 部署服务】。



2.在 【资源队列】 配置中，请从列表中选择带有 “**GPU 虚拟化**” 标签的队列。



3.在 【资源规格】 配置中，填写0-1之间1位小数。



4.完成相关配置后点击确定，即可完成创建虚拟化训练任务。

MIG 虚拟化

创建MIG队列 ： 请提工单创建 MIG 队列。

创建MIG 负载：

- 以在线服务部署为例，创建在线服务部署时，在【资源队列】配置中，请从列表中选择带有 “**MIG 虚拟化**” 标签的队列。如图所示，“加速芯片型号”列中若包含 MIG-1g.10gb 等格式，即表示为 MIG 实例。



- 在 【资源规格】 配置中，填写大于等于1的正整数，1卡代表一个MIG实例。



- 完成相关配置后点击确定，即可完成创建MIG的负载。

队列空闲资源出借

概述

本文介绍百舸全托管资源池如何通过**队列空闲资源出借**与**闲时分布式训练任务**，解决多队列资源忙闲不均、排队阻塞问题，提升全资源池整体算力利用率，同时保障核心业务资源优先供给。

功能概述

- 队列可开启**空闲资源出借**，将闲置算力共享给同资源池其他队列
- 分布式训练任务可开启**使用闲时资源**，优先使用共享闲时资源，减少排队
- 出借资源**可被抢占回收**：当出借队列有非闲时任务排队时，立即回收资源，保障核心业务
- 资源上限：全资源池所有队列均开启出借时，闲时任务最大可用量 = 资源池总资源量

使用前提

- 已登录百度智能云控制台，拥有**全托管资源池**，并可访问全托管资源池相关页面。
- 已具备资源池管理员或 AIHCfullcontrol 权限。

队列配置：开启闲时资源出借

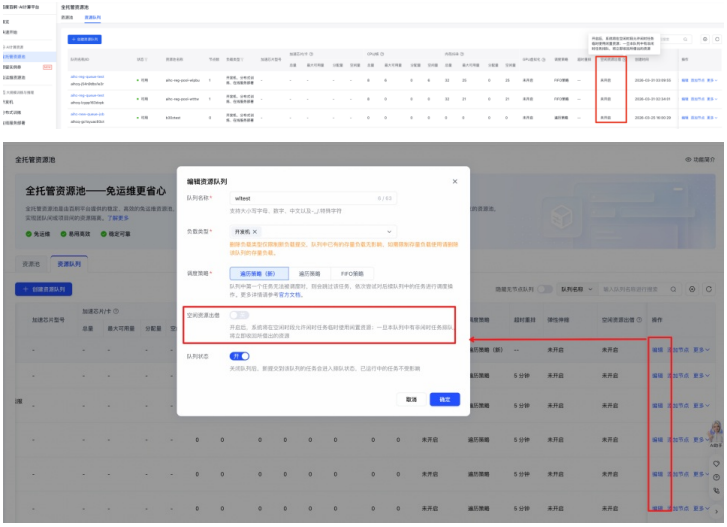
创建队列时配置

1. 进入 **全托管资源池 → 资源队列 → 创建队列**
2. 基础配置中找到**空闲资源出借**开关，默认**关闭**
3. 开启后，系统将在空闲时段允许闲时任务临时使用闲置资源；一旦本队列中有非闲时任务排队，将立即收回所借出的资源



编辑队列开启或关闭空闲资源出借

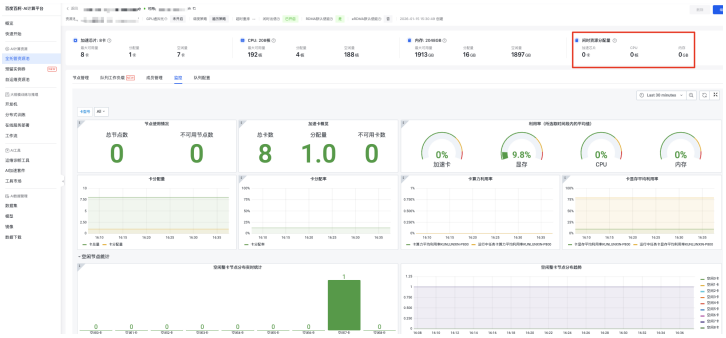
- 队列列表新增**闲时资源出借**字段，展示：已开启 / 未开启
- 存量队列默认：**未开启**
- 编辑关闭出借：不影响已在运行的闲时任务，仅禁止新闲时任务借用



队列详情与监控

队列详情页新增展示：

- 资源概览增加闲时资源分配量：包含本队列闲时任务 + 其他队列借用本队列的闲时任务（可被抢占）
- 当队列开启“空闲资源出借”功能时，闲时任务会使用部分当前空闲的加速芯片、CPU 和内存资源。 这部分被闲时任务占用的资源不计入队列和节点的已分配资源统计中， 因此您看到的“空闲量”仍可用于提交非闲时任务——系统将在需要时自动回收闲时任务所占资源，优先保障您的非闲时任务调度。
- 监控中卡分配趋势：只统计非闲时任务分配量
- 空闲整卡节点统计与分布趋势：闲时任务占用的卡计算在空闲卡中。（空闲任务可以非闲时任务抢占，不影响非闲时任务调度）



训练任务使用：使用闲时资源

详细使用请参考训练任务的使用闲时资源文档

- **操作步骤**
  - 进入分布式训练 → 创建训练任务
  - 选择有权限的**队列**
  - 开启**使用闲时资源**开关
  - 配置实例规格：
    - 不受本队列配额上限限制，仅受**资源池总量**约束
    - 必须与队列**同规格族**匹配
    - 队列无 GPU 则不可选 GPU 资源

- 提交任务，闲时资源充足时自动调度运行

- **抢占与中断机制**
  - 触发条件：出借队列有**非闲时任务排队** → 立即回收出借资源
  - 任务状态：被抢占后标记为**被抢占**，可在 Timeline 查看事件
  - 建议：开启任务通知与自动重试，配合 Checkpoint 实现无缝续跑
- **核心规则与约束**
  - **资源隔离**：闲时资源仅在**同全托管资源池**内共享，不跨资源池
  - **优先级保障**：非闲时任务 > 闲时任务，出借队列业务绝对优先
  - **规格强一致**：闲时任务必须使用与队列**同规格族**资源，不跨规格调度
  - **状态一致性**：关闭出借不中断存量闲时任务，仅拒绝新任务借用

常见问题

- Q1：闲时任务被抢占后怎么办？
  - A：任务会标记为**被抢占**，建议开启自动重试与 Checkpoint，平台会在资源恢复后自动续训。
- Q2：为什么无法开启闲时资源出借？
  - A：已经开启GPU虚拟化的队列不支持开启闲时资源出借。
- Q3：闲时任务可用资源受哪些限制？
  - A：受资源池总资源、芯片类型、可用区与 EHC 集群约束，不跨芯片类型、不跨 AZ、不垮EHC集群。

调度策略与超时重排

概述

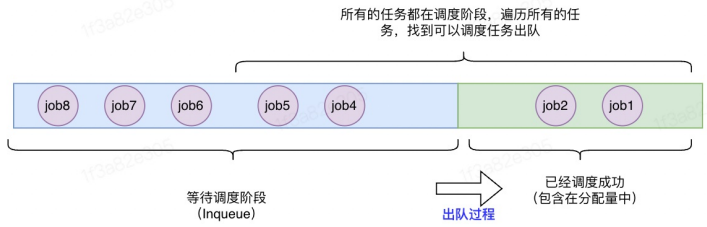
百度百舸 · AI 计算平台支持基于优先级的负载调度，旨在帮助用户在复杂的多任务环境中，通过合理的策略配置，最大化算力资源的利用率与任务出队效率。本文档详细阐述了平台支持的调度策略、任务流转机制以及解决"资源卡死"问题的超时重排功能。

核心概念与任务流转

在百舸平台中，用户可为分布式训练任务、开发机、在线推理服务设置**高、中、低**三个优先级。优先级决定了任务在资源竞争中的权重。平台目前提供三种调度策略：**遍历策略（新）**、**遍历策略**、**FIFO策略**。

一、遍历策略（新）的调度流程

队列状态默认为打开时，所有未调度成功的任务都属于**等待调度阶段**（状态为**调度中**）。



1. **任务流转流程**

遍历策略（新）的调度流程仅含"出队"环节：

环节	定义	说明
出队过程	从**等待调度阶段**（任务状态为**调度中**）进入**调度成功**（Running）	满足具体的资源申请要求（如GPU、CPU等），开始运行。

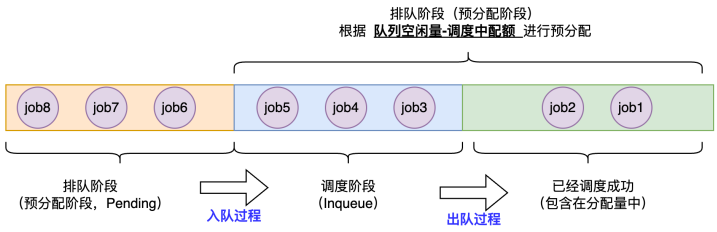
2. **关键判定条件**

- **出队条件**：满足调度中任务的各种资源申请要求。

> **注意**：调度策略仅影响尚未开始执行的任务顺序，不会影响已经开始运行的任务。

二、FIFO策略、遍历策略的调度流程

FIFO策略、遍历策略下的任务从提交到运行，主要经历两个阶段：**排队阶段（即预分配阶段）**和**调度阶段**。



1. **任务流转流程**

任务的调度过程分为"**入队**"与"**出队**"两个关键环节：

环节	定义	说明
入队过程	从**排队阶段**（任务状态为**排队中**）进入**调度阶段**（任务状态为**调度中**）	申请队列配额，计算资源分配量。
出队过程	从**调度阶段**（任务状态为**调度中**）进入**调度成功**（Running）	满足具体的资源申请要求（如GPU、CPU等），开始运行。

2. **关键判定条件**

- **入队条件**：任务申请总资源 <= 队列空闲量 - 调度中任务申请资源配额。
- **出队条件**：满足调度中任务的各种资源申请要求。

> **注意**：调度策略仅影响尚未开始执行的任务顺序，不会影响已经开始运行的任务。

调度策略详解

百舸平台在队列维度提供了三种核心调度策略，您可在资源配额界面进行配置。

遍历策略（新）

适用场景：追求资源利用率最大化，允许任务插队以填补资源碎片的场景。

- **策略逻辑**：
  - 默认所有未调度成功的任务在同一个调度阶段，遍历所有未调度成功的任务，找到可以调度的任务出队调度。
- **优势**：
  - 解决原始调度策略中，若一个任务进入“调度中”但无法真正运行（如因资源碎片无法满足），它将一直占据配额，导致后续所有任务阻塞的风险。
  - 能够真正灵活利用零散资源，让小规格任务有机会在大任务等待期间快速执行。
- **场景示例与调度行为解析：**为了帮助您更直观地理解遍历策略在不同资源竞争环境下的表现，以下提供两个典型场景及其应对建议。
  - **场景1：资源不足时的“队头阻塞”与跳过机制**
    - **场景描述**：队列当前空闲资源为 16 卡。此时，队头任务（任务 A）申请 24 卡，由于资源不足无法进行调度。根据遍历策略的“跳过机制”，系统不会让后续的小任务无限等待，而是会暂时跳过任务 A，允许后续资源需求满足的任务（任务 B）优先被调度。

任务名称	优先级	创建时间	申请卡数	行为解析
任务B	高	11点	16卡	**优先被调度**：资源满足，跳过阻塞的 任务A 先行调度。
任务A	高	10点	24卡	**调度中（等待调度阶段）**：资源不足，暂时滞留在等待调度阶段。
任务C	中/低	12点	16卡	**调度中（等待调度阶段）**：资源不足，暂时滞留在等待调度阶段。

- **场景2：队列空闲配额足够，但由于资源碎片会跳过前序任务，让后续满足碎片的任务优先调度**
  - **场景描述**：队列空闲量 16 卡（分布为：1 整机 + 2 机 × 4 卡）。
    - **任务 A**（10点创建，需 2 整机）因为资源碎片无法调度，处于**调度中**
    - **任务 B**（11点创建，需 1 整机）资源充足，跳过任务A，被优先调度，一直持续运行到14点
    - **任务 C**（12点创建，需 1 整机）任务B调度成功后，只剩2 机 × 4 卡，资源碎片导致无法调度
    - **任务 D**（13点创建，需 4卡 ）资源充足，跳过任务C，被优先调度
  - **任务状态详情**：

任务名称	优先级	创建时间	申请卡数	行为解析
任务A	高	10点	16卡（2机）	**调度中（等待调度阶段）**：资源碎片化，无法启动，但不会阻塞后续任务。
任务B	高	11点	8卡（1机）	**优先被调度**：资源充足，跳过任务A，被优先调度。
任务C	高	12点	8卡（1机）	**调度中（等待调度阶段）**：资源碎片化，无法启动，但不会阻塞后续任务。
任务D	高	13点	4卡	**优先被调度**：资源充足，跳过任务C，被优先调度。

FIFO策略 (先进先出) 适用场景：对任务调度顺序有严格要求，需严格按照创建时间顺序执行的场景。

- **策略逻辑**：
  - **入队**：按优先级（高>中>低）排序，同优先级下按创建时间（早>晚）排序。
  - **出队**：同入队逻辑，优先级高且创建时间早的任务优先获得资源。
- **优势**：保证最早提交的高优任务最先被调度；允许早提交的大规格任务持续积累资源直至成功。
- **局限**：可能导致晚提交的小规格任务因大规格任务“占坑”而长期无法获取资源，降低整体资源碎片利用率。
- **场景示例：**为帮助您快速理解FIFO调度策略，我们提供了以下典型场景示例。
  - **场景1：任务A因资源需求持续占据队头**
    - **场景描述**：队列当前空闲量为16卡，且无调度中任务。用户依次创建任务A、任务B、任务C，其中任务A因资源需求较大，将始终停留在队头位置，直至预分配资源成功并进入调度中阶段。
    - **运维建议：**将任务A的优先级调整为中优，将任务B的排队顺序调整到队首，让任务B入队，顺利进入调度阶段。

任务名称	优先级	创建时间	申请卡数
任务A	高	10点	24卡
任务B	高	12点	16卡
任务C	中	11点	16卡

- **场景2：后提交的高优先级任务B可以排到调度中的中优先级任务A的前面**
  - **场景描述**：T1时刻，队列空闲16卡（1整机+2机×4卡），任务A调度中。T2时刻，提交任务B，因优先级高于任务A，直接进入调度中，满足资源条件后顺利运行。
  - **任务状态详情**：

任务名称	优先级	创建时间	申请卡数	阶段
任务B	高	11点	8卡（1机）	优先级高于任务A，可直接入**队首**，满足资源条件即调度成功。
任务A	中	10点	16卡（2机）	10点提交时进入调度中，由于任务B的插入，导致该任务重新进入排队。
任务C	高	13点	16卡（2机）	排队中（预分配阶段）
任务D	中	12点	16卡（2机）	排队中（预分配阶段）

遍历策略 (Best Effort) 适用场景：追求资源利用率最大化，允许任务插队以填补资源碎片的场景。

- **策略逻辑**：
  - **入队**：遍历**排队中（****即预分配阶段）**的任务。若队首任务无法入队（如资源不足），则跳过并尝试后续任务。
  - **出队**：遍历**调度中**的任务。若队首任务无法出队，跳过并尝试后续任务。
- **优势**：能够灵活利用零散资源，让小规格任务有机会在大任务等待期间快速执行。
- **风险**：若一个任务进入"调度中"但无法真正运行（如因资源碎片无法满足），它将一直占据配额，导致后续所有任务阻塞。
- **场景示例与调度行为解析：**为了帮助您更直观地理解遍历策略在不同资源竞争环境下的表现，以下提供三个典型场景及其应对建议。
  - **场景 1：资源不足时的"队头阻塞"与跳过机制**
  - **场景描述**：队列当前空闲资源为 16 卡。此时，队头任务（任务 A）申请 24 卡，由于资源不足无法立即入队（进入调度阶段）。根据遍历策略的"跳过机制"，系统不会让后续的小任务无限等待，而是会暂时跳过任务 A，允许后续资源需求满足的任务（任务 B）优先入队。
  - **任务状态详情**：

任务名称	优先级	创建时间	申请卡数	行为解析
任务 B	高	11点	16卡	**优先入队**：资源满足，跳过阻塞的 A 先行调度。
任务 A	高	10点	24卡	**阻塞（Pending）**：资源不足，暂时滞留预分配阶段。
任务 C	中	12点	16卡	**排队等待**：等待 B 调度结果及资源释放。

- **场景 2：细粒度资源碎片导致的"调度死锁"**
  - **场景描述**：队列空闲量 16 卡（分布为：1 整机 + 2 机 × 4 卡）。
    1. **任务 A**（10点创建，需 2 整机）率先进入**调度中**阶段，但因缺乏连续的整机资源而无法启动。
    2. **任务 B**（11点创建，需 1 整机）虽资源足够，但受限于调度中的任务A，，预分配资源不够，被卡在**预分配阶段**等待任务 A。
    3. **结果**：若不开启超时重排，任务 A 会一直占用配额却不释放，导致任务 B 及后续任务无法运行，形成死锁。
  - **运维建议**：
    1. **开启超时重排**：配置调度超时阈值。当任务 A 在"调度中"状态超过设定时间，系统将自动将其退回"预分配阶段"，释放调度中的配额，从而让任务 B 顺利进入调度中并成功运行。
    2. **人工干预**：管理员手动停止任务 A，解除阻塞。
  - **任务状态详情**：

任务名称	优先级	创建时间	申请卡数	当前阶段	状态说明
任务 A	高	10点	16卡 (2机)	**调度中**	资源碎片化，无法启动，阻塞后续任务。
任务 B	高	11点	8卡 (1机)	**排队中（预分配阶段）**	预分配资源不够（空闲16卡-调度中16卡 = 0卡），排队等待。
任务 C	高	13点	16卡 (2机)	**排队中（预分配阶段）**	排队等待。
任务 D	中	12点	16卡 (2机)	**排队中（预分配阶段）**	排队等待。

- **场景 3：大任务持续阻塞下的"资源利用优化"**
  - **场景描述**：与场景 2 类似，队列空闲 16 卡（1 整机 + 2 机 × 4 卡）。任务 A（需 2 整机）卡在**调度中**，导致后续任务 B、C、D 全部阻塞。此时，队列后方的**任务 E**（11点创建，仅需 1 整机）本可以运行，却被前方庞大的任务群阻挡。
  - **运维建议**：
    1. **开启超时重排**：任务 A 到D超时后自动退回"预分配阶段"，系统重新扫描队列，此时任务 E 因资源满足且顺序靠前（相对于被退回的 A），可进入调度中并成功运行。
    2. **人工干预**：停止任务 A，并将任务 B 至 D 的优先级调整为"中"或"低"。这将使高优先级的任务 E 在重排序后排在前面，从而顺利进入调度中。
  - **任务状态详情**：

任务名称	优先级	创建时间	申请卡数	当前阶段	优化潜力
任务 A	高	10点	16卡 (2机)	**调度中**	阻塞源，需通过超时或人工移除。
任务 B	高	10点	16卡 (2机)	**排队中（预分配阶段）**	大任务，建议降级优先级以释放通道。
任务 C	高	10点	16卡 (2机)	**排队中（预分配阶段）**	同上。
任务 D	高	10点	16卡 (2机)	**排队中（预分配阶段）**	同上。
任务 E	高	11点	8卡 (1机)	**排队中（预分配阶段）**	**小任务**，若能越过 A-D，即可利用当前碎片资源运行。

超时重排机制 (Timeout Requeue)

为了解决遍历策略中可能出现的"任务卡死"问题（即任务长期占据配额但无法运行），百舸提供了**超时自动重排**功能。

机制原理

当任务在调度阶段（Inqueue）因资源碎片、节点亲和性不满足等原因长期无法调度时，系统会监控其等待时间。一旦达到设定的**超时阈值**，系统将自动对该任务执行重排操作：

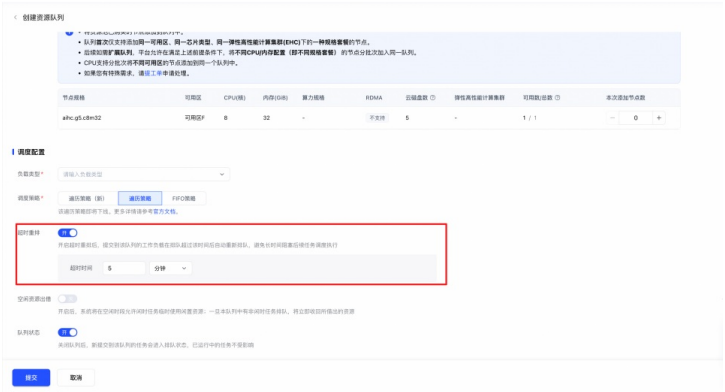
1. 将该任务重新置回预分配阶段（Pending）。
2. 任务进入冷却期，冷却结束后再次参与排队。
3. ⚠风险提示：***当前任务的默认冷却时间为1min。如果排队任务过多，会导致自动重排的高优任务在冷却时间后，依然进入排队的队首，导致排队死锁。***

配置建议

在创建或编辑队列时，可开启超时重排并设置阈值（最小值1分钟）：

- **阈值设置较大**：适用于需要为大规格任务预留资源的场景，避免大任务频繁被挤出。
- **阈值设置较小**：适用于希望快速遍历队列、提高资源碎片利用率的场景，让合适的小任务尽快调度成功。

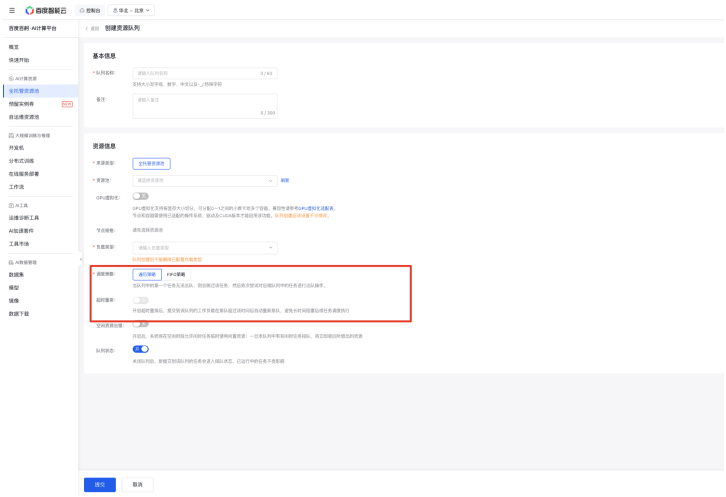
注：下图为超时重排设置

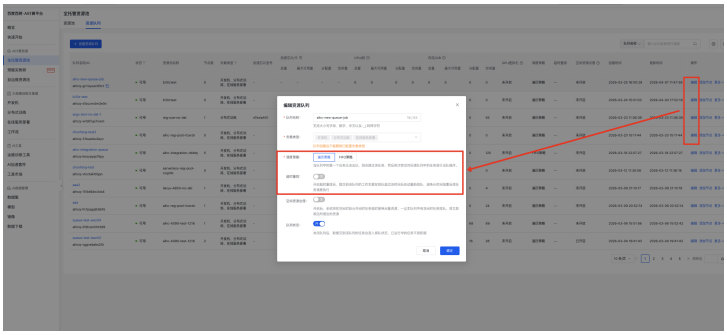


策略配置与最佳实践

1. 配置入口

用户可在**创建资源队列**或**编辑**现有队列的界面中，找到"**调度策略**"选项进行调整。





2. 场景化建议

场景描述	推荐策略	配置说明
资源利用率优先	遍历策略（新）	无需额外配置。通过遍历策略（新）为队列中未调度成功的任务调度空闲资源，避免因资源碎片化导致的浪费。
严格有序开发	FIFO策略	无需额外配置。适用于流水线作业，必须等上一个任务结束才能开始下一个。
高并发推理	遍历策略	侧重快速响应，利用遍历策略填补空闲资源，提升服务启动速度。注意容易有调度中任务阻塞问题，建议开启超时重排。
混合负载生产	遍历策略 + 超时重排	**建议开启超时重排**。***阈值设置较大**：适用于需要为大规格任务预留资源的场景，避免大任务频繁被挤出。* **阈值设置较小**：适用于希望快速遍历队列、提高资源碎片利用率的场景，让合适的小任务尽快调度成功。

3. 故障处理建议

若发现高优先级任务因前方任务"卡死"而无法调度：

1. **推荐**：**开启超时重排功能**，让系统自动处理。
2. **手动**：管理员可手动停止长期阻塞的"头任务"，或调整后续任务的优先级，以释放资源通路。

负载类型配置与使用

概述

本文档介绍全托管资源池中资源队列负载类型的配置与使用方法，涵盖创建资源队列时配置负载类型、在资源队列列表中查看和筛选负载类型，以及在不同业务场景下选择可用资源队列。通过为资源队列配置负载类型，您可以按业务场景对队列进行管理，并在开发机、分布式训练、在线服务部署等页面中使用匹配的资源队列，实现资源的精细化管理。

前提条件

- 已登录百度智能云控制台，并可访问全托管资源池相关页面。
- 已具备 队列管理员、资源池管理员或 AIHCfullcontrol 权限。
- 已在目标地域创建可用的全托管资源池；如需在业务页面选择资源队列，请确保该地域下已存在可用资源队列。

负载类型使用说明

资源队列支持以下负载类型：

业务场景	可见队列规则
开发机	仅显示已配置 **开发机** 负载类型的资源队列
分布式训练	仅显示已配置 **分布式训练** 负载类型的资源队列
在线服务部署	仅显示已配置 **在线服务部署** 负载类型的资源队列

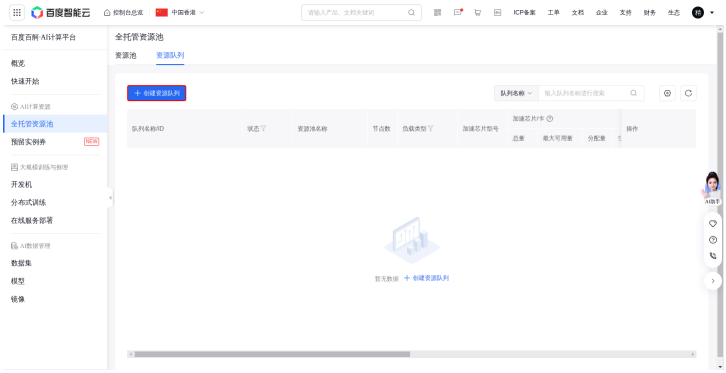
- 一个资源队列可以同时配置多个负载类型。
- 在资源队列提交前，您可以调整已选择的负载类型。
- 编辑资源队列时，可以新增负载类型；已经生效的负载类型不可直接删除。
- 负载类型变更仅影响后续新建、提交或部署操作，不影响已运行的开发机、分布式训练任务和在线服务部署实例。
- 功能上线前已存在的资源队列默认支持全部负载类型。

创建资源队列并配置负载类型

导航路径：**全托管资源池->资源队列**

创建资源队列时设置负载类型

在资源队列列表页，点击 **【创建资源队列】**，进入资源队列创建页面。



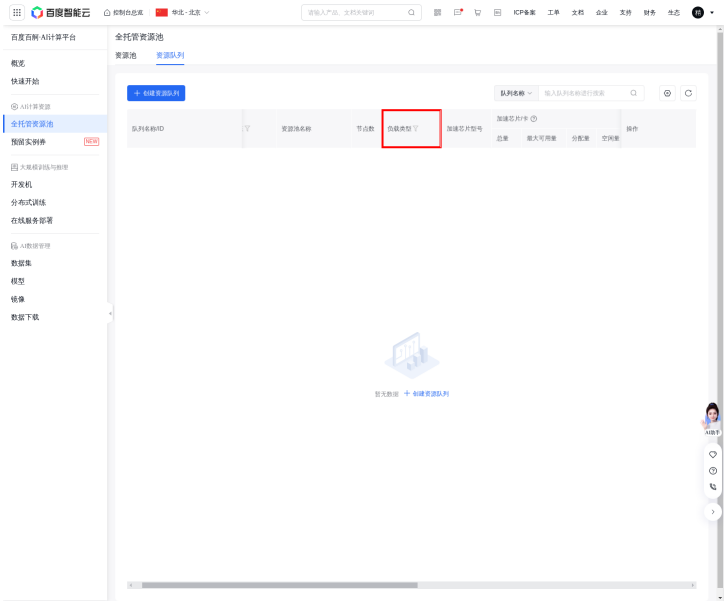


在创建页面填写基础信息，并在 **负载类型** 字段中选择资源队列支持的业务场景。

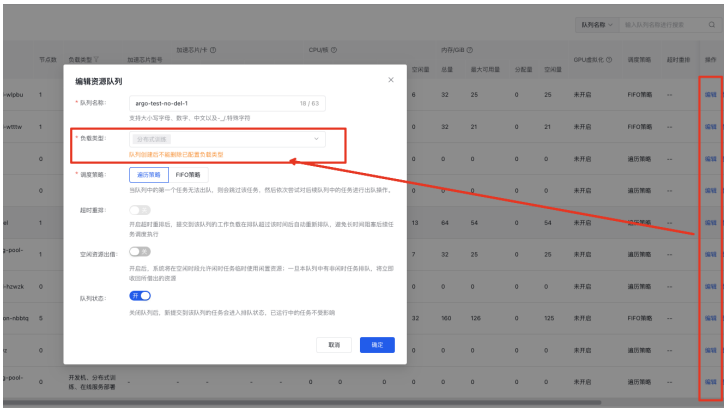
参数	必填	说明
队列名称	是	输入自定义队列名称，用于区分不同资源队列。
资源池	是	选择当前地域下已创建的资源池。未选择资源池时，页面无法提交。
负载类型	是	选择资源队列支持的业务场景，可多选。可选项包括 **开发机**、**分布式训练**、**在线服务部署**。
备注	否	输入便于识别的说明信息。

查看负载类型列&编辑负载类型

进入资源队列列表页后，可在列表中查看新增的 **负载类型** 列，用于识别每个资源队列支持的业务场景。



针对已有队列可以编辑队列信息，新增负载类型。负载类型变更仅影响后续新建、提交或部署操作，不影响已运行的开发机、分布式训练任务和在线服务部署实例。



在业务页面选择可用资源队列

场景入口

- 开发机：**开发机->创建实例**
- 分布式训练：**分布式训练->创建任务**
- 在线服务部署：**在线服务部署->部署服务**

使用规则

- 在业务页面打开资源队列选择控件后，系统仅展示支持当前业务场景的资源队列。
- 如果当前场景下没有可选资源队列，请返回 **全托管资源池->资源队列**，检查队列是否已配置对应的 **负载类型**。
- 如果一个资源队列同时配置了多个负载类型，则该资源队列可以在多个业务场景中复用。

变更影响说明

- 修改资源队列的负载类型后，后续新建的开发机、分布式训练任务和在线服务部署任务会按最新配置选择队列。
- 已经运行中的开发机、分布式训练任务和在线服务部署实例不受影响。

资源报表

概述

随着业务规模的扩大，精细化管理GPU资源、评估投入产出比（ROI）成为各团队的迫切需求。为了帮助您更好地掌握资源使用情况，识别低效任务，优化调度策略，机器学习平台推出了**资源报表**功能。

该功能旨在提供按"人/团队+时间段"维度的卡时统计与资源利用率分析，辅助您进行成本分摊、预算决策及资源优化。

适用范围与权限

- 适用场景
  - 本功能适用于**全托管资源池**与**自运维资源池**。
- 访问权限
  - 仅**资源池管理员**及拥有平台全控权限（AIHCFullControlPolicy）的管理员可见并使用该功能。
- 前置依赖
  - 使用该功能需满足以下条件，否则可能无法生成数据或数据存在延迟：
  - **调度器版本：** 需升级至支持资源报表的最新调度器版本。
    - *注：若版本过低，页面将提示升级。全托管客户需联系百舸技术支持；自运维客户需前往CCE云维与管理的组件管理中心升级 Kube-Scheduler。*
  - **监控实例：** 资源池必须关联有效的 **CProm 监控实例**。

- *注：若未关联CProm，将无法更新最新数据。*

报表数据说明

数据时效性

- **数据延迟：** 报表数据采用T+1模式生成。即每日凌晨生成前一天的数据，您在当天无法查看当天的实时数据。
- **数据可见性起点：** 报表数据的可见范围始于以下三个事件中发生最晚的时刻：
  1. 平台功能正式上线时间。
  2. 调度器版本升级完成时间。
  3. CProm实例关联成功时间。
- **特殊情况：** 若资源池近期发生过CProm实例切换或实例故障，受影响日期的统计数据可能存在偏差，仅供参考。

统计范围与粒度

- **统计粒度：** 天级别统计。
- **时间跨度：** 支持查询最近1年的数据。支持快捷筛选（近7天、近30天、本月）及自定义月份查询。
- **任务类型覆盖：** 涵盖开发机、训练任务、推理服务。
- **历史数据处理：** 即使任务已被删除，或队列已被销毁，其在历史时间段内产生的资源消耗数据仍会被计入统计。

核心指标定义

本章节详细定义了报表中涉及的关键指标，确保您对数据的理解准确无误。

资源池维度报表 聚合该资源池下所有队列的全量负载数据。

指标名称	定义说明	计算逻辑
总占用卡时	反映资源池的真实负载总量。	∑(单个任务占用GPU数 × 实际运行时长) *注：支持按开发/训练/推理类型拆解*
GPU/显存平均利用率	反映资源池的整体利用效率。	统计周期内，所有占用GPU卡的任务运行中的平均利用率。*注：支持按负载类型（开发机 / 训练 / 推理）分别查看GPU利用率与显存利用率。*

队列（团队）维度报表 聚合该队列下所有负载类型及所有用户的执行数据。

指标名称	定义说明	计算逻辑
队列总占用卡时	该队列被实际消耗的资源总量。	∑(本队列所有任务占用GPU数 × 运行时长)
队列GPU/显存利用率	该队列资源的使用强度。	统计周期内，仅针对队列中所有占用GPU卡的任务运行中的平均利用率。

用户维度报表 统计指定用户在某个队列中发起的所有负载任务数据。

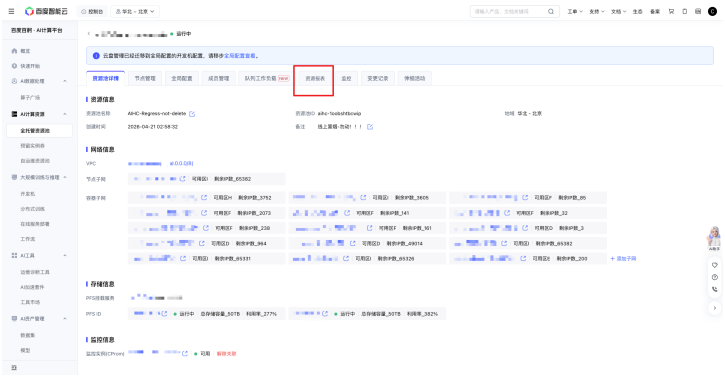
指标名称	定义说明	计算逻辑
用户总占用卡时	指定用户在时间窗口内的资源消耗。	∑(用户所有任务占用GPU数 × 运行时长)
用户GPU/显存利用率	该用户的任务平均利用水平。	该用户所有任务“实际运行期间”，其占用资源的平均利用率。

使用说明

页面入口

登录百度百舸·AI计算平台控制台，在左侧导航栏选择 **AI计算资源 > 全托管资源池**。

在资源池列表中，点击目标资源池名称进入资源池详情页，然后点击顶部导航栏中的 **资源报表** 页签，即可查看该资源池的资源使用统计数据。



时间筛选

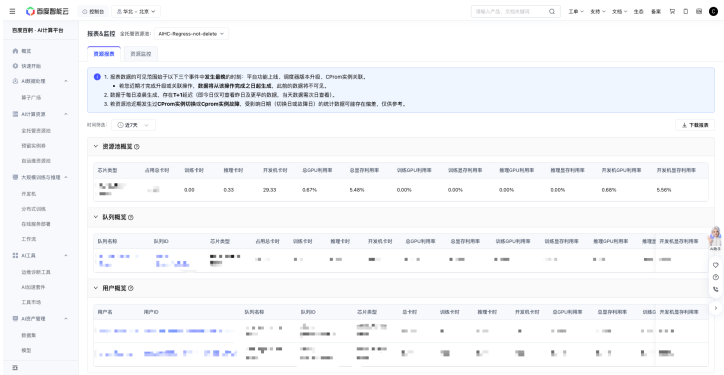
您可以通过顶部的时间选择器筛选数据。

支持维度： 近7天、近30天、本月，或自定义选择月份。

建议单次查询跨度不超过3个月，以保证查询性能与数据准确性。

数据导出

支持将当前页面数据导出为本地文件，方便您线下存档或进一步分析。导出的数据将严格遵循您设定的筛选时间范围。



模型缓存加速配置

概述

- **功能简介**

模型缓存加速服务是百舸·AI计算平台为优化模型加载效率而提供的核心功能。该服务利用计算节点的空闲内存构建缓存池，将公共BOS（Baidu Object Storage）路径中的模型文件预加载至本地内存。通过减少对远端存储的依赖，显著降低模型重复下载时间，从而大幅提升在线服务的部署、扩容及推理任务的启动速度。

- **核心价值**
  - **加速加载：** 直接从内存读取模型，避免重复从BOS拉取，显著缩短冷启动时间。
  - **节省带宽：** 减少对外部存储的高频访问，降低网络流量消耗。
  - **资源优化：** 合理利用节点闲置内存资源，提升整体集群利用率。

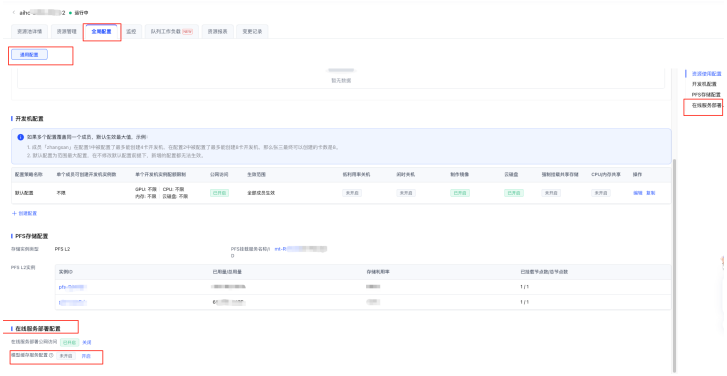
权限与访问

- **访问入口**

登录百度百舸·AI计算平台控制台 → 左侧导航栏选择 **【全托管资源池】** → 选择目标 **【资源池】** 进入详情页 → 点击 **【全局配置】** → 查看 **【模型缓存服务配置】**。

- **权限说明**

仅 **资源池管理员** 或拥有 `AIHCFullControlPolicy` 权限的用户可进行相关配置操作。



模型缓存加速配置说明



模型缓存加速服务默认不开启，需管理员在资源池级别手动配置。**参数配置说明如下：**

参数名称	说明	注意事项
目标队列	选择需要开启缓存加速的计算队列。	需选择处于“可用”状态的队列。
单节点缓存内存	设置每个节点用于缓存的内存大小。	建议不超过节点总内存的20%，建议在100GiB左右，请根据模型大小评估。
目标节点数	选择参与缓存服务的节点数量。	不可超过该队列当前可用节点数。
缓存池总容量	缓存池总容量 = 设置的目标节点数 × 单节点缓存内存。实际生效容量取决于队列中当前可用的节点数量。	若部分节点故障或被移出，实际缓存容量将小于目标值，可能影响模型缓存效果。

> **注意：** 缓存池所占内存被视为系统预留资源。例如，若节点最大可用内存为1000GiB，设置200GiB作为缓存，则用户可提交的工作负载内存上限将降为800GiB。

使用约束与注意事项

- **节点状态要求**
  - 仅 **"可用"** 状态的节点支持模型缓存加速。
  - 自愈中、不可用、已封锁的节点不支持。
  - 节点故障或重启后，缓存内容将清空，首次加载仍需从BOS读取。
- **配置限制**
  - **不可变性：** 一旦配置生效，若保持原队列不变，**不允许修改**"单节点缓存内存"大小，以避免缓存池结构频繁重建。如需调整，需更换目标队列。
  - **资源冲突：** 若节点剩余内存不足以容纳设置的缓存大小（例如节点已有负载占用900GiB，试图设置200GiB缓存），缓存Worker将无法调度。
- **服务范围**
  - 缓存服务对部署在同资源池的所有在线服务生效。
  - 仅对**公共账号BOS**路径生效，挂载其他类型存储或非公共账号BOS路径时加速不生效。
  - 当缓存空间不足或服务不可用时，系统将自动降级为从BOS直接加载，**不会阻塞**服务部署。

服务状态流转与故障排查

模型缓存服务开启后，通常需要 **5-10分钟** 进行初始化。

状态定义

状态	说明	处理建议
可用	缓存服务正常运行，节点数满足预期。	正常使用。
部分可用	实际缓存节点数小于目标节点数。	检查队列中是否有节点被移出或发生故障，建议恢复节点。
未就绪	服务正在初始化或配置异常。	若长时间未就绪，请确认目标队列是否存在，或联系管理员。

若服务长时间处于"未就绪"或转为"部分可用"，请按以下步骤检查：

1. **确认队列状态：** 目标队列是否已被删除或修改。
2. **检查节点健康度：** 目标队列中的可用节点数是否达到配置要求。
   - 如有故障节点，请等待自动恢复或手动干预。
   - 如有被移出节点，请重新加入队列。

在线服务部署的使用说明

开启此功能后，系统将利用计算节点的内存资源构建高速缓存。当您使用公共模型进行部署时，模型文件将直接从节点内存加载，而非从远端存储下载。

适用条件：

- **模型类型：** 仅限**公共模型**（存储于公共账号BOS路径）。
- **资源环境：** 所选的**计算队列**所在的资源池必须已开启模型缓存加速服务。





资源池监控

资源池监控

百舸平台为全托管资源池提供了一套**逐层下钻、全链路覆盖的监控体系**。资源池关联 CProm 实例后，可查看资源池、队列、节点等多层级监控指标，帮助您从宏观到微观全面掌握资源状态，快速定位性能瓶颈，并优化任务调度策略。

关联 CProm 实例

使用资源池监控前，需为资源池关联 CProm 实例。您可以在创建资源池时关联，也可以在资源池创建完成后，在资源池详情页中补充关联。

创建资源池时关联

创建全托管资源池时，可在资源池配置页面通过 **关联 CProm 实例** 参数选择已有 CProm 实例。

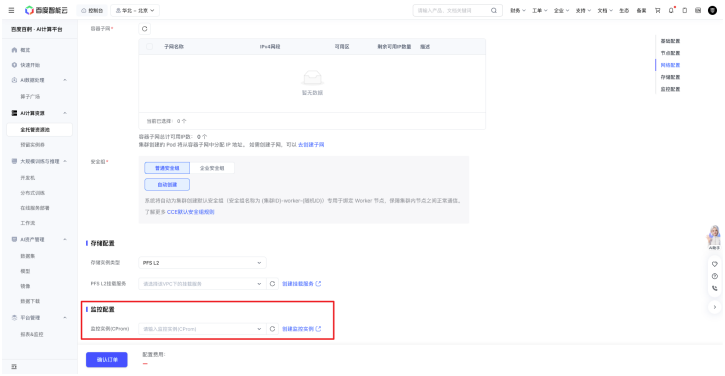
操作步骤：

1. 进入百舸控制台，创建全托管资源池。
2. 在资源池配置页面，找到 **关联 CProm 实例** 参数。

3. 选择需要关联的 CProm 实例。

4. 完成其他资源池参数配置后，提交创建。

创建完成后，可进入资源池监控页面查看监控数据。



资源池创建后关联

如果创建资源池时未关联 CProm 实例，可在资源池详情页中补充关联。

操作步骤：

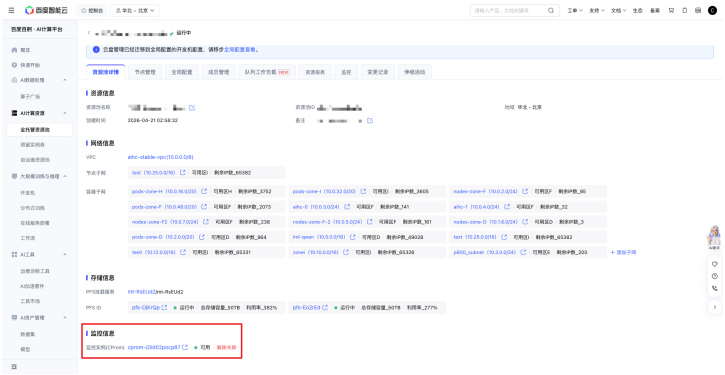
1. 进入百舸控制台，打开目标全托管资源池详情页。

2. 在资源池详情页中找到 CProm 实例关联入口。

3. 点击关联或编辑 CProm 实例。

4. 选择需要关联的 CProm 实例并确认。

关联完成后，可进入资源池监控页面查看资源池、队列和节点维度的监控指标。



监控体系概览

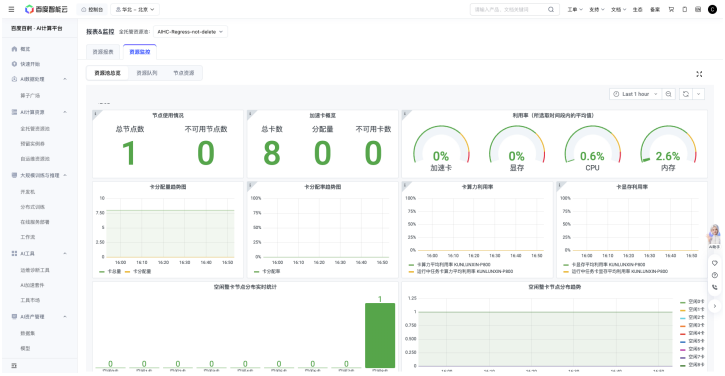
资源池关联 CProm 实例后，监控页面将按资源池、队列、节点和负载联动四个层级展示监控信息，助您高效运维：

1. **资源池层**：宏观审视整体算力、存储及网络资源的使用情况，把握全局健康度。

2. **队列层**：深入分析特定队列的资源利用率与负载排队状况，为调度策略优化提供数据支撑。

3. **节点层**：精准监控单个物理节点的CPU、内存、加速卡等核心指标，快速排查性能瓶颈。

4. **负载联动**：结合工作负载视图，可筛选特定队列任务，并支持手动终止低优先级任务，以保障高优先级任务的快速调度。



资源池与队列监控指标

资源池与队列监控面板提供实时与历史维度的关键指标，帮助您掌握资源运行状态、识别潜在瓶颈。

指标分类	监控指标	指标含义
节点使用情况（实时）	总节点数	当前资源池/队列中包含的物理节点总数。
	不可用节点数	因故障、维护或被封锁而处于不可调度状态的节点数量。队列维度该数值直接影响队列的最大可用资源上限。
加速卡概览（实时）	总卡数	资源池/队列内所有节点上加速卡（如GPU）的物理总数。
	分配量	当前已被任务占用的加速卡数量（包含部分分配的情况），直接反映资源的实际使用压力。
	不可用卡数	因节点故障或硬件异常导致无法被使用的加速卡数量。
资源利用率（平均值）	加速卡利用率	反映所选时间范围内同一类型加速卡的平均算力利用率，用于衡量算力负载水平。 • 计算分两步：先在同一时刻对纳入统计的同类型加速卡求平均，再对所选时间范围内各时刻的结果求平均。 • 数据来源上，原始指标每 15 秒采集一次；资源池总览与物理队列监控每分钟记录一次该时刻的指标值。数据持续采集与计算，无需等待任务结束即可查看。
	显存利用率	加速卡显存的平均占用比例，用于评估内存密集型任务带来的资源压力。
	CPU利用率	节点CPU核心的平均使用率，辅助判断计算瓶颈是否位于CPU侧。
	内存利用率	节点系统内存的平均占用比例，用于评估任务内存需求与节点容量的匹配度。
趋势分析	卡分配量趋势	展示加速卡分配数量随时间的变化趋势。
	卡分配率趋势	展示已分配卡数占总卡数的比例随时间的变化趋势。
	卡算力利用率趋势	展示同一加速卡类型的平均算力利用率随时间的变化。 • 资源池总览与物理队列监控为每分钟记录的指标值，计算对应时刻纳入统计的同类型加速卡的平均利用率。 • 每分钟记录的值为评估时的指标值，并非该分钟内所有原始采样值的平均值。
	卡显存利用率趋势	展示加速卡显存平均利用率随时间的变化曲线。
空闲节点统计	空闲整卡节点分布（实时）	按空闲卡数（0至8卡）统计当前完全空闲的节点数量，用于评估可调度资源的碎片化程度。
	空闲整卡节点分布趋势	展示空闲节点分布随时间的变化，帮助您预测资源释放节奏与最佳调度窗口。
PFS存储监控（资源池维度独有）	总容量 & 已使用	展示各PFS实例的总存储容量与实际使用容量。
	读写吞吐	展示各PFS实例的单位时间读写数据量（读/写吞吐速率）。

节点监控指标

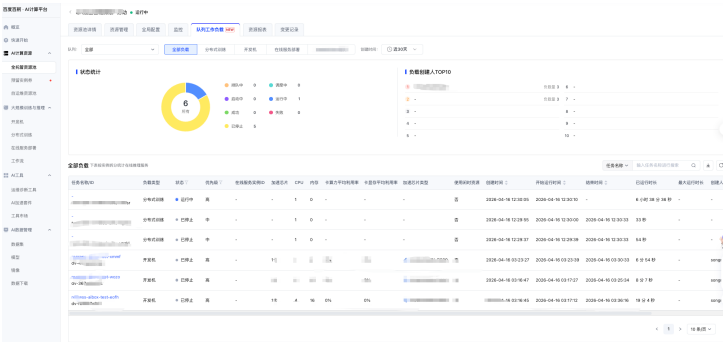
节点监控提供单台物理节点的详细运行数据，是进行深度问题排查的核心工具。

指标分类	监控指标	指标含义
资源分配详情	加速卡/CPU/内存分配量	当前节点已分配资源量与节点资源总量的对比。
资源利用率（平均值&趋势图）	加速卡利用率	展示所选时间范围内当前节点加速卡的平均利用率及变化趋势。 • 节点监控直接查询每 15 秒采集的原始指标，不经过资源池总览与物理队列监控的每分钟记录环节，无需等待任务结束。
	显存利用率	时间范围内显存的平均利用率趋势图。
	CPU利用率	时间范围内CPU的平均利用率趋势图。
	内存利用率	时间范围内内存的平均利用率趋势图。
硬件健康度	卡功耗 (W)	时间范围内GPU加速卡的功耗趋势图。
	卡温度	时间范围内GPU加速卡的温度趋势图。
网络&磁盘	网络收发数据量	单位时间内网络数据的接收与发送量趋势图
	磁盘读写数据量	单位时间内磁盘数据的读取与写入量趋势图
	磁盘读/写IO速率	单位时间内磁盘的读取/写入操作速率趋势图

队列工作负载

资源池监控可以与工作负载联动分析：结合队列工作负载查看功能，筛选特定队列的负载，手动终止低优任务，加速高优任务调度。

更多队列工作负载的介绍请参考相关文档



资源规格模板

概述

资源规格模板是 AIHC 百舸平台为队列管理提供的标准化资源配置能力。管理员可在队列维度预设 GPU 数量与 CPU 数量、内存的搭配规格，算法工程师在创建任务时可直接套用模板，无需手动填写资源参数，从而降低配置门槛、减少资源碎片化，提升集群整体利用率。

功能概述

- **模板配置**：管理员可为队列预设多组资源规格模板，覆盖分布式训练、开发机、在线服务部署等负载场景。
- **自动创建**：新增含 GPU 的队列时，系统自动按节点最大规格生成常用卡数模板，无需手动配置。
- **两种使用模式**：支持**自由创建模式**（参考套用）和**严格规定模式**（强制选择），管理员可按需配置。
- **任务创建集成**：算法工程师创建任务时可通过模板一键回填资源参数，严格规定模式下只能从预设模板中选择。

- **复制/重跑兼容**：严格规定模式下，复制或重跑任务时自动匹配原规格对应的模板，若无匹配则提示重新选择。
- **权限分级管理**：平台管理员、资源池管理员、队列管理员按权限范围管理模板，算法工程师只读查看。

使用前提

- 已在 AIHC 百舸平台创建自运维资源池或全托管资源池，并完成队列配置。
- 需要配置或管理模板的操作者具备相应管理员权限（平台管理员 / 资源池管理员 / 队列管理员）。
- 算法工程师在创建任务时，目标队列已配置资源规格模板。

配置资源规格模板（管理员）

进入模板配置页面 全托管资源池：

1. 进入**全托管资源池**管理页面，选择资源队列。
2. 在**资源队列**列表中，点击目标队列，进入后点击**队列配置**，进入模板配置页面，开启即可配置模板。



系统自动创建模板 新增含 GPU 的队列并开启资源规格模板配置时，系统将根据队列内节点的最大 GPU 规格，自动生成常用卡数的资源模板：

- 自动创建卡数覆盖：1 卡、2 卡、4 卡、…，直至节点最大卡数。
- CPU 与内存按节点最大规格等比分配。
  - 请注意：节点中最大可用内存与 CPU 数量会根据安装插件动态变化，自动创建的模板规格仅为创建时刻的参照值，请管理员根据变化变更模板配置。

示例： 队列内节点为 A800 × 8 卡，系统自动创建 1/2/4/8 卡四个规格模板。

队列显卡变更时，系统将自动删除原 GPU 类型对应的模板，并按新显卡规格自动创建对应模板。



编辑模板

1. 在模板配置页面点击**编辑**，即可进入模板编辑弹窗。
2. 按下表填写各字段：

字段	说明	是否必填
模板名称	自定义名称，同队列内不可重复	是
GPU 类型（芯片型号）	选择该模板对应的显卡型号；GPU 数量大于 0 时必填	条件必填
卡数	单任务使用的 GPU 数量，需大于 0	是
CPU（核）	分配给任务的 CPU 核数	是
内存	分配给任务的内存大小（单位：GiB）	是
适用模块	勾选适用的负载类型：**分布式训练** / **开发机** / **在线服务部署**，可多选	是

3. 填写完成后点击**保存**。

校验规则：

规则	说明
不可重复	同队列内，GPU 类型 + 卡数 + CPU + 内存的组合不可与已有模板重复
名称唯一	同队列内，模板名称不可重复
GPU 类型必填	当卡数 > 0 时，必须选择 GPU 类型（芯片型号）
必填项不可为空	模板名称、卡数、CPU、内存、适用模块均为必填项

复制模板 在模板列表中，点击目标模板行右侧的**复制**按钮，系统将自动创建一条与该模板规格相同的新模板行。修改所需字段后保存即可。

删除模板 在模板列表中，点击目标模板行右侧的**删除**按钮，确认后该模板将被移除。已被任务引用的历史规格不受影响。

配置模板管理模式（管理员）

管理员可在模板配置页面设置该队列（或负载类型）的模板使用模式：

模式	说明
自由创建模式（默认）	开启资源模板配置时默认为该模式，算法工程师创建任务时，资源填写区提供**资源模板**入口供参考套用，也可自行填写资源参数
严格规定模式	若该负载类型已配置模板，算法工程师只能从预设模板列表中选择，不可自定义填写；若某负载类型未配置任何模板，仍允许自定义填写

> 建议在对资源利用率要求较高或集群碎片化严重的场景下，启用**严格规定模式**。

创建任务时使用模板（算法工程师）

自由创建模式

1. 在创建任务页面，进入**资源配置**区域。
2. 点击**资源模板**入口，弹出当前队列下适用于该负载类型的模板列表。
3. 选择目标模板，系统自动回填 **GPU 类型**、**GPU 数**、**CPU 核数**、**内存**。
4. 回填后可根据需要手动调整，也可直接提交。

> 若所选模板的 GPU 类型在当前队列中不存在，系统将提示并禁止提交，请更换其他模板或联系管理员。 **严格规定模式**

1. 在创建任务页面，**资源配置**区域仅支持通过点击**资源模板**填写资源规格。
2. 从列表中选择适合的资源规格模板，系统自动填充对应资源参数。
3. 不支持手动修改资源参数，直接提交任务即可。

> 若该负载类型当前队列下暂无可选模板，系统将切换为自定义填写模式，允许手动输入资源参数。

复制/重跑任务的模板匹配（严格规定模式）

在严格规定模式下，复制或重跑已有任务时，系统将按以下规则处理资源规格：

场景	系统行为
找到完全匹配的模板（GPU 类型 / GPU 数 / CPU / 内存均一致）	自动选中对应模板，无需手动选择
未找到匹配模板，但该负载仍有可选模板	资源规格置空，并提示：**"当前队列已不支持原资源规格，请重新选择，原规格为 GPU×x，CPU×x，内存×xG"**
该负载类型当前队列下已无任何可选模板	切换为自定义填写模式，原规格自动填入，允许手动调整后提交

权限说明

角色	权限范围
平台管理员	可访问并修改所有资源池、所有队列的资源规格模板
资源池管理员	可访问并修改所管理资源池内所有队列的资源规格模板
队列管理员	可修改自己管理队列的资源规格模板
算法工程师	只读查看，不可修改（操作按钮隐藏）

常见问题

Q：新建队列后，系统没有自动生成模板，怎么办？

A：系统仅在队列包含 GPU 节点时自动创建模板。若队列为纯 CPU 队列，或节点资源尚未关联到队列，不会触发自动创建。请确认队列已正确关联 GPU 节点后，再手动添加模板，或联系平台管理员处理。

Q：严格规定模式下，算法工程师反馈没有合适的模板可选，怎么解决？

A：管理员可在模板配置页面为该队列和对应负载类型补充合适的规格模板。若某负载类型暂未配置任何模板，系统将自动允许自定义填写，不影响任务创建。

Q：更换队列显卡后，原有任务的历史记录是否受影响？

A：队列显卡变更仅影响模板列表的自动维护（删除旧模板、创建新模板）。已提交的历史任务及其资源记录不受影响。

Q：复制任务时提示"当前队列已不支持原资源规格"，如何处理？

A：此提示表示原任务所用资源规格在当前队列已无对应模板。请从现有资源规格模板中重新选择一个可用的资源规格模板后再提交。若有特殊需求，可联系队列管理员添加所需规格模板。

Q：资源规格模板适用于哪些资源池类型？

A：资源规格模板功能适用于**自运维资源池**下的物理队列，以及**全托管资源池**下的队列。

RapidFS Master托管实例缓存加速

概述

百舸**全托管资源池**除支持 RapidFS 全托管实例外，也支持通过关联 **RapidFS Master 托管实例**，将计算节点上闲置的本地磁盘构建为高性能分布式缓存层，为 AI训练、推理等场景提供低延迟的数据加速能力。本文仅限于百舸通过 **RapidFS Master 托管实例**的使用说明。

核心优势：

- **加速数据读取**：在分布式训练场景中就近缓存热数据，大幅缩短数据读取时延。
- **运维简单**：部署组与 Cache 节点由平台自动管理，用户无需手动配置。
- **数据安全**：RapidFS 提供只读缓存，写操作直接落入 BOS 对象存储，节点故障时不影响数据可靠性。
- **降低存储成本**：盘活 GPU 节点闲置本地盘，提供更灵活、更轻量化的存储加速方式。

> **加速范围说明**：当前仅支持加速 BOS 对象存储中的数据。

前提条件

在开启缓存加速功能前，请确认以下条件已满足：

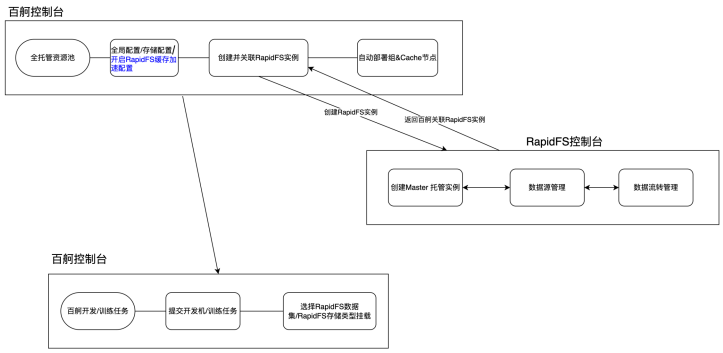
条件	说明
账号权限	需具备资源池**管理员及以上**权限才能进行全局配置操作。
RapidFS 权限	创建 RapidFS 实例需要账号已授予 `RAPIDFSFullControlAccessPolicy` 策略。
RapidFS白名单	RapidFS Master 托管实例目前是白名单功能，请提前联系RapidFS技术人员开通白名单。
本地磁盘要求	资源池中至少有节点拥有 **2 块及以上**物理磁盘（第一块盘不计入可用容量）。
网络要求	RapidFS 实例须与资源池处于**同一地域/VPC。**

> **注意**：若资源池所有节点均无可用本地磁盘，将无法开启该功能。

名词解释

名词	说明
RapidFS	极速文件缓存（Rapid FileSystem），依靠 BOS 作为存储底座，为 AI/大数据业务加速存储访问性能
Master 托管模式	RapidFS 集群中 Master 节点由百度云平台托管，Cache 节点部署在用户自有计算节点上
Master 节点	负责元数据维护、任务调度，为用户读写数据提供数据块定位信息
Cache 节点	负责缓存热数据，执行数据存储与计算任务，数量可线性扩展
部署组	具有相同磁盘配置（磁盘数量 × 磁盘容量）的 Cache 节点集合，由平台自动划分管理
M1 规格	RapidFS Master 小规格，最多托管 **50 个**节点
M2 规格	RapidFS Master 大规格，最多托管 **400 个**节点

使用流程



功能操作概要说明：

1. 在 RapidFS 控制台创建 Master 托管实例，导入数据源并配置数据流转
2. 在百舸资源池全局配置中开启缓存加速并关联 RapidFS 实例，平台自动完成部署组和 Cache 节点的初始化
3. 在百舸控制台开发机或分布式训练任务中挂载 RapidFS 数据集或者相关存储使用

RapidFS控制台相关功能

相关功能简介：

- 创建 RapidFS Master 托管实例，前提条件是：必须已有百舸全托管资源池。也可从百舸全托管资源池的全局配置中跳转到RapidFS控制台进行创建。
- 进行数据源管理：导入数据源&权限组的管理
- 在实际使用中按需进行数据流转的管理

RapidFS 实例要求：

- 实例接入类型必须选择 **百舸全托管资源池，且关联某一个百舸全托管资源池。**
- 与资源池处于同一地域/VPC

操作步骤：

- 创建RapidFS Master托管实例
  - 进入 **RapidFS 控制台**，点击**创建实例，部署模式选择【Master托管】模式**。
  - 在实例创建页面，**接入类型**选择**百舸全托管资源池**，从下拉列表选择您的资源池。
  - 根据资源池节点规模和文件数规模选择 Master 规格，不同Master规格的区别请参考RapidFS控制台说明。
  - VPC、子网等网络参数由系统自动填充。
  - 提交创建，等待约 **3 分钟**实例创建完成。

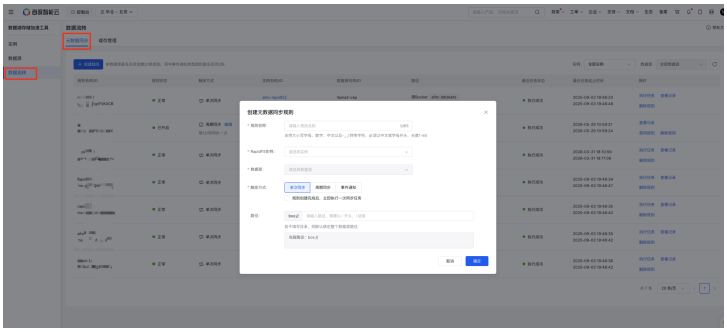
> **Master 规格选择建议**：若未来有节点扩容计划，或预期加速的 BOS 中文件数规模较大，建议直接选择M2规格，避免后续因规格限制影响扩展。



- 导入数据源
  - 创建缓存实例后，需要导入数据源，将对象存储（BOS）的存储桶或存储桶目录以数据源的形式导入缓存实例，以实现加速访问。数据源管理的详细操作说明见RapidFS数据源文档。



- 设置流转策略
  - 用户在百舸控制台使用RapidFS存储或数据集时，可以进一步配置数据流转策略和任务，优化缓存加速效果，详细操作说明见数据流转操作。



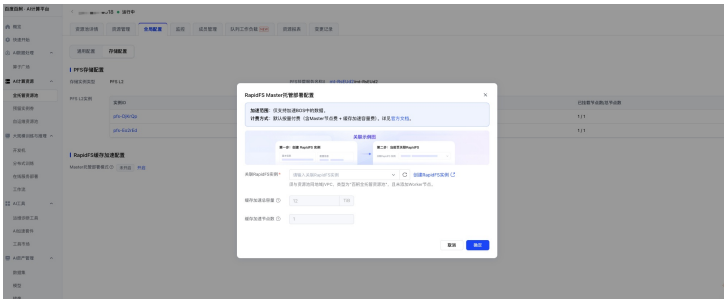
全托管资源池开启缓存加速配置

- 进入全局配置
  - 在百舸控制台，进入目标**全托管资源池**详情页。
  - 点击顶部导航栏中的**全局配置**选项卡，选择**存储配置**子标签页。



- 开启 Master 托管缓存加速
  - 在 **RapidFS 缓存加速配置**区域，找到 **Master 托管部署模式**行。
  - 默认状态为**未开启**，点击**开启**按钮。
  - 系统弹出配置框，按以下说明填写参数。
- 配置参数说明：只能**选择关联****了本资源池的RapidFS Master实例。**

项目	说明
RapidFS 实例	从下拉列表选择处于**运行中**状态的实例；创建中/创建失败的实例不可选
新建实例	若无可用实例，点击**创建 RapidFS 实例**跳转新页面创建（自动携带资源池参数）
刷新列表	若新建实例后未显示，点击刷新图标更新列表



- 确认并完成关联
  - 确认参数无误后，点击**确定**。
  - 配置完成后，**存储配置**页面将展示已关联实例的基本信息。请注意，**总容量数据的显示存在约1至3分钟的延迟**。



开发机和训练任务使用 RapidFS

缓存加速开启后，您可以在**开发机**和**分布式训练**任务中使用 RapidFS 进行数据加速。

方式一：数据集挂载（推荐，支持权限隔离） 推荐使用此方式实现数据集级别的权限隔离，适用于多子用户共享资源池的场景。

前提：需由具有 AssetFullControlPolicy 权限的用户先在 百舸控制台创建**数据集**并指定读/写权限，数据集相关文档请参考数据集文档。

操作步骤：

1. 用户在**百舸控制台/AI资产管理/数据集/** 创建数据集，并指定使用者（子账号）对数据集的读/写权限。
2. 开发人员在百舸**数据集列表**中，可以看到已被授权的 RapidFS存储类型的数据集。
3. 创建开发机或训练任务时，在数据集挂载配置中选择对应的 RapidFS存储类型数据集。
4. 填写挂载路径，提交任务。



方式二：存储挂载

1. 创建开发机或训练任务时，在**存储挂载**配置中，存储类型选择**极速文件缓存 RapidFS**。
2. 在实例列表中，您将看到：
   - 账号下所有**全托管 RapidFS 实例**
   - 以及已关联到当前资源池的**Master托管 RapidFS 实例**
3. 选择对应的 RapidFS 实例，选择数据源，填写挂载路径，提交任务。



RapidFS 缓存加速进阶使用说明 基于 RapidFS 极速文件缓存，可以在 BOS（对象存储）之上构建近计算的存储加速层，大幅提升 AI 训练、推理以及数据处理的存储访问性能。既可以选择由百度云平台完全托管的形态，也可以选择上述性价比更高的 Master 托管形态，从而将计算节点的空闲磁盘资源利用起来。

无论选择哪种形态，创建好 RapidFS 实例后，均需先将 BOS 存储桶或其特定目录导入为一个“数据源”，在百舸上实际是对该“数据源”进行挂载访问。此后可根据业务场景特点，进行数据流转的进阶配置，以实现更优的加速效果。详细说明请参考 RapidFS官方文档。

- 配置元数据同步：如果增量数据会通过 RapidFS 以外的方式直接上传或修改至 BOS，需要配置元数据同步规则，以将 BOS 中的增量变化同步至 RapidFS 使其可被访问。
- 配置缓存预热：为避免冷启动带来的首次访问性能损失，您可以创建“缓存预热规则”，将数据源中指定目录（及其后代目录）下的热点数据，提前预热加载至缓存实例。
- 配置更多自定义策略：除缓存预热外，还可对缓存淘汰、目录级策略等进行更多自定义配置，以适应不同的业务需求。

权限说明

操作	所需权限
开启/配置缓存加速（全局配置）	资源池**管理员**及以上
创建 RapidFS 实例	账号需授予 `RAPIDFSFullControlAccessPolicy` 策略
创建 RapidFS 数据集（权限隔离）	账号需授予 `AssetFullControlPolicy` 策略
在任务中挂载 RapidFS 数据集	数据集已被管理员授权给当前用户

计费说明

费用项	说明
百舸侧	不额外收费，本地磁盘硬件费用已包含在计算节点购买费用中
RapidFS 侧	按**缓存加速容量**计费，**按量付费。**
账单查看	费用计入 **RapidFS 产品账单**，请前往 RapidFS 控制台查看

> 详细计费说明请参考 RapidFS 官方计费文档。

常见问题

Q：为什么开启按钮是灰色的/提示无法开启？

A：可能原因是资源池内所有节点均只有一块本地磁盘或者没有磁盘，没有可用于缓存的额外本地磁盘。

Q：缓存加速总容量与 RapidFS 控制台显示的容量不一致？

A：百舸侧展示的是**预估容量**（基于 节点套餐规格计算，不含折损率），实际可用容量以 RapidFS 控制台"已用量/总容量"为准，两者存在一定差异属正常现象。

Q：选择 M1 规格后，节点数超过 50 怎么办？

A：M1 规格实例最多托管 50 个节点，超出后平台将停止自动扩容。建议在资源池节点数可能超过 50 时，提前在 RapidFS 控制台选择 M2 规格实例。当前版本暂不支持实例解绑和规格变更，该能力将在后续版本支持。

Q：新购或释放节点后，缓存容量会自动更新吗？

A：会的。节点新购、释放、故障等情况下，平台会自动更新 RapidFS 实例的总容量、部署组容量及 Cache 节点列表，无需手动操作。

Q：能否同时关联多个 RapidFS 实例？

A：当前仅支持资源池与 RapidFS 实例 **1:1** 关联。

相关文档

- RapidFS 产品介绍
- RapidFS Master M1/M2 规格说明
- RapidFS 计费说明
- 对象存储 BOS 产品介绍

资源队列CPU节点弹性伸缩

概述

百舸平台支持全托管资源池中的资源队列针对CPU节点进行自动弹性伸缩。开启该功能后，平台将根据队列中的业务负载，自动触发节点的扩容与缩容，帮助用户在保障业务连续性的前提下，实现资源的按需使用与自动释放，最大化资源利用率并降低运维成本。

典型使用场景：

- 任务量突增时，平台自动扩容节点，避免任务长时间排队等待
- 业务低峰期（如夜间），平台自动缩容空闲节点，减少不必要的资源费用
- 无需专人监控负载、手动调整节点数量

使用前提

在使用弹性伸缩功能前，请确认以下条件：

1. **账号权限**：仅平台管理员（拥有 AIHCFullControlPolicy 权限）可开启或关闭弹性伸缩及自动缩容配置。
2. **队列类型**：弹性伸缩仅支持**纯 CPU 节点**，暂不支持GPU节点及CPU-GPU混合队列。
3. **创建时机**：弹性伸缩只能在**创建队列时**开启，已有队列无法事后开启。
4. **机型配置**：当前版本仅支持单一主机型配置，不支持备选机型池。
5. **计费方式**：弹性扩容节点仅支持**按量付费**，不支持预付费。
6. **存储与场景**：暂不支持PFS存储挂载；开发机场景不支持使用弹性伸缩队列。
7. **操作互斥**：已开启弹性伸缩的队列，**禁止**手动添加、移出或转让节点，所有节点变更必须通过弹性伸缩策略执行。

权限要求

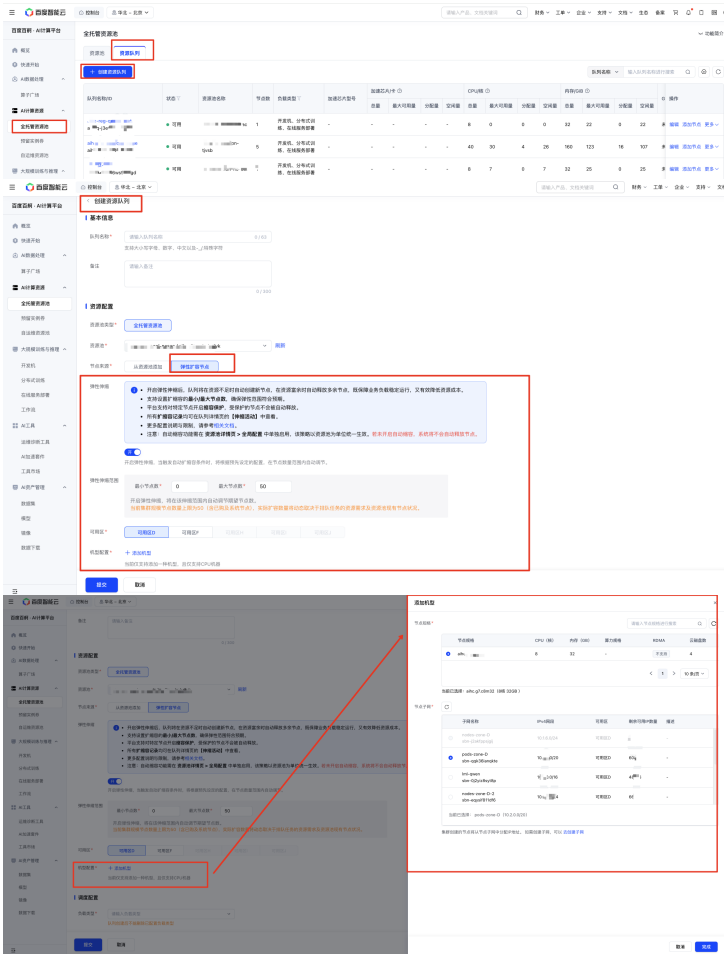
操作类型	所需权限	说明
开启/关闭弹性伸缩	AIHCFullControlPolicy	仅平台管理员可执行
配置自动缩容全局开关	AIHCFullControlPolicy	仅平台管理员可执行
查看伸缩配置与活动日志	普通用户及以上	所有队列成员均可查看
提交负载/任务	普通用户及以上	需遵循弹性队列的任务提交规范

操作步骤

创建支持弹性伸缩的队列

> 弹性伸缩必须在创建队列时开启，创建后无法补充开启。

1. 进入**全托管资源池/资源队列**，点击**创建队列**。
2. 在【节点来源】中选择**弹性扩容节点**。
3. 打开**弹性伸缩**开关。
4. 配置节点数量范围：
   - **最小节点数**：队列保留的最少节点数，可设为 0（即完全空载时释放所有节点）。
   - **最大节点数**：队列可扩容的上限，受资源池规模约束（例如 L50 资源池最大支持 50 个节点）。
5. 添加**机型配置**：选择 CPU 机型（当前仅支持单选）及对应子网。
6. 计费方式默认为**按量付费**，确认后提交创建。



修改弹性伸缩配置

> 适用于已开启弹性伸缩的队列，可调整节点数量范围或关闭功能。

1. 在**队列列表**或**队列详情页**，点击【配置弹性伸缩】。
2. 可修改：**最小节点数** 和 **最大节点数**。
3. 如需关闭弹性伸缩，在配置页关闭开关即可。

注意：

- 关闭弹性伸缩后**不可重新开启**，现有节点将保留。
- 队列处于【节点伸缩中】状态时，配置操作暂时不可用，请等待伸缩完成后再操作。





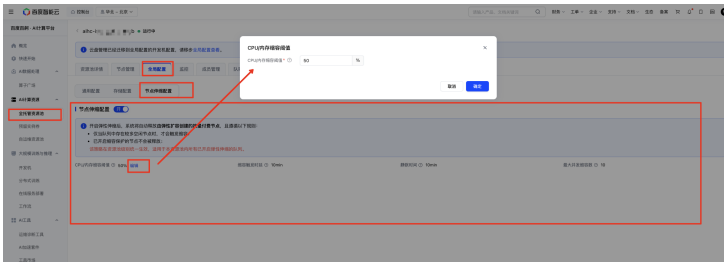
配置全局自动缩容策略

> 自动缩容为资源池级别的全局配置，对资源池内所有已开启弹性伸缩的队列统一生效。

1. 进入**资源池详情页** > **全局配置** > **节点伸缩配置**。
2. 打开**自动缩容**开关（默认关闭）。
3. 配置 **CPU/内存缩容阈值**（默认 50%，范围 0%–100%）：当节点的 CPU 或内存资源请求率持续低于该阈值时，触发自动缩容。
4. 保存配置。未开启此开关时，系统仅支持自动扩容，不会自动释放节点。

以下参数为系统默认值，仅供参考，不支持修改：

参数	默认值	说明
缩容触发时延	10 分钟	资源使用率低于阈值并持续该时长后，才触发缩容
静默时间	10 分钟	扩容完成后，至少等待该时长才会再次执行缩容
最大并发缩容数	10	单次缩容操作最多同时移出的节点数量



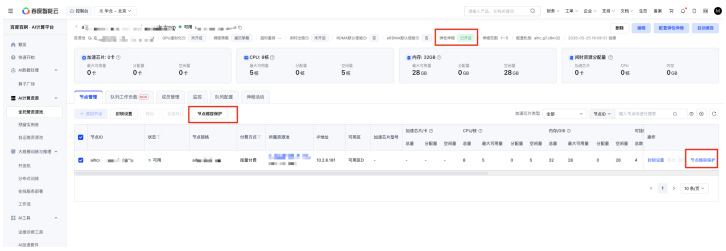
设置节点缩容保护

> 对特定节点开启缩容保护后，该节点不会被自动缩容移出，适用于承载重要任务的节点。

1. 进入**队列详情页** > **节点管理**列表。
2. 勾选需要保护的节点（支持批量选择）。
3. 点击【节点缩容保护】，开启保护。

说明：

- 仅状态为【可用】或【可用（已封锁）】的节点支持该操作。



查看伸缩活动记录

队列级别：

1. 进入**队列详情页**，点击【伸缩活动】标签页。
2. 点击【查看状态】，可查看本次伸缩涉及的节点列表（节点 IP、节点 ID、伸缩状态）。

资源池级别：

在**资源池详情页**可查看资源池范围内的所有伸缩活动，记录中额外包含队列名称和队列 ID，便于跨队列汇总查看。

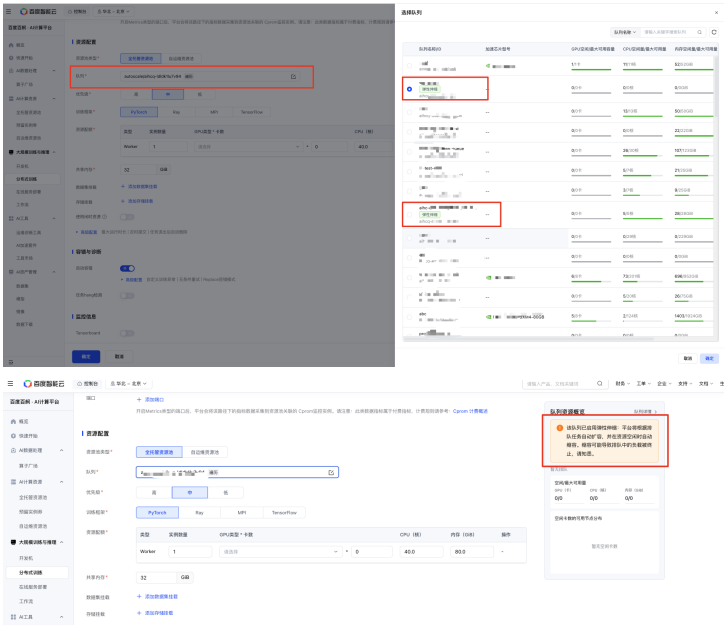


提交任务至弹性伸缩队列

1. 提交训练或推理任务时，选择队列。
2. 弹性伸缩队列会标注【**弹性伸缩**】标识，便于识别。
3. 选中弹性伸缩队列后，系统将弹出提示：

> 该队列已启用弹性伸缩，平台将根据排队任务自动扩容，并在资源空闲时自动缩容。缩容可能导致排队中的负载被终止，请知悉。 **注意：**

1. 开发机不可使用弹性伸缩队列。
2. 系统将根据当前队列负载自动判断是否需要扩容：
   - 若存在排队任务且达到扩容阈值，系统将自动创建新节点。
   - 新节点就绪后，排队任务将自动调度执行。



计费说明

- 弹性扩容的节点采用**按量付费**方式，按小时出账。
- 节点缩容后即停止计费，建议合理配置缩容阈值以控制成本。
- 具体费用可在**费用中心 > 账单**中查看。

注意事项

1. **弹性伸缩仅支持 CPU 节点**，GPU 节点及混合队列不适用。
2. **弹性伸缩只能在创建队列时开启**，已有队列无法补开，请在创建时规划好。
3. **关闭弹性伸缩后不可恢复**，操作前请确认业务需求。
4. **自动缩容策略以资源池为单位生效**，修改配置将影响该资源池内所有弹性伸缩队列。
5. **运行中的任务不会被缩容中断**，仅排队中（尚未开始运行）的任务可能因缩容受到影响。
6. **队列伸缩中状态期间**，配置修改操作不可用，请等待伸缩完成。
7. 弹性伸缩队列**不支持 PFS 存储挂载**，如业务依赖 PFS，请使用普通队列。

常见问题

1. **什么情况下触发CPU节点的扩容？**

- 队列中存在因为资源（cpu、内存）不足而处于pending的pod，且队列的节点未达到队列配置的最大节点数

2. **为什么刚扩容出来的机器满足条件不会被缩容？**

- 刚扩容出的机器会有10分钟的保护时间，过了十分钟后，才会考虑缩容。

3. **为什么修改配置后没有立刻生效？**

- 修改配置（伸缩配置、缩容配置）后伸缩组件会重启，伸缩组内已经存在的节点会被标识成刚扩容出来的节点，重新等待10分钟后，节点才会匹配缩容条件。

4. **为什么符合缩容阈值与缩容时间机器没有被缩容？**

- 首先判断队列内的节点数是否达到设定的最小节点数
- 该机器上的pod是否可以调度到其他节点，如果不能，该机器也不会被缩容
- 是否设置了该节点为不可缩容，如是，则该节点不会被缩容
- 这个节点是否是刚扩容出来的节点，刚扩容出来的节点10分钟内不会被判定缩容

5. **为什么无法在已有队列上开启弹性伸缩？**

- 弹性伸缩只能在创建队列时开启，已创建的队列不支持后续开启。如需使用弹性伸缩，请新建队列并在创建时开启该功能。

6. **关闭弹性伸缩后，现有节点会被删除吗？**

- 不会。关闭弹性伸缩后，队列现有节点将继续保留，不会自动移出。但关闭后无法重新开启弹性伸缩功能。

7. **自动缩容会影响正在运行的任务吗？**

- 不会。自动缩容只会释放空闲节点，运行中的任务不会被中断。仅处于排队状态（尚未分配到节点运行）的任务可能因节点缩容而受到影响。

8. **弹性伸缩队列的费用如何计算？**

- 弹性扩容的节点按实际使用时长计费（按量付费，按小时出账）。节点缩容后即停止计费，具体费用可在费用中心的账单中查看。

9. **为什么【配置弹性伸缩】按钮是灰色不可点击的？**

- 可能有两种原因：① 当前账号不是平台管理员，无操作权限；② 队列处于【节点伸缩中】状态，请等待当前伸缩活动完成后再操作。

10. **队列开启弹性伸缩后，还能手动移出节点吗？**

- 不可以。 开启了弹性伸缩的队列不支持手动移出节点。

11. **最大节点数可以设置为多少？**

- 最大节点数受所在资源池规模上限约束。例如，L50 资源池最多支持 50 个节点。具体上限请以创建队列时页面提示为准。

12. **为什么队列无法手动添加节点？**

- 已启用弹性伸缩的队列处于自动化管理模式，所有节点变更必须通过调整弹性伸缩策略（修改最小/最大节点数）实现，手动操作入口已被禁用。

13. **如何验证弹性伸缩是否生效？**

- 可通过【伸缩活动】Tab查看最近一次扩容/缩容记录，或在队列监控面板观察节点数量随负载变化的趋势曲线。

如有其他问题，请联系平台管理员或提交工单获取支持。

开发机系统盘持久化配置

概述

百舸平台为开发机提供系统盘持久化能力，确保您在开发机中安装的软件、配置的环境变量和代码目录在重启后不会丢失。

平台支持三种系统盘方案：

方案	持久化能力	使用前提	备注
百舸云盘（推荐）	✅ 完整持久化	购买百舸云盘	默认方案
CFS 文件存储（Beta）	✅ 完整持久化	创建CFS实例并关联到相应的资源池	需申请白名单
临时本地盘	❌ 无持久化		停机后数据丢失

> **说明**：持久化方案变更后，仅对**新创建的开发机**生效，存量开发机不受影响。

管理员配置系统盘持久化方案

资源池管理员在资源池的「全局配置 > 开发机配置」中设置系统盘持久化方案。

开启 / 关闭持久化

- **开启**（默认）：新创建的开发机使用所选方案（百舸云盘或 CFS）进行持久化。
  - 百舸云盘与 CFS 可随时切换，切换后**仅对新创建的开发机生效**，存量开发机不受影响。
- **关闭**：新创建的开发机使用**临时本地盘**，开发机停止或变更配置后环境数据丢失。关闭前系统会弹出确认提示，请仔细阅读。

重新开启后，可自动回显上次的持久化配置。

百舸云盘（推荐）

1. 进入资源池「全局配置 > 开发机配置」。
2. 在「系统盘持久化配置」中，确认**百舸云盘**已设为默认。
3. 如需购买或扩容，点击**扩缩容**，按提示完成云盘购买流程。

- 注意事项：百舸云盘最小容量为 **20 GiB**，可用容量低于 20 GiB 时，新开发机将无法申请系统盘。



CFS 文件存储（Beta）

> - CFS 文件存储持久化方案当前处于 **Beta 测试阶段**，需联系百舸技术人员申请开通白名单后方可启用。

- **配置步骤**
  - 进入「全局配置 > 开发机配置 > CFS 持久化配置」，点击**【设为默认】**，初次配置需要关联CFS实例。
  - 在弹窗中选择已有 CFS 实例，或点击「**+新建文件系统**」创建新实例。

- 创建时只需填写文件系统名称，点击**【确定】**完成关联。
- 设置后新创建的开发机将默认使用 CFS 作为系统盘，存量开发机不受影响。

- **注意事项**
  - 每个资源池只能关联 **1 个** CFS 实例；同一 CFS 实例可关联多个资源池。
  - 该CFS 实例**仅用于**开发机系统盘持久化，不支持挂载到训练任务、推理服务等其他工作负载。







临时本地盘 当两种持久化方案均不适用时，可使用**临时本地盘**作为系统盘临时存储。

- 与同节点其他开发机**共享**存储资源。
- 开发机**停止或变更配置**后，环境数据将丢失。
- 不支持自定义申请容量。

> 使用临时本地盘时，请在停止开发机前及时**保存镜像**，以保留环境配置。

创建开发机使用系统盘持久化

创建开发机时，系统根据资源池持久化配置自动确定系统盘类型：

资源池持久化配置	创建开发机时系统盘类型	说明
百舸云盘（默认）	百舸云盘	可申请容量，生命周期内持续计费
CFS 文件存储	CFS 文件存储	创建时检查 CFS 可用状态
未配置持久化	临时本地盘	关机/变配后数据丢失，不支持申请容量

- 注意存量开发机的处理
  - 资源池持久化方案变更后，**存量开发机的系统盘类型不变**，可继续正常使用。
  - 存量使用临时本地盘的开发机，变更配置后系统盘类型仍为临时本地盘。

常见问题

Q：切换持久化方案后，之前的数据会丢失吗？

不会。方案切换只影响新建的开发机，存量开发机的数据和系统盘类型保持不变。

Q：CFS 方案如何申请开通？

CFS 文件系统方案处于 Beta 测试阶段，请联系百舸技术人员，由平台评估后为您开通白名单。

Q：一个 CFS 实例可以关联多个资源池吗？

可以。但每个资源池只能关联一个 CFS 实例。

Q：开发机删除后，CFS 上的数据会同步删除吗？

是的。删除开发机后，该开发机在 CFS 上占用的持久化数据将同步清除，存储空间随之释放。

Q：百舸云盘容量不足怎么办？

进入资源池「全局配置 > 开发机配置 > 云盘管理」，点击扩容并完成购买流程。注意扩容后不支持缩容。

Q：资源池新关联的CFS实例为什么已分配容量不为0？

该CFS实例已经被其他资源池关联过，且有开发机使用该实例进行了系统盘存储。

队列子用户资源配额管理

概述

在全托管资源池中，队列（Queue）级别的资源隔离已经能够保证不同业务团队之间的资源边界，但在同一队列内部，不同子用户之间仍可能出现"一人占满全队列"的资源争夺问题。

子用户资源配额管理功能允许队列管理员在队列维度下，为指定的**子用户**或**用户组**设置资源使用上限（加速芯片卡数、CPU 核数、内存容量），从而实现队列内部的精细化资源治理：

- 防止单个用户突发任务占满整个队列资源；
- 保障队列内多业务、多用户的公平共享；
- 在保留队列总体弹性的同时，控制每位成员的资源天花板。

核心概念

在使用本功能前，建议先了解以下名词：

名词	含义
队列加速芯片总量	队列中所有节点的加速芯片总数
队列最大可用量	队列中所有可用节点（排除故障节点、封锁节点）的加速芯片总数
队列分配量	队列中所有负载（开发机、分布式训练、在线服务）已分配的资源总和
队列空闲量	队列中所有可用节点（排除故障、封锁节点）的空闲芯片总数
子用户资源配额	在特定队列中，分配给该子用户的资源使用**硬性上限**，包含加速芯片、CPU 核数、内存容量。达到上限后，无法调度新任务
子用户资源已用量	该子用户在当前队列中**正在运行**的所有任务（开发机、训练、在线服务等）实际占用的资源总和，随任务启停实时变化
子用户剩余配额	子用户剩余配额 = 子用户资源配额 − 子用户资源已用量；为 0 时表示该用户在该队列的额度已耗尽，此时哪怕队列有剩余资源，该用户的任务也无法调度。

适用范围与权限说明

- 支持的工作负载范围：
  - 子用户配额对该队列中提交的开发机、分布式训练、在线服务部署（推理）的工作负载**总和**生效，不区分负载类型
- 前置条件
  - 队列中**必须存在加速卡资源**。若队列没有 GPU/XPU 资源，开启功能时会提示：**该队列没有加速卡资源，不支持开启该配置。**
  - 调度器组件需达到所需版本。版本不满足时会提示：**当前调度器组件版本暂不支持此功能，请联系百舸技术人员升级后再试。**
- 权限说明

角色	权限
队列管理员及以上	可配置、编辑、删除子用户资源配额
队列开发人员	仅可查看

配置子用户资源配额

- 配置入口
  - 队列详情 → **队列配置** → **资源使用额度配置**。
  - 该开关**默认关闭**。关闭状态下，队列内所有用户默认共享队列的全部资源。
- 开启功能，新增配额规则，点击 **+ 新增配置**，依次填写以下字段：

字段	必填	说明
生效范围类型	是	单选：**用户 / 用户组**（默认：用户）。
生效范围	是	选择"用户"时，从本队列已添加的用户中多选；选择"用户组"时，从本队列已添加的用户组中多选
最高可用加速芯片卡数	是	正整数，**不能超过队列总卡数**
最高可用 CPU 核数	否	正整数，**不能超过队列总 CPU 核数**。不填则不限制
最高可用内存数	否	正整数，**不能超过队列总内存数**。不填则不限制



- 支持编辑规则
  - 在规则列表中点击**编辑**按钮，对应的三个字段进入编辑态，修改后保存。保存时校验"是否完全重复"以及"最高可使用加速芯片卡数"是否合法。
- 支持删除规则
- 支持关闭

配额生效逻辑

- 用户组配额按"**每个成员独立生效**"
  - 为**用户组**设置的配额，表示组内**每个用户**均按该规格独立生效，**而非整组共用**。例如：为某用户组配置"最高 100 卡"，则组内每位用户的上限均为 100 卡，而不是全组合计 100 卡。
- 同一用户存在多条配额时：取"最小交集"，以 GPU 为准
  - 由于一个用户可能同时被列入"用户配置"和某个"用户组配置"，其最终生效配额按以下规则确定：**以 GPU 配额最小的那条规则为准，整条规则（含 GPU/CPU/内存）整体生效。**

> **示例 1：**
>
> - 用户 A 单独配置：30 卡
> - 用户 A 所在用户组配置：100 卡
> - → 用户 A 实际上限：**30 卡**
>
> **示例 2：**
>
> - 用户 A 单独配置：30 卡 / 60 核 / 100 GB
> - 用户 A 所在用户组配置：40 卡 / 50 核 / 60 GB
> - → 比较 GPU，30 卡更小，整条规则生效：**30 卡 / 60 核 / 100 GB**（**不是** 30 卡 / 50 核 / 60 GB）

- 配额校验范围：配额限制的是该用户在本队列中**所有非终止态任务**（启动中 + 运行中）的资源占用总和：
  - **不是**"单任务不超过配额"，而是"该用户全部运行任务的总和不超过配额"；
  - 设置规则前已经在跑的任务**不会被强制杀掉**；
  - 新任务提交时若 已用 + 本次申请 > 配额，则任务**进入排队**（即使队列还有更多空闲资源，也不会被调度）；
  - 已用 + 本次申请 ≤ 配额 → **正常调度**。
- 双层封顶示例：假设一个队列：GPU 100 卡，10 个用户每人配置 20 卡：
  - 所有用户配额合计 200 卡，**远超队列物理总量**，但通过双层约束治理：
    - **单用户封顶**：任何人最多 20 卡，不会被单人占满；
    - **队列全局封顶**：所有人合计不超过 100 卡；
  - 多人同时满配时，超出队列余量的任务进入排队。
- 队列资源变动的影响：队列资源视图本身不受配额影响。需注意的边界情况：
  - 队列初始 100 卡，某子用户配额为 90 卡；
  - 当队列因故障、封锁、节点移出/转让导致最大可用量下降，甚至低于 90 卡时，**子用户配额可能大于队列最大可用量**；
  - 此时该用户的实际可调度资源以**队列空闲量**为准。

提交工作负载时的诊断与拦截

- 事前诊断 · 资源概览：在创建负载页面，如果当前用户在该队列**已配置个人配额**，会展示个人资源概览：
  - **个人空闲量 / 个人可用资源配额**
  - 若管理员只配置了 GPU，未配置 CPU、内存，则个人 CPU、内存配额位置展示为 -
- 事前诊断 · 校验项：当事前诊断命中下述拦截规则时，用户在点击提交时会再次弹出**拦截确认框**，提示用户调整配置或确认排队后提交。
  - 单实例规格 vs. 节点最大可用量

> **单实例的资源申请量超过节点规格的最大可用量，**无法调度**，请修改配置** GPU 最大可用量：1 卡 当前申请：2 卡 超出资源量：1 卡 CPU 最大可用量：12 核 当前申请：20 核 超出资源量：8 核 内存最大可用量：106 GB 当前申请：200 GB 超出资源量：94 GB

* 申请总量 vs. 子用户配额上限

> **该子用户在本队列配置的可用资源上限超过资源申请，**无法调度**，请修改配置** 个人 GPU 配额：1 卡 当前申请：2 卡 超出资源量：1 卡 个人 CPU 配额：12 核 当前申请：20 核 超出资源量：8 核 个人内存配额：106 GB 当前申请：200 GB 超出资源量：94 GB

* 申请总量 vs. 队列最大可用量

> **资源申请量超过当前队列资源的最大可用量，**无法调度**，请修改配置** （展示 GPU/CPU/内存 各维度的最大可用量、当前申请、超出资源量）

- 事前诊断 · 空闲资源判断：不满足时，在创建页面提示需要排队：
  - 个人空闲配额是否满足（**新增**）

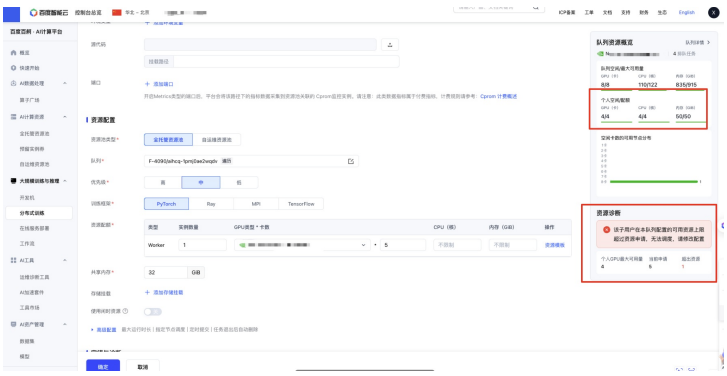
> **当前用户在本队列的可用空闲资源无法满足资源申请，请检查配置或者等待排队** 个人 GPU 空闲：1 卡 当前申请：2 卡 超出资源量：1 卡 个人 CPU 空闲：12 核 当前申请：20 核 超出资源量：8 核 个人内存空闲：106 GB 当前申请：200 GB 超出资源量：94 GB

- 队列空闲资源总量

> **当前队列中空闲资源无法满足资源申请，请检查配置或者等待排队** （展示满足规格的节点数、可调度实例数等）

- 队列空闲资源碎片分布

> 在节点维度比对 GPU/CPU/内存可用量、CDS 可挂载数、PFS L2 服务挂载情况，给出可调度实例数与缺口。



典型场景与最佳实践

- 场景一：保障多团队公平使用 100 卡队列
  - 队列：GPU 100 卡 / CPU 800 核 / 内存 6144 GB；
  - 10 名工程师，每人配额 GPU 20 卡；
  - 效果：单人最多 20 卡，整队列累计不超过 100 卡，超出部分自动排队，避免单人独占。
- 场景二：用户组批量授权 + 个别用户特殊额度
  - 为用户组"算法组"统一配置 50 卡；
  - 为用户组中的负责人 A 单独配置 80 卡；
  - 由于"取最小交集，以 GPU 为准"，A 实际生效配额为**50 卡**（用户组配置更小）；
  - 若需要 A 享受更高额度，应将用户组配置调高，或将 A 移出用户组。
- 场景三：仅约束 GPU，不约束 CPU/内存
  - 配额配置：GPU 30 卡，CPU 与内存留空；
  - 用户在该队列的 CPU/内存上限由**资源规格模板**约束，不再单独限制；
  - 在事前诊断的"个人资源概览"中，个人 CPU/内存配额展示为 -。
- 场景四：队列资源缩水后的配额行为
  - 初始队列 100 卡，用户 A 配额 90 卡；
  - 因部分节点故障，队列最大可用量降至 70 卡；
  - 用户 A 的配额仍为 90 卡，但实际可调度量受队列空闲量限制；
  - 建议管理员定期复核配额与队列实际容量的匹配关系。

常见问题（FAQ）

Q1：开启功能后，队列里之前已经在跑的任务会被杀掉吗？ 不会。配额仅作用于**新任务的调度**，已在运行的任务不会被强制中断。

Q2：用户组配额是组内共享，还是每人独立？ 每人独立。组内每位用户都按该规格各自享有上限。

Q3：用户既被单独配置，又属于某个用户组，最终听谁的？ 取**最小交集，以 GPU 为准**：选 GPU 配额最小的那条规则**整体**生效，而不是逐字段取最小。

Q4：配额加起来超过了队列总量，可以保存吗？ 可以。系统支持**超配**（合计超过队列总量），通过"单人封顶 + 队列全局封顶"的双层约束治理资源争抢。

Q5：关闭"资源使用额度配置"开关后，规则会消失吗？ 不会消失，仅是停用。再次开启时已有规则会重新生效。

Q6：达到配额后再提交任务会发生什么？ 任务进入**排队**，即便队列还有更多空闲资源也不会被调度，直到该用户已用资源回落、低于配额为止。

Q7：开发机被多人共享，校验谁的配额？ 开发机的**所有者**。即便其他成员有公开管理权进行更配，校验仍以所有者的个人配额为准。建议仅由所有者执行更配操作。

Q8：开启功能时提示"调度器组件版本暂不支持此功能"怎么办？ 联系百舸技术支持升级调度器组件后重试。

查看资源池、队列、节点操作云审计记录

概述

云审计（BCT）会记录用户对百舸资源池、队列、节点等资源的关键操作行为（如创建、更新、删除、关联监控实例等），支持对操作记录进行合规检查、操作审核和事件回溯。本文介绍如何在云审计控制台查看资源池或节点相关的操作记录。

前提条件

- 已在 IAM 中单独获取云审计（BCT）的事件查看权限（BCTFullControlPolicy 或 BCTReadPolicy），该权限与百舸（AIHC）权限策略互相独立，需要在 IAM 控制台单独申请。
- 待查询的操作发生在最近 180 天内。

核心说明

限制项	说明
记录范围	仅记录已接入云审计的服务的创建、修改、删除类操作
保留时长	支持查询最近 180 天内的操作记录
记录方式	仅通过控制台或已接入服务的 API 调用产生记录，不涉及资源运行状态数据

- 仅支持查询已接入云审计的服务和事件，未接入的操作不会被记录。
- 云审计不记录资源的运行状态或监控数据，仅记录操作行为本身。

资源池/队列/节点支持审计的操作类型 在云审计控制台的事件名称筛选框中，可直接填写下表中的事件名称进行精确查询。

资源类型	支持审计的操作	事件名称
资源池	创建资源池	CreateResourcePool
	创建活动资源池	CreateCampaignResourcePool
	资源池列表	ListResourcePool
	资源池详情	GetResourcePool
	更新资源池	UpdateResourcePool
	删除资源池	DeleteResourcePool
	安装 Cprom	BindCpromConsoleAPIV2
	卸载 Cprom	UnbindCpromConsoleAPIV2
	获取资源池创建步骤	GetResourcePoolCreateSteps
	创建资源池预检	CreateResourcePoolPrecheck
	查询资源池预检结果	GetResourcePoolPrecheck
	查询资源池存储信息	GetResourcePoolStorageInfo

	查询资源池监控信息	GetResourcePoolMonitorInfo
队列	创建资源队列	CreateResourceQueueConsoleAPIV2
	资源队列列表	ListResourceQueue
	资源队列详情	GetResourceQueue
	查询队列节点列表	GetResourceQueueNodes
	更新资源队列	UpdateResourceQueueConsoleAPIV2
	删除资源队列	DeleteResourceQueueConsoleAPIV2
	节点转移	TransferNodesConsoleAPIV2
	校验负载类型支持	CheckWorkloadTypeSupport
节点	订购节点（购买/释放）	OrderNodesConsoleAPIV2
	节点转预付费	ChangeNodesToPrepayConsoleAPIV2
	删除节点	DeleteNodeConsoleAPIV2
	批量删除节点	BatchDeleteNodesConsoleAPIV2
	节点列表	GetNodeList
	节点详情	GetNode
	封锁/解封节点	CordonNodeConsoleAPIV2
	批量封锁节点	BatchCordonNodesConsoleAPIV2 BatchUnCordonNodesConsoleAPIV2
	节点 MIG 设置	MigNodesConsoleAPIV2

		DisableNodeMigConsoleAPIV2
预留实例券	开启/关闭自动续费	OperationAutoRenewSpec

- 上表仅覆盖当前已接入云审计的操作类型，未列出的操作暂不支持审计查询。

操作步骤

导航路径：百度智能云管理控制台 -> 云审计 BCT -> 操作记录

步骤 1：登录百度智能云管理控制台，选择"云审计 BCT"

步骤 2：进入"操作记录"页面，按需设置筛选条件，在请选择云服务处选择【aihc】

参数	必填	说明
事件名称	否	按具体操作事件名称筛选，如资源池创建、节点封锁、关联监控实例
资源名称/ID	否	按具体资源池、队列或节点的名称或 ID 筛选
时间范围	否	选择查询的起止时间，最长可查询 180 天内记录

步骤 3：在事件列表中查看匹配的操作记录

步骤 4：单击【查看事件】，展开查看事件详情，包括访问密钥、区域、错误码、事件名称、事件源、事件时间、请求 ID、源 IP 地址、操作用户名等信息。查询成功后，您可以在事件列表中确认对应资源池或节点的历史操作是否符合预期。

变更影响说明

- 不影响：查看操作记录为只读操作，不会对资源池、队列、节点的运行状态产生任何影响。
- 不支持：未接入云审计的操作类型（如部分只读查询类接口）不会出现在操作记录中。

资源池产品事件

概述

在资源池详情页的【产品事件】Tab 中，您可以查看当前资源池产生的产品事件。您可以按时间范围查看事件，按产品事件名称、节点 ID 和事件 ID搜索事件，并通过事件级别筛选、查看事件详情和刷新事件列表。

本功能当前适用于全托管资源池。

前提条件

- 当前用户已登录百度百舸·AI 计算平台控制台。
- 当前用户具备目标全托管资源池的查看权限。
- 如需添加报警策略，当前用户的账号需已配置 BCM 的报警策略相关权限（`BCMFullControlAccessPolicy`）。您可参照 IAM 为子用户授权 中的步骤完成该权限的配置。

核心说明

功能入口

导航路径：百度百舸·AI 计算平台控制台 -> 全托管资源池 -> 目标资源池 -> 【产品事件】

进入资源池详情页后，在页面顶部单击【产品事件】Tab，即可查看该资源池的产品事件。

事件列表

产品事件列表包含以下字段：

字段	说明
事件 ID	事件的唯一标识，支持精准搜索
产品事件	产品事件名称，支持模糊搜索
事件级别	事件严重程度，支持筛选
事件详情	事件的具体信息
节点 ID	产生事件的节点标识，支持精准搜索
触发时间	事件产生的时间
发起人	触发或发起该操作的用户

事件级别

事件级别与 BCM 云监控的定义保持一致，详细说明以 BCM 云监控-事件监控 为准。

已支持的产品事件

当前已接入的产品事件如下表所示（适用于全托管资源池），事件的详细信息可在百度百舸·AI 计算平台控制台的资源池详情页【产品事件】Tab 中查看，也可在 BCM 云监控-事件监控 中查看。

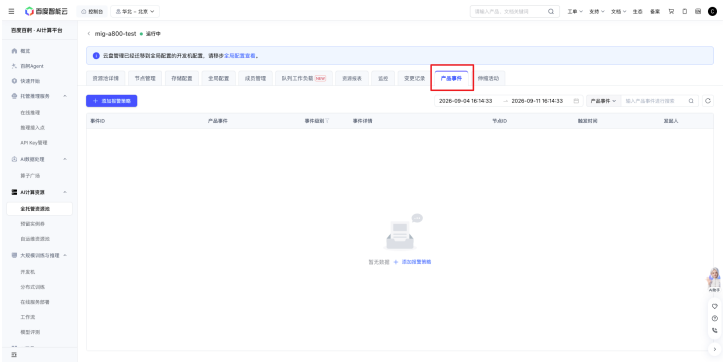
事件中文名称	事件英文名称（事件唯一标识）	事件级别	说明
节点状态由可用变更为自愈中	MRPInstanceAutomaticSelfHealing	预警	节点发生故障并进入自愈流程，请关注后续事件通知
节点自愈待开始离线修复	MRPInstanceOfflineRepairPending	警告	节点需进行离线修复，请检查是否有需要迁移的负载，如有请先完成迁移
节点自愈开始离线修复	MRPInstanceOfflineRepairStarted	通知	节点已开始离线修复，请等待修复完成，恢复可用时将另行通知
节点状态由自愈中变更为可用	MRPInstanceRecovered	通知	节点完成自愈并恢复为可用状态
节点封锁操作	MRPInstanceCordoned	预警	节点被封锁，不再调度新的工作负载
节点解封锁操作	MRPInstanceUncordoned	通知	节点解除封锁，恢复正常调度
节点容量水位超阈值	MRPInstanceCapacityThresholdExceeded	预警	节点系统盘等容量水位超过阈值，需关注受影响的负载任务

操作步骤

步骤 1：打开产品事件页面

1. 登录百度百舸·AI 计算平台控制台。
2. 在左侧导航栏选择【全托管资源池】。
3. 在资源池列表中单击目标资源池。
4. 在资源池详情页顶部单击【产品事件】Tab。

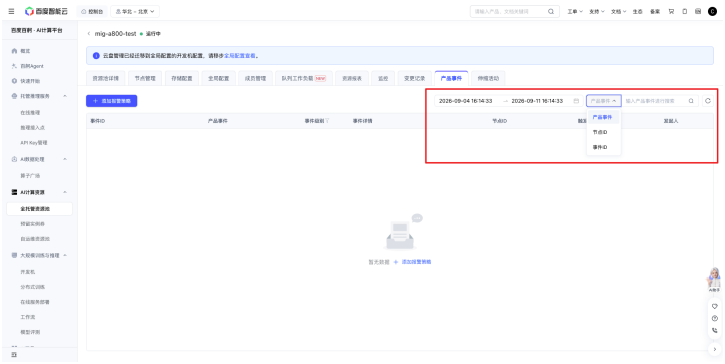
进入【产品事件】页面后，确认页面已显示目标资源池的事件列表。



步骤 2：筛选或搜索产品事件

1. 设置时间范围，系统默认展示近一周的产品事件。
2. 按事件级别筛选产品事件。
3. 在搜索框中选择搜索类型，输入产品事件名称进行模糊搜索，或输入节点 ID、事件 ID进行精准搜索。
4. 单击刷新按钮，获取最新事件。

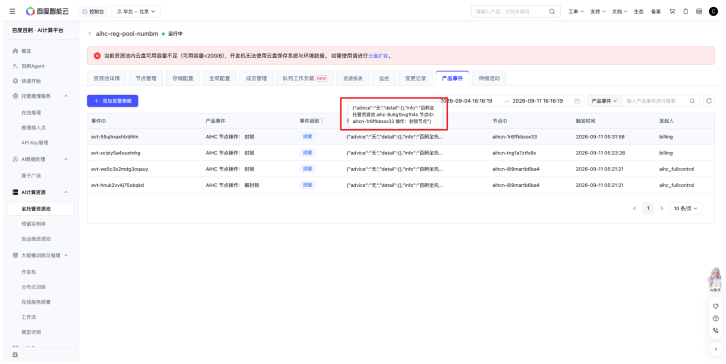
列表展示符合条件的结果后，可以通过分页查看产品事件。



步骤 3：查看事件详情

1. 在产品事件列表中找到目标事件。
2. 单击目标事件对应的事件详情。
3. 查看事件的具体信息。

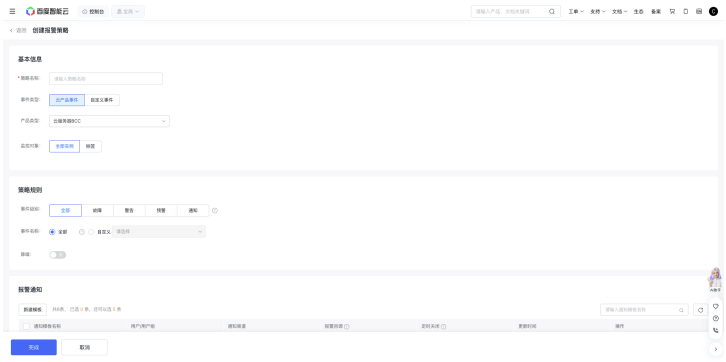
以页面展示的事件详情为准。



步骤 4：添加报警策略

1. 在【产品事件】页面单击【添加报警策略】。
2. 在新打开的 BCM 页面中配置报警策略。
3. 按照 BCM 页面提示完成提交。

如需添加报警策略，请先参照 IAM 为子用户授权 为当前账号配置 BCM 报警策略相关权限（BCMFullControlAccessPolicy）。



使用须知

- 【产品事件】Tab 仅用于查看资源池的产品事件，并提供筛选、搜索、详情查看、刷新、分页和报警策略跳转功能，不改变资源池、节点或工作负载的运行状态。
- 添加报警策略后，系统跳转至 BCM 页面进行配置，报警策略由 BCM 负责管理。
- 产品事件功能当前适用于全托管资源池，队列的节点列表不支持查看产品事件。

删除资源池

概述

当资源池不再使用时，您可以在百度百舸·AI 计算平台控制台删除资源池。删除资源池前，需要先清理资源池中的工作负载、队列和相关计算资源，避免因存量资源未释放导致删除失败。资源池删除后无法恢复，**执行删除前，请确认资源池已不再承载业务，并已备份需要保留的数据。**

前提条件

- 当前用户已具备AIHCFullControlPolicy权限。
- 已停止向待删除资源池及其队列提交新的开发机、训练任务或在线服务。
- 已备份需要保留的数据、日志、环境和其他业务内容。
- 已完成对应的删除前检查。

核心说明

删除顺序

平台当前采用逐级清理方式删除资源池，不支持直接级联删除资源池中的全部工作负载和队列。全托管资源池的删除顺序为：删除工作负载 -> 删除队列 -> 释放节点 -> 删除资源池

删除前校验

检查项	全托管资源池
工作负载	所有队列中的开发机、训练任务、评测任务、工作流和在线服务等均需删除
队列	资源池中的全部队列均需删除
节点	必须从资源池释放

> 删除过程中请勿向相关队列提交新的工作负载。平台会在删除队列时再次校验存量工作负载；如果出现新提交的任务，需要删除后重试。

操作步骤

步骤 1：停止提交新的工作负载

通知资源池使用者停止向待删除资源池及其队列提交开发机、训练任务、工作流和在线服务，避免清理期间产生新的存量资源。

步骤 2：删除工作负载

进入资源池下各队列的工作负载列表，依次删除队列中的工作负载，包括开发机、训练任务、评测任务、工作流和在线服务等。



删除前，请先保存需要保留的数据和环境。删除工作负载后，工作负载将无法访问，且删除操作无法撤销。

在队列详情的工作负载页确认各类工作负载数量均为 0。

步骤 3：删除队列

进入资源队列列表，按照以下规则删除队列：

1. 确认队列中不存在任何工作负载。

2. 单击目标队列操作栏中的【删除】，核对提示信息并确认删除。

3. 删除期间不要向该队列提交新的工作负载。

如果平台提示队列中存在工作负载，请按照提示信息删除工作负载。队列删除后无法恢复，其计算节点将返回所属资源池。

步骤 4：释放节点

释放资源池中的全部节点，当前只支持释放后付费节点和包年包月的到期的节点。

注意：包年包月节点在到期前不支持直接释放。请先前往退订管理中心找到对应节点并完成退订，**退订完成后会自动销毁资源，无需再进行释放。**

步骤 5：删除资源池

导航路径：百度百舸·AI 计算平台控制台 -> 对应资源池类型 -> 资源池列表

1. 找到目标资源池，单击操作栏中的【删除】。

2. 阅读删除确认信息，核对资源池名称及影响范围。

3. 返回资源池列表，刷新页面并确认目标资源池已不存在。

资源池删除可能需要一定时间，请以控制台显示的状态和结果为准，避免重复提交删除操作。

删除失败处理

提示	处理方法
资源池中存在节点，无法删除	将节点从队列移出，并释放全托管资源池中的全部节点后重试
资源池中存在队列，无法删除	清空队列中的工作负载，并删除全部队列后重试，注意打开隐藏无节点队列查看是否有没有删除的队列。
删除操作不可用	检查队列、工作负载和节点是否已完成清理

自运维资源池

创建自运维资源池

概述

本文将介绍如何创建自运维计算资源池。

自运维计算资源池是百舸内部管理资源的基础单元，一个资源池对应一个CCE集群，一个PFS实例和一个CPROM实例。

在资源池中，用户可以创建多个队列，从而在不同的队列中处理不同业务的工作任务。通常情况下，企业内部一个独立的团队或者项目建议创建一个独立的资源池，实现团队间或项目间的资源隔离。

百舸平台支持用户基于实际使用场景创建具备不同规格GPU套餐的资源池，详细GPU套餐规格请见：百舸平台支持的GPU套餐。

操作步骤

1. 登录百舸AI计算平台AIHC控制台。

2. 在左侧菜单栏选择资源池，进入资源池列表页面，点击**创建自运维计算资源**。



3. 在购买资源池页面，根据实际需求完成相关参数配置。

- 基本信息



配置项	必填/选填	配置说明
资源池名称	必填	输入资源池名称，支持大小写字母、数字、中文以及-_ /.特殊字符，必须以字母开头，长度1-65。
当前地域	必填	选择集群所在的物理区域。
可用区	必填	该可用区中包括您所需的VPC、节点配置/类型、存储实例和监控大盘实例。

- 网络配置



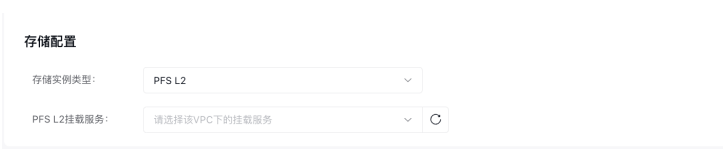
配置项	必填/选填	配置说明
资源池网络	必填	选择集群中节点网络所在的VPC，您可以在当前可用区新建VPC，每个可用区均已创建默认私有网络。
子网	必填	选择集群中节点网络所在的子网，您可以在当前可用区新建子网，每个可用区均已创建默认私有网络。
APIServer公网访问	必填	开启后将为集群API Server绑定公网IP。
容器网络模式	选填	默认支持VPC-ENI网络模式。
容器子网	必填	请您务必在选择网段前，合理有效地规划集群的网络，避免出现网络冲突，否则可能存在风险。CCE集群网络说明及规划
Cluster IP网段	必填	配置当前集群的ClusterIP网段
安全组	必填	用户可使用普通安全组或企业安全组

- 节点配置



配置项	必填/选填	配置说明
节点来源	必填	支持新建节点或直接添加已有服务器资源（已有HPAS节点仅能在创建资源池完成后才能添加）。
付费方式	必填	预付费：先付费后使用服务。您可在账户余额中预先充值，通过账户余额支付购买；也可直接通过银行卡、百度钱包等支付购买。后付费：先使用服务后付费。根据您的使用情况，按分钟实时计费并扣费。因此为保证服务的正常使用，请在使用后付费支付方式的产品前向账户中预先充值。
购买时长	必填	选择购买节点的时间
自动续费	必填	若您已选购预付费服务，为节省您的操作时间，避免因续费失败造成的服务停服给您带来的不便，百度智能云建议您开通自动续费业务。开通自动续费后，百度智能云将在实例到期前7/5/3/1/0天定时进行自动扣款续费，续费成功或失败都将向您发送短信和邮件提醒。
续费周期	必填	开通自动续费后，选择您需要的自动续费周期（按年或按月）与续费周期数量（N个月或N年），点击“确定”。
节点选择	必填	根据不同的业务场景，为用户提供了多种不同规格的GPU节点配置/类型。
节点数量	必填	当前最少支持2个节点。
登陆凭证	必填	支持多种登陆方式，SSH密钥对是一种比常规密码更安全的登录云服务器的方式，更多内容详见帮助文档>
可用性检查	必填	开启后，将在资源池创建成功后自动进行可用性测试，以保障集群资源池&节点的运行状态和任务提交效果。

- 存储配置



配置项	必填/选填	配置说明
存储实例类型	选填	选择当前VPC内子网下已有的PFS存储实例与集群进行关联。
PFS L2挂载服务	选填	选择当前VPC内子网下已有的PFS存储实例与集群进行关联。

4. 确认资源池配置信息，然后单击提交。下单成功后，您可以在**自运维资源池列表**页面点击**资源池名称**查看详情。

1111

管理自运维资源池

概述

本文介绍如何查看自运维资源池详情、查看资源池状态和删除资源池，有助于您更为全面地了解资源池的信息，并进行操作。

查看资源池详情

操作步骤

1. 登录百舸AI计算平台AIHC控制台。

2. 在左侧菜单栏选择资源池，进入资源池列表页面，资源池列表页面已为您提供基本节点数量统计信息，若需继续查看更多详情，找到您想要查看的实例，单击资源池名称。



4. 资源池详情页面。

- 查看数据概览。您可以查看节点/GPU状态统计。
  - 节点状态统计：已分满、空闲可用、空闲受限可用（有污点）、空闲不可用（封锁或Not Ready）的节点数。
  - GPU状态统计：已分配、空闲可用、空闲受限可用（有污点）、空闲不可用（封锁或Not Ready）的GPU卡数。

状态名称	说明
已分配	可用卡数为0
空闲可用	无需加容忍即可使用
空闲不可用（有污点）	需要加容忍才能使用
空闲不可用（封锁或Not Ready）	因封锁或Not Ready空闲但不可用

您可以点击“空闲可用”、“空闲受限可用（有污点）”、“空闲不可用（封锁或Not Ready）”的GPU卡数跳转到对应的节点查看具体信息。

- 查看基本信息。您可以查看资源池名称、ID、地域及创建时间，并开启或关闭资源池保护功能。
- 查看资源信息。您可以查看计算资源集群、存储资源、监控实例信息。
  - 计算资源集群（CCE）：名称、节点数量、可用空闲卡数/总卡数、CCE集群状态。
  - 存储资源（PFS）：名称、总存储容量、存储利用率、PFS状态。
  - 监控实例：名称、CPROM状态。

您可以点击“计算资源集群（CCE）”、“关联存储资源（PFS）”、“关联监控实例（CPROM）”跳转到对应界面查看具体信息。

查看资源池状态

资源池的状态反映了资源池所处的不同阶段，而不同状态下用户可以执行的操作也会有所不同。资源池的状态包括：部署中、部署失败、运行中、升级中、升级失败、删除中和删除失败。当资源池成功创建后，其状态将显示为“运行中”。用户可以根据资源池的状态进行相应的管理和操作，以满足需求。

操作步骤

1. 登录百舸AI计算平台AIHC控制台。

2. 在左侧菜单栏选择**自运维资源池**，进入资源池列表页面。

3. 在**自运维资源池列表**页面，查看资源池具体状态。



删除资源池

删除资源池可以帮助您清理不必要的资源，并释放相关的计算和存储资源，以提高整体资源利用效率。删除资源池可以帮助您维护一个干净和有序的资源环境，以便更好地管理和利用云计算资源。

> 对于开启删除保护的资源池，需要先在资源池列表右侧**更多**操作中将该资源池的删除保护关闭，再进行删除。



操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**自运维资源池**，进入**自运维资源池列表**页面。
3. 在资源池列表页面，单击右侧**更多**按钮，对想要删除的资源池选择**删除**。



4. 在删除资源池弹框，选择删除资源类型。

> 资源池删除时将根据您的选择，自动删除或或保留节点、磁盘等资源。
> 若资源正在被其他资源池或云产品使用，将自动为您保留。 保留资源将继续进行计费，请您自行到对应控制台进行查看和操作。
> **AI算力资源紧张，资源释放后可能无法再次购买，请谨慎操作。**



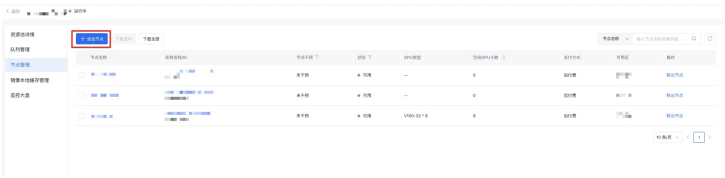
创建节点

概述

节点是百度百舸平台中的重要组成部分，在百度百舸平台中，节点可以看作是计算、存储和容器等服务的一个集合，用户可以根据实际需求选择合适的节点来执行AI任务。同时，百度百舸平台还提供了丰富的API和工具，方便用户对节点进行管理和监控，以及对AI任务进行调度和优化。
本文介绍如何查看节点详情、添加和移出节点，有助于您更为全面地了解并节点队列。

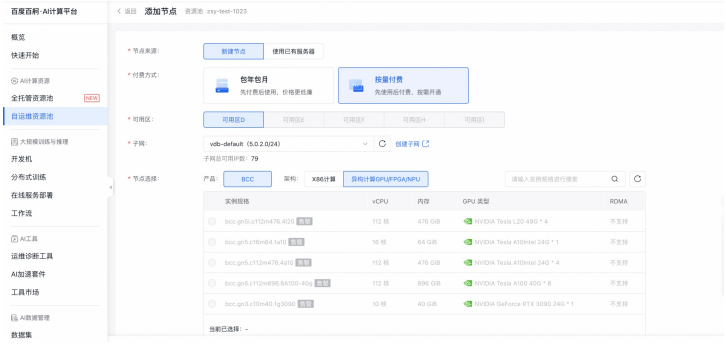
操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**，找到您想要查看的资源池名称，单击ID。
3. 单击**节点管理**页面，点击**添加节点**。



4. 在添加节点页面，根据实际需求完成相关参数配置。

配置项	必填/选填	配置说明
节点来源	必填	新建节点或使用已有服务器，使用已有服务器支持选择已有BCC、EBC、BBC、HPAS节点
付费方式	必填	预付费或后付费
可用区	必填	当前资源池支持的容器子网所在的可用区，如可用区不可选，请先在资源池详情中添加对应可用区的容器子网
子网	必填	选择节点所在子网
节点选择	必填	选择要加入资源池中的节点套餐规格
节点数量	必填	本次加入资源池的节点数量
登录凭证	必填	选择登录节点的凭证，支持密钥对、用户自定义密码和创建后设置
封锁节点	选填	勾选时，本次加入资源池的节点默认开启封锁，新的任务不会调度到该节点上，通常在新节点加入资源池时开启，新节点检查符合预期后可关闭封锁。



5. 确认节点配置信息，然后单击提交。创建成功后，您可以在**节点管理**页面点击**节点名称**查看详情。

管理节点

概述

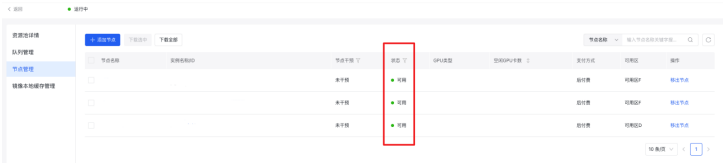
本文介绍如何查看节点状态、查看节点详情、移出节点，以及节点内容器组的相关操作，有助于您更为全面地了解节点的信息，并进行操作。

查看节点状态

节点的状态反映了节点所处的不同阶段，而不同状态下用户可以执行的操作也会有所不同。当节点成功创建后，其状态将显示为“创建中”、“创建失败”、“可用”、“不可用”、“移出中”、“移出失败”、“可用（已封锁）”。用户可以根据节点的状态进行相应的管理和操作，以满足需求。

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**，找到您想要查看的资源池名称，单击ID。
3. 单击节点**管理页面**，查看节点具体状态。

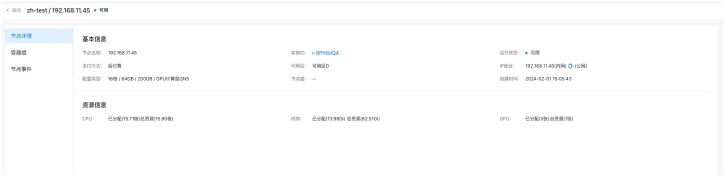


查看节点详情

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**。
3. 单击**节点管理**页面。
4. 找到您想要查看的节点名称，单击ID，进入节点详情页面。

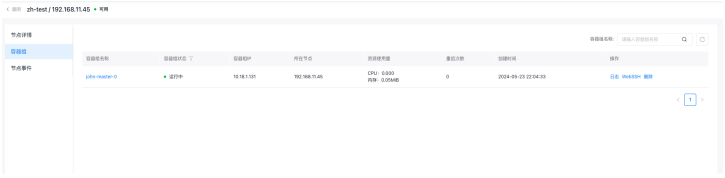
- 查看基本信息。您可以查看节点名称、实例ID、运行状态、支付方式、可用区、IP地址、配置类型、节点组、创建时间等基本信息。
- 查看资源信息。您可以查看CPU、内存、GPU的总资源与已分配资源。



5. 单击**容器组**页面。

您可以查看容器组名称、容器组状态、容器IP、所在节点、资源使用量、重启次数、创建时间、操作。

- 容器组状态包括“已成功”、“生效中”、“运行中”、“已失败”。
- 资源使用量包括CPU使用量和内存使用量。
- 操作包括查看容器组日志、WebSSH和删除容器组。



移出节点

移出节点可以帮助您清理不必要的资源，并释放相关的计算和存储资源，以提高整体资源利用效率。移出节点可以帮助您维护一个干净和有序的资源环境，以便更好地管理和利用云计算资源。

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**，找到您想要查看的资源池名称，单击ID。
3. 单击**节点管理**页面。
4. 点击**移出节点**。





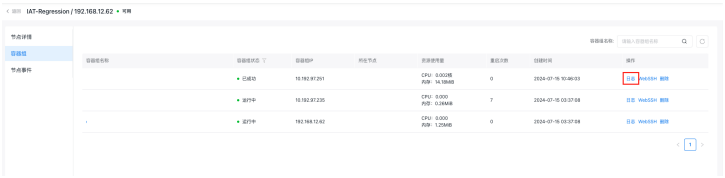
查看节点容器组日志

概述

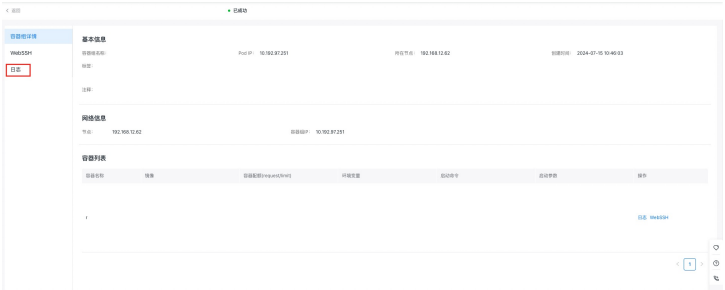
您可以选择您想要查看的节点内容器运行过程中产生的各种日志信息。

操作步骤

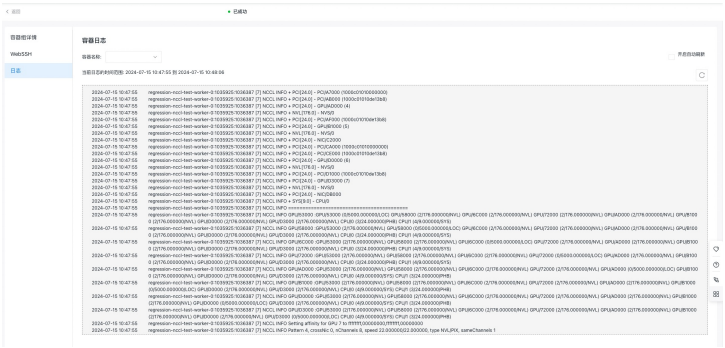
1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**。
3. 单击**节点管理**页面。
4. 找到您想要查看的节点名称，单击ID。
5. 找到左侧栏**容器组**，单击进入页面。
6. 找到您想要查看的容器组名称，单击日志；



或者单击ID，进入容器组详情页面，单击左侧栏日志部分。



7. 查看容器日志：选择想要查看日志的容器名称、当前日志的时间范围、是否开启自动刷新和刷新按钮



WebSSH接入节点容器组

概述

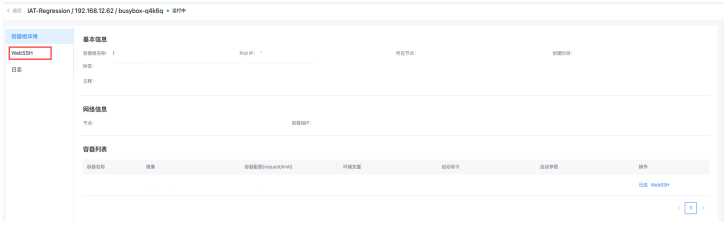
通过WebSSh的方式连接您想要接入的容器

操作步骤

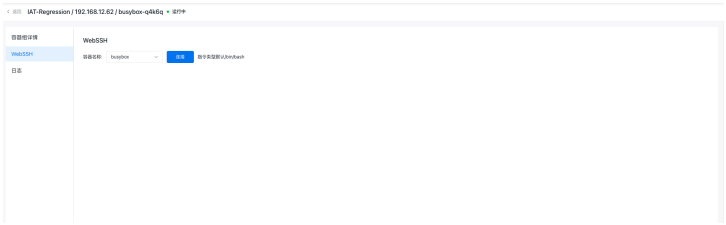
1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**。
3. 单击**节点管理**页面。
4. 找到您想要查看的节点名称，单击ID。
5. 单击左侧栏容器组，找到您想要查看的容器组名称，单击WebSSH。



或者单击容器组ID，进入容器组详情页面，左侧菜单栏选择**WebSSH**。



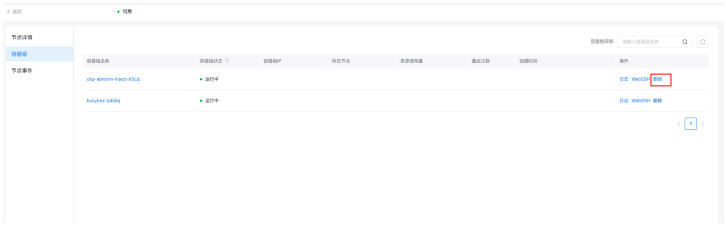
6. 选择您想要接入的容器名称，单击**连接**，（指令类型默认/bin/bash）。



删除容器组

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**资源池**。
3. 单击**节点管理**页面。
4. 找到您想要查看的节点名称，单击ID。
5. 点击左侧栏容器组进入页面，找到您想要删除的容器组名称，单击删除。
6. 在确认删除窗口中单击确认。





查看监控大盘

接入监控实例并启用采集任务

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧导航栏**自运维资源池**中选择您想要查看监控大盘的**资源池名称**，并点击列表中的**监控**按钮。



3. 若您**所选资源池**未关联过监控实例，点击后会提示您**未关联监控实例**，点击**确定**，跳转至**资源池详情**。



4. 在**资源池详情**中点击**资源信息**中**关联监控实例(CPROM):快速接入**。



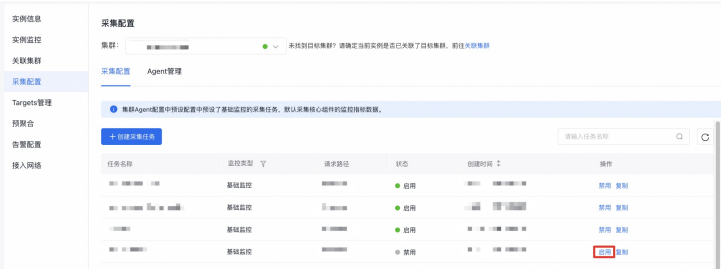
5. 选择**新建实例**后，点击**确定**。



6. 接入成功后在**资源池详情**页面下方点击**关联监控实例(CPROM)**，跳转至**Prometheus监控服务**。



7. 在左侧导航栏中点击**采集配置**后，右侧选择**目标集群**，在下方**采集配置**列表中找到所需要的任务名称，在右侧**操作**中点击**启用**。任务状态由禁用转为启用。



各大盘需启用的采集任务

大盘名称	采集任务					
	volcano	kubelet	gpu-dcgm	kubernetes-pods	cadvisor	kubernetes-pods-kube-state-metrics
资源统计	√	√	√	√	√	√
节点资源	√	√	√	√	√	√
工作负载	√	√	√	√	√	√
调度器组件稳定性	√	√	√	√	√	√
GPUManager稳定性	—	—	—	—	—	√

NVIDIA芯片资源观测

资源池总览

GPU资源池总览，包含节点使用情况、集群GPU卡分配情况、GPU卡使用情况、CPU&内存使用情况、GPU卡分配数、GPU卡分配率、GPU卡利用率、GPU卡显存利用率、节点信息、占用GPU的工作负载信息。 **前提条件**

- AI Job Scheduler版本 >= 1.7.9
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧导航栏**自运维资源池**中选择您想要查看监控大盘的**资源池名称**，并点击**监控**按钮。



3. 跳转至**资源观测**页面，选择**资源池总览**。

资源池总览如图所示：



您可以点击右上角**按钮**，自行设定**监控时间**、**手动刷新**、**自动刷新**。

资源池总览具体说明

节点使用情况

监控项	说明
总节点数	集群中所有节点
已分配节点数	可用GPU卡数为0的节点
空闲节点数	GPU卡数大于0的节点，含污点节点
不可用节点数	封锁或not ready节点



集群卡分配情况

监控项	说明
总卡数	集群中所有节点上的GPU卡数
已分配卡数	已分配使用中的GPU卡数
空闲卡数	GPU卡数大于0的节点，含污点节点空闲卡数
不可用卡数	集群中节点上的不可用卡数



卡使用情况

监控项	说明
卡平均利用率	当前集群内所有节点上GPU卡平均利用率实时值，卡平均利用率=sum(所有节点GPU卡利用率)/所有节点GPU卡数
卡显存平均利用率	当前集群内所有节点上GPU卡显存平均利用率实时值，显存平均利用率=sum（所有节点GPU卡显存利用率）/所有节点GPU卡数



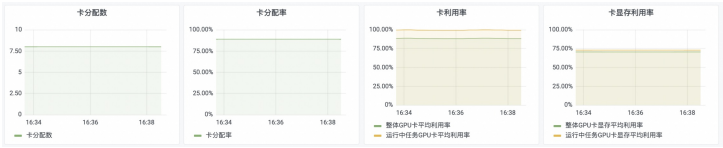
CPU&内存

监控项	说明
CPU核数	当前集群内CPU总核数
CPU平均利用率	当前集群内所有CPU平均利用率实时值
内存总量	当前集群内存总量
内存平均利用率	当前集群内所有内存平均利用率实时值



利用率 & 分配率

监控项	说明
卡分配数	已分配GPU卡数
卡分配率	分配率=已分配GPU卡数/总GPU卡数
整体GPU平均利用率	当前集群内所有节点上GPU卡平均利用率实时值，平均利用率=sum(所有节点GPU卡利用率)/所有节点GPU卡数
运行中任务GPU平均利用率	GPU平均利用率=sum(已分配GPU卡利用率)/已分配GPU卡数
整体GPU显存平均利用率	当前集群内所有节点上GPU卡显存平均利用率实时值，显存平均利用=sum(所有节点GPU卡显存利用率)/所有节点GPU卡数
运行中任务GPU显存平均利用率	GPU显存平均利用率=sum(已分配GPU卡显存利用率)/已分配GPU卡数



节点信息

监控项	说明
节点名称	当前集群内的节点名称
节点状态	当前节点状态
已分配卡数	当前集群内节点上分配的GPU卡数
GPU利用率	当前节点内所有GPU平均利用率实时值
GPU显存平均利用率	当前节点内所有GPU显存平均利用率实时值
空闲卡数	当前集群内节点上空闲的GPU卡数
CPU核数	当前节点内CPU总核数
内存总量	当前节点内寸总量



占用GPU的工作负载信息

监控项	说明
负载名称	当前集群占用GPU的工作负载名称
类型	当前集群占用GPU的工作负载类型
命名空间	当前集群占用GPU的工作负载所在命名空间
开始时间	当前集群占用GPU的工作负载开始时间
运行时长	当前集群占用GPU的工作负载运行时长
GPU配给卡数	当前集群占用GPU的工作负载配给GPU卡数
显存平均利用率	当前集群占用GPU的工作负载内所有GPU卡显存平均利用率实时值
GPU平均利用率	当前集群占用GPU的工作负载内GPU卡平均利用率实时值
内存使用量	当前集群占用GPU的工作负载的内存使用量
CPU核数	当前集群占用GPU的工作负载的CPU核数



节点资源

GPU节点资源，包含占用GPU的Pod数量、GPU卡分配情况、GPU卡使用情况、GPU&Memory总量、GPU&Memory利用率、GPU卡平均利用率、GPU卡利用率、卡平均显存利用率、卡显存利用率、GPU利用率、Memory利用率和运行中的占GPU的Pod列表。

前提条件

- AI Job Scheduler版本 >= 1.7.9
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧导航栏**自运维资源池**中选择您想要查看监控大盘的**资源池名称**，并点击**监控**按钮。



3. 跳转至**资源观测**页面，选择**节点资源**。

节点资源如图所示：



您可以点击右上角**按钮**，自行设定**监控时间**、**手动刷新**、**自动刷新**。

节点资源具体说明

占用卡的Pod数量

监控项	说明
占用GPU的Pod数量	当前节点内占用GPU资源的Pod数量



卡分配情况

监控项	说明
总卡数	当前节点内全部GPU卡数
分配数	当前节点内已分配的GPU卡数
卡分配率	分配率=已分配GPU卡数/总GPU卡数



卡使用情况

监控项	说明
卡平均利用率	当前节点内所有GPU卡平均利用率实时值，卡平均利用率=sum(所有GPU卡利用率)/所有GPU卡数
卡显存平均利用率	当前节点内所有GPU卡显存平均利用率实时值，显存平均利用率=sum(所有GPU卡显存利用率)/所有GPU卡数



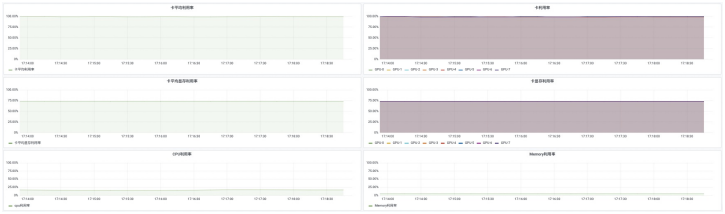
CPU&Memory总量/利用率

监控项	说明
CPU核数	当前节点内CPU总核数
CPU平均利用率	当前节点内所有CPU平均利用率实时值
内存总量	当前节点内存总量
内存利用率	当前节点内所有内存平均利用率实时值



利用率

监控项	说明
卡平均利用率	当前节点内所有GPU卡平均利用率实时值，卡平均利用率=sum(所有GPU卡利用率)/所有GPU卡数
卡利用率	当前节点内所有GPU卡利用率实时值
卡平均显存利用率	当前节点内所有GPU卡显存平均利用率实时值，显存平均利用率=sum(所有GPU卡显存利用率)/所有GPU卡数
卡显存利用率	当前节点内所有GPU卡显存利用率实时值
CPU利用率	当前节点内所有CPU利用率实时值
Memory利用率	当前节点内所有内存利用率实时值



运行中的占GPU的Pod列表

监控项	说明
负载名称	当前节点运行中的占GPU的Pod所在工作负载名称
类型	当前节点运行中的占GPU的Pod任务类型
命名空间	当前节点运行中的占GPU的Pod所在命名空间
Pod名称	当前节点运行中的占GPU的Pod名称
GPU配给卡数	当前节点运行中的占GPU的Pod的配给GPU卡数
GPU平均利用率	当前节点运行中的占GPU的Pod内GPU卡平均利用率实时值
GPU显存平均利用率	当前节点运行中的占GPU的Pod内GPU卡显存平均利用率实时值
内存使用量	当前节点运行中的占GPU的Pod的内存使用量
CPU核数	当前节点运行中的占GPU的Pod的CPU核数



工作负载资源

工作负载资源，包含任务属性、GPU卡数、GPU资源使用情况、GPU节点使用概要。

前提条件

- AI Job Scheduler版本 >= 1.7.9
- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

使用方法

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧导航栏**自运维资源池**中选择您想要查看监控大盘的**资源池名称**，并点击**监控**按钮。



3. 跳转至**资源观测**页面，选择**工作负载资源**。

工作负载资源如图所示：



您可以点击右上角**按钮**，自行设定**监控时间**、**手动刷新**、**自动刷新**。

工作负载资源具体说明

任务属性

监控项	说明
负载名称	当前工作负载的名称
类型	当前工作负载的类型
命名空间	当前工作负载所在命名空间
开始时间	当前工作负载的开始时间
运行时长	当前工作负载的运行时长



卡数&GPU资源使用情况

监控项	说明
卡数	当前工作负载的GPU卡数
GPU利用率	当前工作负载所有GPU的平均利用率实时值
显存利用率	当前工作负载所有显存的平均利用率实时值
显存使用量	当前工作负载所有显存的使用量实时值



GPU节点使用概要

监控项	说明
命名空间	当前工作负载中GPU节点所在命名空间
节点IP	当前工作负载中GPU节点IP
Pod名称	当前工作负载中GPU节点中运行Pod名称
GPU配给卡数	当前工作负载中GPU节点的GPU配给卡数
GPU平均利用率	当前工作负载中GPU节点的GPU平均利用率
显存使用量	当前工作负载中GPU节点的显存使用量
显存平均利用率	当前工作负载中GPU节点的显存平均利用率



大盘告警配置管理

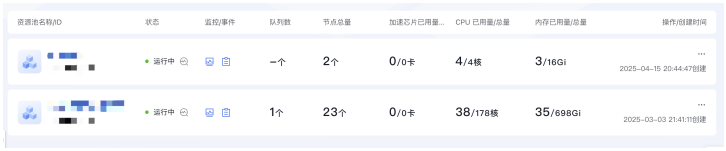
通过创建监控告警，您可以制定针对特定指标数值进行告警。当告警事件的条件满足后，系统会自动通知您告警信息。

前提条件

- 已接入监控实例
- 需启用采集任务,具体参考文档：接入监控实例并启用采集任务

创建告警

1. 登录百舸AI计算平台AIHC控制台，点击**自运维资源池列表**，选择资源池名称，进入资源池详情页面。



2. 点击关联监控实例(CPROM)进入监控实例详情



3. 在左侧控制栏中选择告警配置，点击创建告警。



4. 配置告警策略信息：

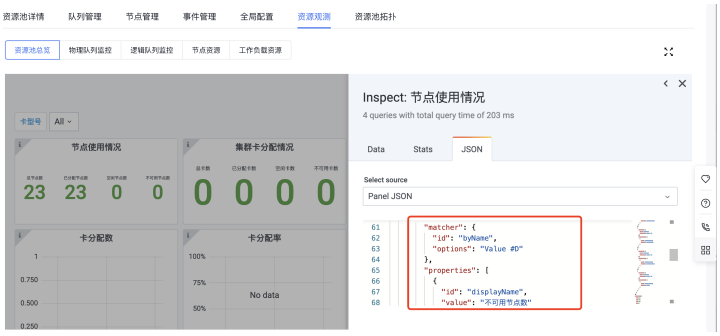
告警模板	告警名称	告警规则	告警内容	标签	注释
资源统计-节点不可用报警	自定义	(count((volcano_node_idle{clusterID="您的资源池ID",resource_type="totalGPU"}>0 and on (node_name) volcano_node_status{node_reason="NotReady"}) or (volcano_node_idle{resource_type="totalGPU"}>0 and on (node_name)volcano_node_status{node_reason="Unschedulable"})) > 0	百度智能云-百舸AI计算平台-告警消息，资源池：{{ $labels.clusterID }} ，不可用节点 > 0 ，请关注	标签名称：clusterID 标签值：您的资源池ID	自定义

4.1. 告警规则获取：

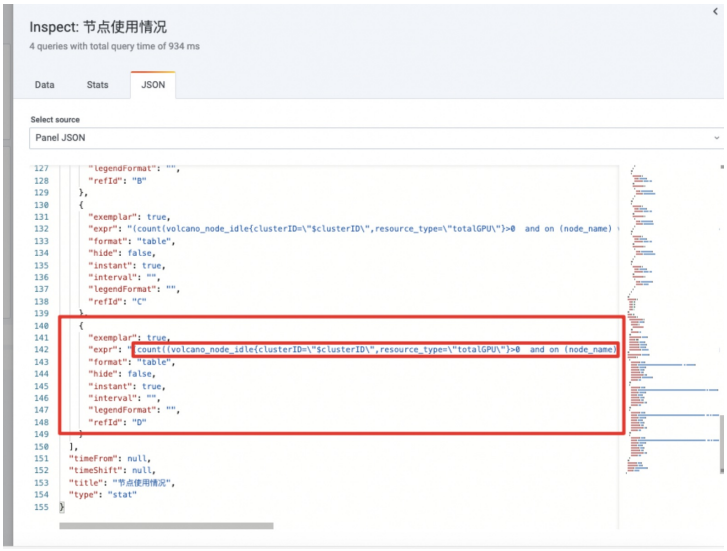
4.1.1. 进入观测资源页面点击所需仪表盘设置，点击弹窗中 Inspect 中的 Panel JSON



4.1.2. 在找到您所需指标对应options和value，例如：图中所选options为value #D，value为不可用节点数



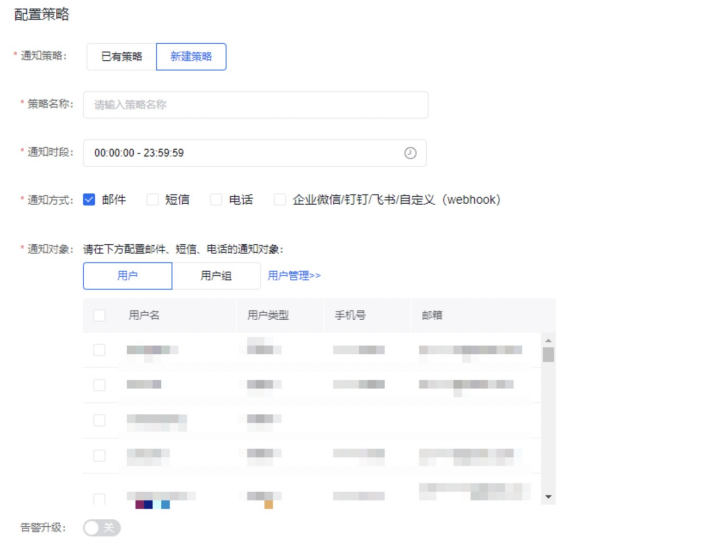
4.1.3. 通过options对应的refId（图中为D）找到所需指标字段，复制指标对应expr字段（删除路径中的"\"，将$clusterID替换为您的集群ID）



4.2. 填入告警规则中，在字段后自定义添加告警条件，例如：(example) > 0 为当example的值大于0时发出告警



5. 配置通知策略，包括策略名称、通知时段、通知周期、通知方式、通知对象、告警升级等，可选择已有通知策略模板或新建自定义通知模板。



6. 点击“创建”，创建告警策略和对应的通知策略。

管理告警策略

1. 创建告警策略成功后，在“监控实例 > 告警配置”中可查看当前监控实例的告警策略列表，展示告警名称、告警规则、持续时间、通知策略、状态等信息。



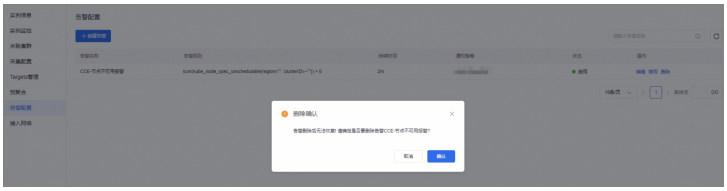
2. 在告警策略的操作列中，点击“编辑”，即可修改告警策略的信息。



删除告警策略

创建告警策略成功后，在“监控实例 > 告警配置”中可查看当前监控实例的告警策略列表，在告警策略的操作列中，点击“删除”，在二次确认弹窗中，点击“确认”，即可删除告警策略，相关告警规则自动失效。

> 注：告警策略暂不支持自定义模板保存和管理，删除后将无法找回，请谨慎操作。



镜像本地缓存管理

概述

镜像本地缓存是指在资源池所有节点预部署指定镜像，用于提升百舸任务的启动速度和稳定性。

使用限制

该功能仅支持在自运维资源池中使用。

创建镜像缓存

您可以通过创建镜像缓存实现指定镜像预部署。 **操作步骤**

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**自运维资源池**，找到需要使用镜像缓存功能的资源池名称，单击进入**资源池详情页**。
3. 单击“全局配置”-“镜像管理”，点击“创建镜像缓存”。



4. 在“创建镜像缓存”弹窗中配置镜像地址和用户名、密码（仅私有镜像仓库需要授权）。



5. 确定完成即可。

查看&管理镜像缓存

创建镜像缓存后，您可以通过镜像缓存列表查看已配置镜像的缓存进度。 **操作步骤**

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**自运维资源池**，找到需要使用镜像缓存功能的资源池名称，单击进入**资源池详情页**。
3. 3. 单击“全局配置”-“镜像管理”，查看镜像缓存列表，可以查看已配置镜像的缓存进度。



4. 点击“查看”可查看节点缓存详情。



5. 点击“删除”可删除镜像缓存配置。



故障自愈

功能概述

大模型训练过程中服务器会有极小概率出现硬件级别故障，特别是在大规模算力使用场景下，算力规模越大发生硬件故障的可能性越高。当发生硬件故障时可能会影响节点和节点上服务的正常运行。

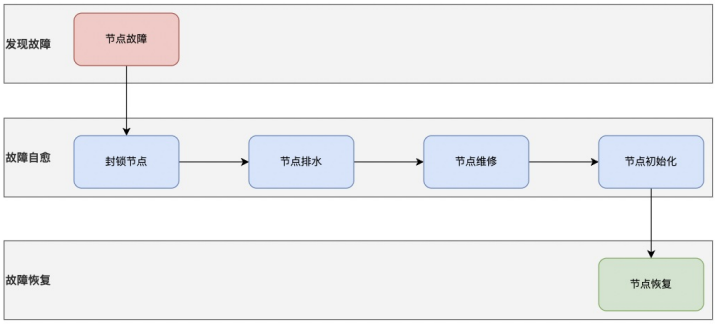
针对这一场景，百舸提供故障自动感知、上报能力，并配合业务情况针对不同的故障信息执行一系列自愈操作（如封锁节点、重启、机器维修等），降低故障对业务的影响以及充分保证节点的可用性和业务的稳定运行。

功能说明

故障自愈范围 百舸提供针对以下故障的自愈能力：

故障分类	故障名称	故障说明	故障影响	自愈策略
GPU	GPUUnhealthyExt	GPU带外故障	GPU异常	封锁节点，排水，维修
	GPUUnhealthy	GPU带内故障	GPU异常	封锁节点，排水，重启或维修
主板	MainboardUnhealthyExt	主板带外故障	主板异常	封锁节点，排水，维修
内存	MemoryUnhealthyExt	内存带外故障	内存异常	封锁节点，排水，维修
	MemoryUnhealthy	内存带内故障	内存异常	封锁节点，排水，维修或重启
网卡	NICUnhealthy	内存带内故障	内存异常	封锁节点，排水，维修或重启

故障自愈流程



使用说明

使用前提 资源池已经安装百度云 CCE CCE Node Remedier 组件和 CCE Node Problem Detector 组件

- CCE Node Remedier：节点自动维修系统，实现节点故障自愈，要求组件版本>=0.2.3
- CCE Node Problem Detector ：实时检测节点上各种异常情况，并将检测结果上报至集群 >=0.8.30

开启故障自愈

1. 选择“资源池详情 > 节点管理 > 故障自愈”，点击“立即开启”。



2. 故障自愈配置

（1）故障检测：开启故障自愈后，会自动开启故障节点检测。

（2） 节点封锁：自动封锁故障节点，避免后续新任务调度到故障节点无法运行。可配置自动封锁节点数量上限，达到上限后，新检测到的故障节点不会被自动封锁。

（3）节点自愈：针对GPU、内存、主板、网卡等故障进行自愈恢复处理，可单独关闭节点自愈功能，关闭节点自愈后，故障节点仅被封锁，不进入节点自愈流程。



3. 消息订阅：开启“故障自愈 > 节点自愈”后，支持开启消息订阅功能，故障节点进入自愈流程会触发消息通知，支持邮件、电话、短信、webhook等方式通知对应用户。



自愈活动

开启故障自愈后，检测到故障节点并触发自愈流程后，会生成一条自愈活动记录，可点击自愈活动ID查看自愈活动详情。





队列管理

队列功能概述

队列是一个资源池中部分资源的集合，用于工作负载（训练任务、模型服务）的运行。用户可将资源池划分成若干个独立队列（逻辑队列、物理队列）来运行不同工作负载。资源池创建成功后会默认生成default队列。

队列分为逻辑队列和物理队列，逻辑队列仅通过配额限制资源使用，与物理节点无绑定；物理队列绑定节点，实现队列间资源物理隔离。一级逻辑队列和物理队列下都可再创建二级队列，形成层级队列，方便用户细粒度资源管理。

本文将介绍如何通过百舸控制台创建、更新和使用队列。

前提条件

在 CCE 组件管理页面，选择云原生 AI 栏目，CCE AI Job Scheduler 组件版本按照下图升级到最新版本（1.7.23 以上），以使用相关队列功能。



逻辑队列

普通队列 普通队列通常是为不同业务部门分配资源配额最常见的队列形式，是将固定数量的配额资源分配给不同部门使用，实现基础的资源分配能力。 **弹性队列** 弹性队列是帮助平台型客户更高效为不同业务部门分配资源配额的队列形式。在弹性队列中允许队列闲置配额资源借出给其他队列使用，实现资源出借回收、任务超发抢占等能力，以提高GPU/NPU资源的利用率。具体的任务超发、抢占资源的策略见附录详情。

物理队列

通过物理队列对GPU/NPU资源进行物理隔离，支持客户保障重点业务的资源供给，避免资源抢占。同时在物理队列下可以创建多个子逻辑队列。

队列功能说明

配额设置说明

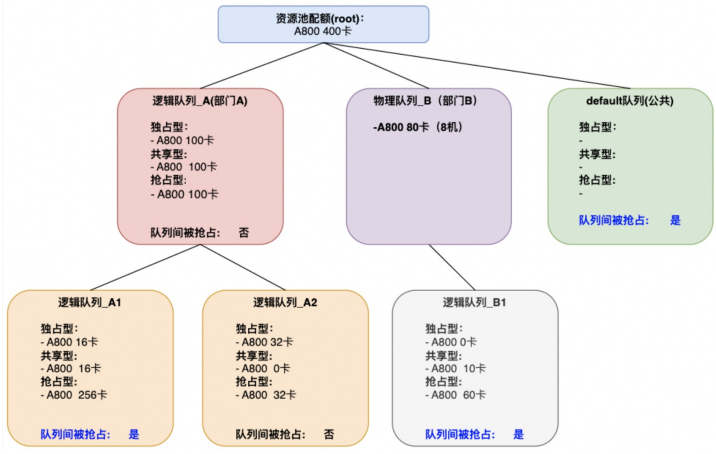
弹性队列需要设置三类资源配额：独占型、共享型和抢占型，以更好地优化成本并提高GPU/NPU资源的利用率。

- 独占型：队列独占资源，仅限本队列使用，不会被其他队列抢占。
- 共享型：队列可对其他队列共享的资源。开启队列间抢占功能时，其他队列可使用该队列中空闲的共享型资源。
- 抢占型：队列可从资源池或其他队列空闲资源中抢占的资源，独占+共享型资源不足时可抢占其他队列空闲资源。



物理队列仅需选择GPU型号和节点数量即可，支持指定节点加入到物理队列中，物理队列下可再创建子队列（逻辑队列）。

针对以上配额的概念，可参考以下示例便于理解：



调度策略说明

弹性队列支持两种调度策略：

- 遍历策略（BestEffortFIFO）：当队列中的第一个任务无法出队，则会跳过该任务，然后依次尝试对后续队列中的任务进行出队操作。弹性队列选择Fifo调度策略时，需要注意合理配置独占型资源配额，配置独占型资源不足且队首任务申请资源较多时，可能会出现长期排队资源无法满足的情况。
- Fifo策略（StrictFIFO）：当队列中的第一个任务无法出队，系统将反复尝试对第一个任务进行出队操作，而不会跳过。

队列内任务排队策略：

- 队内任务排队策略：按照优先级进行排序，同优先级下按照创建PodGroup的时间（约等于任务的创建时间）顺序进行排序。
- 容错任务重提后按照优先级和任务创建时间重新排队。

队列限制说明

队列配额遵循以下的限制：

- 单个子队列"独占型+共享型+抢占型"配额之和不允许大于父队列"独占型+共享型+抢占型"配额之和
- 同一父队列下的子队列"独占型"配额之和不允许大于父队列"独占型"配额
- 同一父队列下的子队列"独占型+共享型"配额之和不允许大于父队列"独占型+共享型"配额

层级队列遵循以下限制：

- 不支持指定default 队列为父队列
- 不支持指定第二层队列为父队列
- 不支持不允许向已存在运行任务的队列添加子队列
- 目前仅支持创建1级子队列，队列结构即"资源池配额-父队列-子队列"。

删除队列遵循以下限制：

- 层级队列中不允许直接删除父队列，需要先删除子队列后，再删除父队列；
- 有任务正在运行的队列不允许删除

使用说明

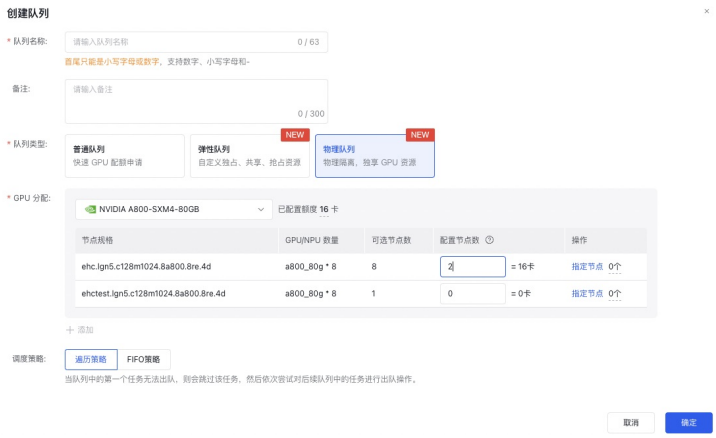
普通/弹性队列配置项说明

配置项	说明
队列名称	队列名称，首尾只能是小写字母或数字，支持数字、小写字母和-
父队列	该队列所属的父队列名称，默认为资源池配额，也可选择其他没有正在运行任务的一级队列。
队列类型	按需选择普通队列或弹性队列，弹性队列支持闲置资源借出回收，借用资源池空闲资源，如无此需求可选择普通队列
GPU分配	普通队列直接申请配额，弹性队列按需配置独占型、共享型、抢占型配额。
调度策略	弹性队列按需选择遍历策略或Fifo策略
队列间抢占	是否开启队列间抢占策略，借用的资源是否接受被抢占。默认关闭



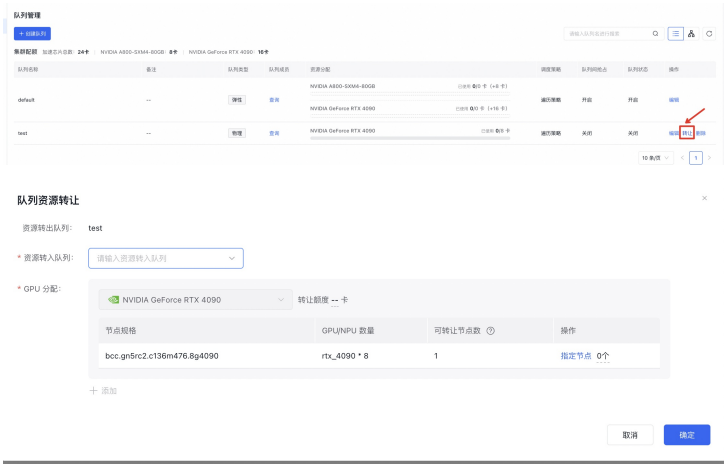
物理队列配置项说明

配置项	说明
队列名称	队列名称，首尾只能是小写字母或数字，支持数字、小写字母和-
GPU分配	选择要加入该物理队列的GPU类型及节点数量，支持指定节点加入物理队列。
调度策略	物理队列按需选择遍历策略或Fifo策略。



物理队列资源转让

指定节点转让到其他物理队列，仅在队列关闭状态下支持资源转让。



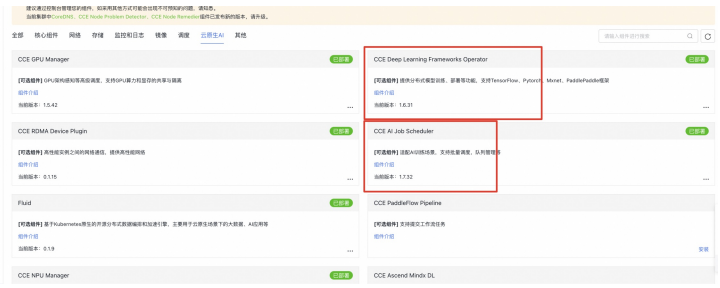
碎片治理

概述

为优化资源利用率并降低计算成本，您可以开启百舸碎片治理功能，有效解决分布式训练、小规模推理等场景下的非整机任务导致的资源碎片问题，实现资源高效回收与复用。

前提条件

在 CCE 组件管理页面，选择云原生 AI 栏目，CCE AI Job Scheduler、CCE Deep Learning Frameworks Operator 组件版本按照下图升级到最新版本，以使用碎片治理功能。



功能说明

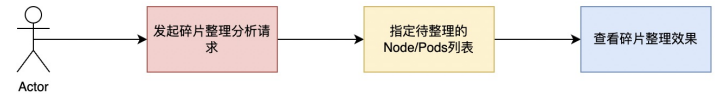
碎片诊断

1. 诊断工作负载：分布式训练任务和自定义服务部署。
2. 诊断输入：物理队列和节点，支持对指定物理队列或节点进行碎片诊断（仅支持治理含加速芯片的队列和节点），定位碎片资源分布情况。
3. 不可治理的原因：
   - 关联分析任务已不存在
   - 任务关闭重调度
   - 预调度失败
   - 达到任务最低保障副本
   - 单负载的工作负载
   - 无工作负载管理的Pod
   - 关联Pod已不存在
   - 错误的整理类型
   - 推理服务摘流失败
   - 推理服务终止失败

- 该Pod已在重调度中，重复的重调度
- Pod迁移失败
- 该工作负载已达到最大pod迁移数

碎片治理

诊断完成后，可以基于诊断结果进行碎片治理：



操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**自运维资源池**，找到您想要查看的资源池名称，单击ID。
3. 单击**全局配置**页面，打开**碎片诊断与治理**功能。



4. 点击立即治理，开始对指定物理队列或节点进行碎片诊断并治理。



5. 针对已完成的诊断，点击“治理”，在弹窗中选择需要治理的工作负载，开启碎片治理。





6. 治理完成后，点击“详情”，查看碎片治理结果详情信息。



开启GPU显存共享

概述

本文介绍如何开启并使用GPU显存共享功能，开启后支持多任务共享同一张卡的显存和算力资源。

前提条件

开启GPU显存共享功能的节点基础环境符合兼容性要求，详情参考https://cloud.baidu.com/doc/CCE/s/9lrrdyikg 。 如节点基础环境不符合要求，需要从资源池中移出，在BCC控制台进行操作系统重装。

节点基础环境建议版本：

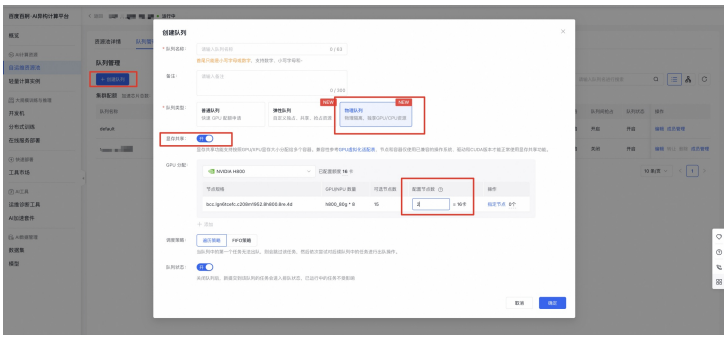
- OS：Ubuntu 22.04
- CUDA：12.2
- Driver：535.x

2. 组件版本要求：

- CCE AI Job Scheduler：1.7.34以上
- CCE GPU Manager：1.5.46以上

创建物理队列并开启显存共享功能

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**自运维资源池**，进入**资源池列表**页面，找到您想要查看的资源池，进入资源池详情。
3. 进入**队列管理**页面，创建物理队列，并开启“显存共享”开关，选择要添加到队列的节点规格和数量。

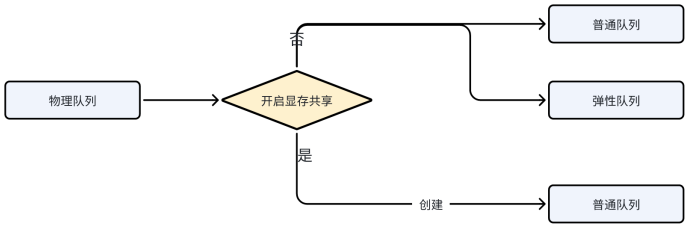


4. 单击**确定**，完成队列创建。
5. 队列列表页面能看到队列带有“显存共享”标签。



开启显存共享的物理队列创建子队列

平台支持在物理队列下继续创建子队列，从而实现更精细化的资源管理。



用户在创建普通队列时，父队列可以选择开启【显存共享】的物理队列（需要确保父队列没有任务），支持给子队列分配小数卡，如下图所示。



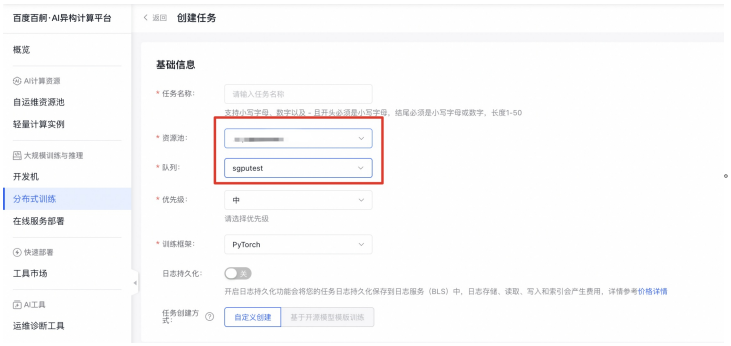
用户可以在物理队列的子队列上提交开发机、训练任务和在线推理服务。

提交训练任务

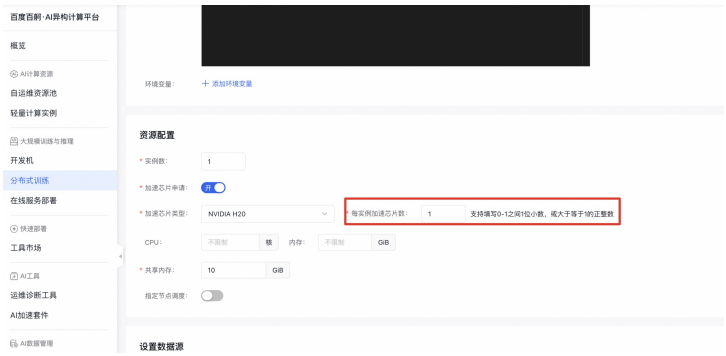
1. 进入百舸分布式训练页面并创建任务 https://console.bce.baidu.com/aihc/tasks



2. 选择开启了显存共享的资源池和队列



3. 资源配置中填写需要使用的算力资源，支持填写0-1之间的一位小数，如0.1即代表申请10%的算力和显存。



CCR镜像免密拉取功能

概述

CCR镜像免密拉取是指在资源池中完成CCR实例免密拉取配置后，使用该资源池的算力资源创建工作负载时无需输入镜像用户名密码即可免密直接使用。

操作步骤

1. 登录百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择**自运维资源池**，找到需要配置免密拉取功能的资源池名称，单击进入**资源池详情页**。
3. 单击【全局配置】-【通用配置】，点击镜像免密拉取右侧的【编辑】按钮。



4. 在“免密拉取配置”弹窗中配置CCR关联实例和免密配置的方式。支持2种免密配置方式：
   - 自动创建访问凭据：系统自动使用当前操作用户身份来创建访问凭据，请确保用户持续有效，否则无法拉取镜像；
   - 指定访问凭据：系统将使用指定用户的访问凭据，请确保用户及访问凭据持续有效，否则无法拉取镜像。



5. 确定完成即可。

资源规格模板

概述

资源规格模板是 AIHC 百舸平台为队列管理提供的标准化资源配置能力。管理员可在队列维度预设 GPU 数量与 CPU 数量、内存的搭配规格，算法工程师在创建任务时可直接套用模板，无需手动填写资源参数，从而降低配置门槛、减少资源碎片化，提升集群整体利用率。

功能概述

- **模板配置**：管理员可为队列预设多组资源规格模板，覆盖分布式训练、开发机、在线服务部署等负载场景。
- **自动创建**：新增含 GPU 的队列时，系统自动按节点最大规格生成常用卡数模板，无需手动配置。
- **两种使用模式**：支持**自由创建模式**（参考套用）和**严格规定模式**（强制选择），管理员可按需配置。
- **任务创建集成**：算法工程师创建任务时可通过模板一键回填资源参数，严格规定模式下只能从预设模板中选择。
- **复制/重跑兼容**：严格规定模式下，复制或重跑任务时自动匹配原规格对应的模板，若无匹配则提示重新选择。
- **权限分级管理**：平台管理员、资源池管理员、队列管理员按权限范围管理模板，算法工程师只读查看。

使用前提

- 已在 AIHC 百舸平台创建自运维资源池或全托管资源池，并完成队列配置。
- 需要配置或管理模板的操作者具备相应管理员权限（平台管理员 / 资源池管理员 / 队列管理员）。
- 算法工程师在创建任务时，目标队列已配置资源规格模板。

配置资源规格模板（管理员）

进入模板配置页面 自运维资源池：

1. 进入**自运维资源池**管理页面，选择目标资源池。
2. 在**队列管理**列表中，找到目标队列，点击**操作**列中的**更多，**在下拉菜单中选择**队列配置**。
    1. 注意：自运维资源池仅物理队列支持配置资源规格模板。
3. 在队列配置中点击**资源模板**开关，进入该队列的模板配置页面。



系统自动创建模板 新增含 GPU 的队列并开启资源规格模板配置时，系统将根据队列内节点的最大 GPU 规格，自动生成常用卡数的资源模板：

- 自动创建卡数覆盖：1 卡、2 卡、4 卡、…，直至节点最大卡数。
- CPU 与内存按节点最大规格等比分配。
    - 请注意：节点中最大可用内存与 CPU 数量会根据安装插件动态变化，自动创建的模板规格仅为创建时刻的参照值，请管理员根据变化变更模板配置。

示例： 队列内节点为 A800 × 8 卡，系统自动创建 1/2/4/8 卡四个规格模板。

队列显卡变更时，系统将自动删除原 GPU 类型对应的模板，并按新显卡规格自动创建对应模板。



编辑模板

1. 在模板配置页面点击**编辑**，即可进入模板编辑弹窗。
2. 按下表填写各字段：

字段	说明	是否必填
模板名称	自定义名称，同队列内不可重复	是
GPU 类型（芯片型号）	选择该模板对应的显卡型号；GPU 数量大于 0 时必填	条件必填
卡数	单任务使用的 GPU 数量，需大于 0	是
CPU（核）	分配给任务的 CPU 核数	是
内存	分配给任务的内存大小（单位：GiB）	是
适用模块	勾选适用的负载类型：**分布式训练** / **开发机** / **在线服务部署**，可多选	是

3. 填写完成后点击**保存**。

校验规则：

规则	说明
不可重复	同队列内，GPU 类型 + 卡数 + CPU + 内存的组合不可与已有模板重复
名称唯一	同队列内，模板名称不可重复
GPU 类型必填	当卡数 > 0 时，必须选择 GPU 类型（芯片型号）
必填项不可为空	模板名称、卡数、CPU、内存、适用模块均为必填项

复制模板 在模板列表中，点击目标模板行右侧的**复制**按钮，系统将自动创建一条与该模板规格相同的新模板行。修改所需字段后保存即可。

删除模板 在模板列表中，点击目标模板行右侧的**删除**按钮，确认后该模板将被移除。已被任务引用的历史规格不受影响。

配置模板管理模式（管理员）

管理员可在模板配置页面设置该队列（或负载类型）的模板使用模式：

模式	说明
自由创建模式（默认）	开启资源模板配置时默认为该模式，算法工程师创建任务时，资源填写区提供**资源模板**入口供参考套用，也可自行填写资源参数
严格规定模式	若该负载类型已配置模板，算法工程师只能从预设模板列表中选择，不可自定义填写；若某负载类型未配置任何模板，仍允许自定义填写

> 建议在对资源利用率要求较高或集群碎片化严重的场景下，启用**严格规定模式**。

创建任务时使用模板（算法工程师）

自由创建模式

1. 在创建任务页面，进入**资源配置**区域。
2. 点击**资源模板**入口，弹出当前队列下适用于该负载类型的模板列表。
3. 选择目标模板，系统自动回填 **GPU 类型**、**GPU 数**、**CPU 核数**、**内存**。
4. 回填后可根据需要手动调整，也可直接提交。

> 若所选模板的 GPU 类型在当前队列中不存在，系统将提示并禁止提交，请更换其他模板或联系管理员。 **严格规定模式**

1. 在创建任务页面，**资源配置**区域仅支持通过点击**资源模板**填写资源规格。
2. 从列表中选择适合的资源规格模板，系统自动填充对应资源参数。
3. 不支持手动修改资源参数，直接提交任务即可。

> 若该负载类型当前队列下暂无可选模板，系统将切换为自定义填写模式，允许手动输入资源参数。

复制/重跑任务的模板匹配（严格规定模式）

在严格规定模式下，复制或重跑已有任务时，系统将按以下规则处理资源规格：

场景	系统行为
找到完全匹配的模板（GPU 类型 / GPU 数 / CPU / 内存均一致）	自动选中对应模板，无需手动选择
未找到匹配模板，但该负载仍有可选模板	资源规格置空，并提示：**"当前队列已不支持原资源规格，请重新选择，原规格为 GPU×x，CPU×x，内存×xG"**
该负载类型当前队列下已无任何可选模板	切换为自定义填写模式，原规格自动填入，允许手动调整后提交

权限说明

角色	权限范围
平台管理员	可访问并修改所有资源池、所有队列的资源规格模板
资源池管理员	可访问并修改所管理资源池内所有队列的资源规格模板
队列管理员	可修改自己管理队列的资源规格模板
算法工程师	只读查看，不可修改（操作按钮隐藏）

常见问题

Q：新建队列后，系统没有自动生成模板，怎么办？

A：系统仅在队列包含 GPU 节点时自动创建模板。若队列为纯 CPU 队列，或节点资源尚未关联到队列，不会触发自动创建。请确认队列已正确关联 GPU 节点后，再手动添加模板，或联系平台管理员处理。

Q：严格规定模式下，算法工程师反馈没有合适的模板可选，怎么解决？

A：管理员可在模板配置页面为该队列和对应负载类型补充合适的规格模板。若某负载类型暂未配置任何模板，系统将自动允许自定义填写，不影响任务创建。

Q：更换队列显卡后，原有任务的历史记录是否受影响？

A：队列显卡变更仅影响模板列表的自动维护（删除旧模板、创建新模板）。已提交的历史任务及其资源记录不受影响。

Q：复制任务时提示"当前队列已不支持原资源规格"，如何处理？

A：此提示表示原任务所用资源规格在当前队列已无对应模板。请从现有资源规格模板中重新选择一个可用的资源规格模板后再提交。若有特殊需求，可联系队列管理员添加所需规格模板。

Q：资源规格模板适用于哪些资源池类型？

A：资源规格模板功能适用于**自运维资源池**下的物理队列，以及**全托管资源池**下的队列。

开发机

创建开发机

开发机是为模型开发者提供在线编译、代码调试、模型开发功能。支持使用云端WebIDE在线开发，集成了Notebook、VSCode、Terminal等多种开发环境，也支持通过SSH连接开发机远程开发。对实例生命周期进行管理，关机释放算力，持久化保存环境，实现开箱即用高效开发。

前置条件

如需使用此功能，请先联系售后工程师开通白名单。

操作步骤

1. 登录百舸AI计算平台，进入开发机列表页面
2. 点击**【创建开发机】**在创建页面填写相关参数，具体参数如下：

参数	说明
实例名称	支持小写字母、数字以及 - 且开头必须是小写字母，结尾必须是小写字母或数字，长度1-50；
资源池类型	选择开发机需要部署到的资源池。 **自运维资源池**：自运维资源池，需要先创建资源池购买用于AI推理的计算资源，请参考自运维资源池 **全托管资源池**：免运维资源池，您无需运维和管理资源池和算力资源，由平台保障资源池稳定性，请参考全托管资源池
资源池/队列	选择资源池和队列或资源队列
资源规格	支持创建CPU或GPU开发机，创建GPU开发机时需指定指定CPU、内存、GPU卡数。所选队列中若存在CPU机器，未使用加速芯片时如需优先调度到CPU机器，请勾选【优先使用CPU机型】。为勾选此按钮将优先调度到含有加速芯片的机型。
指定节点调度	对资源规格、节点驱动版本有特殊要求的场景可以使用指定节点调度功能，目前仅支持自运维资源池的物理队列指定节点调度
镜像地址	平台预置开发机镜像，您可以使用平台的预置镜像也可以自定义镜像。
百舸云盘	云盘用于存储开发环境、代码、无需共享的数据文件等， 开发机开关机时会云盘中的数据不会丢失。请填写开发机需要使用的云盘容量，至少100GiB，不能缩容。
启动命令	可选配置。输入开发机启动后自动执行的 Shell 命令。
环境变量	可选配置。设置容器运行时的全局变量，可以添加多组键值对（Key-Value），用于在不修改代码的情况下传递配置参数。
存储挂载	支持挂载PFS、CFS、Bos对象存储、数据集，相同存储类型的挂载源路径不能重复
工作目录	WebIDE和Notebook的启动目录，用户无设置的工作目录权限时平台会创建并使用 $HOME/workspace 目录作为工作目录。
启用SSH	选择是否需要通过SSH远程连接开发机，可启用SSH原生方式直连登录开发机实例，若配置了自定义镜像，请确认镜像安装了sshd。
启用自定义端口	支持将实例中启动的自定义服务通过自定义端口向外提供访问，需注意实例停止后该服务也会中断。
公钥	1. 启用SSH后需填写。远程开发需输入公钥，通常填写发起 SSH 登录机器上的~/.ssh/id_rsa.pub文件内容即可。若该文件不存在，可用 ssh-keygen命令生成。 2. 按照回车换行的方式逐个添加公钥，最多支持添加10个。发起SSH登录时，请注意检查私钥是否配对。
绑定BLB	1. 可以通过绑定BLB和映射端口，使开发机可被SSH访问、公网访问、对外提供自定义服务。 2.绑定支持公网的BLB实例，开发机可以被公网访问。如需开发机访问公网，需参考文档设置。
访问端口	1. 支持添加自定义端口，最多设置5个访问端口 2. 平台提供的预置镜像提供VSCode和JupyterLab，设置WebIDE端口时需要选择使用哪种WebIDE
资源抢占	开启后，当资源出现不足时，将抢占其它低优任务资源
共享内存	共享内存(shared memory)是Linux中不同进程之间实现高效数据请输入CPU交换和共享的方式， 在容器中其默认值为64MB,您可以在创建pod时指定其大小，以满足算法需求。开发机默认设置为申请内存的一半， 如果设置了CPU内存为共享模式，那么共享内存的默认值为10G。
所有者	支持修改所有者，默认创建人为开发机所有者，所有者对开发机有全部管理和使用权限。
可见范围	队列内可见：管理员及所有子账号均可见 所有者可见：仅管理员及所有者可见详情信息，其它子账号仅可见列表信息。
公开管理权	开启后，所在资源池和队列的管理员及成员均可管理并使用该开发机。

确认以上配置参数无问题后，点击**【确定】**。

管理开发机

开发机生命周期管理

开发机的的生命周期指从创建开始到删除结束，开发机实例所经历的状态。在不同状态，开发机的登录、操作不同。以下介绍开发机状态说明及状态转换说明：

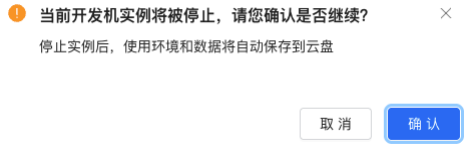
状态	描述信息
创建中	创建开发机的配置信息
排队中	开发机正在排队流程中，若资源不足会一直处于排队状态
部署中	调度成功开始部署
运行中	开发机已创建完成，可以通过 WebIDE或 SSH 连接开发机
停止实例中	开发机正在停止当前运行实例中
已停止实例	开发机成功停止当前实例，停止实例后不会丢失之前的操作和环境
开启实例中	开发机正在开启当前实例
更新中	开发机当先实例正在更新
扩容中	开发机当前实例正在扩容
制作镜像中	正在保存当前开发机环境
删除中	正在删除开发机
重启中	正在重新启动当前开发机
已释放	已经释放当前开发机资源
已失败	拉取镜像失败，内部请求参数异常
异常	开发机异常，无法正常使用，可在状态后查看异常原因。

变更配置

如需修改开发机的配置，请点击【变更配置】可修改开发机的资源配置、存储挂载、访问端口、访问权限等信息。

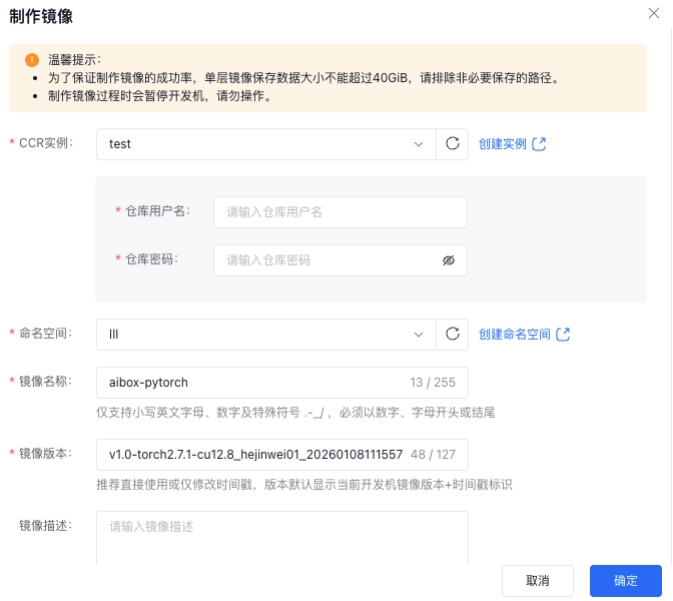
开发机停止实例

若开发机暂时不再使用，可以停止当前实例，来关闭开发机。开发机停止实例后，会保存环境配置、数据等信息。但会释放开发机占用的算力资源，再次开机可能会出现算力资源不足排队等待的情况。



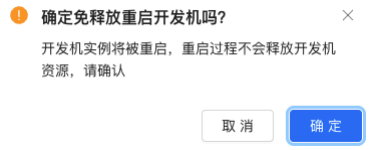
开发机制作镜像

如需基于当前现有的开发机实例快速创建一个相同镜像资源，可以使用制作镜像功能。



开发机免释放重启

在资源池详情中设置免释放重启，开发机实例将被重启，但重启过程中不会释放开发机资源，无须等待排队。



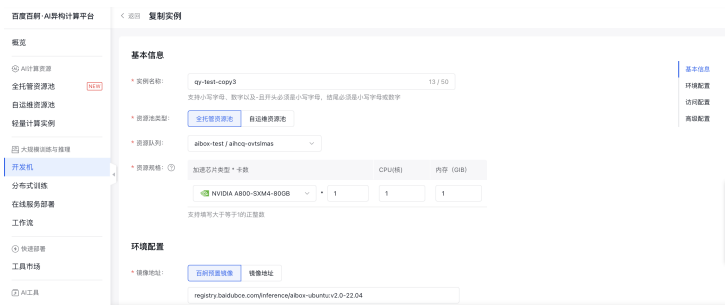
开发机定时停止实例

在资源池详情中设置定时停止实例策略，支持自定义时长，最低时长不得低于1小时，仅支持整时数，例如1.5小时则视为2小时。



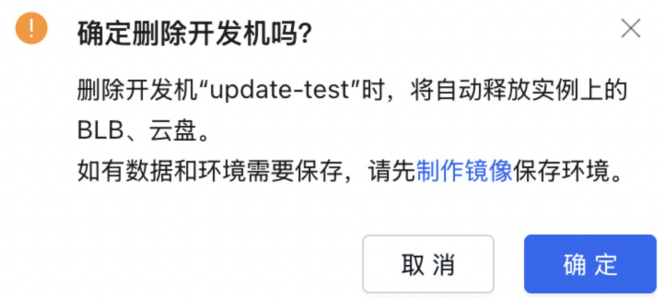
复制开发机

如需基于当前现有的开发机快速创建一个新开发机，可以使用复制实例功能。



删除开发机

若开发机使用已经结束，可以删除开发机。开发机删除后，会释放开发机的算力。如需保存开发环境和数据，请先制作镜像保存。



网络配置

如何通过公网访问开发机内的服务

概述

本文档介绍在开发机实例中配置自定义服务并开放公网访问的方法。当您在开发机实例中部署了服务（如模型 API、WebUI），并希望直接在浏览器访问、免登录分享给协作者测试，或需要从外网连接调用时，可以通过自定义服务功能配置公网访问能力，使服务暴露在互联网上。

注意： 此功能主要用于开发测试阶段。

前提条件

1. 已登录百度智能云控制台，并具备创建或更新开发机、创建应用型 BLB、创建或绑定弹性公网 IP（EIP）以及配置安全组的权限。

2. 账号无欠费状态，且 BLB、EIP、端口和 IP 组等配额满足创建与绑定要求。

3. 已创建或准备创建开发机实例，并已选择资源池。开发机所在 VPC 需要在选择资源池后确认。

4. 已规划服务在开发机容器内监听的端口、BLB 监听端口和传输协议，并确认安全组允许对应端口访问。## 实现原理

开发机通过 BLB 作为流量出入口。要将服务暴露到公网，需要提前创建一个绑定了弹性公网 IP（EIP）的 BLB 负载均衡实例。

为了让开发机实例内的服务（如运行在指定端口的 API）能从公网访问，需要以下资源协同工作：

1. **弹性公网 IP（EIP）**：提供固定的公网 IP 地址，是公网访问的入口。

2. **BLB 负载均衡**：负责将公网请求（EIP:端口）映射到 VPC 内的开发机实例（私有 IP:端口）。

3. **安全组**：相当于开发机实例的防火墙。您必须显式添加规则，允许公网流量访问服务监听的特定端口。这是访问成功的关键一步。

核心流程：公网用户（浏览器/工具）-> EIP:端口 -> BLB 负载均衡 -> 安全组规则检查 -> 开发机实例私有 IP:端口 -> 您的服务

产品限制说明

- **开发机服务配置**：每个开发机实例最多可配置 10 个服务，WebIDE 默认占用 1 个。
- **BLB 配额限制**：每个新建的 BLB 实例默认最多提供 50 个端口和 IP 组配额。如果端口和 IP 组被占满，BLB 将无法被绑定；每个 BLB 只能被一个集群绑定。
- **计费说明**：BLB 负载均衡和 EIP 从创建起即开始计费。即使开发机实例已停止，只要 BLB 实例和 EIP 实例未被删除，它们仍会持续产生费用。

如何配置公网访问

步骤一：创建带弹性公网 IP 的应用型 BLB

开发机要启用公网服务，需要绑定一个带弹性公网 IP（EIP）的应用型 BLB 负载均衡实例。如果已有符合条件的 BLB 实例，可跳过此步骤。

进入 **负载均衡 BLB->应用型实例**，查看当前地域下的应用型 BLB 实例。若需要新建，点击 **【创建负载均衡】** 进入创建流程。



如果创建时页面提示账号存在欠款，需先处理欠款后再继续购买和创建 BLB。



开发机只能选择同 VPC 的应用型 BLB。只要 VPC 相同，子网和可用区可按实际资源选择。如果从开发机创建或更新页面点击 **【创建 BLB】** 跳转到 BLB 创建页面，控制台会自动带入当前地域和应用型实例类型，但仍需确认 VPC 与开发机所在 VPC 一致。

创建 BLB 时建议按以下规则配置：

参数	必填	说明
类型	是	选择应用型 BLB。
地域	是	保持与开发机所在地域一致。从开发机页面跳转创建时，建议不要变更默认地域。
VPC	是	选择开发机所在 VPC。开发机所在 VPC 需要在选择资源池后确认。
弹性公网 IP	是	创建或绑定 EIP，作为公网访问入口。
计费模式	是	BLB 到期可能影响开发机服务访问。计费模式选择包年包月时，建议开启自动续费。

创建好 BLB 后，继续创建或绑定弹性公网 IP（EIP），确保该 BLB 具备公网访问入口。

步骤二：创建或更新开发机访问配置

进入 **百度百舸·AI 计算平台->大规模训练与推理->开发机**，创建开发机或进入已有开发机的变更配置页面，在 **访问配置** 区域开启 **启用自定义端口**。

1. 绑定 BLB 实例。

开启 **启用自定义端口** 后，**绑定 BLB** 为必填项。点击 **请选择 BLB** 下拉框，选择符合条件的带 EIP 的应用型 BLB 负载均衡实例。如果没有可选实例，可点击 **【创建 BLB】** 跳转创建。





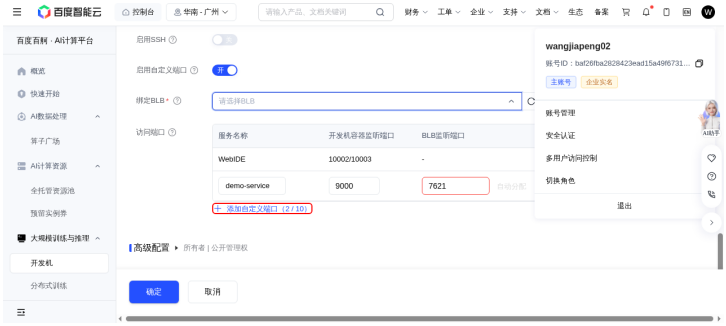
2. 配置访问端口。

点击 **【添加自定义端口】**，在访问端口表格中填写服务名称、开发机容器监听端口、BLB 监听端口，并选择传输协议。



参数	必填	说明
服务名称	是	自定义服务标识，用于区分不同访问服务。
开发机容器监听端口	是	服务在开发机实例内容器中实际监听的端口，需要与应用服务启动端口一致。
BLB 监听端口	是	面向公网访问的监听端口。BLB 监听端口只能同时被绑定一次。
传输协议	是	根据服务访问协议选择，需与服务实际协议一致。

如果已绑定 BLB，可点击 **【自动分配】** 自动获取空闲的 BLB 监听端口。



3. 完成开发机创建或更新。

确认资源队列、资源规格、BLB 绑定和访问端口等必填项均已配置后，点击 **【确定】** 提交创建或更新。若页面提示前置条件未满足，请按提示补齐后重新提交。



测试连通性

进入 **百度百舸·AI 计算平台->大规模训练与推理->开发机**，打开目标开发机实例的 Terminal。

1. 在开发机 Terminal 中执行如下命令，启动 Python 内置的 HTTP 服务。

```
echo 'Hello, World!' > index.html
python -m http.server 9000 --bind 0.0.0.0
```

2. 在开发机列表点击开发机实例名称，进入实例详情页，在 **访问配置** 中查看公网访问地址。公网访问地址格式为 `http://<EIP>:<BLB监听端口>`。

3. 在浏览器中访问公网访问地址。当页面返回 `Hello, World!` 时，表示服务访问配置成功。

常见问题

Q：配置了公网访问，但浏览器访问或外网连接失败（无法访问此网站 / Connection refused / Timeout）？

- **检查安全组规则。**
  - 确认安全组已添加入方向规则。
  - 确认端口范围包含您配置的服务监听端口。
  - 确认授权对象包含您的公网 IP 或 0.0.0.0/0。
  - 确认协议类型（TCP/UDP）与服务协议一致。
- **确认服务在运行且监听正确端口。**
  - 在开发机实例的 Terminal 内执行 `netstat -tunlp | grep <端口号>`，查看端口是否被监听。
  - 确保应用服务进程已启动，并在监听配置的端口。
- **检查 BLB 和 EIP 状态。**
  - 确认 BLB 实例和 EIP 实例状态正常。
  - 确认账号无欠费状态。
  - 确认 BLB 监听端口未被其他服务占用。
- **在开发机实例内部测试服务。**
  - 执行 `curl http://localhost:<端口号>` 或 `curl http://<实例私有IP>:<端口号>`，确认服务本身可以正常响应。

SSH远程连接开发机

概述

本文档介绍如何通过 SSH 远程连接开发机。开启开发机 SSH 服务并绑定负载均衡 BLB 后，您可以通过 SSH 安全连接到开发机，进行文件传输、代码编辑等操作。

前提条件

- 开发机处于"运行中"状态。
- 开发机已开启 SSH 服务，并绑定 BLB 和端口映射；如果使用自定义镜像，需要提前安装并启用 `sshd`。
- 如果要使用公网连接，需要开发机实例绑定带 EIP（弹性公网 IP）的负载均衡 BLB。

具体操作步骤

1. 生成 SSH 密钥对

在本地终端执行以下命令，生成 RSA 2048 位 SSH 密钥对。

```
ssh-keygen -t rsa -b 2048 -C "your_email@example.com"
```

参数说明如下：

参数	必填	说明
-t rsa	是	指定使用 RSA 算法。
-b 2048	是	指定密钥长度为 2048 位。
-C "your_email@example.com"	否	添加注释，通常填写邮箱，便于识别密钥用途。

2. 指定密钥对保存位置（可选）

执行命令后，系统会提示输入保存路径，默认位置为：

```
 ~/.ssh/id_rsa      # 私钥
~/.ssh/id_rsa.pub   # 公钥
```

若需保存到其他路径，可在提示时输入自定义路径。

3. 开启开发机 SSH 并绑定 BLB

进入 **百度百舸 · AI 计算平台->开发机->创建实例** 页面，或进入已有开发机的访问配置页面，在 **访问配置** 区域开启 SSH 并配置 BLB。

步骤 1：开启 SSH 并填写公钥

在 **访问配置** 区域，将 **启用 SSH** 开关切换为开启状态。如果开发机使用自定义镜像，需要提前确保镜像内已安装并启用 `sshd`。

开启 SSH 后，在 **公钥** 输入框中完整粘贴本机公钥文件内容。页面支持按回车换行添加多个公钥，最多支持 30 个。





步骤 2：选择或创建绑定 BLB

在 **绑定 BLB** 配置项中点击 **【请选择 BLB】** 下拉框，选择用于对外提供 SSH 访问流量的负载均衡 BLB。开发机通过负载均衡 BLB 对外提供流量访问，支持多个开发机复用一个 BLB。

若下拉列表中没有可用 BLB，可点击 **【创建 BLB】** 创建负载均衡实例。如需公网访问，应确保该 BLB 已绑定 EIP（弹性公网 IP），创建完成后返回开发机配置页重新选择。







步骤 3：配置 SSH 访问端口

开启 SSH 后，**访问端口** 表格会新增 SSH 服务行，开发机容器监听端口为 22，传输协议为 TCP。绑定 BLB 后，为 **BLB 监听端口** 配置端口：可手动输入未被占用的端口，也可点击 **【自动分配】** 生成空闲端口。

每个 BLB 最多支持映射 50 个端口，请确保配置的端口未被其他开发机或服务占用。





完成以上配置并生效后，开发机即成功开启 SSH 服务。

4. 使用 SSH 密钥访问开发机

在本地终端执行以下命令，通过 SSH 密钥访问开发机。

```
ssh -i /这里填私钥地址 username@ip_address -p port_number
```

参数说明如下：

参数	必填	说明
/这里填私钥地址	否	私钥文件路径。如果未自定义修改过私钥保存位置，可省略 -i /这里填私钥地址。
username	是	登录用户名，默认为 root。
ip_address	是	访问地址。内网访问时填写 BLB 内网 IP；公网访问时填写 BLB 绑定的 EIP（弹性公网 IP）。
port_number	是	配置的 BLB 监听端口。

通过BLB、EIP及NAT网关提升公网访问速率

概述

在百舸平台进行 AI 开发时，开发机存在需要与公网进行交互的场景。

1. 下载大型数据集、拉取 Docker 镜像、安装 pip/conda 依赖包等，需要开发机从公网下载资源的场景，依赖于 **弹性公网 IP（EIP）** 以及 **NAT 网关** 的组合。通过升级这两个组件的规格参数，可以显著提升网络带宽和并发处理能力。

2. 进行远程调试、开发机对外公开端口提供 Web 服务等，需要开发机通过端口输出数据的场景，依赖于 **负载均衡（BLB）** 以及 **弹性公网 IP（EIP）** 的组合。通过升级这两个组件的规格参数，可以显著提升网络带宽和并发处理能力。

原理

公网流量进入开发机的路径通常为：**公网请求 -> EIP -> BLB -> 开发机**。若您希望提升性能，需同步调大 EIP 的带宽和 BLB 的规格参数。

开发机下载公网资源的路径通常为：**开发机 -> 路由（NAT 网关的 SNAT 设置）-> EIP**。若您希望提升这部分性能，提升带宽请调整 EIP 带宽设置，提升并发连接数或转发能力请调整 NAT 网关的 CU 设置。

如果场景对性能有较严格的需求，可以考虑使开发机独占 EIP 资源（通过设置路由和 BLB），和其他开发机或推理服务隔离开来，追求获得最大网络性能。

前提条件

1. 公网访问开发机场景：开发机已启用自定义端口，并成功绑定支持公网的 BLB。未绑定请参考文档 开发机实例开启访问公网。
2. 开发机访问公网下载场景：已开通开发机访问公网能力。如需开通，请参考 容器实例开启访问公网。
3. 已拥有需要调整的 EIP、BLB 或 NAT 网关实例，并具备对应资源的变更权限。

操作步骤

> **注意：** 调整 EIP 带宽、BLB 性能规格或 NAT 网关性能容量可能产生费用变化，请以控制台变更配置页面和订单确认页面展示的信息为准。### 公网访问开发机

调整 EIP 带宽

调整 EIP 带宽共有两种方式。

方式一：从开发机详情进入

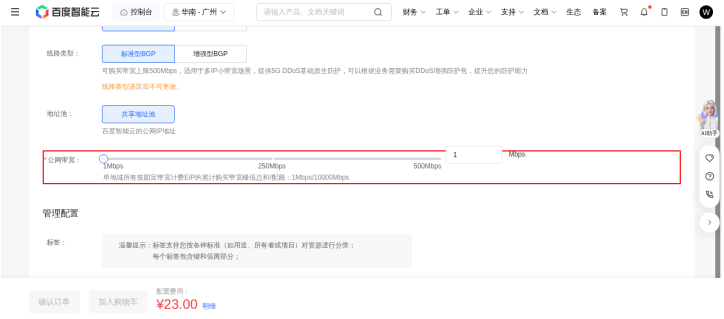
1. 进入 **百度百舸 · AI 计算平台->开发机**，点击目标开发机名称进入实例详情页。
2. 在 **访问配置** 中找到已绑定的 EIP 或 BLB 公网访问地址，点击地址进入 **弹性公网 IP** 实例详情。
3. 点击 **【调整带宽】**，进入变更配置页面。
4. 根据业务需要调整 **公网带宽**，并根据页面提示确认费用和提交变更。

方式二：从 EIP 实例列表进入

1. 进入 **弹性公网 IP->实例**，通过搜索框按实例名称、实例 ID 或公网 IP 检索目标 EIP 实例。

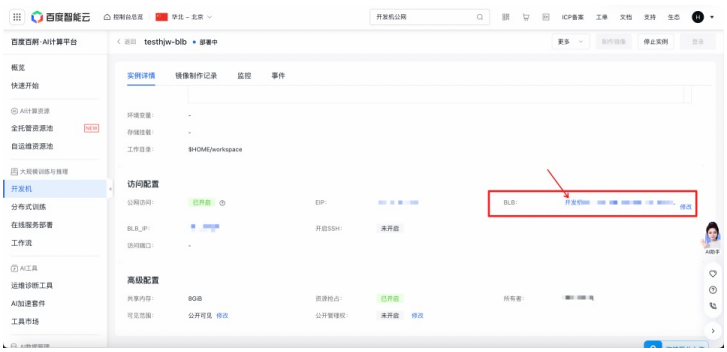


2. 在目标 EIP 实例的操作入口点击 **【调整带宽】** 或 **【带宽调整】**，进入变更配置页面。
3. 根据业务需要调整 **公网带宽**。公网带宽用于控制 EIP 可用的公网带宽上限，单位为 Mbps；调整后按页面提示确认费用并提交变更。

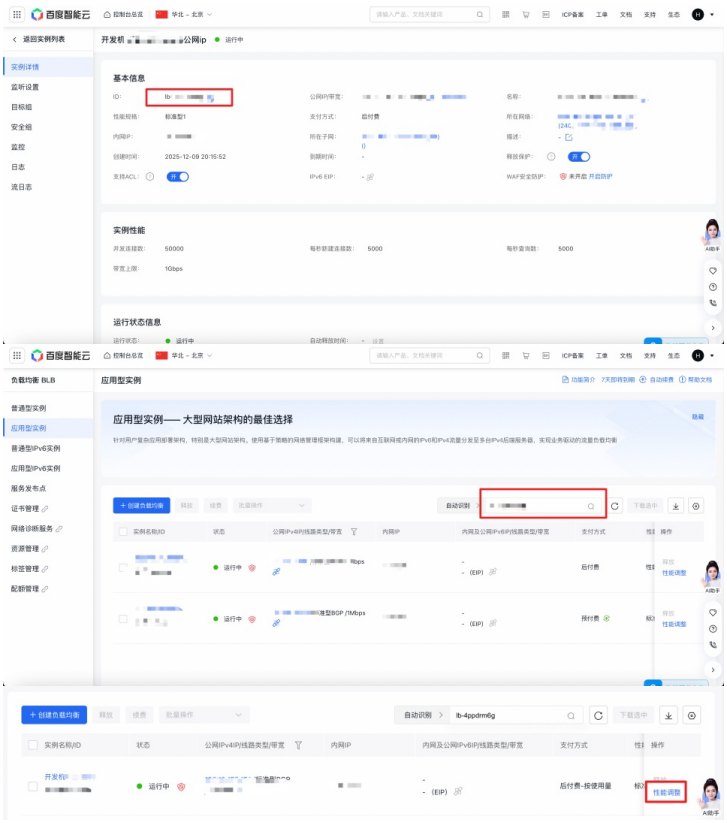


调整 BLB 性能规格

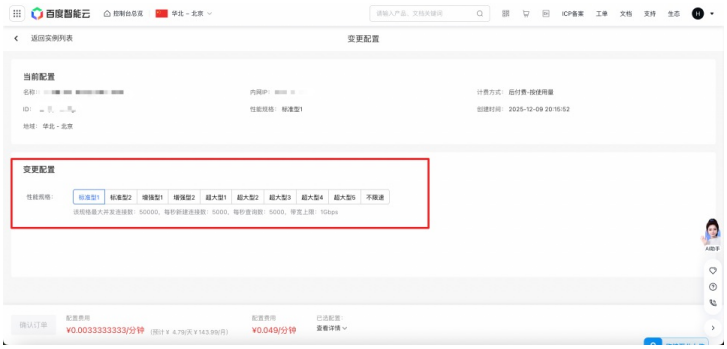
1. 打开开发机实例详情，在访问配置中找到BLB配置，点击BLB地址进入弹性公网实例详情；



2. 复制实例id，返回上级菜单，进行检索找到具体实例，点击性能调整，进入变更配置页面。



3. 根据需求变更配置。



开发机访问公网

如何为开发机开通从公网下载，请参考 容器实例开启访问公网。

调整 EIP 带宽

开发机访问公网下载时，EIP 公网带宽决定从公网下载资源的带宽上限。调整方式与"公网访问开发机"中的 EIP 带宽调整相同：进入 **弹性公网 IP->实例**，找到目标 EIP，点击 **【调整带宽】** 或 **【带宽调整】**，按需设置 **公网带宽** 并提交变更。

配置 NAT 网关

若您想提升连接数等性能，则需要调整 NAT 网关的性能容量。

新建 NAT 网关时设置性能容量

1. 进入 **私有网络 VPC->网络连接->NAT 网关**，点击 **【创建 NAT 网关】**。



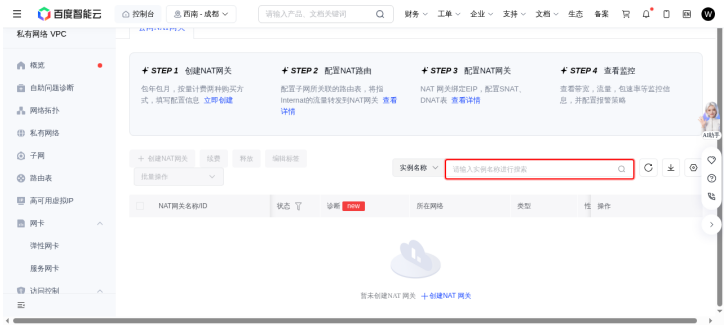
2. 在创建页面的 **配置信息** 区域，找到 **性能容量**，根据业务连接规模选择或输入 CU 数值。CU 会影响 NAT 网关的新建连接数、并发连接数和转发能力。



3. 完成其他配置后，根据页面提示确认费用并提交创建。

编辑已有 NAT 网关的性能容量

1. 进入 **私有网络 VPC->网络连接->NAT 网关**，在 **私有网络** 选择框中选择目标 NAT 网关所在的 VPC；也可以在 **实例名称** 搜索框中输入目标 NAT 网关名称进行检索。



2. 找到目标 NAT 网关后，在实例行的操作入口点击 **【更多】**，选择 **【网关升级】**。

3. 进入性能容量变配页面后，按需调整 CU 规格，并根据页面提示确认费用和提交变更。

开发机如何访问互联网

概述

本文档介绍开发机访问互联网的配置方法。在开发机使用过程中，在安装依赖、下载数据集等场景下往往需要访问互联网。开发机支持通过配置获得访问互联网的能力。

> 开发机实质上是运行在资源池上的一个容器，百舸平台不支持仅针对某个开发机开通访问互联网权限，需要在资源池维度开启容器实例访问公网。

前提条件

- 在资源池上配置 容器实例开启访问公网。

操作步骤

如果开发机已经可以正常访问互联网，可直接使用，无需继续配置。

如果开发机还不能访问互联网，则可能存在以下情况，需要根据实际情况进行对应操作：

- 已经配置了容器实例开启访问公网，但开发机不能访问公网，开发机子网未添加权限。
- 未配置容器实例开启访问公网。

步骤 1：查看资源池的子网信息

进入百度百舸控制台，选择 **AI计算资源->全托管资源池**，进入资源池列表。



已有资源池时，点击资源池名称进入详情页，在详情页查看 **容器子网** 字段，记录开发机所在子网信息。



自运维资源池同样在资源池详情页通过 **容器子网** 字段查看子网信息。如果详情页不直接显示网段，可在后续选择网段时通过子网名称匹配。

步骤 2：按子网信息配置容器实例访问公网

根据上一步获取到的子网信息，打开 容器实例开启访问公网 文档。

- 若尚未配置容器实例访问公网，按文档创建 EIP、NAT 网关、路由表和 SNAT 条目。
- 若已经配置过访问公网但开发机仍无法访问公网，重点检查 SNAT 条目中的源网段是否包含开发机所在子网，并将开发机子网补充到配置中。



存储挂载与使用

常用存储对比&选用推荐

本文旨在帮助您在不同的应用场景中，选择适合的存储产品

选用推荐 1. 云磁盘 CDS

- 接口形式：块设备接口
- 特点：高性能、低时延，支持高效随机读写；一对一挂载关系，不支持多Pod共享
- 适用场景：需要频繁更新、细粒度更新的数据、虚拟机root盘、I/O密集型业务（如文件系统、数据库等）；在百舸平台上，开发机优先推荐挂载CDS作为数据持久化方案

2. 文件存储 CFS

- 接口形式：Posix兼容接口
- 特点：层级目录结构，支持数据共享、高吞吐、高IOPS；多对一关系，支持多读多写
- 适用场景：大数据分析、媒体处理、内容管理等需要文件共享的场景

3. 并行文件存储PFS

- 特点：高性能并行文件系统，适合高性能计算场景
- 与CFS对比：PFS提供更高性能，适合计算密集型应用
- 适用场景：对训练性能敏感的场景推荐选用PFS

4. 对象存储 BOS

- 接口形式：HTTP接口
- 特点：多对一关系，支持一写多读；扁平结构，数据以对象形式存储，支持公网访问，适合海量数据存储和分发
- 适用场景：图片、视频、备份文件等读多写少的数据存储；数据分发、备份、大数据分析、音视频处理等

选用建议：

1. 需要海量数据存储和分发：选择BOS
2. 需要高性能块存储：选择CDS
3. 需要共享文件存储：选择CFS
4. 需要高性能并行文件系统：选择PFS

更多详细信息可参考：

- 存储类型对比

存储挂载和数据集挂载

开发机数据存储空间有限，当您想要扩展开发机实例存储空间、持久化存储数据、共享数据时，可以通过为开发机实例挂载数据集或直接挂载存储路径的方式实现。

并行文件存储PFS挂载与使用

并行文件存储服务PFS简介

并行文件存储服务PFS (Parallel Filesystem Service)，是百度智能云提供的完全托管、简单可扩展的并行文件存储系统，针对高性能计算场景提供TBps级吞吐、千万级IOPS能力的同时还能保证亚毫秒级的延时。同时，百度智能云PFS提供简单、易操作的接口，免去部署、维护费用的同时，最大化提升您的业务效率。PFS与对象存储BOS深度联动，提供冷热数据分级存储能力，在保持海量数据在BOS中低成本存储的同时，获得高性能文件访问能力，适用于AI训练与推理、自动驾驶、高性能计算和视频渲染等场景。

了解更多：并行文件存储服务PFS

使用前提

百舸集群已关联目标并行文件存储PFS实例。

如果没有并行文件存储PFS实例：

1. 新建并行文件存储PFS实例和挂载服务： 创建PFS -> 创建挂载服务->为挂载服务绑定已创建的PFS实例

2. 在资源池详情页面中，关联对应的挂载服务



在开发机中挂载并行文件存储PFS

步骤：创建或更新开发机->选择对应的资源池/队列->存储挂载中"添加存储挂载"->存储类型选择PFS->即可在下拉框中找到已经被集群绑定的PFS->选择目标PFS实例->填写源路径、目标路径->设置读写权限->完成开发机创建或更新

本文以下图配置为例：



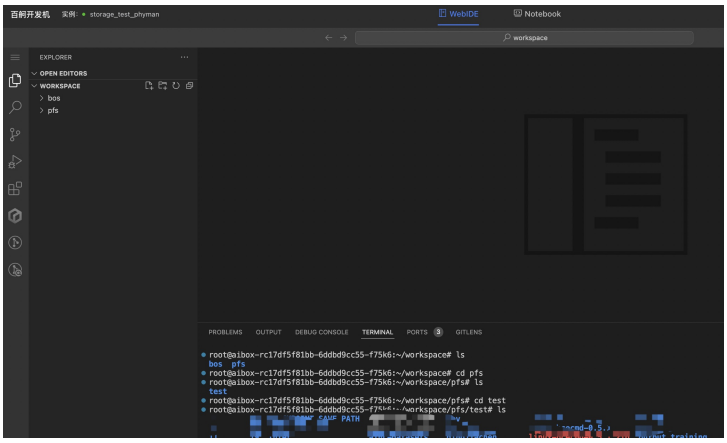
注意：

- 需要确保填写的源路径在PFS中存在，不建议源路径以根目录挂载"/"
- 目标路径可选择默认路径，或者配置到/root/workspace/**工作目录下，方便被读写
- 如果勾选"只读"，将无法在目标路径下创建和修改文件

在开发机中使用并行文件存储PFS

1. 从开发机列表页，登录对应开发机，进入webIDE页面

2. 打开terminal，进入/root/workspace/pfs/test，即可读写文件



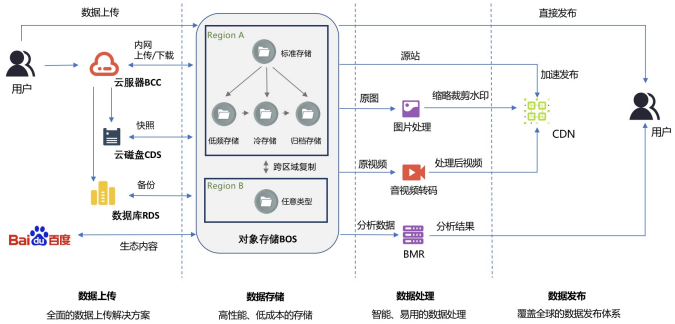
对象存储BOS挂载与使用

什么是对象存储BOS

百度对象存储 BOS(Baidu Object Storage) 提供稳定、安全、高效以及高扩展存储服务，支持单文件最大 48.8 TB 的文本、多媒体、二进制等任何类型的数据存储。

数据多地域跨集群的存储，以实现资源统一利用，降低使用难度，提高工作效率。"存储+计算框架"让您的数据加上"动力"引擎，让数据在传输、存储、处理和发布 4 个环节有机融为一体。

BOS 适用于数据分发、数据备份、大数据分析、音视频文件处理等各类场景。

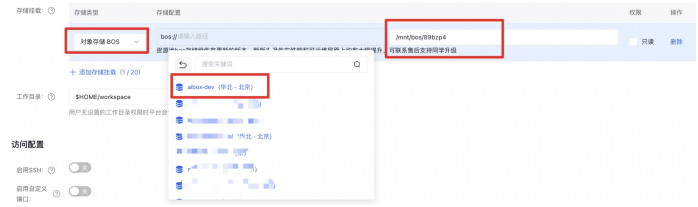


了解更多：对象存储BOS介绍

在开发机中挂载对象存储BOS

1. 进入开发机创建或更新页面

2. 在存储挂载中，选择BOS类型，选择对应的对象存储



- 存储类型选择"对象存储 BOS"->源路径：选择目标BOS、选择目标目录->目标路径：填写配置挂载目录（开发机容器内的文件路径）->设置挂载目录读写权限
  - 挂载目录默认以 /mnt/bos/ **开头，如果希望在webIDE工作目录下直接可视操作，可以改为 /mnt/workspace/bos/**
  - 挂载目录默认对当前用户有"读写"权限，如果勾选了"只读"，那么该目录不可写
  - 原则上，多个源路径之间、目标路径之间，请规避父子关系
- 如果没有现成的对象存储，可以选择 新建对象存储BOS

3. 完成开发机创建或更新，开发机进入"运行中"，对象存储 BOS 即挂载完成

在开发机中使用对象存储BOS

概述

本文旨在帮助您熟悉对象存储 BOS 在开发机中的使用。本文以验证工作目录挂载和只读权限为例，说明 BOS 挂载目录在开发机中的读写行为。

前提条件

- 已拥有百度百舸 · AI 计算平台的访问权限。
- 已创建可登录的开发机实例。
- 开发机已完成对象存储 BOS 挂载配置，且包含读写权限挂载目录和只读权限挂载目录。## 对象存储 BOS 在开发机中的使用

步骤一：登录开发机

1. 登录百度百舸 · AI 计算平台控制台，进入 **大规模训练与推理->开发机**。

2. 在开发机列表中，选择已配置 BOS 挂载的开发机，点击 **【登录】**。

3. 进入开发机后，打开 Terminal。

当 BOS 挂载目录位于 `workspace` 下时，可以直接在工作目录下进行可视化操作。

步骤二：验证挂载目录读写权限

- 进入配置为读写权限的 BOS 挂载目录，可以正常读取和写入文件。
- 进入配置为只读权限的 BOS 挂载目录，执行 `touch file.txt`。如果返回权限错误，表示该挂载目录为只读权限，符合预期。

高级功能

BOS 提供了命令行工具 BCE CMD，帮助开发者更好地在容器内使用 BOS。

BCE CMD 提供了丰富的功能，您不仅可以通过 BCE CMD 完成 Bucket 的创建和删除、Object 的批量上传、下载、删除和拷贝等；当您从 BOS 下载文件或者上传文件到 BOS 遇到问题时，还可以使用 BCE CMD 的子命令 `bosprobe` 进行错误检查。

相关功能参见 BOS 命令行工具使用指南。

将 Conda 和 Pip 包持久化存储至 PFS

概述

本文档介绍在百舸平台使用开发机时，如何通过挂载 PFS 或数据集实现 Conda 开发工具的持久化，支持开发环境快速复用与迁移。

在云上的 AI 开发环境中，您可能经常通过 `conda install` 或 `pip install` 安装 Python 依赖包。直接在容器内安装时，这些包会写入容器的可写层，导致以下问题：

- 镜像过大：提交镜像时包含大量临时文件和缓存；
- 环境不可复现：新开发机实例启动后需重新安装依赖；
- 资源浪费：重复下载相同包，浪费时间和网络开销。

前提条件

- 已具备使用百度百舸 AI 计算平台的权限，并可进入目标地域的 **百度百舸 AI 计算平台->大规模训练与推理->开发机** 和 **百度百舸 AI 计算平台->大规模训练与推理->分布式训练** 页面。
- 目标地域已存在可用资源队列、资源规格和可挂载的 PFS 存储。
- 若创建实例时无法选择 **资源队列** 或 **存储挂载**，请先完成资源池、资源队列和 PFS 存储准备后再继续。## PFS 目录结构规范

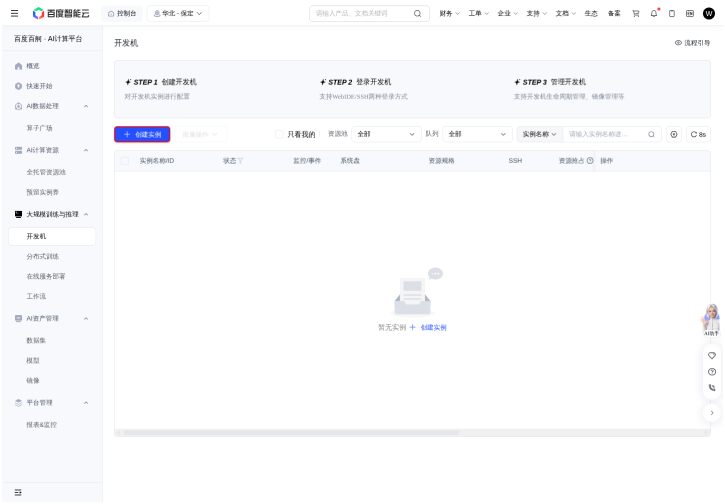
挂载目录建议采用统一规范命名并保持一致，可采用 `/mnt/pfs/<project>/<user>` 格式。

在 PFS 上安装 Conda

创建开发机

1. 登录百度百舸 AI 计算平台，进入 **百度百舸 AI 计算平台->大规模训练与推理->开发机**。

2. 单击 **【创建实例】**，进入创建实例页面。



3. 按需配置开发机参数。

参数	必填	说明
资源队列	是	选择目标地域下可用于开发机的资源队列。
资源规格	是	选择满足开发环境运行需求的资源规格。
镜像地址	是	选择与后续复用场景同架构、且包含 glibc 和 /bin/bash 的镜像。
存储挂载	是	添加 PFS 挂载，并将目标路径设置为 /mnt/pfs/project1。
访问配置	是	保留 WebIDE 服务访问配置，便于后续登录开发机。

4. 提交创建后，通过 WebIDE 登录开发机。

安装 Miniconda 到 PFS

```
###### 进入 PFS 挂载目录，创建 Conda 安装目录
cd /mnt/pfs/project1
mkdir -p user1/conda

###### 获取安装脚本
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

###### 安装
bash Miniconda3-latest-Linux-x86_64.sh -b -p /mnt/pfs/project1/user1/conda/miniconda
```

初始化 Conda

```
source /mnt/pfs/project1/user1/conda/miniconda/etc/profile.d/conda.sh
```

验证 Conda 安装

```
conda info | grep 'root prefix'
###### 期望输出：
###### root prefix : /mnt/pfs/project1/user1/conda/miniconda
```

创建开发环境

```
conda create -n py310 python=3.10 -y

###### 默认路径：
###### /mnt/pfs/project1/user1/conda/miniconda/envs/py310

###### 激活环境
conda activate py310

###### 根据需求安装环境依赖
pip install numpy -i http://mirrors.baidubce.com/pypi/simple/ --trusted-host mirrors.baidubce.com
pip install torch -i http://mirrors.baidubce.com/pypi/simple/ --trusted-host mirrors.baidubce.com
```

通过挂载 PFS 使用开发环境

创建开发机或训练任务时挂载同一 PFS，目标路径使用 `/mnt/pfs/project1`。

注意：

1. 镜像选择需要与之前操作安装的开发机镜像同架构（x86_64），且必须包含 glibc 以及 /bin/bash。

2. PFS 挂载路径需要和之前安装 Conda 时的开发机 PFS 挂载路径一致。

在开发机中使用

进入 **百度百舸 AI 计算平台->大规模训练与推理->开发机**，创建或打开已挂载 PFS 的开发机。创建开发机时，在 **存储挂载** 中选择同一 PFS，并将目标路径设置为 /mnt/pfs/project1。

通过 WebIDE 或 SSH 登录开发机，打开 Terminal 执行：

```
###### 初始化 Conda
source /mnt/pfs/project1/user1/conda/miniconda/etc/profile.d/conda.sh

###### 激活环境
conda activate py310

###### 查看之前安装的包
pip show torch
```

在训练任务中使用

进入 **百度百舸 AI 计算平台->大规模训练与推理->分布式训练**，单击 **【创建任务】**。创建训练任务时，选择与安装 Conda 时同架构且包含 glibc 和 /bin/bash 的镜像，并挂载同一 PFS，目标路径设置为 /mnt/pfs/project1。

在 **执行命令** 中填写启动命令：

```
###### 初始化 Conda
source /mnt/pfs/project1/user1/conda/miniconda/etc/profile.d/conda.sh

###### 激活环境
conda activate py310

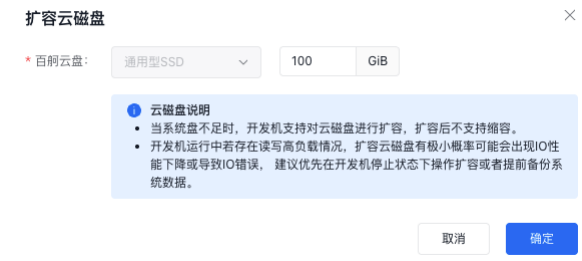
###### 后续添加训练任务命令等
###### ...
```

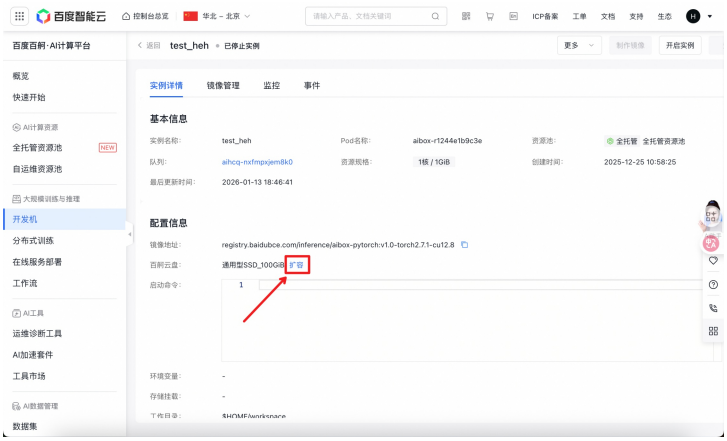
扩容系统盘使用场景说明

使用开发机实例时，平台要求至少申请20GB，推荐100GB的百舸云盘作为系统盘。如果不满足使用需求，您可以通过扩容系统盘来解决。本文为您介绍扩容系统盘的操作方法。

开发机扩容

如需要对当前开发机扩容云磁盘，可以使用扩容功能，在开发机实例详情扩容云磁盘。





注意事项

> 当系统盘不足时，开发机支持对云磁盘进行扩容，扩容后不支持缩容。 开发机运行中若存在读写高负载情况，扩容云磁盘有极小概率可能会出现IO性能下降或导致IO错误， 建议优先在开发机停止状态下操作扩容或者提前备份系统数据。

制作开发机镜像

平台提供通过控制台基于开发机制作镜像和在开发机容器内制作镜像两种方式。在开发机内制作镜像时，使用自定义镜像创建的开发机需要先安装PodMan工具，使用平台预置镜像创建的开发机已经内置PodMan无需安装。

制作镜像注意事项

- 百舸开发机和百舸提供的预置镜像都是以amd64架构为基础，若需要使用其他来源的镜像作为基础镜像或安装二进制包，请使用amd64或x64的基础镜像和二进制包；
- 请勿在镜像中捆绑GPU驱动（如libcuda.so）；
- 通过控制台制作镜像写入的数据不能超过40GiB。

基于控制台界面制作镜像制作镜像

1. 登录百舸AI计算平台，进入开发机列表页面
2. 选择相应的开发机实例，点击【制作镜像】
3. 选择需要保存的CCR镜像仓库，如列表无内容，需要先创建CCR镜像仓库，并创建用户名密码
4. 填写相关参数，保存镜像到CCR镜像仓库，等待镜像制作完成
5. 镜像制作完成后，可点击开发机实例，进入详情，查看和管理镜像列表，此处仅为列表信息，镜像实际保存在CCR仓库中

在开发机内制作镜像

使用预置镜像创建的开发机 使用预置镜像创建的开发机，目前已内置PodMan。PodMan使用命令与Docker基本一致，制作镜像相关命令示例如下：

```
##### 登录镜像仓库
podman login --username=xxxxxx ccr-2nf6531g-pub.cnc.bj.baidubce.com
##### 拉取镜像
podman pull image
##### 使用Dockerfile制作镜像,需要一个--isolation=chroot 参数
podman build --isolation=chroot -t imageName .
##### 推送镜像到远程仓库
podman push imagename
```

更多命令可参考官网文档：https://docs.podman.io/en/latest/Commands.html

自定义镜像开发机内制作镜像 不同系统版本PodMan的安装和配置方式有所差别。

Ubuntu22.04 步骤一：安装podman

```
apt-get update -y
apt-get install -y podman
```

步骤二：制作镜像

```
##### 登录镜像仓库
podman login ccr-abc12345-vpc.cnc.bj.baidubce.com --username=xxxxxx
##### 拉取镜像
podman pull ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/image:tag
##### 使用Dockerfile制作镜像（省略创建和编写Dockerfile步骤，建议将RUN指令合并以减少层数，稳定依赖（如系统包）安装在前，频繁变动的代码复制在后，以充分利用Docker构建缓存）
##### --isolation=chroot 轻量级的文件系统隔离，启动更快、资源消耗更低
##### --format docker 设置为docker模式，避免出现shell类型报错
podman build --format docker --isolation=chroot -t ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/new_image:new_tag .
##### 推送镜像到远程仓库
podman push ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/new_image:new_tag
```

Ubuntu20.04 步骤一：安装podman

```
apt-get update -y
apt-get install curl vim gnupg
echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_20.04/ /" | tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
curl -s -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_20.04/Release.key | apt-key add -
apt-get update
apt-get install -y podman
```

步骤二：修改配置文件

/etc/containers/storage.conf修改 driver="vfs"

```
# that use the containers/storage library.
# See man 5 containers-storage.conf for more information
# The "container storage" table contains all of the server options.
[storage]

# Default Storage Driver, Must be set for proper operation.
driver = "vfs"

# Temporary storage location
runroot = "/run/containers/storage"

# Primary Read/Write location of container storage
graphroot = "/var/lib/containers/storage"

# Storage path for rootless users
#
# rootless_storage_path = "$HOME/.local/share/containers/storage"

[storage.options]
# Storage options to be passed to underlying storage drivers
```

步骤三：制作镜像

```
##### 登录镜像仓库
podman login ccr-abc12345-vpc.cnc.bj.baidubce.com --username=xxxxxx
##### 拉取镜像
podman pull ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/image:tag
##### 使用Dockerfile制作镜像（省略创建和编写Dockerfile步骤，建议将RUN指令合并以减少层数，稳定依赖（如系统包）安装在前，频繁变动的代码复制在后，以充分利用Docker构建缓存）
##### --isolation=chroot 轻量级的文件系统隔离，启动更快、资源消耗更低
##### --format docker 设置为docker模式，避免出现shell类型报错
podman build --format docker --isolation=chroot -t ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/new_image:new_tag .
##### 推送镜像到远程仓库
podman push ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/new_image:new_tag
```

Centos8 步骤一：安装podman

```
##### 进入root，切换至yum.repos.d目录
cd /etc/yum.repos.d/
##### 创建新文件夹并将源文件备份为repo.bak
mkdir backup && mv *repo backup/
##### 下载国内yum源文件
curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-8.repo
##### 更新下载yum源地址
sed -i -e"s|mirrors.cloud.aliyuncs.com|mirrors.aliyun.com|g " /etc/yum.repos.d/CentOS-*
sed -i -e "s|releasever|releasever-stream|g" /etc/yum.repos.d/CentOS-*
##### 生成缓存
yum clean all && yum makecache\
yum install podman -y
```

步骤二：修改配置文件 /etc/containers/storage.conf修改 driver="vfs"

```
# See man 5 containers-storage.conf for more information
# The "container storage" table contains all of the server options.
[storage]

# Default Storage Driver, Must be set for proper operation.
driver = "vfs"

# Temporary storage location
runroot = "/run/containers/storage"

# Primary Read/Write location of container storage
# When changing the graphroot location on an SELINUX system, you must
# ensure  the labeling matches the default locations labels with the
```

步骤三：制作镜像

```
##### 登录镜像仓库
podman login ccr-abc12345-vpc.cnc.bj.baidubce.com --username=xxxxxx
##### 拉取镜像
podman pull ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/image:tag
##### 使用Dockerfile制作镜像（省略创建和编写Dockerfile步骤，建议将RUN指令合并以减少层数，稳定依赖（如系统包）安装在前，频繁变动的代码复制在后，以充分利用Docker构建缓存）
##### --isolation=chroot 轻量级的文件系统隔离，启动更快、资源消耗更低
##### --format docker 设置为docker模式，避免出现shell类型报错
podman build --format docker --isolation=chroot -t ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/new_image:new_tag .
##### 推送镜像到远程仓库
podman push ccr-abc12345-vpc.cnc.bj.baidubce.com/xyz/new_image:new_tag
```

如何减小镜像体积 如果制作的镜像体积过大，可以通过以下方式多阶段构建或清理apt缓存、临时文件

```
##### 多阶段构建例子，本质上就是基础环境单独生成一个镜像，然后从这个镜像里面只拿走我们需要的部分
##### 当前只是显示格式的范例，直接使用python:3.9-slim这样的官网镜像在开发机内无法下载，建议以国内的源或者使用推荐的百度CCR源为基础

##### ========== 第一阶段：构建阶段 ==========
##### 这个阶段可以很"胖"，因为它不会进入最终镜像
FROM python:3.9-slim AS builder
WORKDIR /build

##### 在这个阶段，放心地安装所有构建工具
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc g++ git

##### 将依赖文件单独复制，以利用Docker缓存
COPY requirements.txt .
##### 安装依赖（会下载whl包，产生缓存，所以要在后面追加rm -rf ~/.cache/pip/*命令清理缓存，不过这只是范例，因为当前是构建阶段，留下缓存也不会有问题）
RUN pip install --user --no-cache-dir -r requirements.txt && \
    rm -rf ~/.cache/pip/*
##### 假设requirements.txt包含: torch, torchvision, numpy

##### 复制应用代码
COPY app.py .

##### ========== 第二阶段：运行阶段 ==========
##### 这个阶段必须非常"瘦"，它才是最终发布的镜像
FROM python:3.9-slim
WORKDIR /app

##### 从"构建阶段"只复制安装好的依赖成品和应用程序
COPY --from=builder /root/.local /root/.local
COPY --from=builder /build/app.py .

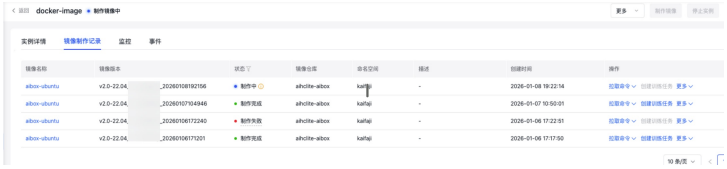
##### 确保PATH能找到我们复制的本地依赖
ENV PATH=/root/.local/bin:$PATH

#####  （可选）安装最小的运行时系统依赖
RUN apt-get update && apt-get install -y --no-install-recommends \ # 使用 --no-install-recommends：在 apt-get install 时加上此参数，可以避免安装非严格依赖的"推荐"包，进一步减少不必要的内容。
    # 例如PyTorch可能需要的libgomp1
    libgomp1 && \
    apt-get clean && \ # 删除已下载的 .deb 安装包文件
    rm -rf /var/lib/apt/lists/* && \ # 删除 apt-get update 后下载的软件包元数据列表文件
    rm -rf /tmp/* /var/tmp/* # 删除临时目录下的文件。一些安装脚本或解压操作会在这里留下残留
##### 合并RUN指令：务必像示例一样，将 update、install、clean 放在同一条 RUN 指令中（用 && 连接）。这样做是为了让清理操作发生在同一个Docker镜像层里，如果分开成多条 RUN，即使清理了，被删除的文件在之前的镜像层中仍然存在，无法真正减小镜像体积。

CMD ["python", "app.py"]
```

查看制作镜像记录

制作镜像过程较长，可以在开发机详情中查看制作镜像的进度，平台也保存了制作失败的镜像记录，可以查看制作失败的原因。



模型开发环境

管理第三方库

概述

在开发机实例中，您可以通过 Terminal 直接安装、查看、卸载、更新 Python 第三方库。变更会保存在开发机实例的磁盘中；若创建时挂载了云磁盘或百舸云盘，停止实例后数据会保留。

前提条件

- 已进入目标开发机实例的 Terminal。
- 开发机环境中已安装 Python3、pip 或 pip3。
- 若预置镜像提供多个 conda 环境，请先激活目标环境。## 安装第三方库

开发机提供的开发环境包括 Python3、pip 和 apt。安装第三方库时，默认安装至当前 Python 环境；若预置镜像提供多个 conda 环境，需先激活目标环境再安装。具体命令格式如下所示。

```
###### 在当前 Python 环境安装（默认）。
pip3 install <yourLibraryName>

###### 若使用 conda 多环境，先激活目标环境再安装，例如：
###### conda activate <环境名>
###### pip3 install <yourLibraryName>
```

需要将 `<yourLibraryName>` 替换为待安装的第三方库名称。例如，使用 `pip3 install bottle` 命令，安装 `bottle` 库。

如果您在 Jupyter Notebook 的 Cell 中执行安装命令，请在命令前加上英文感叹号 `!`，例如：`!pip3 install bottle`。

重要

如果成功安装了三方包但是在 Notebook 中依然找不到，请尝试重新加载 Kernel。

查看第三方库

使用以下命令查看已安装的第三方库。

```
pip3 list
```

卸载第三方库

使用以下命令卸载已安装的第三方库。

```
pip3 uninstall <yourLibraryName>
```

需要将 `<yourLibraryName>` 替换为已安装的第三方库名称。

如需跳过交互确认，可添加 `-y` 参数。

```
pip3 uninstall -y <yourLibraryName>
```**说明**

只能卸载自己安装的第三方库。

更新第三方库

使用以下命令更新已安装的第三方库。

```bash
pip3 install --upgrade numpy==<versionNumber>
```

需要将 `<versionNumber>` 替换为待安装的 `numpy` 版本号。

更新完成后，可执行以下命令验证版本。

```
python3 - <<'PY'
import numpy
print(numpy.__version__)
print(numpy.__file__)
PY
```## 查看或更改 pip 源

查看 pip 源执行如下命令查看 pip 配置文件。

```bash
pip3 config list
```

返回结果中 `global.index-url` 参数的值即为全局 pip 源地址。返回示例如下：

```
global.index-url='http://mirrors.baidubce.com/pypi/simple/'
global.trusted-host='mirrors.baidubce.com'
install.trusted-host='mirrors.baidubce.com'
```

安装第三方库时临时设置 pip 源

安装第三方库时，可以通过 `-i` 参数临时指定 pip 源。本方式只对当前安装命令生效，不会修改全局 pip 配置。

```
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple <yourLibraryName>

###### 如果出现 SSL 验证失败的错误，请尝试加上 --trusted-host 参数，示例如下：
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple --trusted-host pypi.tuna.tsinghua.edu.cn <yourLibraryName>
```

需要将 <yourLibraryName> 替换为待安装的第三方库名称。

如果您在 Jupyter Notebook 的 Cell 中执行安装命令，请在命令前加上英文感叹号！，例如：!pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple bottle。### 更改全局 pip 源

更改全局 pip 源。pip 配置文件默认路径为 ~/.config/pip/pip.conf，执行如下命令编辑配置文件。

```
vim ~/.config/pip/pip.conf
```

以修改为清华 pip 源为例，修改 index-url 和 trusted-host 参数，然后保存并退出。修改后的 pip.conf 如下：

```
[global]
trusted-host=pypi.tuna.tsinghua.edu.cn
index-url = https://pypi.tuna.tsinghua.edu.cn/simple/

[install]
trusted-host=pypi.tuna.tsinghua.edu.cn
```

开发机支持R语言

开发机实例集成了开源JupyterLab，您可以在开发机实例中安装R Kernel，即可在Notebook中运行R脚本来做数据分析。本文为您介绍如何在开发机实例中安装R Kernel。

前提条件

已创建开发机实例，详情请参见创建开发机实例。

操作步骤

步骤一：进入开发机开发环境

1. 登录百舸控制台，在左侧导航栏，选择**模型开发与训练 > 开发机**。

2. 单击需要打开的实例操作列下的**登录**。

3. 在开发机实例页面，进入Terminal，完成后续操作。

步骤二：安装R语言

执行如下命令，使用Ubuntu的包管理工具apt安装R语言环境。

> **提示**：代码执行过程中，当遇到 Proceed（[y]/n） 时，请输入 y 并按回车。

```
apt update
apt install r-base
```

步骤三：安装IRkernel

安装IRkernel，即R语言的Jupyter Kernel。启动R环境（在终端中输入 R 并回车）后，执行以下命令：

```
install.packages('IRkernel')
IRkernel::installspec()
```

安装完成后退出R环境。

```
q()
```

步骤四：验证安装结果

返回Launcher页面并刷新页面，查看R环境是否已安装完成。

在Notebook区域下新打开一个Launcher页，即可看到R语言的kernel选项。选择**R**，您就可以新建一个Notebook，开始使用R语言在JupyterLab里工作。

示例：用R语言做数据分析

依次执行如下命令，创建一个R Kernel的Notebook文件，使用内置的mtcars数据集，计算每种车型的平均马力（hp），并使用ggplot2包绘制马力与里程（mpg）之间的关系图。

```
install.packages("ggplot2")
```

```
###### 加载所需的包
library(ggplot2)

###### 使用mtcars数据集
data(mtcars)

###### 打印前几行数据进行查看
head(mtcars)

###### 计算每种车型的平均马力
average_hp_by_cyl <- aggregate(mtcars$hp, by=list(mtcars$cyl), FUN=mean)
colnames(average_hp_by_cyl) <- c("cylinders", "average_hp")

###### 打印出平均马力
print(average_hp_by_cyl)

###### 使用ggplot2绘制马力与里程的关系图
ggplot(mtcars, aes(x=mpg, y=hp)) +
  geom_point() +        # 添加点
  geom_smooth(method="lm") +  # 添加线性回归拟合线
  labs(title="马力与里程之间的关系", x="每加仑里程 (mpg)", y="马力 (hp)")  # 添加标题和轴标签
```

在开发机中使用Tensorboard

概述

本文介绍如何在百舸开发机中安装并使用 Tensorboard 工具。通过配置自定义端口，可将开发机内容器中的 Tensorboard 服务暴露到公网访问入口，并通过浏览器查看训练日志、标量、直方图、图像、文本、计算图等可视化内容。

前提条件

- 已登录百度智能云控制台。
- 已进入 **百度百舸·AI 计算平台->大规模训练与推理->开发机** 页面。
- 已准备可用的百舸开发机环境，且开发机中具备 Python 和 pip 环境。
- 如需公网访问 Tensorboard，需具备可绑定的 BLB；若当前无可选 BLB，可先创建 BLB。## 环境准备

端口配置

为了确保能够在公网环境访问Tensorboard服务，需要配置自定义端口，将服务暴露到公网：



Tensorboard 安装

通过 `pip install` 安装 Tensorboard。

```
pip install tensorboard
```

Tensorboard 日志准备

首先需要确保 Tensorboard 日志已经存在于开发机目录中。若无可用日志，可通过如下脚本快速生成。该示例脚本通过随机数据生成包括标量、直方图、图像、文本、计算图、PR 曲线和嵌入数据等多种数据类型的 Tensorboard 日志。

```
import torch
import torch.nn as nn
import numpy as np
from torch.utils.tensorboard import SummaryWriter
import datetime

###### Set random seeds
torch.manual_seed(42)
np.random.seed(42)

###### Create TensorBoard writer
log_dir = "logs/random_demo/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
writer = SummaryWriter(log_dir)

###### 1. Scalar data - Simulating training metrics
for step in range(100):
    # Simulate loss decreasing
    loss = 2.0 * np.exp(-step / 20) + np.random.rand() * 0.2
    # Simulate accuracy increasing
    accuracy = 0.3 + 0.7 * (1 - np.exp(-step / 30)) + np.random.rand() * 0.05

    writer.add_scalar('Loss/train', loss, step)
    writer.add_scalar('Accuracy/train', accuracy, step)

    # Add more scalars
    writer.add_scalar('Learning_rate', 0.001 * np.exp(-step / 50), step)
    writer.add_scalar('Batch_time', 0.1 + np.random.rand() * 0.05, step)

###### 2. Histogram data - Simulating weight distributions
for step in range(0, 100, 20):
    # Simulate neural network weights
    weights = torch.randn(1000) * (1.0 - step/100)
    biases = torch.randn(100) * 0.1
    gradients = torch.randn(500) * 0.01

    writer.add_histogram('weights/layer1', weights, step)
    writer.add_histogram('biases/layer1', biases, step)
```

```
    writer.add_histogram('gradients', gradients, step)

###### 3. Image data - Random images
for step in range(0, 100, 25):
    # Generate random RGB images (3, 32, 32)
    random_images = torch.rand(8, 3, 32, 32)  # 8 images, 3 channels, 32x32
    writer.add_images('Random_images', random_images, step)

    # Generate random grayscale images
    gray_images = torch.rand(8, 1, 28, 28)
    writer.add_images('MNIST_like_images', gray_images, step)

###### 4. Text data - Example text
sample_text = """
This is a TensorBoard demonstration log.
Generated using PyTorch with random data.
Includes various data types: scalars, histograms, images, text, etc.
"""
writer.add_text('Experiment_Info', sample_text, 0)
writer.add_text('Hyperparameters',
                f"Batch size: 64\nLearning rate: 0.001\nOptimizer: Adam", 0)

###### 5. Computation graph - Simple model structure
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.fc1 = nn.Linear(10, 20)
        self.fc2 = nn.Linear(20, 5)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = SimpleModel()
dummy_input = torch.rand(1, 10)
writer.add_graph(model, dummy_input)

###### 6. PR curve (Precision-Recall Curve)
num_samples = 1000
probabilities = torch.rand(num_samples)
labels = torch.randint(0, 2, (num_samples,)).float()
writer.add_pr_curve('Binary_Classification', labels, probabilities, 0)

###### 7. Embedding data - High-dimensional data visualization
num_embeddings = 100
embedding_dim = 16
features = torch.randn(num_embeddings, embedding_dim)
metadata = [f'class_{i%5}' for i in range(num_embeddings)]

writer.add_embedding(features, metadata=metadata,
                     tag='Random_embeddings', global_step=0)

###### 8. Audio data - Random audio
try:
    sample_rate = 44100
    audio_data = torch.rand(1, sample_rate) * 2 - 1  # Normalize to [-1, 1]
    writer.add_audio('Random_audio', audio_data, 0, sample_rate=sample_rate)

    t = torch.linspace(0, 1, sample_rate)
    sine_wave = 0.3 * torch.sin(2 * np.pi * 440 * t)  # 440Hz A tone
    writer.add_audio('Sine_wave_440Hz', sine_wave.unsqueeze(0), 0, sample_rate=sample_rate)
except Exception as e:
    print(f"Audio generation skipped: {e}")

###### 9. Scalar groups
for step in range(50):
    writer.add_scalars('Losses', {
        'train_loss': 1.0 / (step + 1),
        'val_loss': 1.5 / (step + 1),
        'test_loss': 2.0 / (step + 1)
    }, step)

    writer.add_scalars('Accuracies', {
        'train_acc': min(0.95, 0.5 + step * 0.01),
        'val_acc': min(0.92, 0.5 + step * 0.009),
        'test_acc': min(0.90, 0.5 + step * 0.008)
    }, step)

###### 10. Custom scalar layout
layout = {
    'Training': {
        'Loss': ['Multiline', ['Loss/train', 'Loss/validation']],
        'Accuracy': ['Multiline', ['Accuracy/train', 'Accuracy/validation']]
    },
    'Distributions': {
        'Weights': ['Multiline', ['weights/layer1']]
    }
}
writer.add_custom_scalars(layout)

###### Close the writer
writer.close()

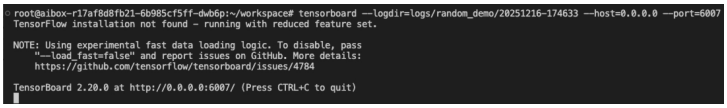
print(f"TensorBoard logs generated to: {log_dir}")
```

启动 Tensorboard

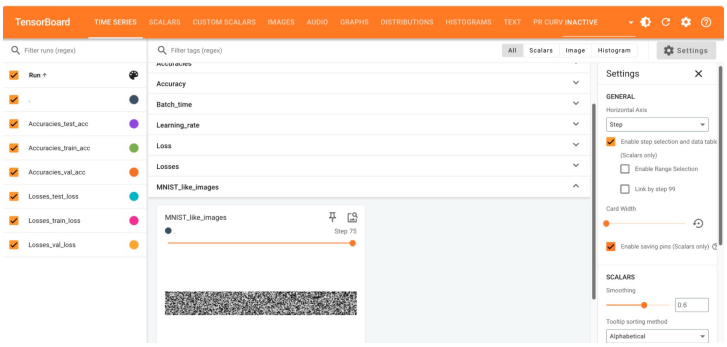
在已准备 Tensorboard 日志的开发机环境中，执行如下命令启动 Tensorboard 服务，并指定日志路径、监听地址和服务内部端口。

Plain Text 复制

```
tensorboard --logdir={log_folder} --host=0.0.0.0 --port={tensorboard服务内部端口}
```



启动成功后，在浏览器访问 http://{开发机公网 IP}:{BLB 监听端口}/ 即可打开 Tensorboard 页面。



开发机监控

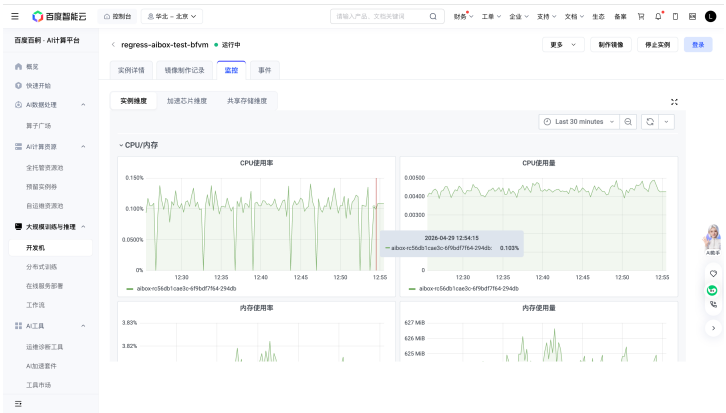
开发机实例支持查看和监控资源使用情况，提供多个维度的加速芯片、CPU/内存、网络、磁盘等详细监控指标，帮助您实时掌握实例负载状态。

使用百舸控制台查看监控页面

在开发机实例列表页单击实例名称进入详情页，单击切换至监控页签。在此监控页面您可查看实例的实例维度、加速芯片维度、共享存储维度的详细监控指标。

> 各维度的监控指标项不同，方便用户在不同维度视角结合运行的程序感知开发机的状态。

实例维度



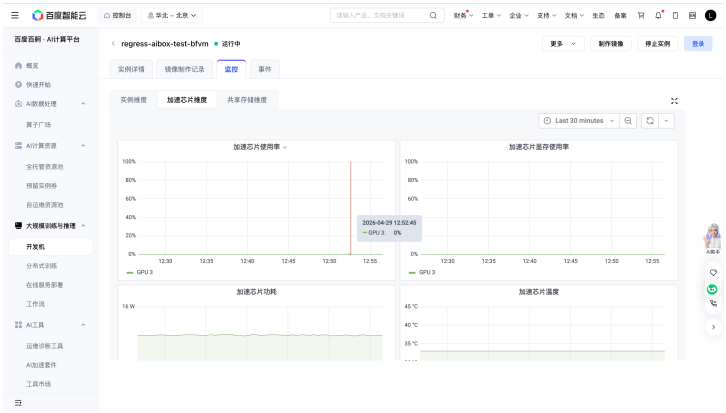
监控指标项：

- 加速芯片
  - 加速芯片使用率
  - 加速芯片显存使用率
  - 加速芯片功耗
  - 加速芯片温度
  - 加速芯片SM设备使用率
  - 加速芯片Tensor pipe使用率
  - PCIE发送数据速率
  - PCIE接收数据速率
  - NVLink接收数据速率
  - NVLink发送数据速率
- CPU/内存
  - CPU使用率
  - CPU使用量
  - 内存使用率
  - 内存使用量
  - CPU使用率（节点维度）
  - 内存使用率（节点维度）
- 网络

- 主网卡健康状态
- 主网卡接收端丢包速率
- 主网卡发送端丢包速率
- 主网卡发送数据速率
- 主网卡接收数据速率
- 主网卡发送包速率
- 主网卡接收包速率

- 磁盘
  - 磁盘设备读/写速率
  - 磁盘设备读/写IO速率

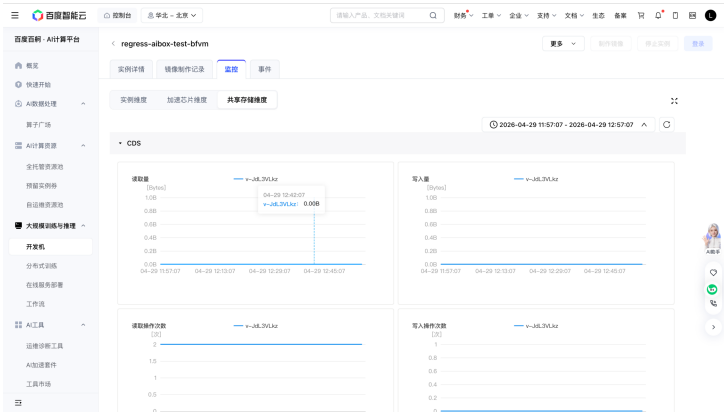
加速芯片维度



监控指标项：

- 加速芯片使用率
- 加速芯片显存使用率
- 加速芯片功耗
- 加速芯片温度
- 加速芯片SM设备使用率
- 加速芯片Tensor pipe 使用率
- PCIE传输数据的速率
- PCIE接收数据的速率
- NVLink发送数据速率
- NVLink接收数据速率

共享存储维度



监控指标项：

- CDS
  - 读取量
  - 写入量
  - 读操作次数
  - 写操作次数
  - 磁盘使用率

在开发机中将模型部署为在线服务

概述

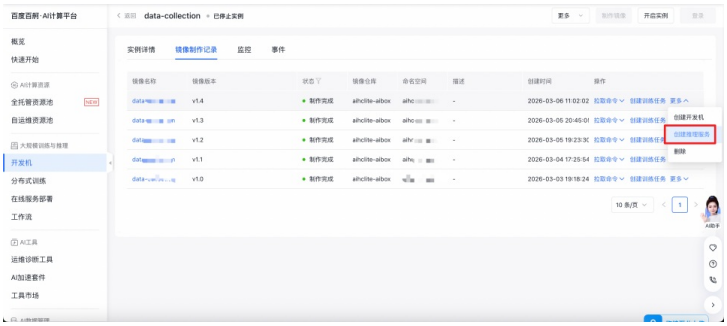
本文档介绍如何将开发机中调试完成的模型环境直接封装为镜像，并快速部署为在线推理服务（Inference Service）。通过此流程，无需复杂的 Docker 构建过程，即可将实验环境转化为可对外提供 API 调用的生产服务。

前提条件

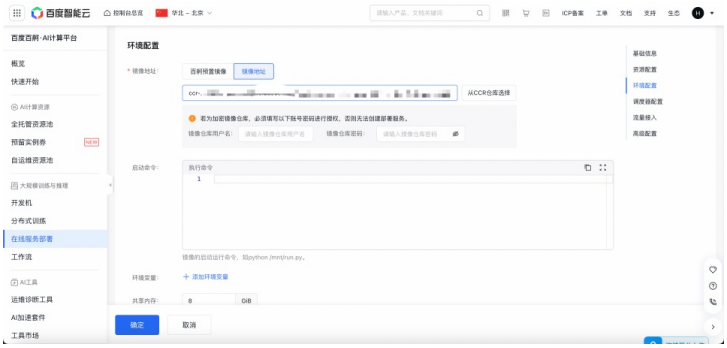
- 已创建开发机实例，并在其中完成模型代码调试。
- 模型文件和推理代码（如 `app.py`）已保存在开发机环境中。
- 推理代码已配置监听端口（如 8080 或 8000）。

操作步骤

1. 在控制台进入 **百度百舸 · AI 计算平台->大规模训练与推理->开发机**。2. 选择已完成调试的开发机实例，在开发机中制作镜像。详情参见 开发机中制作镜像。
2. 镜像制作完成后，点击开发机实例名称，进入实例详情页面。
3. 选择 **【镜像制作记录】** 标签页，在列表中找到需要部署的镜像版本。



3. 创建推理服务：在“镜像制作记录”列表中找到需要部署的镜像版本，在右侧“操作”列中，点击 “更多” 下拉菜单，选择 “创建推理服务” 选项。
4. 填写服务配置信息。配置项可参考 在线服务部署。从镜像制作记录发起创建时，无需填写 **镜像地址**；如需更换镜像，可点击 **【百舸预置镜像】**，或点击 **【百舸预置镜像】** 后切回 **镜像地址** 以刷新镜像信息。



5. 配置完成后，点击确定提交，即可成功在开发机中将模型部署为在线服务。

开发机拉取海外镜像

背景说明

在使用百度百舸（AI计算平台）创建开发机或训练任务时，如果您填写的容器镜像地址（如 docker.io）或模型下载地址（如 huggingface.co）位于海外，可能会受限于跨国网络环境（防火墙、带宽延迟等），导致以下问题：

- 拉取超时/失败：实例长期处于“创建中”或直接报错。
- 下载速度极慢：严重影响开发与部署效率。

平台处理机制

目前百舸平台对网络访问的处理逻辑如下：

- 无需 VPN 的资源：如果目标镜像/模型地址在国内网络环境下可直连访问，平台将直接进行拉取。
- 需 VPN/受限资源：对于被防火墙阻断或需要特殊网络环境才能访问的海外资源，平台暂不支持自动加速拉取。此类情况建议通过“中转上传”的方式解决。
  - 针对无法拉取的海外镜像（如 gcr.io、部分 docker.io 镜像），建议您先将其下载到本地，或上传至百度智能云容器镜像服务 (CCR)，再从百舸平台内进行拉取。

操作步骤

1. 准备镜像（在您的本地环境/中转机）
2. 支持上传至百度智能云 CCR镜像仓库（参见外部镜像迁移至企业版实例。），或直接跳转第3步。
3. 在百舸中选用镜像



- 对于上传至百度智能云 CCR镜像仓库的镜像，则直接从CCR仓库选择即可。

- 对于未上传至CCR镜像仓库的镜像，直接填写镜像地址。

备注：若为加密镜像仓库，必须填写账号密码进行授权，否则无法创建开发机，并且，因为在关机后再开机、变更配置、打包docker镜像等多个场景下，都会需要使用账号密码重新拉取镜像，所以请尽量使用长久有效的账号密码，不然某些操作可能出现报错

在开发机中安装工具引导

功能概述

欢迎使用百舸开发机，开发机镜像已预装了常用工具，但由于开发场景不同，若遇到某个软件没有安装您可以通过文档快速补齐所需环境。

前置条件

1. 开发机镜像

平台提供的开发机镜像（比如 Ubuntu 22.04 / 20.04）已经配置 apt 源、网络和基础环境，文档中的命令可以直接运行。 若您使用自定义镜像，自定义镜像可能裁剪了某些包、修改了 apt 源，导致 apt install 失败，可能是由于兼容性问题或配置差异导致，如果遇到这种情况，可以联系百舸技术支持人员。

2. 开发机开启公网访问

apt install使用的是百度内部镜像，不需要公网，但nodejs、python、java等语言的依赖库，依然需要从公网下载，所以绝大部分情况下，开发机需要能够访问公网。
一个最简单的检查方法：在开发机终端执行

```
ping www.baidu.com -c 4
```

如果看到类似 64 bytes from ... 的回复，说明网络是通畅；如果超时或报错，请检查 VPC 的路由表、安全组/防火墙规则，开放公网出方向。

> **小提示：** 百舸全托管资源池的 VPC 默认已开通公网出方向，一般不需要额外配置。

3. 更新apt 平台默认只设置了apt镜像地址，还需要执行一下apt update命令，镜像地址才会生效，创建开发机之后请先执行一下apt update，以保证后续的apt install命令运行正常。

安装常用软件概览

类型	提供软件
编程语言	Python3.12及pip、uv包管理 Nodejs20安装和npm、yarn包管理 Golang1.22安装和go mod包管理 Rust1.77安装和cargo包管理 Java21安装和Maven、Gradle包管理
数据库软件	SQLite PostgreSQL客户端 Redis客户端
系统命令和基础软件	常用命令包

编程软件及依赖库

Python Python推荐版本 开发机启动后会自动安装miniforge，相当于python3.12版本，不需要额外安装 **Python包管理器pip** pip会跟随python3.12自动安装，不需要额外安装 包管理的镜像也会使用国内镜像，不需要修改 **Python包管理器uv**

uv安装 使用国内镜像安装最新版uv（uv版本独立于python版本）

```
curl -sL https://mirrors.ustc.edu.cn/github-release/astral-sh/uv/LatestRelease/uv-installer.sh | sh

##### 安装后验证
uv --version
```

> **注意：** uv 不直接依赖 Miniforge，但可在 Miniforge 创建的 Conda 环境中使用。关键在于确保 uv 使用的是 Conda 环境中的 Python。百舸托管 VPC 默认已开通公网出方向，一般不需要额外配置。

方法 1：在 Conda 环境中安装 uv（推荐）

```
##### 创建并激活 Conda 环境
conda create -n myproject python=3.11 -y
conda activate myproject

##### 在此环境中安装 uv（确保使用环境内 Python）
pip install uv  # 或使用 conda install -c conda-forge uv（如有）

##### 初始化项目（将在当前环境创建 .venv）
cd myproject/
uv init
```

> ⚠ **注意：** 此时 uv 会基于 myproject 环境中的 Python 创建虚拟环境，不会使用系统或 base 环境的包。

方法 2：让 uv 使用 Miniforge 的 base Python
若希望 uv 直接使用 Miniforge 的 base Python（不创建新虚拟环境），可

```
##### 激活 base 环境
conda activate base

##### 在当前目录初始化项目，并复用 base 环境的 Python
uv init
uv venv --system-site-packages  # 允许访问 base 环境的包
```

> **注意：** 此方式适合快速开发，但牺牲了环境隔离性，不推荐用于生产或团队协作。

uv国内镜像设置 将下面的命令添加到 Shell 配置文件中（例如 ~/.bashrc, ~/.zshrc）：

```
##### 设置 PyPI 默认镜像源
export UV_DEFAULT_INDEX="https://pypi.tuna.tsinghua.edu.cn/simple"
##### 设置 Python 解释器下载镜像
export UV_PYTHON_INSTALL_MIRROR="https://cnb.cool/astral-sh/python-build-standalone/-/releases/download/"
```

之后执行 `source ~/.bashrc` 使其生效。

Nodejs Nodejs推荐版本 建议安装版本为Nodejs20，请使用下面的脚本，用官方deb包版本替换apt源然后安装

```
##### 安装 Node.js 20.x
curl -fsSL https://deb.nodesource.com/setup_20.x | bash
apt install -y nodejs
```

Nodejs包管理器npm npm安装 上面安装nodejs20时，会自动安装npm10.8.2，请参照上面的Nodejs20安装步骤

npm国内镜像配置 npm在国内可直接访问，但国内有些场景访问速度会比较慢，如果出现超时或者访问缓慢，可尝试通过下面的方式设置国内镜像

```
##### 命令行配置
npm config set registry https://registry.npmmirror.com

##### 查看是否配置成功
npm config get registry
##### 预期输出：https://registry.npmmirror.com
```

Nodejs包管理器yarn yarn安装

```
##### yarn需要使用npm安装，所以需要先完成【npm安装】
npm install --global yarn
```

yarn国内镜像配置 yarn在国内可直接访问，但国内有些场景访问速度会比较慢，如果出现

```
##### 查看当前配置
yarn config get registry

##### 设置为国内镜像源
yarn config set registry https://registry.npmmirror.com

##### 验证配置生效
yarn config list | grep registry
##### 预期输出：registry: 'https://registry.npmmirror.com/'
```

Golang Golang推荐版本 golang推荐版本为1.22，可通过下面的命令安装，apt提供从1.12到1.24的版本

```
apt instal -y golang-1.22-go

##### 安装完成后，需要将go命令所在文件夹设置进PATH，才能直接调用go命令
echo 'export PATH=/usr/lib/go-1.22/bin:$PATH' >> ~/.bashrc
echo 'export GOROOT=/usr/lib/go-1.22' >> ~/.bashrc

##### 执行上面的命令后，开启新的bash即可使用go命令，当前窗口还得执行下面的source才能起效
source ~/.bashrc
```

Golang包管理器gomodules gomodules安装 gomodules的使用命令为go mod（例如go mod tidy），golang安装完成就可用，不需额外安装 **gomodules国内镜像配置**

```
##### 安装golang成功后

##### 启用 Go Modules (Go 1.11+ 通常默认开启)
go env -w GO111MODULE=on

##### 配置 GOPROXY 为国内加速地址
go env -w GOPROXY=https://goproxy.cn,direct

##### 配置私有仓库代理（可选）
##### go env -w GOPRIVATE=git.mycompany.com,*.example.com
```

Rust Rust推荐版本 Rust推荐版本为1.77，可通过下面的命令安装，apt提供多个版本，可通过apt search自行查看：

```
apt install -y rustc-1.77

##### 安装完成后，需要将rustc命令所在文件夹设置进PATH，才能直接调用rustc命令
echo 'export PATH=/usr/lib/rust-1.77/bin/:$PATH' >> ~/.bashrc

##### 执行上面的命令后，开启新的bash即可使用rustc命令，当前窗口还得执行下面的source才能起效
source ~/.bashrc
```

Rust包管理器cargo cargo安装 cargo安装也使用推荐版本1.77，可通过下面的命令安装，apt提供多个版本，可通过apt search自行查看：

```
apt install -y cargo-1.77

##### cargo的所在路径和同版本rustc相同，如果之前有安装过rustc，就可以跳过这一步了
##### echo 'export PATH=/usr/lib/rust-1.77/bin/:$PATH' >> ~/.bashrc

##### 执行上面的命令后，开启新的bash即可使用rustc命令，当前窗口还得执行下面的source才能起效
##### source ~/.bashrc
```

cargo国内镜像配置

```
##### 以下内容写入 ~/.cargo/config.toml
[source.crates-io]
replace-with = "tuna"

[source.tuna]
registry = "sparse+https://mirrors.tuna.tsinghua.edu.cn/crates.io-index/"
```

Java Java推荐版本 Java推荐版本为21，可通过下面的命令安装，apt提供21，11、17、18、21、25、8版本

```
apt install -y openjdk-21-jdk
```

Java包管理器Maven Maven安装

```
cd /tmp
wget https://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz
tar -zxvf apache-maven-3.9.9-bin.tar.gz -C /opt
ln -s /opt/apache-maven-3.9.9 /opt/maven
echo 'export MAVEN_HOME=/opt/maven' > /etc/profile.d/maven.sh
echo 'export PATH=${MAVEN_HOME}/bin:${PATH}' >> /etc/profile.d/maven.sh
source /etc/profile.d/maven.sh

##### 最后使用下面的命令查看安装结果
mvn -version
```

Maven全局镜像配置 最推荐的配置方式是修改Maven的全局配置文件 `settings.xml` ，这样对所有项目都能生效。

1.找到配置文件

`~/.m2/settings.xml`如果文件或目录不存在，请手动创建 `.m2` 文件夹和`settings.xml` 文件。

2.编辑配置文件

在 `settings.xml` 文件的 `<mirrors>`标签内，添加清华源作为镜像：

```
<!-- 这些并不是完整的settings.xml内容，只是要添加在<mirrors>标签内的部分 -->
<mirror>
  <id>tuna</id>
  <name>TUNA Maven Mirror</name>
  <url>https://mirrors.tuna.tsinghua.edu.cn/maven-central/</url>
  <mirrorOf>central</mirrorOf>
</mirror>
```

> **小提示 ：** 如果希望镜像源覆盖所有官方仓库请求，可以将 标签内的值改为 *。这样会拦截所有请求，适合网络环境较差时使用，但可能影响部分特殊依赖的获取。

3.验证配置是否生效

执行以下命令，若没有错误且构建日志中的下载链接指向清华源，则表示配置成功。

```
mvn clean compile
```

Java包管理器Gradle Gradle安装

```
cd /tmp
wget https://mirrors.tuna.tsinghua.edu.cn/gradle/8.10.2/gradle-8.10.2-bin.zip
apt install -y unzip
unzip -d /opt/gradle gradle-8.10.2-bin.zip
ln -s /opt/gradle/gradle-8.10.2 /opt/gradle/latest
echo 'export GRADLE_HOME=/opt/gradle/latest' > /etc/profile.d/gradle.sh
echo 'export PATH=${GRADLE_HOME}/bin:${PATH}' >> /etc/profile.d/gradle.sh
source /etc/profile.d/gradle.sh

##### 最后使用下面的命令查看安装结果
gradle -v
```

Gradle全局镜像配置（推荐） 为了不在每个项目中重复配置，推荐使用全局 `init.gradle` 脚本。

1. 创建配置文件 `~/.gradle/init.gradle`

2. 编辑配置文件

将以下内容复制进 `init.gradle` 文件。这个脚本会自动拦截 Gradle 对 Maven 中央仓库的请求，并替换为清华源，而项目中的 `build.gradle`文件无需做任何修改。

```
allprojects {
  repositories {
    maven {
      url 'https://mirrors.tuna.tsinghua.edu.cn/maven-central/'
    }
    mavenCentral()
  }
}
```

数据库软件

数据库	apt软件包名	是否默认安装
SQLite	bsqlite3	✖
PostgreSQL客户端	postgresql-client	✖
Redis客户端	redis-tools	✖

系统命令和基础软件

文件与目录管理 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | ls | 列出目录内容 | coreutils |✔| | cd | 切换目录 | coreutils |✔| | rm | 删除文件或目录 | coreutils |✔| | mkdir | 创建新目录 | coreutils |✔| | cp | 复制文件或目录 | coreutils |✔| | mv | 移动或重命名文件或目录 | coreutils |✔| | pwd | 显示当前目录 | coreutils |✔| | cat | 显示文件内容 | coreutils |✔| | mkdir | 创建目录 | coreutils|✔| | rmdir | 删除空目录 | coreutils|✔| | tac | 反向显示文件内容 | coreutils |✔| | head | 显示文件开头部分 | coreutils |✔| | tail | 显示文件结尾部分 | coreutils |✔| | find | 查找文件或目录 | findutils |✔| | grep | 搜索文本 | grep |

✓| | tar | 归档工具 | tar |✓| | zip | 压缩文件 | zip |✗| | unzip | 解压文件 | unzip |✓|

网络与通信 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | ping | 测试网络连通性 | iputils-ping |✓| | curl | 传输数据 | curl |✓| | wget | 下载文件 | wget |✓| | netstat | 显示网络连接、路由表等 | net-tools |✓| | ip | 显示/操作路由、设备等 | iproute2 |✗| | ss | 查看套接字统计信息 | iproute2 |✗| | traceroute | 追踪数据包路由路径 | traceroute |✗| | dig | DNS 查询工具 | dnsutils |✗| | nslookup | DNS 查询工具 | ndnsutils |✗|

系统信息监控 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | ps | 显示当前进程快照 | procps |✓| | top | 实时显示系统资源使用情况 | procps |✓| | htop | 交互式进程查看器 | htop |✗| | free | 显示内存使用情况 | procps |✓| | df | 报告文件系统磁盘空间使用情况 | coreutils |✓| | du | 估算文件和目录的磁盘使用情况 | coreutils |✓| | uptime | 显示系统运行时间 | procps |✓| | uname | 显示系统信息 | coreutils |✓|

硬件信息 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | lspci | 列出所有 PCI 设备 | pciutils |✗| | lsusb | 列出所有 USB 设备 | usbutils |✗| | lscpu | 显示 CPU 架构信息 | util-linux |✓| | lsblk | 列出块设备信息 | util-linux |✓|

文本处理 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | sed | 流编辑器 | sed |✓| | awk | 模式扫描和处理语言 | awk |✓| | sort | 对文本文件行排序 | coreutils |✓| | uniq | 去除重复行 | coreutils |✓| | cut | 从每行中剪切出选中的部分 | coreutils |✓| | jq | 操作json信息 | jq |✗| | rg | ruby版本的grep命令| ripgrep |✗|

软件开发与版本控制 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | git | 版本控制 | git |✓| | vim | 文本编辑器 | vim |✓| | nano | 简单易用的文本编辑器 | nano |✗| | gcc | C 编译器 | gcc |✗| | g++ | C++ 编译器 | g++ |✗| | make | 项目构建工具 | make |✗|

> gcc默认会安装gcc的11.2.0版本 建议使用下面的命令安装gcc的12.3.0版本：

```
 ##### 安装gcc-12版本
apt install -y gcc-12

##### gcc-12安装后默认只能用gcc-12命令调用，执行下面的命令，就可以用gcc命令调用gcc12版本
ln -s gcc-12 /usr/bin/gcc
```

其他实用工具 | 命令 | 功能说明 | apt 软件包名 |是否默认安装| |------|------|------|------| | tree | 以树状结构显示目录内容 | tree |✗| | man | 查看命令手册 | man-db |✓| | sudo | 以其他用户身份执行命令 | sudo |✗| | screen | 终端复用器 | screen |✗| | tmux | 终端复用器（更现代） | tmux |✗|

分布式训练

训练任务生命周期

概述

本文介绍分布式训练任务以及实例状态，帮助您更好地理解任务的运行状况。

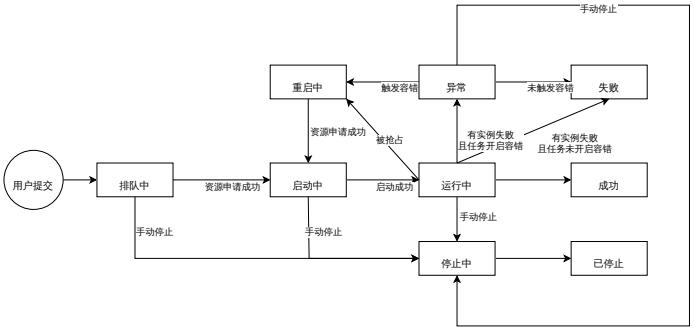
实例状态

状态	说明
排队中	Pod 已经被创建，但是还没有被调度到节点上。
启动中	Pod 已经被调度到节点，正在进行镜像拉取、存储挂载、创建容器等操作。
运行中	Pod 进入 running 状态，正在执行训练代码。
失败	Pod 失败退出。
成功	Pod 成功退出。
异常	Pod 状态未知。

任务状态

状态	说明
排队中	任务已创建，等待调度阶段（允许修改优先级），实例未创建或者所有实例状态为 排队中。
启动中	Pod 调度完成，实例的状态均为 启动中，或者包含部分 运行中 的状态。
运行中	所有 Pod 均为 运行中 状态。
停止中	用户操作停止任务，任务正在停止，正在将实例置为 失败 或者删除实例。
已停止	任务已经停止，所有实例状态为 已停止。
失败	任务失败退出，存在>= 1 个实例状态为 失败。
成功	任务成功结束，所有实例状态为 成功。
异常	存在 >= 1 个实例状态为 异常。
重启中	任务容错或者被抢占，且所有实例状态为 排队中。

任务状态流转图



> **注意：** 为确保资源的有效利用，避免因任务配置问题导致任务无法启动而长时间无效占用资源，平台针对 **启动中** 状态的任务设置了启动超时时间，默认 1 小时。启动超过 1 小时未进入运行态的任务，平台会自动将任务直接置为失败。当前仅针对 PyTorchJob 任务生效。

创建分布式训练任务

概述

本文档介绍在百舸 AI 计算平台的控制台创建训练任务的方法。训练任务支持 **自定义创建** 和 **基于开源模型训练模版创建** 两种方式，基于模板创建方式已预置模型、镜像和训练代码，您可以根据业务需求选择合适的会创建方式。

前提条件

- 已成功创建资源池。详情请参见 创建资源池。
- 已在资源池中准备可用于提交训练任务的队列。

操作步骤

1. 登录 百舸 AI 计算平台。
2. 在导航栏选择 **大规模训练与推理->分布式训练**，进入训练任务列表页面后，单击 **【创建任务】**。

基本参数

在 **基本信息** 区域填写训练任务的基础信息。

参数	必填	说明
任务名称	是	填写任务名称。支持大小写字母、数字、中划线 -、下划线 _ 和小数点 .，长度不超过 256 个字符。
可见范围	是	选择任务的可见范围，支持 **队列内可见** 和 **仅创建人可见**。选择 **队列内可见** 时，队列管理员和成员可见任务；选择 **仅创建人可见** 时，仅队列管理员及创建人可见任务。
创建方式	是	选择 **自定义创建** 或 **基于开源模型训练模版创建**。**自定义创建** 适用于自定义训练镜像以及参数的场景；**基于开源模型训练模版创建** 使用平台预置开源模型的训练镜像和代码，可直接使用。

选择 **基于开源模型训练模版创建** 时，还需要按业务训练目标选择模型和训练模版相关配置。

参数	必填	说明
模型	是	选择需要训练的开源模型及版本。
芯片类型	是	选择训练任务使用的芯片类型。
训练模板	是	选择平台提供的训练模板。
训练模式	是	按模板支持的能力选择训练模式。
训练方法	是	选择模板支持的训练方法。

环境信息

填写训练的配置信息。不同创建方式需要填写的配置信息不同。

自定义创建

参数	必填	说明
镜像地址	是	填写训练镜像。您可以直接输入镜像地址，也可以选择百舸预置镜像，或单击 **【从 CCR 仓库选择】** 从镜像仓库选择镜像。目前支持 CCR 企业版镜像和百舸预置镜像。更多信息，请参考 容器镜像服务 CCR。
执行命令	是	容器启动后执行的命令，支持Shell指令。 若挂载训练代码，挂载路径是代码文件在容器中的位置，执行命令需要使用挂载路径引用代码文件，如代码文件为/workspace，代码文件为demo.py，则执行命令为python/workspace/demo.py。
环境变量	否	单击 **【添加环境变量】**，按需填写变量名和变量值；支持配置多个环境变量。

基于开源模型训练模版创建

参数	必填	说明
镜像地址	是	AIAK 训练加速模版预置默认训练镜像，自动填充且不支持修改。
执行命令	是	AIAK 训练加速模版提供默认参数，需要将数据集、CHECKPOINT、TOKENIZER 以及 Tensorboard 的 PATH 替换成用户自定义路径。
环境变量	否	AIAK 训练加速模版提供默认环境变量，通常不建议修改。

执行命令中需要替换的参数如下：

```
 DATA_PATH=$USER_DATA_PATH
CHECKPOINT_LOAD_PATH=$USER_CHECKPOINT_LOAD_PATH
CHECKPOINT_SAVE_PATH=$USER_CHECKPOINT_SAVE_PATH
TOKENIZER_PATH=$USER_TOKENIZER_PATH
TENSORBOARD_PATH=$TENSORBOARD_PATH
```

> 您如果需要对模型参数做进一步修改，可以在执行命令中直接修改提交。

资源配置

选择训练所需的计算资源、内存资源和存储资源等，可以设置闲时资源、指定节点、优先级等

参数	必填	说明
资源池类型	是	选择全托管资源池或自运维资源池。
队列	是	选择任务部署的资源池和队列信息。
优先级	是	选择任务优先级，支持设置为高、中、低。 队列资源不足时会按照设置的优先级排队调度，优先调度高优先级的任务。
训练框架	是	当前支持 Ray、PyTorch、MPI 和 TensorFlow 训练框架。
资源配额	是	填写任务不同角色节点的实例数量以及规格，支持设置实例申请的 GPU 类型/数量、CPU 和内存。如不申请 GPU，不选择GPU类型并且数量填写为0。
共享内存	否	共享内存（shared memory）用于 Linux 中不同进程之间数据交换和共享，以提高应用程序的性能和效率。
数据集挂载	否	选择训练所需的数据集，平台支持公共数据集和用户自定义数据集。 · 用户自定义数据集：从并行文件存储PFS、对象存储BOS、极速文件缓存RapidFS，可以设置只读权限。 · 公共数据集：平台预置的公开数据集。
存储挂载	否	读取训练所需的数据、存放Checkpoint等训练过程和结果文件。 支持以下存储：并行文件存储 PFS、文件存储 CFS 以及对象存储 BOS，自运维资源池额外支持本地盘。
使用闲时资源	否	开启后，任务可以使用其他队里和本队列的空闲资源，提高资源池的资源利用率。 若出借资源的队列有任务需要使用此资源，资源会被回收，当前任务会中断后重新排队。
指定节点调度	否	支持指定节点运行任务。 默认不指定节点
最大运行时长	否	设置任务运行的最长时长。在完成配置后，超过该时长的任务将停止运行。默认不限制运行时长。
定时提交	否	开启后可在指定时间自动提交训练任务。您只能设置当前时间之后的提交时间，默认不开启定时提交任务。
任务退出自动删除	否	配置任务的自动回收策略。开启后，仅针对成功/失败状态的任务生效，您可以自定义配置保留时长。

容错与诊断

为训练异常和中断等情况提供重试策略和诊断机制。

参数	必填	说明
自动容错	否	开启训练容错。详见 训练容错。
任务 hang 检测	否	用于检测训练任务长时间无进展等异常情况，开启后，任务hang时，平台通过函数调用栈与进程IO监控指标，多维度判断任务状态。详见训练任务Hang检测。

监控信息

平台提供配置 Tensorboard、任务告警和日志持久化能力，实现训练监控。

参数	必填	说明
Tensorboard	否	为该任务开启 Tensorboard。开启后需要指定日志存储路径，并行文件存储PFS、文件存储CFS、对象存储BOS、极速文件缓存RapidFS，支持该路径需要与代码中的 Tensorboard 日志路径保持一致，否则 Tensorboard 无法获取数据。详见 训练效果监控 Tensorboard。
任务告警	否	百舸平台针对训练任务状态提供告警通知机制，支持短信、邮件、微信、飞书以及自定义 webhook 通知方式。详见 配置任务的消息通知。
日志持久化	否	**注意：** 开启日志持久化功能会将您的任务日志持久化保存到日志服务（BLS）中，日志存储、读取、写入和索引会产生费用，详情参考 价格详情。

提交任务

确认任务名称、镜像地址、执行命令、资源队列、资源配额、容错与监控配置无误后，单击 **【确定】**，完成任务创建。任务创建成功后，在详情页可以查看任务状态、监控和日志。

管理分布式训练任务

概述

本文档介绍百舸 AI 计算平台对训练任务的管理，支持对训练任务执行停止、重跑、复制、删除等操作。

前提条件

- 如需停止训练任务，需准备处于运行中、启动中或排队中的训练任务。
- 如需复制或删除训练任务，需准备已有训练任务。
- 如需重跑训练任务，需准备处于失败或成功状态的训练任务。

操作步骤

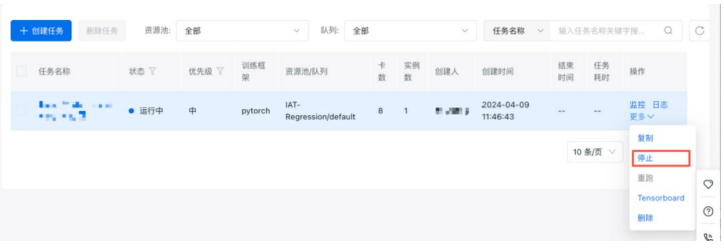
1. 登录 百舸 AI 计算平台
2. 进入 **大规模训练与推理->分布式训练** 页面

停止训练任务

当您在任务提交后发现参数配置错误或者效果不符合预期，可以停止任务，释放资源。

操作步骤如下：

1. 在训练任务列表中，选择需要停止的任务，在操作栏单击 **【停止】**。
2. 停止后资源将被释放，仍可查看日志、监控和 TensorBoard 信息。



1. 任务运行阶段，停止训练任务，会保留 Worker 实例，仍然支持查看当前任务的日志、监控和 TensorBoard 数据。

> 2. 任务启动、排队阶段，停止训练任务，会删除 Worker 实例。此状态下操作停止，需要保证资源池中 CCE Deep Learning Frameworks Operator 组件版本在 1.6.22 及以上。如组件版本不满足要求，请按照页面提示完成升级。

复制训练任务

在大模型训练中，需要频繁调整参数以获得最佳训练效果，百舸控制台提供了任务复制功能。可快速复制当前任务信息，您可在当前任务基础上修改相关参数再次提交，提高模型训练效率。

操作步骤如下：

1. 在训练任务列表中，选择需要复制的任务，在操作栏单击 **【复制】**。



2. 系统跳转到任务创建页面，并自动填充当前任务相关信息。

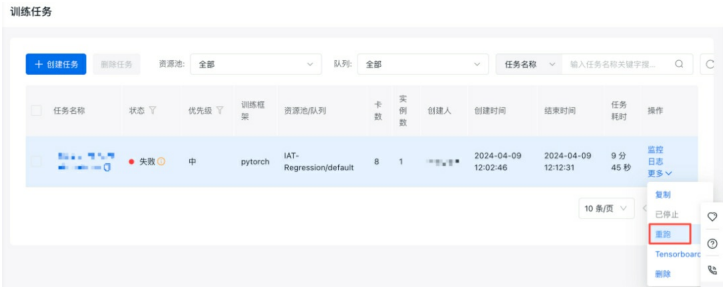
3. 确认参数后，单击 **【完成】**，即可完成任务复制。

重跑训练任务

如果您的训练任务失败并且希望重新启动运行，可以使用重跑功能。

> 仅支持处于失败/成功状态的任务使用重跑功能。

1. 在训练任务列表中，选择需要重跑的任务，在操作栏 **【更多】** 中单击 **【重跑】**。



2. 系统会自动提交一个与原始任务除任务名称外，其他配置完全相同的任务。

> 系统会自动生成任务名称，新的任务名称命名规则：原始任务名称-4 位随机字符。

3. 在创建后新任务的任务详情页面，可以在 **任务重跑** 字段中查看该任务的重跑信息，包括当前任务基于哪个最原始的任务重跑以及重跑次数。



4. 单击 **重跑次数**，可以查询详细的任务重跑记录。



删除训练任务

删除任务会同步释放任务使用的资源。

> **注意：** 任务删除后无法恢复，请谨慎操作。

1. 在训练任务列表中，选择需要删除的任务，在操作栏 **【更多】** 中单击 **【删除】**。



2. 在删除确认弹窗中单击 **【确认】**，完成删除。

查看训练详情

概述

本文介绍如何查看任务详情，有助于您更为全面地了解任务的信息。

前提条件

已登录 百舸 AI 计算平台，且当前账号下已存在可查看的训练任务。

操作步骤

1. 在控制台页面选择 **大规模训练与推理->分布式训练**，进入任务列表页面。
2. 在任务列表页面，找到需要查看的任务，在 **任务名称/ID** 列单击目标任务名称，进入任务详情页面。
3. 在任务详情页面查看任务信息和实例信息。



   1. 查看任务信息
      在基本信息栏，您可以查看任务名称、资源池队列、训练框架、GPU 类型及卡数、任务创建时间信息。
   2. 查看实例信息
      在任务详情下方的实例信息栏，您可以查看角色、实例名称、状态、重启次数、实例 IP、Host IP 、创建时间和执行时长。

训练框架说明

发起Ray分布式训练

概述

Ray 是一款专为 AI 工作负载打造的开源分布式计算框架，能够统一基础设施，高效管理、执行和优化从数据处理、模型训练到模型服务的全流程 AI 计算需求。凭借对 Python 的天然友好性，算法开发者可轻松在集群上扩展应用程序与机器学习工作负载。

百舸平台为用户提供开箱即用的 Ray 环境。您无需更改代码或调整资源，只需使用预置 Ray 镜像，或使用包含 Ray Runtime 的自定义镜像，并提供入口命令（entrypoint）与依赖定义，即可运行 Ray 作业。平台会自动创建并管理 Ray 集群，完成节点检查、资源调度、日志与指标采集，并提供 Dashboard。

功能限制

1. 暂不支持容错与诊断，支持 Ray 原生的自动重试。
2. Ray Dashboard 中 Metrics 功能的启用需要用户对接自己的 Grafana。可参考 自定义监控指标采集 实现 Ray Metrics 采集和监控。

前提条件

准备训练镜像

Ray 集群由 Head 管理节点与 Worker 计算节点构成。百舸将基于您指定的统一镜像，分别构建 Head 节点与 Worker 节点容器，并自动完成 Ray 集群的部署与初始化。待集群就绪后，系统会自动启动一个同镜像的 Submitter 节点，用于向 Ray 集群提交您的脚本。

Ray 版本支持范围：平台已对 Ray 2.7~2.49 版本完成验证，更高版本暂未验证。

1. 使用百舸官方 Ray 镜像：百舸已经预置 Ray 官方镜像，您可以直接使用



2. 使用自定义的镜像：需要镜像中包含 Ray[default]关键依赖，建议您基于百舸预置的 Ray 镜像构建自定义的镜像

准备启动命令和训练脚本

百舸训练任务的执行命令会被用作 `ray job submit` 所提交的 `entrypoint` 命令。启动命令可以填写多行或单行，无需在执行命令中写 `ray start` 或 `ray job submit`。

例如，执行命令可填写 `python /workspace/sample.py`，其中 `sample.py` 为运行的 Python 脚本文件。示例内容如下：

```
 import ray
import os
import requests

ray.init()


@ray.remote
class Counter:
    def __init__(self):
        # Used to verify runtimeEnv
        self.name = os.getenv("counter_name")
        self.counter = 0

    def inc(self):
        print(f"current counter {self.counter} counter_name from runtimeENV is {self.name}")
        import time
        time.sleep(1)
        self.counter += 1

    def get_counter(self):
        return "{} got {}".format(self.name, self.counter)

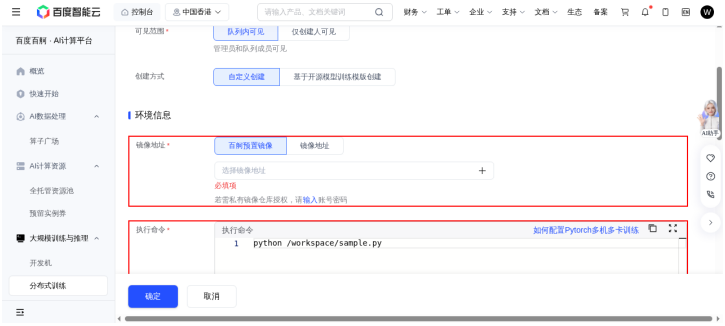

counter = Counter.remote()

for _ in range(50000):
    ray.get(counter.inc.remote())
    print(ray.get(counter.get_counter.remote()))
```

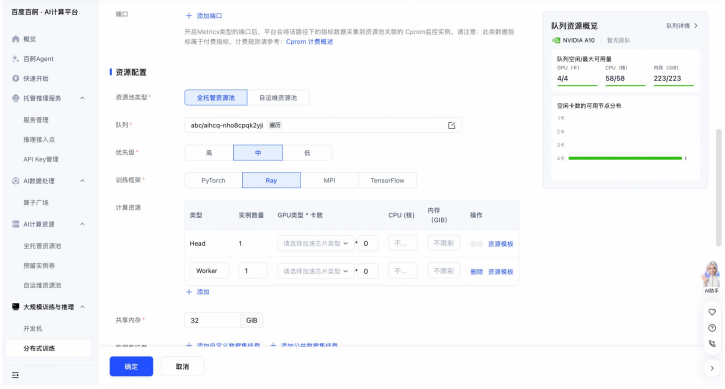
提交 Ray 任务

操作步骤

1. 登录 百舸 AI 计算平台。
2. 在控制台导航栏选择 **大规模训练与推理->分布式训练**，进入训练任务列表页，点击 **【创建任务】**。
3. 在新建任务页面的 **环境信息** 区域配置镜像和执行命令。
    1. 镜像地址：可选择 **百舸预置镜像**，也可以切换为 **镜像地址** 使用自定义镜像；
    2. 执行命令：填写 Ray 作业入口命令，例如 `python /workspace/sample.py`。该命令会作为 Ray Job 的 entrypoint 执行，无需手动填写 `ray start` 或 `ray job submit`。



4. 在 **资源配置** 区域选择队列、优先级和训练框架。
    1. 选择可用训练队列。
       点击 **队列** 输入框可打开 **选择队列** 抽屉，选择满足 Ray 任务资源需求的训练队列。
    2. 训练框架：选择 **Ray**框架。
    3. 计算资源: 配置 Head 和 Worker 节点的资源，支持手动输入资源，或者通过资源模板快速选择。
    4. 共享内存：支持配置共享内存资源。



参数说明

参数	必填	说明
镜像地址	是	可以选择百舸预置 Ray 镜像，也可以使用自定义镜像。
执行命令	是	本任务需要执行的命令，例如 `python /workspace/sample.py`。该命令会作为 Ray Job 的 entrypoint 执行。
训练框架	是	选择 **Ray**。
计算资源	是	配置任务的节点类型为 Head 和 Worker。 • Head 节点个数限制为 1，仅用于运行 entrypoint 脚本； • 支持自定义 Worker 类型名称，支持设置多组 Worker； • 每个 Ray 任务会自动生成一个 Submitter 节点执行启动命令，可通过 Submitter 节点查看任务运行日志。
RuntimeENV	否	RuntimeEnv 用于动态安装 Ray 任务所需的软件依赖、配置环境变量或工作目录等，详见 Ray 官方文档。生产环境中，建议使用已经打包好的镜像执行任务，避免因临时安装依赖库造成任务失败。

日志与监控

查看日志

1. 在控制台导航栏选择 **大规模训练与推理->分布式训练**，进入训练任务列表页。
2. 在任务列表中查看 **监控/日志** 列。当存在 Ray 任务时，可在该列进入任务的监控或日志详情。
   任务运行日志可通过 Submitter 节点的标准输出查看，Ray 集群系统日志可通过 Head/Worker 节点的文本日志查看。

监控配置

1. 在 **分布式训练** 页面点击 **【创建任务】**，进入创建任务页面。
2. 在**环境信息**区域配置端口，Ray Metrics 需要通过 自定义监控指标采集 对接 Cprom 后，才能进行指标采集与查询
3. 在 **监控信息** 区域查看 **Tensorboard** 和 **任务告警** 配置。

常见问题

1. 我还需要在执行命令中写 `ray start` 或 `ray job submit` 吗？

不需要。

在平台提交 Ray Job 时：

- 平台会在 Head 节点容器内自动执行 `ray start --head ...` 启动 Ray 集群。
- 平台会在 Worker 节点容器内自动执行 `ray start --address=<head-service>:6379 --block`，将 Worker 加入集群。
- Ray 集群就绪后，系统将自动启动一个 Submitter 节点，用于向 Ray 集群提交脚本。

2. 1 个 Ray Job 会创建几个 Ray 集群？集群什么时候销毁？

1 个 Ray Job 对应 1 个独立的 Ray 集群。Ray Job 结束后，对应的 Ray 集群会自动销毁。

3. Ray Job 结束之后，是否可以查询 Ray Dashboard？

目前仅支持在 Ray Job 处于运行中状态时查看 Ray Dashboard。Ray Job 结束后查看 Ray Dashboard 的功能正在开发中，敬请期待。

发起MPI分布式训练

概述

本文档介绍在百舸平台中使用 **MPI（Message Passing Interface）** 类型工作负载运行分布式训练任务的基本原理、启动命令配置方法、控制台配置流程以及常见问题处理方式。

MPI 是一种成熟、稳定的进程通信模型，广泛应用于高性能计算（HPC）与深度学习分布式训练。百舸平台提供的 MPI 类型工作负载，使用户可以在 Kubernetes 环境中以原生方式运行 MPI 训练任务，适用于：

- 多节点、多 GPU 训练。
- 需要精细控制进程拓扑与通信方式的任务。
- 基于 NCCL / RDMA 的高性能通信场景。
- 框架层（PyTorch / Horovod / Megatron / DeepSpeed 等）对 MPI 兼容良好的训练任务。

MPI 训练基本原理

MPI 进程模型

MPI 采用 **多进程（Multi-Process）** 模型：

- 每个训练进程拥有独立的进程空间。
- 进程之间通过 MPI 通信库进行点对点或集合通信。
- 每个进程有唯一的 RANK。
- 所有进程共享 WORLD_SIZE。

常见逻辑关系如下：

概念	含义
WORLD_SIZE	全局进程总数
RANK	当前进程在全局中的编号
LOCAL_RANK	当前节点内的进程编号
NODE_RANK	节点编号

在 GPU 训练中，通常 1 个 MPI 进程绑定 1 张 GPU。

工作机制

MPI Operator 主要由以下角色组成：

1. **MPI Launcher**
   - 负责执行 `mpirun`。
   - 生成 `hostfile`。
   - 拉起 Worker Pod 中的 MPI 进程。
2. **MPI Worker**
   - 实际运行训练进程。
   - 提供计算资源（CPU / GPU / RDMA）。

整体流程如下：

1. **MPIJob 提交：** 用户通过百舸平台提交 MPIJob 自定义资源，配置任务命令、Worker 数量、资源规格及镜像信息，触发任务调度流程。
2. **Launcher Pod 启动：** 集群优先调度 Launcher Pod，作为 MPI 任务的主控入口，其成功启动是后续 Worker 调度的前提。
3. **hostfile 自动注入：** MPI Operator 在检测到 Launcher 与 Worker Pod 就绪后，自动收集 Worker 节点信息并生成标准 `hostfile`，注入至 Launcher 容器。
4. **mpirun 启动 Worker 进程：** Launcher Pod 内执行 `mpirun`，基于 `hostfile` 通过 SSH 或 PMI 协议在各 Worker Pod 中拉起 MPI 进程。
5. **MPI / NCCL 通信执行：** 所有 Worker 进程启动完成后，进入业务执行阶段，通过 MPI 或 NCCL 完成参数同步与计算协同，直至任务结束。

镜像依赖与环境准备

参数	要求
运行用户	root
MPI	镜像中必须包含 mpirun
SSH	支持 root 用户 SSH 免密登录
sshd	Worker 容器中必须可启动 sshd
配置文件	/etc/ssh/sshd_config 中关闭 StrictModes

本文测试镜像如下：

```
registry.baidubce.com/cce-plugin-dev/deepspeed-training:23.08-gpu-py310-cu122-ubuntu22.04
```

MPI 启动方式与命令结构

基本 `mpirun` 结构

```
mpirun \
--hostfile /etc/mpi/hostfile \
-np 4 \
-bind-to none \
-map-by slot \
-x NCCL_DEBUG=INFO \
-x NCCL_IB_DISABLE=1 \
python /app/train.py
```

参数说明

OpenMPI 的 `mpirun` 参数较多，以下按功能分类说明核心参数。

1. 进程管理类

参数	完整写法	作用	场景使用建议
-np	--np <num>	指定总 MPI 进程数	值需匹配 Worker Pod 数 × 每个 Pod 进程数。（如 2 Worker × 2 进程 = -np 4）
-npernode	--npernode <num>	每个节点启动的进程数	推荐使用：显式控制每个 Worker Pod 的进程数（如 -npernode 2），避免进程分配不均
--hostfile	--hostfile <path>	指定节点列表文件	路径固定使用 /etc/mpi/hostfile（MPI Operator 自动生成，包含所有 Worker Pod 的 IP / 端口）。
--host	--host <host1:slot1,host2:slot2>	手动指定节点	禁用：MPI Operator 已自动生成 hostfile，手动指定会冲突。
-oversubscribe	--oversubscribe	允许进程数超过节点 slot 数	谨慎使用：仅当 K8s 资源充足时临时调试，生产环境禁用（易导致资源耗尽）

2. 通信配置类

参数	完整写法	作用	场景使用建议
-mca pml <layer>	--mca pml <layer>	指定进程间通信层（PML）	-mca pml ob1（OpenMPI 4.x 稳定版，适配 K8s 网络）；禁用 ucx（K8s 集群中易出网络问题）
-mca btl <layers>	--mca btl <layers>	指定字节传输层（BTL）	-mca btl tcp,self（K8s 集群内通信基于 TCP，self 仅兜底）；禁用 openib（无 InfiniBand 时会报错）
-mca btl_tcp_if_exclude <iface>	-	排除指定网卡	避免使用本地回环或容器网卡通信。
-mca btl_tcp_if_include <iface>	-	仅使用指定网卡	若集群有专用通信网卡（如 eth0），可指定 -mca btl_tcp_if_include eth0
-mca oob_tcp_if_exclude <iface>	-	排除 OOB（带外通信）网卡	同 btl_tcp_if_exclude，推荐同步配置。

3. 资源绑定类

参数	作用	MPI Operator 场景使用建议
-bind-to <type>	绑定进程到 CPU 核心、NUMA 节点或不绑定	推荐使用 -bind-to none，禁用绑定，交给K8s调度；禁用core/socket，易导致资源冲突
-map-by <type>	进程分配策略	推荐使用 -map-by slot，按 slot 均匀分配进程，适配 K8s Pod 资源模型；禁用 node/core（分配不均）。
--report-bindings	打印进程绑定信息	调试或验证时使用，确认进程是否被错误绑定。

4. 环境变量传递类

Launcher Pod 的环境变量不会自动同步到 Worker，需要显式传递核心变量。

参数	作用	场景使用建议
-x <var>	传递单个环境变量到所有 Worker	至少传递 -x PATH（保证命令可执行）、-x LD_LIBRARY_PATH（保证依赖库找到）；GPU 任务需加 -x NCCL_DEBUG=INFO（调试）、-x NCCL_SOCKET_IFNAME=eth0（指定 NCCL 通信网卡）
-x <var>=<value>	传递并覆盖环境变量值	推荐：如 -x NCCL_DEBUG=INFO、-x CUDA_VISIBLE_DEVICES=0
--env-file <file>	从文件读取环境变量	环境变量较多时使用，文件中每行格式为 VAR=VALUE。

5. 调试 / 日志类

参数	作用	场景使用建议
-v / --verbose	打印详细启动日志	调试时使用，排查进程启动失败问题。
--debug	打印调试级日志	仅深度排查时使用（日志量较大）。
-mca orte_base_help_aggregate 0	禁用日志聚合，每个进程单独输出日志。	调试时使用，用于定位单个 Worker 的日志问题。
--display-map	打印进程分配映射，展示哪个进程运行在哪个 Worker。	用于验证进程分配是否符合预期。

6. 容错 / 超时类

参数	作用	场景使用建议
-timeout <sec>	设置任务超时时间	-timeout 3600（1 小时），防止任务卡死后长期占用资源。
-mca plm_rsh_no_tree_spawn 1	禁用树形启动，直接启动所有进程。	Kubernetes 集群中稳定性更高。
-mca orte_abort_on_non_zero_status 1	任意进程非 0 退出时，整个任务终止	生产环境保证任务原子性。
--leave-sessions-attached	进程退出后保留会话，便于排查退出现场。	调试时使用，生产环境不建议使用。

7. 其他实用类

参数	作用	场景使用建议
-quiet	静默模式，仅打印错误日志。	生产环境推荐使用，减少日志量。
--tag-output	为每个进程输出添加标签（进程ID）	分布式任务排障时推荐使用，便于区分不同进程日志。
-wdir <path>	指定所有进程的工作目录	推荐使用，统一设置为业务代码目录，如 -wdir /app。

hostfile 说明

百舸平台会自动为 MPI 任务注入 /etc/mpi/hostfile，内容如下：

```
job-xxx-worker-0.job-xxx-worker.default.svc slots=8
job-xxx-worker-1.job-xxx-worker.default.svc slots=8
```

用户无需手动创建或修改 hostfile。

MPI 任务实践

本文档介绍在百舸平台中使用 DeepSpeed 进行分布式训练的最佳实践。本文以 2 个 Worker 节点、每个节点 2 张 GPU 的典型集群配置为场景，展示如何正确配置和启动训练任务。本方案中的 MPI 核心参数配置具有通用性，可适配各类 MPI 任务，使用时根据训练代码和参数进行调整即可。

训练镜像

镜像中包含 CIFAR10 数据集，路径为 /root/code/training/data。训练代码路径为 /root/code/training/cifar10_deepspeed.py，参考 https://github.com/deepspeedai/DeepSpeedExamples/blob/master/training/cifar/cifar10_deepspeed.py。

```
registry.baidubce.com/cce-plugin-dev/deepspeed-training:23.08-gpu-py310-cu122-ubuntu22.04
```

启动命令详解

1. 启动命令

```
mpirun -np 4 \
  -bind-to none -map-by slot \
  -mca pml ob1 -mca btl ^openib \
  -x NCCL_DEBUG=INFO \
  python /root/code/training/cifar10_deepspeed.py \
  --deepspeed \
  --epochs 128 \
  --stage 2 \
  --dtype fp16 \
  --log-interval 100
```

2. 基础 MPI 命令说明

```
mpirun -np 4 \
  --hostfile /etc/mpi/hostfile \
  --allow-run-as-root \
  -bind-to none -map-by slot \
  -mca pml ob1 -mca btl ^openib \
  python /root/code/training/cifar10_deepspeed.py
```

参数解析如下：

- -np 4：启动 4 个进程，对应 2 个 Worker 节点 × 每个 Worker 节点 2 张 GPU。
- --hostfile /etc/mpi/hostfile：指定集群节点配置文件。
- --allow-run-as-root：在容器化环境中允许以 root 身份运行。

- `-bind-to none -map-by slot`：将进程按 slot 映射到各个 GPU。
- `-mca pml ob1 -mca btl ^openib`：使用 TCP 通信而非 InfiniBand，提高兼容性。

3. DeepSpeed 相关参数说明

```
 --deepspeed \
--epochs 128 \
--stage 2 \
--dtype fp16 \
--log-interval 100 \
--micro-batch-size 2 \
--gradient-accumulation-steps 1
```

参数解析如下：

- `--deepspeed` ：启用 DeepSpeed 训练引擎。
- `--epochs` ：设置训练轮数。
- `--stage` ：设置 ZeRO 优化阶段，Stage 2 可在保持训练速度的同时减少显存占用。
- `--dtype` ：设置训练精度，使用 fp16 可减少显存使用并加速计算。
- `--log-interval` ：设置日志打印间隔。
- `--micro-batch-size`：设置每个 GPU 上的批次大小。
- `--gradient-accumulation-steps`：设置梯度累积步数，用于模拟更大批次训练。

控制台配置

1. 进入分布式训练任务创建页

在百舸控制台选择 **大规模训练与推理->分布式训练**，进入分布式训练任务列表页，点击 **【创建任务】**。在创建任务页面填写基础信息。

2. 配置镜像和执行命令

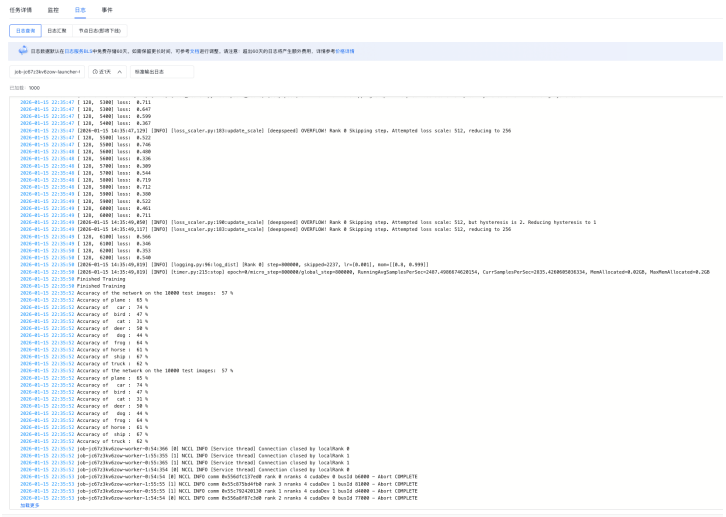
在 **环境信息** 区域选择 **【镜像地址】**，填写训练镜像地址，镜像地址可以使用registry.baidubce.com/cce-plugin-dev/deepspeed-training:23.08-gpu-py310-cu122-ubuntu22.04。执行命令可使用本文"启动命令详解"中的 MPI + DeepSpeed 启动命令，并根据实际训练脚本路径、训练轮数、精度类型等参数进行调整。

3. 配置 MPI 训练资源

在 **资源配置** 区域选择MPI训练框架，设置worker数量和资源规格，可以配置2个节点，每个节点2张卡。
确认 Worker 实例数量与 GPU 卡数配置符合总进程数设置。例如启动命令中配置 `-np 4` 时，资源规格应能提供 4 张 GPU。



4. 运行任务 点击确定按钮提交任务开始运行。



常见问题（FAQ）

1. sshd 启动失败：Missing privilege separation directory: /run/sshd

原因： 容器中 `/run/sshd` 不存在。

解决方案： 构建镜像时添加如下命令。

```
 mkdir -p /run/sshd
chmod 0755 /run/sshd
```

2. mpirun 无输出或任务挂起

常见原因：

- Worker Pod 中 `sshd` 未运行。
- SSH 端口未监听。
- SSH 认证失败。

排查命令：

```
ps aux | grep sshd
ss -lntp | grep :22
```

3. SSH 认证失败：`Permission denied (publickey)`

建议配置：

- 使用 root 用户。
- 确认 `/root/.ssh/id_rsa` 存在。
- 确认 `/root/.ssh/authorized_keys` 已配置公钥。
- 在 `sshd_config` 中关闭 `StrictModes`。

4. 禁止 root 运行

错误信息如下：

```
mpirun has detected an attempt to run as root
```

解决方案：在启动命令中添加如下参数：

```
--allow-run-as-root
```

5. MPI 环境中错误使用 DeepSpeed 启动

MPI 已负责进程管理和 rank 分配。如果再次使用 DeepSpeed 启动器，会导致二次进程拉起和 rank 冲突。

```
###### 正确做法
mpirun -np 8 python train.py

###### 错误做法
mpirun -np 8 deepspeed train.py
```

关键：MPI 模式下，DeepSpeed 只作为训练库使用，需要禁用其启动器功能。

6. mpirun 直接执行 shell 脚本失败

`mpirun` 不会自动解析脚本解释器，远程节点可能无法识别脚本格式，需要显式指定 `bash` 解释器执行脚本。

```
###### 正确做法
mpirun -np 8 bash run.sh

###### 错误做法
mpirun -np 8 run.sh
```

建议在脚本首行添加如下内容，确保使用 `bash` 而非 `sh`。

```
###### !/usr/bin/env bash
```

发起Pytorch分布式训练

概述 DistributedDataParallel（DDP） 是 PyTorch 官方推荐的 **多卡 / 多机分布式训练框架** ，基于 **多进程 + Ring AllReduce** 实现，用于加速大模型、大数据集的训练。

百舸平台提供了 PyTorch 训练框架的完整支持，覆盖单机训练和分布式训练场景。本文介绍如何在百舸平台提交 PyTorch 训练任务的关键要点，包括PyTorch的分布式进程模型、工作机制和启动命令。

PyTorch 训练基本原理

PyTorch 分布式进程模型

PyTorch 分布式训练采用 **多进程（Multi-Process）** 模型，核心基于 `torch.distributed` 模块实现进程间通信：

- 每个训练进程拥有独立的 Python 解释器和 GPU。
- 进程之间通过 NCCL / Gloo 等后端进行集合通信，包括 AllReduce、Broadcast 等。
- 每个进程有唯一的 `RANK`，同一节点内的进程有 `LOCAL_RANK`。

常见逻辑关系如下：

概念	含义
`WORLD_SIZE`	全局进程总数。
`RANK`	当前进程在全局中的编号，取值范围为 `0` ~ `WORLD_SIZE - 1`。
`LOCAL_RANK`	当前节点内的进程编号，取值范围为 `0` ~ 本节点 GPU 数 - 1。
`MASTER_ADDR`	主节点（`RANK = 0`）的地址。
`MASTER_PORT`	主节点通信端口。

PyTorch 训练框架工作机制

对于 PyTorch 训练框架任务，百舸平台的工作机制如下：

1. **环境变量注入** 平台自动向所有训练容器注入 MASTER_ADDR、MASTER_PORT、WORLD_SIZE、RANK、LOCAL_RANK 等分布式训练所需的环境变量。详见 PyTorch 环境变量 页面，您可以在代码中直接应用上述环境变量。
2. **进程拉起** 平台根据实例数量与单实例 GPU 卡数，自动完成多节点进程调度与拉起。
3. **通信建联** 训练进程启动后，通过 torch.distributed.init_process_group() 初始化进程组，利用平台注入的环境变量自动完成节点间通信建联。
4. **训练执行** 所有进程完成初始化后进入训练循环，通过 NCCL / Gloo 后端完成梯度同步。

典型启动命令示例

torchrun 启动

```
torchrun \
--nproc_per_node $NPROC_PER_NODE \
--nnodes $WORLD_SIZE \
--node_rank $RANK \
--master_addr $MASTER_ADDR \
--master_port $MASTER_PORT \
your_training_script.py
```

accelerate launch 启动

```
num_processes=$((WORLD_SIZE * NPROC_PER_NODE))
echo $num_processes
accelerate launch --same_network --multi_gpu --gpu_ids all \
--num_machines $WORLD_SIZE --num_processes $num_processes \
--machine_rank $RANK --main_process_ip $MASTER_ADDR \
--main_process_port $MASTER_PORT \
your_training_script.py
```

PyTorch 任务实践：基于 AIAK-Training 训练 Qwen-2.5-VL-72B

可参考 Qwen-2.5-VL-72B 多模态大模型 SFT 训练 文档，在百舸平台使用 PyTorch 框架 + AIAK-Training（Megatron）提交多机多卡训练任务。

配置训练任务的消息通知

概述

百舸平台针对训练任务的状态变化提供通知机制，配置后当训练任务状态匹配已配置的告警规则时，系统会按照所选通知策略触发告警通知。本文将介绍如何配置任务状态变化通知。

前提条件

1. 资源池已经接入百度云 Prometheus 监控服务。请参考 资源池接入 Prometheus 监控实例。

通知策略说明

基于百度云 Prometheus 监控服务 的 通知策略 能力。配置任务告警时，您可以选择已有通知策略，也可以新建通知策略，系统会在任务状态匹配告警规则后，按照所选通知策略发送告警通知。

参数	描述
任务状态	支持任务进入以下状态时通知： 1. 运行状态 2. 任务失败退出 3. 任务成功结束 4. 任务 hang（需要先开启 hang 检测功能） 5. 任务发生容错 6. 任务所在节点出现故障。
通知策略	通知渠道支持邮件、短信、电话、钉钉、企业微信、飞书等机器人 Webhook 地址

操作指南

1. 登录 百舸AI计算平台，在导航栏选择 **大规模训练与推理->分布式训练**，进入分布式训练任务列表页面。
2. 单击 **【创建任务】**，进入创建任务页面。
3. 在创建任务页面的 **监控信息** 区域，打开**任务告警**开关，选择需要通知的任务状态和通知策略。如需选择任务Hang，请开启任务Hang检测；如需创建新的通知策略，需要设置策略名称和通知方式。



4. 完成任务基础信息、环境信息、资源配置和任务告警配置后，单击 **【确定】** 创建训练任务。
5. 任务创建成功后，当训练任务状态匹配已配置的告警规则时，系统会按照所选通知策略触发告警通知。

使用闲时资源

概述

本文档介绍在百度百舸 AI 计算平台中，分布式训练任务使用闲时资源的方法。

用户在百舸中购买的资源以资源队列的形式划分给不同的业务，不同业务的资源队列可能会出现忙闲不均的场景，某些业务的资源队列在特定时间段内可能闲置，而其他业务的资源队列可能因缺少资源而排队，导致整个资源池维度上出现资源浪费。

为了解决上述痛点，百舸提供了分布式训练使用闲时资源的能力，提升资源池的整体利用率，实现原理如下：

- 允许管理员将队列的空闲资源出借给同资源池的其他队列，用于创建训练任务，提升资源池的整体利用率。
- 开启闲时资源的任务，使用当前队列或者其他队列的空闲资源，不受本资源队列总量和剩余资源数量的限制。
- 当借用的空闲资源需要被原资源队列的任务使用时，闲时计算任务将会被终止，自动归还资源。

使用限制

1. 仅支持全托管资源池的训练任务。

使用步骤

创建任务开启使用闲时资源

1. 在百舸控制台选择 **大规模训练与推理->分布式训练**，单击 **【创建任务】**，进入分布式训练任务创建页面。
2. 在**资源配置**区域开启**使用闲时资源**。
   开启后，任务可以使用本队列的空闲资源，也可以使用同资源池内其他已开启空闲资源出借队列的空闲资源。系统调度时会优先使用本队列的空闲资源。



说明：

1. 使用闲时资源的任务能够使用其他队列的空闲资源，因此资源选择不受本 **队列资源总量** 限制，只要未超过资源池资源上限即可，但仍受以下限制：
   - 使用闲时资源的任务使用的资源型号，不能超出本队列限制。如本队列 GPU 仅有 A800，则创建使用闲时资源的任务也仅能申请 GPU A800 资源。
   - 如本队列中未分配 GPU 资源，则创建使用闲时资源的任务时也不能申请 GPU。
2. 任务调度优先级：本队列内非闲时任务高于闲时任务。
3. 闲时资源使用模式：任务可以使用本队列的空闲资源，也可以使用其他队列出借的空闲资源（队列已经开启空闲资源出借），也可以同时使用。系统在调度上优先使用本队列的空闲资源。
4. 资源回收说明：当借用的空闲资源需要被回收时，优先回收提交时间最晚的任务占用的资源。

配置告警

当闲时资源被回收时，您可配置任务告警通知策略，第一时间获取通知。任务状态选择**闲时资源回收导致任务中断**。闲时任务的资源被回收的记录，也可以通过任务timeline、任务事件查询



开启自动容错

使用闲时资源的任务，资源回收而导致任务被中断后，平台会尝试重新申请资源启动任务，请确认代码中包含 checkpoint 机制，确保任务可以顺利续跑。

极端情况下任务会直接失败退出，这里推荐开启 **自动容错** - **无条件重试** 能力进行兜底，详见 训练容错。

> 仅支持 PyTorchJob。



查询闲时任务资源回收情况

任务使用的闲时资源被回收后，在任务详情页面的实例状态会被标记为 **被抢占**。也可以通过任务timeline、任务事件查询闲时资源被收回的记录。

训练任务优先级管理

概述

本文档介绍百度百舸 AI 计算平台训练任务优先级规则配置功能。

为保障高优用户和高优任务的资源需求，百舸平台针对训练任务提供优先级规则配置能力。队列管理员可在队列详情中开启该功能，并为不同用户、用户组或队列角色设定可使用的最高任务优先级，确保高优任务优先获得资源。

使用限制

仅支持全托管资源池

使用步骤

开启优先级规则配置

1. 进入百度百舸 AI 计算平台，进入 **AI 计算资源->全托管资源池->资源队列**列表。
2. 选择目标资源队列进入详情页，点击 **【队列配置】**。
3. 页面内可见 **优先级规则配置** 开关，默认处于关闭状态；关闭状态下，队列成员创建任务不受优先级规则管控。



4. 开启 **优先级规则配置** 开关，系统会自动创建默认规则：
   - 针对队列成员：**队列内算法开发人员** 最高可创建中优先级任务；**队列管理员** 可创建高优先级任务。
   - 全局配置：未匹配任何规则的用户，其可创建任务的最高优先级受全局配置限制，默认为中优先级。
5. 添加或编辑规则。
   您可以自定义配置规则名称，生效范围，以及针对该用户/用户组/队列角色配置最高优先级，例如您可以：
   - 为指定用户创建高优先级规则
   - 为指定用户组（高级 xx 员）创建高优先级规则
   - 为指定用户/用户组创建低优先级规则（例如实习生等）
   - 目前仅支持为训练任务配置规则



> 为了查询效率与使用体验，平台限制每个队列创建五条规则。

规则限制

1. 受限制的角色仅能创建不超过其最高优先级限制的训练任务
2. 在默认规则限制下，队列内算法开发成员只能创建中、低优先级训练任务；复制高优先级任务时，系统会默认将优先级重置为中。
3. 在默认规则限制下，队列管理员支持创建高优先级训练任务；队列管理员可将排队中的任务优先级调为高。

配置训练任务低利用率自动停止策略

概述

本文档介绍在百度百舸 · AI 计算平台中配置训练任务低利用率自动停止策略的方法。

在分布式训练任务量大、集群资源紧张的场景下，队列管理员可通过该策略自动停止低利用率训练任务，减少算力闲置浪费，降低训练成本，提升集群整体资源利用率与任务调度效率。

使用说明

- 支持全托管资源池和自运维资源池的队列
- 仅支持队列管理员、资源池管理员开启该策略
- 自动停止策略作用维度为队列，即仅对配置的队列任务生效

使用步骤

1. 登录百度百舸 · AI 计算平台，进入 **AI 计算资源->全托管资源池->资源队列**。



2. 点击目标队列名称进入队列详情，进入 **【队列配置】** Tab页面，策略默认关闭，关闭状态下，队列成员创建的训练任务不会被该策略自动停止。开启低利用率自动停止策略，在弹窗内配置触发规则。



3. 配置规则。
   对于每个资源队列，您可以配置两条通过或逻辑连接的规则组，每个规则组内部通过与逻辑连接。

   - 或逻辑：满足任意一个规则组即可触发自动停止。
   - 与逻辑：满足规则组内所有条件才会触发自动停止。

   您可以按运行时长、GPU 资源利用率等条件配置规则组，使队列中的训练任务在满足低利用率自动停止规则时被自动停止。
   您也可以通过配置排除策略来实现高优先级任务豁免，来避免一些长期任务因为训练中周期性/不同原因导致的低利用率触发关闭规则。
   如下图所示，队列中高、中优先级的任务都不会因为触发规则而关闭任务



4. 查看被停止的任务
   队列管理员可以点击【停止记录】按钮来查看该队列触发规则停止的任务记录列表。



   同时，被自动停止的任务，在任务timeline以及任务状态的浮窗可以看到任务被停止的原因



在训练任务中使用云存储

概述

在百舸异构计算平台提交分布式训练任务时，您可以通过存储挂载的方式将云存储（BOS、CFS、PFS、RapidFS）挂载到训练容器中，从而在训练过程中直接读写存储中的数据。本文介绍如何在百舸训练任务中配置并使用 BOS、CFS、PFS 和 RapidFS 存储挂载。

前提条件

- 已登录 百舸异构计算平台。
- 已创建资源池和队列。
- 已准备对应的云存储实例：
  - **PFS**：已创建 PFS 实例和挂载服务，并将挂载服务绑定到资源池。请参考 创建 PFS 和 创建挂载服务。
  - **CFS**：已创建 CFS 文件系统，并已获取 CFS 挂载地址。请参考 创建文件系统。
  - **BOS**：已创建 BOS 存储桶。请参考 创建存储桶。

- **RapidFS**：已创建 RapidFS 实例和数据源。请参考 创建 RapidFS 实例。

使用 PFS 存储

并行文件存储 PFS 是百舸平台训练任务的 **推荐存储方案**，提供最高性能的文件访问能力。

确认 PFS 实例已关联资源池

1. 在控制台导航中选择 **AI计算资源->全托管资源池**，进入资源池列表页面。
2. 选择目标资源池名称，进入资源池详情页面。
3. 在 **存储挂载** 区域，确认已关联目标 PFS 实例的挂载服务。若未关联，请参考 创建挂载服务 完成绑定。

在训练任务中配置 PFS 存储挂载

1. 在控制台导航中选择 **大规模训练与推理->分布式训练**，进入分布式训练任务列表页面。
2. 单击 **【创建任务】**，进入任务创建页面。
3. 完成 **基本信息**、**环境信息** 和 **资源配置** 后，在 **资源配置** 区域查看 **存储类型** 配置项。
4. 单击 **【添加存储挂载】**，配置 PFS 存储挂载：

参数	必填	说明
存储类型	是	选择 **并行文件存储PFS**。
文件系统	是	从下拉框中选择已绑定到资源池的 PFS 实例。
源路径	是	填写 PFS 中已存在的目录路径，不建议以根目录 `/` 挂载。
挂载路径	是	填写训练容器内的挂载路径，执行命令中的数据路径需与该路径对应，推荐配置在 `/mnt/pfs/` 目录下。
权限	否	默认读写，勾选 **只读** 后无法在挂载路径下创建或修改文件。

使用 BOS 存储

对象存储 BOS 适合海量数据的长期存储和分发，在百舸训练任务中支持通过挂载方式访问 BOS 数据。BOS 存储挂载基于 bosfs2.0 实现。

在训练任务中配置 BOS 存储挂载

1. 在控制台导航中选择 **大规模训练与推理->分布式训练**，进入分布式训练任务列表页面。
2. 单击 **【创建任务】**，进入任务创建页面。
3. 完成 **基本信息** 和 **环境信息** 后，在 **资源配置** 区域查看 **存储类型** 配置项。
4. 单击 **【添加存储挂载】**，在 **存储类型** 中选择 **BOS**，并按需配置 BOS 存储挂载。

参数	必填	说明
存储类型	是	选择 **BOS**。
BOS存储路径	是	选择需要挂载到训练任务中的 BOS 存储桶和源路径。
挂载路径	是	填写训练容器内的挂载路径，推荐配置在 `/mnt/bos/` 目录下。
权限	否	默认读写，勾选 **只读** 后无法在挂载路径下创建或修改文件。

BOS 挂载使用限制

- 文件读取限制：新写入的文件必须在文件关闭后才会完全上传到 BOS。如果读取正在顺序写入且尚未关闭的文件，可能导致报错。（支持随机写的小文件不受限制）
- 重命名目录相关请求不保证原子性，在正在执行 `mv` 的目录下创建文件，可能导致文件仍保留在原始目录下。
- 多个客户端访问同一个或多个挂载点时，数据一致性需由使用侧维护，可能出现多端数据不一致情况。

更多信息请参考 bosfs2.0。

使用 CFS 存储

文件存储 CFS 提供 POSIX 兼容的文件访问接口，支持多读多写，适合需要文件共享但性能要求不如 PFS 的场景。

参数	必填	说明
存储类型	是	选择 **CFS**。
CFS 挂载点	是	选择已创建的 CFS 文件系统挂载点地址（CFS Server）。
源路径	是	填写 CFS 文件系统中的源目录路径。
挂载路径	是	填写训练容器内的挂载路径，推荐配置在 `/mnt/cfs/` 目录下。
权限	否	默认读写，勾选 **只读** 后无法在挂载路径下创建或修改文件。

使用 RapidFS 存储

参数	必填	说明
存储类型	是	选择 **RapidFS**。
RapidFS 实例	是	选择需要挂载到训练任务中的 RapidFS 实例。
数据源	是	选择 RapidFS 实例的数据源。
挂载路径	是	填写训练容器内的挂载路径，推荐配置在 `/mnt/rapidfs/` 目录下。

使用限制

- 通过 RapidFS 读写文件时，避免对单个文件进行边读边写操作，否则会对性能产生较大影响。
- 通过 RapidFS 写入的文件，只能在文件 `flush` 后才能被读取。
- 多个客户端写入同一个文件时，只会有一个客户端写入成功，其他客户端会写入失败。

训练任务可观测

训练任务日志

概述

你可以在百舸 AI 计算平台查询分布式训练任务日志。任务日志支持日志查询和节点日志（即将下线），可用于检索任务运行日志、查看日志上下文、配置日志展示字段、查看各节点最新日志。

前提条件

- 已登录百度智能云控制台，并具备访问百舸 AI 计算平台和日志服务 BLS 的权限。
- 已创建并提交分布式训练任务，且任务已产生运行日志。

操作步骤

1. 登录 百舸AI计算平台，在导航菜单选择 **大规模训练与推理->分布式训练**，进入 **分布式训练** 任务列表页面。
2. 在任务列表中找到目标任务，在任务行的 **监控/日志** 列单击日志图标，进入节点日志页面后查看任务日志。

日志查询

日志数据默认在 日志服务BLS 中免费存储 60 天。如需保留更长时间，可自行调整。

> **注意：** 超出 60 天的日志将产生额外费用，详情参考 价格详情。

日志检索

平台支持日志关键字的检索能力。

- 关键字不区分大小写
- 支持多个关键字检索。输入多个关键字时，系统会查询同时包含这些关键字的日志。如查询内容为`service not available`，则表示查询包含关键字`service`、 `not`、`available`的日志。
- 支持前缀匹配的模糊检索方式，`*` 代表多个任意字符的词，`?` 代表单个字符的词。例如：`abc*`代表查询以 `abc`开头的词。`ab?d` 代表查询以 `ab` 开头、`d`结尾且中间包含单个字符的词。

上下文查询

日志检索后，支持查看日志上下文，获取更多信息。日志检索完成后，可在结果列表中点击所选日志的上下文图标，打开**查看上下文**弹窗查看该日志前后的上下文内容。上下文区域支持填写 **高亮显示**，并可以设置是否展示**行号**和**时间**。

配置日志展示字段

在日志结果区域配置日志展示字段。结果表默认展示 **实例名称** 和 **时间戳**，如需调整展示内容可按需配置。

调整日志存储时间

持久化日志默认免费存储60天，在此期间免费。 如需保留更长时间，可通过 BLS控制台， **日志管理** > **日志集合** 页面，修改名称为 **aihcp-aijob-log** 日志集的存储周期

节点日志（即将下线）

在日志页面，您可以选择实例查看当前日志或者选择2个不同实例对日志进行参照对比。您可以对任务日志进行刷新或 **开启自动刷新**。

> 节点日志仅展示最近5000条日志，可以通过 **日志查询** 入口查看全部日志。

训练任务事件

概述

本文档介绍如何查询训练任务的事件信息。任务事件包括任务在启动和运行阶段，任务以及关联资源的变更情况，可以弥补资源监控在实时性和场景上的欠缺，有助于运维人员日常观察任务变更以及定位问题。

前提条件

- 已创建分布式训练任务，并具备查看该任务详情的权限。

操作步骤

1. 登录 百舸 AI 计算平台 AIHC 控制台。
2. 在导航栏选择 **大规模训练与推理->分布式训练**，进入 **任务列表** 页面。
3. 在任务列表中，单击需要查看事件的 **任务名称**，进入任务详情页面。
4. 单击 **【任务事件】** 页签，查看事件信息。



训练任务事件字段定义如下：

参数	说明
首次出现时间（FIRST SEEN）	事件首次出现的时间。
最后出现时间（LAST SEEN）	事件最后出现的时间。
事件类型（TYPE）	标记任务级别，取值为 Normal 或 Warning。
资源类型/名称（OBJECT）	事件所涉及的对象以及对象名称。
原因（REASON）	当前发生事件的简短描述，一般为枚举值，主要在程序内部使用。
消息（MESSAGE）	当前发生事件的详细描述信息。
事件源（SOURCE）	报告此事件的组件，如 Scheduler、Kubelet 等。
出现次数（COUNT）	事件发生的次数。

任务事件中支持的对象如下：

> 如训练任务失败，用户可以通过训练任务事件，查询任务所在的节点是否有故障事件，初步排除或定位节点故障的原因。

参数	说明
job	任务自身事件。
podgroup	任务基于 volcano 调度时，关联的 podgroup 事件。
pod	任务创建的 pod 事件。
service	pytorchjob 或 mpijob 自动创建的 service 事件。
pv/pvc	用户关联存储卷 pv/pvc 时，展示对应事件。
node	任务所在节点的故障或运维事件。除原生的节点事件外，百舸通过 Node Problem Detector / Node Remedier 组件，拓展了包含节点故障信息以及运维信息的事件。

训练任务资源监控

概述

本文档介绍如何在百舸平台查询训练任务监控。百舸平台针对训练任务的资源使用提供全面的监控指标，帮助您掌握训练任务的资源负载情况。

前提条件

- 资源池已经接入百度云 Prometheus监控服务。请参考 资源池接入Prometheus监控实例。
- 已经创建训练任务，详情可参考 创建任务。

查询入口

1. 登录 百舸控制台。
2. 左侧菜单栏单击 **分布式训练**，进入训练任务列表页面
3. 选择需要查询的训练任务，单击 **监控**，即可查询该任务的监控大盘

监控指标及说明

当前任务资源监控主要支持任务维度、实例维度、GPU 卡维度、RDMA 网卡维度以及节点维度的监控。

任务维度

面板组	指标	描述
GPU	GPU 算力使用率（任务维度）	指定任务的 GPU 算力使用率
GPU	GPU 显存使用率（任务维度）	指定任务的 GPU 显存使用率
GPU	GPU SM 设备使用率（任务维度）	指定任务的 GPU 设备 SM 处于 Active 的比例
GPU	PCIE 发送数据速率（任务维度）	指定任务的 GPU 设备 PCIE 上的发送数据速率
GPU	PCIE 接收数据速率（任务维度）	指定任务的 GPU 设备 PCIE 上的接收数据速率
GPU	NVLink 发送数据速率（任务维度）	指定任务的 GPU 设备 NVLink 上的发送数据速率
GPU	NVLink 接收数据速率（任务维度）	指定任务的 GPU 设备 NVLink 上的接收数据速率
GPU	GPU 功耗（任务维度）	指定任务的 GPU 设备功耗
GPU	GPU 温度（任务维度）	指定任务的 GPU 设备温度
GPU	GPU Tensor pipe 使用率（任务维度）	指定任务的 GPU 设备 Tensor pipe 处于 Active 的周期分数，较高的值表示 Tensor Cores 的利用率较高
网络	RDMA 网卡健康状态（任务维度）	指定任务的 RDMA 网卡状态（100% 为健康，低于 100% 为部分异常）
网络	RDMA 网卡发送端丢包速率（任务维度）	指定任务的 RDMA 网卡发送端丢包速率
网络	RDMA 网卡接收端丢包速率（任务维度）	指定任务的 RDMA 网卡接收端丢包速率
网络	RDMA 网卡接收数据速率（任务维度）	指定任务的 RDMA 网卡的接收数据速率
网络	RDMA 网卡发送数据速率（任务维度）	指定任务的 RDMA 网卡的发送数据速率
网络	RDMA 网卡发送包速率（任务维度）	指定任务的 RDMA 网卡的发送包速率
网络	RDMA 网卡接收包速率（任务维度）	指定任务的 RDMA 网卡的接收包速率
CPU/内存	CPU 使用率（任务维度）	指定任务的 CPU 使用率（占节点）
CPU/内存	内存使用率（任务维度）	指定任务的内存使用率（占节点）
存储	磁盘设备读/写速率（任务维度）	指定任务的磁盘读/写数据的速率
存储	磁盘设备读/写 IO 速率（任务维度）	指定任务的磁盘读/写数据的 IOPS

实例维度

面板组	指标	描述
GPU	GPU 算力使用率（Pod 维度）	指定 Pod 的 GPU 算力使用率
GPU	GPU 显存使用率（Pod 维度）	指定 Pod 的 GPU 显存使用率
GPU	GPU SM 设备使用率（Pod 维度）	指定 Pod 的 GPU 设备 SM 处于 Active 的比例
GPU	PCIE 发送数据速率（Pod 维度）	指定 Pod 的 GPU 设备 PCIE 上的发送数据速率
GPU	PCIE 接收数据速率（Pod 维度）	指定 Pod 的 GPU 设备 PCIE 上的接收数据速率
GPU	NVLink 发送数据速率（Pod 维度）	指定 Pod 的 GPU 设备 NVLink 上的发送数据速率
GPU	NVLink 接收数据速率（Pod 维度）	指定 Pod 的 GPU 设备 NVLink 上的接收数据速率
GPU	GPU 功耗（Pod 维度）	指定 Pod 的 GPU 设备功耗
GPU	GPU 温度（Pod 维度）	指定 Pod 的 GPU 设备温度
GPU	GPU Tensor pipe 使用率（Pod 维度）	指定 Pod 的 GPU 设备 Tensor pipe 处于 Active 状态的周期分数，较高的值表示 Tensor Cores 的利用率较高
网络	RDMA 网卡健康状态（Pod 维度）	指定 Pod 的 RDMA 网卡状态（100% 为健康，低于 100% 为部分异常）
网络	RDMA 网卡发送端丢包速率（Pod 维度）	指定 Pod 的 RDMA 网卡发送端丢包速率
网络	RDMA 网卡接收端丢包速率（Pod 维度）	指定 Pod 的 RDMA 网卡接收端丢包速率
网络	RDMA 网卡接收数据速率（Pod 维度）	指定 Pod 的 RDMA 网卡的接收数据速率
网络	RDMA 网卡发送数据速率（Pod 维度）	指定 Pod 的 RDMA 网卡的发送数据速率
网络	RDMA 网卡发送包速率（Pod 维度）	指定 Pod 的 RDMA 网卡的发送包速率
网络	RDMA 网卡接收包速率（Pod 维度）	指定 Pod 的 RDMA 网卡的接收包速率
CPU/内存	CPU 使用率（Pod 维度）	指定 Pod 的 CPU 使用率（占节点）
CPU/内存	CPU 使用量（Pod 维度）	指定 Pod 的 CPU 使用量
CPU/内存	内存使用率（Pod 维度）	指定 Pod 的内存使用率（占节点）
CPU/内存	内存使用量（Pod 维度）	指定 Pod 的内存使用量
存储	磁盘设备读/写速率（Pod 维度）	指定 Pod 的磁盘读/写数据的速率
存储	磁盘设备读/写 IO 速率（Pod 维度）	指定 Pod 的磁盘读/写数据的 IOPS

GPU 卡维度

指标	描述
GPU 算力使用率（卡维度）	GPU 设备算力使用率
GPU 显存使用率（卡维度）	GPU 设备显存使用率
GPU 显存使用量（卡维度）	GPU 设备显存使用量
GPU SM 设备使用率（卡维度）	GPU 设备 SM 处于 Active 的比例
PCIE 发送数据速率（卡维度）	GPU 设备 PCIE 上的发送数据速率
PCIE 接收数据速率（卡维度）	GPU 设备 PCIE 上的接收数据速率
NVLink 发送数据速率（卡维度）	GPU 设备 NVLink 上的发送数据速率
NVLink 接收数据速率（卡维度）	GPU 设备 NVLink 上的接收数据速率
GPU 功耗（卡维度）	GPU 设备功耗
GPU 温度（卡维度）	GPU 设备温度
GPU Tensor pipe 使用率（卡维度）	GPU 设备 Tensor pipe 平均处于 Active 状态的周期分数，较高的值表示 Tensor Cores 的利用率较高

RDMA 网卡维度

指标	描述
RDMA 网卡健康状态（网卡维度）	RDMA 网卡状态（1 为健康，0 为异常）
RDMA 网卡发送端丢包速率（网卡维度）	RDMA 网卡发送端丢包速率
RDMA 网卡接收端丢包速率（网卡维度）	RDMA 网卡接收端丢包速率
RDMA 网卡接收数据速率（网卡维度）	RDMA 网卡的接收数据速率
RDMA 网卡发送数据速率（网卡维度）	RDMA 网卡的发送数据速率
RDMA 网卡发送包速率（网卡维度）	RDMA 网卡的发送包速率
RDMA 网卡接收包速率（网卡维度）	RDMA 网卡的接收包速率

节点维度

面板组	指标	描述
Overview	节点状态	节点的健康状态，1 表示健康，0 表示异常
GPU	GPU 算力使用率（节点维度）	指定节点的 GPU 设备算力使用率
GPU	GPU 显存使用率（节点维度）	指定节点的 GPU 设备显存使用率
GPU	GPU SM 设备使用率（节点维度）	指定节点的 GPU 设备 SM 处于 Active 的比例
CPU/内存	CPU 使用率（节点维度）	指定节点的 CPU 使用率
CPU/内存	CPU 使用量（节点维度）	指定节点的 CPU 使用量
CPU/内存	内存使用率（节点维度）	指定节点的内存使用率
CPU/内存	内存使用量（节点维度）	指定节点的内存使用量
磁盘	磁盘分区存储空间总容量/已使用容量/使用率	指定节点磁盘分区存储空间总容量/已使用容量/使用率
磁盘	磁盘分区 inode 总数量/已使用量/使用率	指定节点磁盘分区 inode 总数量/已使用量/使用率
磁盘	磁盘设备读速率（节点维度）	指定节点上磁盘设备读速率
磁盘	磁盘设备写速率（节点维度）	指定节点上磁盘设备写速率
磁盘	磁盘设备 IO 利用率（节点维度）	指定节点上磁盘设备 IO 利用率
磁盘	磁盘设备读 IO 速率（节点维度）	指定节点上磁盘设备读 IO 速率
磁盘	磁盘设备写 IO 速率（节点维度）	指定节点上磁盘设备写 IO 速率

RDMA网络高精度监控

概述

本文档介绍百舸平台大模型并行训练场景下的 RDMA 网络高精度监控能力。大模型训练场景中，通常会使用并行训练策略，降低训练过程中的通信开销，加速训练效率。为方便高效地观测和排查大模型并行训练通信中的问题，百舸平台提供高精度 RDMA 网络监控能力，最高支持 10ms 精度，为故障诊断排除、训练性能调优等提供数据支撑。

监控指标说明

监控面板	监控指标	指标说明
RDMA 网卡监控（250ms 精度）	RDMA 网卡发送数据速率（250ms 精度）	指定任务的 RDMA 网卡发送数据速率（250ms 精度）。
RDMA 网卡监控（250ms 精度）	RDMA 网卡接收数据速率（250ms 精度）	指定任务的 RDMA 网卡接收数据速率（250ms 精度）。
RDMA 网卡监控（10ms 精度）	RDMA 网卡发送数据速率（10ms 精度）	指定任务的 RDMA 网卡发送数据速率（10ms 精度）。
RDMA 网卡监控（10ms 精度）	RDMA 网卡接收数据速率（10ms 精度）	指定任务的 RDMA 网卡接收数据速率（10ms 精度）。

前提条件

1. 资源池内 CCE Deep Learning Frameworks Operator 组件版本 >= 1.6.17。查询组件版本或者升级组件版本，可以通过 **百舸资源池详情** > **计算资源集群（CCE)** > **组件管理** 操作。
2. 资源池已经接入百度云 Prometheus 监控服务。请参考 资源池接入 Prometheus 监控实例。
3. 创建训练任务前，已准备支持 RDMA 的 GPU 资源和可用队列。

使用限制

1. 仅支持满足如下条件的任务开启高精度监控：
   - 任务实例数 >= 2 个。
   - 任务开启 RDMA 加速。
2. RDMA 网卡监控（250ms 精度）的监控指标保留最近 30 天，RDMA 网卡监控（10ms 精度）仅保留最近 1 天。

操作步骤

1.创建训练任务&开启高精度监控

1. 登录 百舸AI计算平台AIHC控制台。
2. 点击 左侧菜单栏 **分布式训练** ，进入训练任务创建页面
3. 点击 **创建任务**，进入创建任务页面
4. 在 **创建任务** > **高级设置** 模块中，开启 **RDMA高精度监控**
5. 点击 **完成** 即完成任务创建。

2 .查看高精度监控

1. 进入训练任务列表页面，选择需要查询的训练任务，单击 **监控** > **高精度监控**，即可查询该任务的高精度RDMA监控指标。

训练性能监控

概述

本文档介绍百舸平台训练性能监控功能。百舸平台提供训练过程中的吞吐、训练分阶段耗时以及训练 Loss 指标，帮助客户实时掌控训练的性能状态，快速发现性能异常，提升性能异常排查效率。

训练性能监控说明

目前针对于性能可观测提供了以下指标：

训练吞吐

训练过程中单位时间内模型能够处理的数据量，它是衡量训练性能的关键指标之一。

监控项	说明
训练吞吐（tokens/卡/秒）	每张 GPU 卡每秒能够处理的 tokens 数量
训练吞吐（B tokens/台/天）	每台服务器每天能够处理的 tokens 数量（Billion）

训练分阶段耗时

提供训练过程中每次迭代中不同操作（计算时间、IO 时间、通信时间等）的耗时统计（最小值以及最大值），便于对比分析，定位异常。

面板组	监控项	描述
TP+前向/反向耗时	forward-backward-time	完成一次前向传播和反向传播所需的总时间
TP+前向/反向耗时	forward-compute-time	模型进行前向传播，即从输入到输出计算预测结果所需的时间
TP+前向/反向耗时	backward-compute-time	模型进行反向传播计算预测结果所需的时间
DP 通信耗时	grads-reduce-scatter-time	在分布式训练中，将梯度规约并分散到不同的计算节点上的时间（通信）
DP 通信耗时	params-all-gather-time	在分布式训练中，将不同计算节点上的参数收集起来的时间（通信）
DP 通信耗时	layernorm-grads-all-reduce-time	在分布式训练中，对层归一化（Layer Normalization）操作的梯度进行全规约（all-reduce）操作的时间
DP 通信耗时	embedding-grads-all-reduce-time	在分布式训练中，对嵌入层（Embedding Layer）的梯度进行全规约操作的时间
PP 通信耗时	forward-recv-time	在分布式训练或者模型并行中，接收前向计算所需数据的时间（通信）
PP 通信耗时	forward-send-time	在分布式训练或者模型并行中，发送前向计算结果数据的时间（通信）
PP 通信耗时	backward-recv-time	在分布式训练或者模型并行中，接收用于反向传播计算的梯度数据的时间（通信）
PP 通信耗时	backward-send-time	在分布式训练或者模型并行中，发送梯度数据以进行下一步计算的时间（通信）
PP 通信耗时	forward-send-backward-recv-time	在模型并行中，同时进行前向传播的发送操作和反向传播的接收操作所需的时间（通信）
PP 通信耗时	backward-send-forward-recv-time	在模型并行中，同时进行反向传播的发送操作和前向传播的接收操作所需的时间（通信）
优化器耗时	optimizer-time	优化器在训练过程中执行更新步骤所需的时间
优化器耗时	optimizer-copy-to-main-grad-time	梯度数据从辅助存储复制到主存储所用时间
优化器耗时	optimizer-unscale-and-check-inf-time	梯度进行缩放，并检查梯度是否包含无穷大值或 NaN 值所用时间
优化器耗时	optimizer-count-zeros-time	统计梯度中的零值数量时间
优化器耗时	optimizer-inner-step-time	实际的参数更新步骤所用时间
优化器耗时	optimizer-copy-main-to-model-params-time	主存储中的参数复制回模型参数存储所用时间
其他操作耗时	batch-generator-time	从数据集中生成一个批次数据所需的时间，可能包括数据读取、预处理等

训练 Loss

提供训练过程中的 lm loss 的监控。

监控项	说明
lm loss	“lm loss” 一般指的是 “Language Model Loss”。lm loss 是用来衡量模型生成的文本与实际文本之间的差异程度。在机器学习中，我们通常希望最小化这个损失函数，以便模型能更好地学习数据的分布。

前提条件

1. 使用新版本的 AIAK 训练加速镜像，要求镜像版本 >= v1.2.7.3。

```
A800: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.3_release
H800: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu22.04-cu12.2-torch2.1.0-py310_v1.2.7.3_release
```

2. 资源池内 CCE Deep Learning Frameworks Operator 组件版本 >= 1.6.18。查询组件版本或者升级组件版本，可以通过 **百舸资源池详情->计算资源集群（CCE）->组件管理** 操作。

3. 资源池已经接入百度云 Prometheus监控服务。请参考 资源池接入Prometheus监控实例。

4. 已准备可用于提交分布式训练任务的资源池、训练队列和资源配额。

操作步骤

1.在Cprom中实例中配置采集任务

> 与训练集合通信带宽监控的采集任务配置一致，如已经配置，则此步骤忽略

在资源池关联的Cprom监控实例中，配置采集任务。

1. 登录 百舸AI计算平台AIHC控制台。

2. 点击 **全托管资源池** 或 **自运维资源池**，找到想要任务所在的资源池名称，点击进入资源池的详情页面。

3. 在 **资源详情** 中，点击 **关联监控实例(CPROM)** 的实例ID，进入Cprom实例的详情页面。

4. 单击 **采集配置**，进入采集任务配置页面。

5. 点击 **创建采集任务**，采集任务的详情如下，点击 **确认** 即完成创建。

```
  job_name: "ftagent"
scheme: http
scrape_interval: 15s
honor_timestamps: true
metrics_path: /metrics
kubernetes_sd_configs:
  - role: pod
relabel_configs:
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
    action: keep
    regex: true
  - source_labels: [__meta_kubernetes_pod_label_training_kubeflow_org_operator_name]
    action: keep
    regex: "pytorchjob-controller"
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
    action: replace
    target_label: metrics_path
    regex: (.+)
  - source_labels: [__meta_kubernetes_pod_host_ip]
    action: replace
    target_label: host_ip
    regex: (.+)
  - source_labels: [__meta_kubernetes_pod_node_name]
    action: replace
    target_label: node_name
    regex: (.+)
  - source_labels: [__meta_kubernetes_namespace]
    action: replace
    target_label: namespace
  - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
    action: replace
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    target_label: __address__
  - action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
```

2.创建训练任务

您需要在百舸控制台，使用符合要求的AIAK训练镜像创建训练任务。参照 创建任务。

3.查看性能监控

1. 登录百舸AI计算平台AIHC控制台。
2. 左侧菜单栏单击 **分布式训练**，进入训练任务列表页面。
3. 选择需要查询的训练任务，单击 **监控 > 性能监控**，即可查询该任务的性能监控大盘。

训练集合通信带宽监控

概述

本文档介绍如何在百舸平台查询训练过程中的集合通信带宽监控。百舸平台支持实时集合通信带宽统计，可以在训练过程中对集合通信性能进行实时观测，展示集合通信在不同阶段的性能表现，为故障诊断排除、训练性能调优等提供数据支撑。

> 集合通信监控指标是基于百度云 高性能集合通信库 BCCL 实现，需要您的训练任务集成 BCCL 通信库。

监控指标说明

集合通信带宽监控（通信组维度）

指标	说明
集合通信带宽（avg/max）	训练过程中，通信组维度的集合通信带宽平均值和最大值。集合通信带宽 = 集合通信传输数据量 / 集合通信完成时间，统计的是整个集合通信期间的平均带宽。
单次集合通信数据量	单次集合通信传输的数据量。

集合通信带宽监控（GPU 维度）

指标	说明
集合通信带宽（avg/max）	训练过程中，通信组中 GPU 维度的集合通信带宽平均值和最大值。集合通信带宽 = 集合通信传输数据量 / 集合通信完成时间，统计的是整个集合通信期间的平均带宽。
网络传输带宽（avg/max）	训练过程中，通信组中 GPU 维度的网络传输带宽平均值和最大值。网络传输带宽 = 集合通信传输数据量 / 网络传输完成时间。仅统计集合通信期间网络传输的带宽。
单次集合通信数据量	单次集合通信传输的数据量。

> **说明：** 集合通信完成时间 = 网络传输完成时间 + GPU 操作及等待时间。

使用限制

1. 仅支持 PyTorch 训练框架的任务。
2. 集合通信带宽监控暂不支持机内集合通信带宽监控。

前提条件

1. 训练任务使用百度云 高性能集合通信库 BCCL。支持以下两种方式：
   - 使用新版本的 AIAK 训练加速镜像，镜像版本需不低于 v2.1.1.5。AIAK 镜像中已集成 BCCL 通信库，默认采用 NCCL 通信库，可以通过设置环境变量开启 BCCL：
     `export LD_LIBRARY_PATH=$BCCL_PATH:$LD_LIBRARY_PATH`
     可使用满足版本要求的 AIAK 训练加速镜像地址：

```
registry.baidubce.com/aihc-aiak/aiak-training-llm:ubuntu22.04-cu12.3-torch2.2.0-py310-bccl1.2.7.2_v2.1.1.5_release
```

- 在自定义训练镜像中一键注入 BCCL 软件，详情请参见 BCCL：百度自研高性能集合通信库。

2. 资源池内 CCE Deep Learning Frameworks Operator 组件版本需不低于 1.6.17。您可以在 **AI 计算资源 -> 全托管资源池 -> 资源池名称 -> 计算资源集群（CCE）-> 组件管理**中查询或升级组件版本。

3. 资源池已接入百度云 Prometheus 监控服务。接入方式请参见 资源池接入 Prometheus 监控实例。

操作步骤

在 CPROM 实例中配置采集任务

> 与训练性能监控的采集任务配置一致，如已配置，则可跳过此步骤。

1. 登录 百舸 AI 计算平台 AIHC 控制台，在左侧导航栏选择 **AI 计算资源 -> 全托管资源池**，进入资源池列表页面。

2. 单击目标资源池名称，进入资源池详情页面。

3. 在 **资源详情** 区域，单击 **关联监控实例（CPROM）** 的实例 ID，进入 CPROM 实例详情页面。

4. 单击 **【采集配置】**，进入采集任务配置页面。

5. 单击 **【创建采集任务】**，配置采集任务信息，然后单击 **【确认】** 完成创建。

```
 job_name: "ftagent"
scheme: http
scrape_interval: 15s
honor_timestamps: true
metrics_path: /metrics
kubernetes_sd_configs:
  - role: pod
relabel_configs:
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
    action: keep
    regex: true
  - source_labels: [__meta_kubernetes_pod_label_training_kubeflow_org_operator_name]
    action: keep
    regex: "pytorchjob-controller"
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
    action: replace
    target_label: metrics_path
    regex: (.+)
  - source_labels: [__meta_kubernetes_pod_host_ip]
    action: replace
    target_label: host_ip
    regex: (.+)
  - source_labels: [__meta_kubernetes_pod_node_name]
    action: replace
    target_label: node_name
    regex: (.+)
  - source_labels: [__meta_kubernetes_namespace]
    action: replace
    target_label: namespace
  - source_labels:
      [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
    action: replace
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    target_label: __address__
  - action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
```

创建训练任务

您需要在百舸控制台，使用包含百舸BCCL软件的训练镜像，创建训练任务。点击查看如何 创建任务。

查看集合通信监控

1. 登录 百舸AI计算平台AIHC控制台。

2. 左侧菜单栏单击 **分布式训练**，进入训练任务列表页面

3. 选择需要查询的训练任务，单击 **监控 > 集合通信带宽监控，** 即可查询该任务的集合通信监控大盘

训练效果监控Tensorboard

概述

TensorBoard 是一个用于可视化 TensorFlow 模型训练过程和结果的工具，可以帮助开发人员和研究人员更好地理解和调试模型，从而提高模型的性能和效率。百舸 AI 计算平台 AIHC 中集成了 Tensorboard 能力，本文档介绍如何在百舸 AI 计算平台 AIHC 中创建训练任务并开启 Tensorboard 服务，以及如何查询 Tensorboard。

前提条件

- 已登录百舸 AI 计算平台 AIHC 控制台。
- 已准备可用的资源池、训练队列及日志路径对应的存储资源。
- 如需查询 Tensorboard，需已有开启 Tensorboard 的训练任务。## 功能说明

1. 百舸 Tensorboard 服务使用百度云的鉴权体系进行安全访问限制，保证数据安全。

2. 为避免资源池残留大量 Tensorboard 实例，百舸对 Tensorboard 实例采用定时回收机制，默认保留 24 小时。如已经被回收，Tensorboard 实例会在下次查询时触发实时创建，预计需要 1 分钟。

使用限制

已经创建的训练任务，暂不支持开启 Tensorboard。

操作步骤

创建任务开启 Tensorboard 服务

1. 在百舸 AI 计算平台 AIHC 控制台，通过 **大规模训练与推理->分布式训练** 进入分布式训练任务列表页，点击 **【创建任务】** 按钮。
2. 在创建任务页面填写任务基础信息、环境信息、资源配置等必填项后，在 **监控信息** 区域，将 **Tensorboard** 开关切换为 **开** 状态。开启时平台会检测资源池依赖组件是否已安装；如未安装 Tensorboard 依赖组件，需按页面提示先完成依赖组件安装后再开启。

> **说明：** 资源池内的训练任务，首次开启 Tensorboard，需要资源池安装 CCE Ingress NGINX Controller 组件，并会自动创建一个 BLB 作为 Tensorboard 服务的访问入口。后续此资源池内其他训练任务开启 Tensorboard，可以复用此 BLB。

3. 开启 Tensorboard 后，查看 **日志路径** 配置。日志路径由平台自动生成，默认路径为 `/output/training_logs`，支持通过 `$AIHC_TENSORBOARD_LOG_PATH` 环境变量在训练代码中引用。

> **注意：** 请保持该路径与代码中的 Tensorboard 日志路径一致，否则 Tensorboard 无法获取数据。

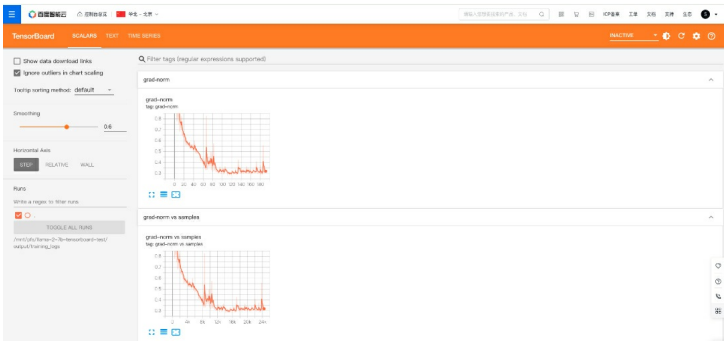
4. 确认任务基础信息、环境信息、资源配置、日志路径等配置无误后，点击 **【确定】** 按钮，完成训练任务创建并开启 Tensorboard 服务。

查询 Tensorboard

训练任务开启 Tensorboard 后，可在百舸 AI 计算平台 AIHC 控制台进入任务关联的 Tensorboard 实例。

1. 在百舸 AI 计算平台 AIHC 控制台，通过 **大规模训练与推理->分布式训练** 进入分布式训练任务列表页。
2. 在目标训练任务的操作项中点击 **【更多】** 菜单，选择 **【Tensorboard】** 入口，跳转至 Tensorboard 面板。
3. 进入 Tensorboard 面板后，查看训练过程和分析报告。

> 已失败、成功、运行中的任务均支持通过 Tensorboard 查询训练效果。



支持自定义监控指标采集

概述

百舸对接百度智能云 Prometheus 监控服务（CProm），通过自定义端口的方式支持分布式训练自定义指标的采集。用户只需在代码中暴露相关指标，并且在创建任务时显式声明指标的端口以及路径，平台可自动将该路径下的指标数据采集到资源池关联的 CProm，后续用户可在 Grafana 配置监控大盘查询。

前提条件

1. 分布式训练任务所在的资源池已经关联 CProm。
2. 已准备可用于提交分布式训练任务的资源池和资源队列。
3. 用户需要在代码中主动暴露指标，实现一个 HTTP 接口，该接口返回符合 Prometheus 文本格式 的指标数据。

操作步骤

创建任务并开启 Metrics 类型端口

1. 登录 百舸AI计算平台AIHC控制台
2. 进入 **分布式训练** 页面，点击 **创建任务** 按钮
3. 在 **环境信息** 模块，添加Metrics 类型的端口，用户需要填写指标的暴露的端口以及路径

> 说明：开启Metrics类型的端口后，平台会将该路径下的指标数据采集到资源池关联的 Cprom监控实例。请注意：此类数据指标属于付费指标，计费规则请参考： Cprom 计费概述

步骤 3：提交训练任务

完成任务名称、镜像地址、执行命令、资源配置、队列等必填项后，点击 **【确定】** 提交训练任务。如果队列无可选资源，请先完成资源池和资源队列准备后再提交。

采集指标验证

在 CProm 控制台，登录 CProm 监控实例绑定的 Grafana 来查询验证指标。

> 本地的 Grafana 访问 CProm 监控示例，请参考 这里

步骤 1：进入 CProm 监控实例绑定的 Grafana

登录 CProm 控制台，进入资源池关联的 Prometheus 监控实例详情页，打开该实例绑定的 Grafana。

步骤 2：使用 Explore 查询指标

登录 Grafana 后，点击 **【Explore】**，选择 CProm 对应的数据源，在查询输入框中输入待查询指标名称并执行查询。若页面返回时序数据或表格数据，表示 Prometheus 已成功抓取该指标。

配置Grafana监控大盘

登录 Grafana ，进入 创建dashboard 的流程：

1. 登录 Grafana → 点击左侧菜单栏 Dashboards → 选择 New → 点击 Import。
2. 选择以下任一导入方式：（本文档使用本地导入 train_grafana_dashboard.json）
   - 上传 JSON 文件：点击「Upload JSON file」→ 选择本地 Dashboard 的文件。
   - 输入 Grafana.com ID/URL：若使用官方或社区模板，粘贴模板 ID或模板 URL → 点击 Load。
   - 粘贴 JSON 文本：将 Dashboard 的 JSON 内容复制到文本框 → 点击 Load。
3. 加载完成后，配置关键参数：
   - Name：Dashboard 的名称
   - Folder：Dashboard存放的文件夹，方便分类管理。
   - UID：保持自动生成即可，如需通过 API 调用可自定义。
4. 点击 Import → 完成 Dashboard 导入。

附录：监控指标内置标签说明

百舸平台在监控指标体系中内置的标准标签（label），用于对训练任务与资源进行统一标识与多维度区分。平台会自动在系统级与自定义监控指标中附加这些标签，无需用户手动维护，主要用于任务溯源、按租户、队列、地域等维度进行统计与运维分析。

各标签含义说明如下。

标签名	说明
job_name	训练任务名称，用于区分不同训练作业。
jobid	训练任务唯一 ID，用于任务级精确定位与问题排查。
clusterID	集群 ID，仅自运维资源池有此标签。
aihc_id	托管资源池 ID，用于标识任务运行的百舸托管资源池。
workload_type	工作负载类型，训练默认为 train。
region	地域信息，标识任务所在的云区域。
user_name	提交任务的用户名，用于展示与审计。
user_id	用户唯一标识，用于多租户隔离与统计分析。
custom_monitor	标识是否开启自定义指标能力，值为 true 表示任务上报了自定义监控指标。
pod_name	运行任务的 Pod 名称，用于定位具体运行实例。
namespace	Pod 所在命名空间，区分不同业务空间。
queue	托管资源池：队列 ID；自运维资源池：队列名称，用于调度与排队策略分析。
train_framework	训练框架类型，例如 pytorch、mpi、tensorflow 等，用于框架级统计。

训练容错

概述

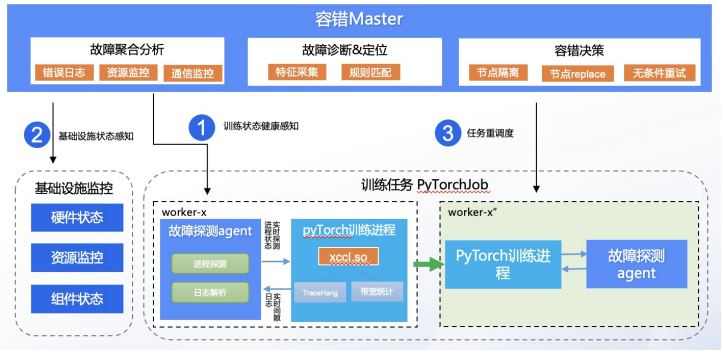
本文档介绍百舸 AI 计算平台训练任务自动容错能力的功能说明、使用前提、开启方式，以及容错通知和容错事件的查看方法。在大模型训练场景中，大多数训练任务是多机多卡、结合多种并行策略的分布式训练，通常训练规模大、运行时间长。训练过程中，可能会出现由于基础设施环境偶发异常导致训练业务中断的情况。为保障大规模分布式训练任务稳定运行，百舸 AI 计算平台提供任务自动容错能力，支持对训练任务进行异常感知、容错判断和自动恢复。

功能说明

为了保障大规模分布式训练任务的稳定运行，百舸提供容错监控、诊断定位和自动恢复能力。

百舸容错架构由故障探测 Agent 和中心化的 Master（资源池维度）组成：

- 故障探测 Agent 作为 sidecar 运行在客户业务 Pod 内，与训练进程完全独立，用于探测训练进程。
- 中心化的 Master 组件汇聚故障探测 Agent 的探测信息，从任务全局视角诊断、定位异常，并最终做出相应决策，使任务能够快速从异常中恢复。



目前百舸训练容错可以覆盖多类场景的故障感知定位和自动恢复能力：

- 训练进程报错异常退出。
- 训练心跳失联。
- NCCL/CUDA 系统报错日志。
- Pod 被误删除或驱逐。
- 训练进程 hang。

前提条件

- 仅支持 PyTorch 训练框架。
- 训练任务的 `restartPolicy` 需配置为 `Never`。
- 资源池已经开启 自动故障隔离能力。
- 资源池已经安装百度云 CCE Deep Learning Frameworks Operator 组件和 CCE AI Job Scheduler 组件：
  - CCE Deep Learning Frameworks Operator：主流深度学习框架 operator 组件，要求组件版本 >= 1.6.10。
  - CCE AI Job Scheduler：任务调度组件，支持调度管理各种 AI 任务，要求组件版本 >= 1.7.7。
- 通过 CLI 提交任务前，需准备运行中的资源池、可用队列和可访问的数据源。

> 百舸资源池已经默认安装以上组件，无需手动安装。

提交任务&开启容错

通过百舸控制台提交任务

在百舸控制台 **创建任务** > **容错&诊断**，开启容错。示例如下：



高级配置说明

参数	说明
任务 hang 检测	用于检测任务运行时是否出现 Hang 异常。如果所有实例的 Stdout 和 Stderr 日志在指定时间内没有更新，平台会初步判断任务 hang，并在任务事件中记录。可结合任务告警进行通知。
自定义训练异常	适用于用户自定义训练异常并触发容错的场景。训练进程出现自定义异常时，可在训练日志中打印预设关键字。平台在训练实例日志中精确匹配到关键字后，会启动训练容错。
无条件重试	任务在未检测到节点故障的场景下失败时，可开启无条件重试，通过重调度任务尝试容错恢复。重试次数最大可设置为 3 次。
Replace 容错模式	容错时仅销毁并重调度故障 Pod，正常 Pod 不受影响，可减少任务重启恢复时长。

> **注意：** 请合理设置容错与重试策略，避免任务被频繁容错重启。

Replace 容错模式说明

1. 开启 Replace 容错模式后，百舸会尝试替换训练镜像中的 elastic agent，并将任务切换为 c10d 模式。
2. 开启要求：PyTorch 版本 >= 2.0.0，Python 版本 >= 3.8.0。
3. 兼容性检查：开启 Replace 容错模式后，系统会自动检测 PyTorch 和 Python 版本。若符合要求，则启用 **Replace 容错模式**；否则自动使用默认容错模式。
4. 限制：如任务 master0 实例故障，则 Replace 容错模式不生效，并自动退化为 **默认容错模式**。

通过 CLI 提交任务并开启容错

AIHC CLI v1.6.0 使用 `job create` 子命令创建训练任务。开启容错时，可配置 `--enable-fault-tolerance`；如需兼容已有任务脚本，也可保留 `--env faulttolerance=true`。

```
aihc job create -p <资源池 ID> -q <队列 ID 或队列名称> \
--name <任务名称> \
--priority <优先级> \
--script-file <训练脚本路径> \
--image <训练镜像地址> \
--label <标签键>=<标签值> \
--gpu <GPU 资源名称>=<数量> \
--env CUDA_DEVICE_MAX_CONNECTIONS=1 \
--env faulttolerance=true \
--enable-fault-tolerance \
--ds-type <数据源类型> \
--ds-name <数据源名称> \
--ds-mountpath <数据源挂载路径>
```

容错监控

容错通知

在 **创建任务** > **高级配置** > **告警** 的选项中，配置任务容错告警通知，支持通过邮件、短信、电话、钉钉、企业微信、飞书等机器人 Webhook 地址等通知方式，可参考： 配置训练任务的消息通知

容错事件

训练任务开启自动容错后，当任务异常触发容错判断时，会在任务层面生成容错的事件记录，记录容错发生的时间以及原因。

用户可以通过 **百舸控制台** > **训练任务** > **任务事件** 查询详细信息

RDMA的使用

如何验证镜像是否支持RDMA

概述

本文档介绍如何验证当前镜像是否支持 RDMA 能力。您可以根据以下步骤，确认某个镜像是否满足 RDMA 使用条件。当前主流训练容器镜像通常基于 Ubuntu 构建，本文以 Ubuntu 容器环境为例进行说明。

> **说明：** 百舸分布式训练中，预置的 PyTorch 镜像已经默认支持 RDMA 能力，推荐基于百舸预置的 PyTorch 镜像构建您的自定义镜像。

步骤一：测试环境配置

进入路径：**百舸 AI 异构计算平台->分布式训练任务->目标任务->WebShell**。

1. 执行如下命令安装测试软件包。

```
apt update && apt install -y infiniband-diags
```

如果 apt update 长时间阻塞或失败，请先检查容器网络、软件源和代理配置，确认软件源可访问后再继续执行。

2. 使用 ibstatus 命令查看网卡速率。这里我们测试的是A800实例。可以看到本例中网卡（mlx5_1）速率（rate）为 100Gb/s，这是符合预期的。

```
 ###### ibstatus
Infiniband device 'mlx5_0' port 1 status:
    default gid:     0000:0000:0000:0000:0000:0000:0000:0000
    base lid:     0x0
    sm lid:         0x0
    state:         4: ACTIVE
    phys state:     5: LinkUp
    rate:         100 Gb/sec (4X EDR)
    link_layer:     Ethernet

Infiniband device 'mlx5_1' port 1 status:
    default gid:     0000:0000:0000:0000:0000:0000:0000:0000
    base lid:     0x0
    sm lid:         0x0
    state:         4: ACTIVE
    phys state:     5: LinkUp
    rate:         100 Gb/sec (4X EDR)
    link_layer:     Ethernet

Infiniband device 'mlx5_2' port 1 status:
    default gid:     0000:0000:0000:0000:0000:0000:0000:0000
    base lid:     0x0
    sm lid:         0x0
    state:         4: ACTIVE
    phys state:     5: LinkUp
    rate:         100 Gb/sec (4X EDR)
    link_layer:     Ethernet
```

3. 执行如下命令检查是否安装 RDMA 相关库。

```
dpkg -l perftest ibverbs-providers libibumad3 libibverbs1 libnl-3-200 libnl-route-3-200 librdmacm1
```

输出示例

```
 ###### dpkg -l perftest ibverbs-providers libibumad3 libibverbs1 libnl-3-200 libnl-route-3-200 librdmacm1
dpkg-query: no packages found matching perftest
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                    Version      Architecture Description
+++-=======================-============-============-===============================================================
ii  ibverbs-providers:amd64 39.0-1       amd64        User space provider drivers for libibverbs
ii  libibumad3:amd64        39.0-1       amd64        InfiniBand Userspace Management Datagram (uMAD) library
ii  libibverbs1:amd64       39.0-1       amd64        Library for direct userspace use of RDMA (InfiniBand/iWARP)
ii  libnl-3-200:amd64       3.5.0-0.1    amd64        library for dealing with netlink sockets
ii  libnl-route-3-200:amd64 3.5.0-0.1    amd64        library for dealing with netlink sockets - route interface
ii  librdmacm1:amd64        39.0-1       amd64        Library for managing RDMA connections
```

上述输出信息中包含了已安装（如 ibverbs-providers:amd64、 libibumad3:amd64 等）和未安装（ perftest）的软件。 如有软件包未安装，请继续执行第4步的操作安装软件；如已经安装全部软件，则可以直接验证是否支持RDMA

4. 执行如下命令安装 RDMA 相关软件包。

```
apt update && apt install -y perftest ibverbs-providers libibumad3 libibverbs1 libnl-3-200 libnl-route-3-200 librdmacm1
```

5. 执行如下命令再次查看软件包安装情况

```
dpkg -l perftest ibverbs-providers libibumad3 libibverbs1 libnl-3-200 libnl-route-3-200 librdmacm1
```

输出示例

```
 ###### dpkg -l perftest ibverbs-providers libibumad3 libibverbs1 libnl-3-200 libnl-route-3-200 librdmacm1
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                    Version      Architecture Description
+++-=======================-============-============-===============================================================
ii  ibverbs-providers:amd64 39.0-1       amd64        User space provider drivers for libibverbs
ii  libibumad3:amd64        39.0-1       amd64        InfiniBand Userspace Management Datagram (uMAD) library
ii  libibverbs1:amd64       39.0-1       amd64        Library for direct userspace use of RDMA (InfiniBand/iWARP)
ii  libnl-3-200:amd64       3.5.0-0.1    amd64        library for dealing with netlink sockets
ii  libnl-route-3-200:amd64 3.5.0-0.1    amd64        library for dealing with netlink sockets - route interface
ii  librdmacm1:amd64        39.0-1       amd64        Library for managing RDMA connections
ii  perftest                4.4+0.37-1   amd64        Infiniband verbs performance tests
```

步骤二：验证镜像是否支持 RDMA

进入路径：**百舸 AI 异构计算平台->分布式训练任务->目标任务->WebShell**。

在分布式训练任务中，以调试的形式进入容器 WebShell。按以下步骤进行验证。如果测试环境配置已完成，且容器具备 RDMA 资源，但命令报错或者结果不符合预期，则镜像可能不支持 RDMA。您可以更换为官方镜像，或重新安装相关软件包并检查是否遗漏配置项。

1. 在容器中执行以下脚本获取 GID 信息。

```
 ###### !/bin/bash
#
###### Copyright (c) 2016 Mellanox Technologies. All rights reserved.
#
###### This Software is licensed under one of the following licenses:
#
###### 1) under the terms of the "Common Public License 1.0" a copy of which is
###### available from the Open Source Initiative, see
###### http://www.opensource.org/licenses/cpl.php.
#
###### 2) under the terms of the "The BSD License" a copy of which is
###### available from the Open Source Initiative, see
###### http://www.opensource.org/licenses/bsd-license.php.
#
###### 3) under the terms of the "GNU General Public License (GPL) Version 2" a
###### copy of which is available from the Open Source Initiative, see
###### http://www.opensource.org/licenses/gpl-license.php.
#
###### Licensee has the right to choose one of the above licenses.
#
###### Redistributions of source code must retain the above copyright
###### notice and one of the license notices.
#
###### Redistributions in binary form must reproduce both the above copyright
###### notice, one of the license notices in the documentation
###### and/or other materials provided with the distribution.
#
###### Author: Moni Shoua <monis@mellanox.com>
#

black='\E[30;50m'
red='\E[31;50m'
green='\E[32;50m'
yellow='\E[33;50m'
blue='\E[34;50m'
magenta='\E[35;50m'
cyan='\E[36;50m'
white='\E[37;50m'

bold='\033[1m'

gid_count=0

###### cecho (color echo) prints text in color.
###### first parameter should be the desired color followed by text
function cecho ()
{
        echo -en $1
        shift
        echo -n $*
        tput sgr0
}

###### becho (color echo) prints text in bold.
becho ()
{
        echo -en $bold
        echo -n $*
        tput sgr0
}

function print_gids()
{
        dev=$1
        port=$2
        for gf in /sys/class/infiniband/$dev/ports/$port/gids/* ; do
                gid=$(cat $gf);
                if [ $gid = 0000:0000:0000:0000:0000:0000:0000:0000 ] ; then
                        continue
                fi
                echo -e $(basename $gf) "\t" $gid
        done
}

echo -e "DEV\tPORT\tINDEX\tGID\t\t\t\t\tIPv4  \t\tVER\tDEV"
echo -e "---\t----\t-----\t---\t\t\t\t\t------------  \t\t---\t---"
DEVS=$1
if [ -z "$DEVS" ] ; then
        DEVS=$(ls /sys/class/infiniband/)
fi
for d in $DEVS ; do
        for p in $(ls /sys/class/infiniband/$d/ports/) ; do
                for g in $(ls /sys/class/infiniband/$d/ports/$p/gids/) ; do
                        gid=$(cat /sys/class/infiniband/$d/ports/$p/gids/$g);
                        if [ $gid = 0000:0000:0000:0000:0000:0000:0000:0000 ] ; then
                                continue
                        fi
                        if [ $gid = fe80:0000:0000:0000:0000:0000:0000:0000 ] ; then
                                continue
                        fi
                        _ndev=$(cat /sys/class/infiniband/$d/ports/$p/gid_attrs/ndevs/$g 2>/dev/null)
                        __type=$(cat /sys/class/infiniband/$d/ports/$p/gid_attrs/types/$g 2>/dev/null)
```

```
                _type=$(echo $__type| grep -o "[Vv].*")
                if [ $(echo $gid | cut -d ":" -f -1) = "0000" ] ; then
                    ipv4=$(printf "%d.%d.%d.%d" 0x${gid:30:2} 0x${gid:32:2} 0x${gid:35:2} 0x${gid:37:2})
                    echo -e "$d\t$p\t$g\t$gid\t$ipv4  \t$_type\t$_ndev"
                else
                    echo -e "$d\t$p\t$g\t$gid\t\t\t$_type\t$_ndev"
                fi
                gid_count=$(expr 1 + $gid_count)
            done #g (gid)
        done #p (port)
done #d (dev)

echo n_gids_found=$gid_count
```

输出示例

```
 ###### sh cmd.sh
DEV    PORT    INDEX    GID                    IPv4          VER    DEV
---    ----    -----    ---                    ------------  ---    ---
mlx5_0    1    44    0000:0000:0000:0000:0000:ffff:ac10:0942    172.16.9.66     v1    eth0
mlx5_0    1    45    0000:0000:0000:0000:0000:ffff:ac10:0942    172.16.9.66     v2    eth0
mlx5_1    1    4    0000:0000:0000:0000:0000:ffff:1912:0106    25.18.1.6      v1    roce1
mlx5_1    1    5    0000:0000:0000:0000:0000:ffff:1912:0106    25.18.1.6      v2    roce1
mlx5_2    1    4    0000:0000:0000:0000:0000:ffff:1912:0116    25.18.1.22      v1    roce2
mlx5_2    1    5    0000:0000:0000:0000:0000:ffff:1912:0116    25.18.1.22      v2    roce2
mlx5_3    1    4    0000:0000:0000:0000:0000:ffff:1912:0126    25.18.1.38      v1    roce3
mlx5_3    1    5    0000:0000:0000:0000:0000:ffff:1912:0126    25.18.1.38      v2    roce3
mlx5_4    1    4    0000:0000:0000:0000:0000:ffff:1912:0136    25.18.1.54      v1    roce4
mlx5_4    1    5    0000:0000:0000:0000:0000:ffff:1912:0136    25.18.1.54      v2    roce4
mlx5_5    1    4    0000:0000:0000:0000:0000:ffff:1912:0166    25.18.1.102     v1    roce5
mlx5_5    1    5    0000:0000:0000:0000:0000:ffff:1912:0166    25.18.1.102     v2    roce5
mlx5_6    1    4    0000:0000:0000:0000:0000:ffff:1912:01c6    25.18.1.198     v1    roce6
mlx5_6    1    5    0000:0000:0000:0000:0000:ffff:1912:01c6    25.18.1.198     v2    roce6
mlx5_7    1    4    0000:0000:0000:0000:0000:ffff:1912:0216    25.18.2.22      v1    roce7
mlx5_7    1    5    0000:0000:0000:0000:0000:ffff:1912:0216    25.18.2.22      v2    roce7
mlx5_8    1    4    0000:0000:0000:0000:0000:ffff:1912:0246    25.18.2.70      v1    roce8
mlx5_8    1    5    0000:0000:0000:0000:0000:ffff:1912:0246    25.18.2.70      v2    roce8
n_gids_found=18
```

2. 在容器中输入以下测试命令，启动 ib_write_bw 服务端。

```
 ###### ib_write_bw -d mlx5_1 -x <gid>  -p 18516
```

信息从上述第一步的返回中获取，这里我们测试 mlx5_1网卡，选择v2版本，这里 gid 为 5。

返回结果如下时，表示服务端已启动并等待客户端连接：

```
 ************************************
* Waiting for client to connect... *
************************************
```

3. 保持服务端进程运行，在同一容器的另一个终端中继续输入如下命令。

```
 ib_write_bw -d mlx5_1 127.0.0.1 -x <gid> -p 18516 --report_gbits //<gid>:请求端gid需和服务端一致
```

输出示例

```
 ###### ib_write_bw -d mlx5_1 127.0.0.1 -x 5 -p 18516 --report_gbits
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF        Device         : mlx5_1
 Number of qps   : 1        Transport type : IB
 Connection type : RC        Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API     : ON
 TX depth        : 128
 CQ Moderation   : 1
 Mtu             : 4096[B]
 Link type       : Ethernet
 GID index       : 5
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0xb1e5 PSN 0x893dde RKey 0x016006 VAddr 0x007efdfae11000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:25:18:01:06
 remote address: LID 0000 QPN 0xb1e4 PSN 0x344ea9 RKey 0x007905 VAddr 0x007f9f69f3d000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:25:18:01:06
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
 65536      5000             100.77             100.77             0.192207
---------------------------------------------------------------------------------------
```

带宽值（BW peak、BW average）在100Gb/s左右，符合预期。

这里不同GPU机型的RDMA带宽规格不同，您可以从机器套餐规格中获取。

如无输出或报错说明镜像不支持RDMA，请重新安装相关软件包，检查是否有配置项的遗漏。

RDMA：使用高性能网络进行分布式训练

概述

本文档介绍在百舸环境下使用 RDMA 网络进行分布式训练的方法。

RDMA（Remote Direct Memory Access）是新一代网络通信技术，允许计算机之间直接进行内存到内存的数据传输，而不需要经过操作系统或中央处理器处理。在大规模分布式训练中，使用 RDMA 可以有效降低网络传输中服务器端数据处理的延迟，实现高吞吐、低延迟的网络通信，提升训练效率。

NCCL 是 NVIDIA 的集合通信库，能实现 Collective 通信和点对点通信。NCCL 内部已经实现 RDMA 通信，同时 NCCL 可以根据环境中的网卡类型和拓扑关系，自行选择最优的通信路径，目前主流分布式训练框架都已支持 NCCL。

前提条件

- 已创建百舸集群，且集群中至少有 2 台具有 RDMA 网络的 GPU 实例。
- GPU 实例镜像中包含 OFED 和 NVIDIA 驱动。推荐使用百度智能云提供的 GPU 镜像，该镜像已包含 OFED 驱动，无需手动安装。
- 业务镜像中需要使用 NCCL 依赖库。推荐使用 NVIDIA GPU Cloud（NGC）提供的基础镜像。NGC 提供的基础镜像通常包含 NCCL 依赖库，并且已经预先配置和优化常用的深度学习框架和工具。使用 NGC 基础镜像可以简化设置和配置过程，并确保能够顺利使用 NCCL 进行 GPU 加速计算和深度学习任务。

提交任务开启RDMA

1. 登录 百舸AI计算平台AIHC控制台
2. 进入 **分布式训练** 列表页面，点击 **创建任务**， 进入创建任务的流程
3. 在 **创建任务 > 资源配置** 模块，选择 **开启RDMA** 即可，开启后系统将训练实例自动调度任务到支持RDMA网络的节点上，启用RDMA加速

平台预置 NCCL 环境变量

开启 RDMA 后，百舸平台基于百度智能云内部大规模分布式训练经验，在任务运行时会自动注入推荐的 NCCL 环境变量，无需手动填写。您也可以根据业务场景自定义调整。

> 强烈建议您使用平台预置系统内的 NCCL 默认变量，可以获得较优性能和较高的稳定性。

A800/A100 NCCL 预置环境变量

环境变量	网络类型
NCCL_IB_HCA=动态值	RoCE/IB
NCCL_SOCKET_IFNAME=动态值	RoCE/IB
NCCL_IB_GID_INDEX=动态值	RoCE/IB
NCCL_TOPO_FILE=动态值	RoCE/IB（仅 BCC 形态自动注入）
NCCL_IB_TIMEOUT=22	RoCE/IB
NCCL_IB_QPS_PER_CONNECTION=8	RoCE/IB
NCCL_DEBUG=INFO	RoCE/IB
NCCL_IB_ADAPTIVE_ROUTING=1	IB
NCCL_DEBUG_SUBSYS=INIT,ENV,GRAPH	RoCE/IB

H800 NCCL 预置环境变量

环境变量	网络类型
NCCL_IB_HCA=动态值	RoCE/IB
NCCL_SOCKET_IFNAME=动态值	RoCE/IB
NCCL_IB_GID_INDEX=动态值	RoCE/IB
NCCL_TOPO_FILE=动态值	RoCE/IB（仅 BCC 形态自动注入）
NCCL_IB_TIMEOUT=22	RoCE/IB
NCCL_IB_QPS_PER_CONNECTION=1	RoCE/IB
NCCL_DEBUG=INFO	RoCE/IB
NCCL_IB_ADAPTIVE_ROUTING=1	RoCE/IB
NCCL_DEBUG_SUBSYS=INIT,ENV,GRAPH	RoCE/IB

NCCL 环境变量说明

NCCL 关键环境变量说明如下表所示，更多 NCCL 环境变量说明，请参见 NCCL。

环境变量	说明
NCCL_IB_HCA	环境中的 RDMA 网卡。
NCCL_SOCKET_IFNAME	指定用于通信的 IP 接口。
NCCL_IB_GID_INDEX	设置 RDMA 通信优先级。
NCCL_IB_DISABLE	是否关闭 RDMA 通信：0 代表 NCCL 使用 IB/RoCE 传输；1 代表禁止 NCCL 使用 IB/RoCE 传输，此时 NCCL 将退回到使用 IP 套接字。
NCCL_DEBUG	NCCL 日志级别，此变量通常用于调试。
NCCL_IB_TIMEOUT	网络断点重连超时时间，计算公式：4.096 µs * 2 ^ timeout。
NCCL_IB_RETRY_CNT	网络断点重连重试次数。
NCCL_DEBUG_FILE	日志输出文件。
NCCL_IB_QPS_PER_CONNECTION	每个 IB queue pair 的并发连接数。
NCCL_IB_ADAPTIVE_ROUTING	NVIDIA NCCL 是否使用 InfiniBand 自适应路由功能的环境变量，默认是 0 或 false，配置为 1 或 true 表示开启自适应路由。
NCCL_TOPO_FILE	网络拓扑文件，NCCL 通信库会依据网络拓扑文件构建拓扑关系。

训练任务Hang检测

概述

本文档介绍百舸 AI 计算平台 AIHC 的 Hang 检测能力。Hang 检测面向大规模分布式训练场景，可在任务运行期间自动巡检、实时感知异常挂起，并通过函数调用栈分析快速定位性能瓶颈与故障根因。面向算法工程师与任务运维人员，该功能可降低分布式训练 Hang 问题排查成本，提升任务运行效率与稳定性。

前提条件

- 已具备百舸 AI 计算平台 AIHC 控制台登录权限，以及分布式训练任务的创建或查看权限。
- 执行 Hang 分析前，需已有可进入详情页的分布式训练任务。

功能介绍

- 支持多维度 Hang 检测：在基于日志迭代间隔判断是否 Hang 的基础上，支持结合训练进程调用栈、进程 IO 等多维度指标综合判断，提升 Hang 检测的准确性。
- Hang 分析能力：
  - **无侵入采集** 无需修改业务代码或用户代码，自动采集训练进程、DataLoader 子进程调用栈，支持手动或自动触发指标采集。
  - **可视化分析定位** 自动生成函数聚合火焰图，支持按进程类型筛选。通过多 Rank 聚合函数火焰图对比分析，快速锁定异常节点、异常函数。

使用限制

1. 当前仅支持 PyTorch 框架的训练任务。
2. 仅支持申请了整机 GPU 的训练任务。

使用步骤

开启 hang 检测

1. 登录 百舸AI计算平台AIHC控制台。
2. 进入 **分布式训练** 列表页面，点击 **创建任务**。
3. 资源配置模块，训练框架选择 **Pytorch**
4. 容错与诊断模块，开启自动容错，在高级设置中，开启 hang检测：

- 基于日志的检测，可以自定义日志间隔的阈值
- 高级诊断，在日志检测达到阈值后，平台会额外通过函数调用栈与进程IO监控指标，多维度判断任务状态，提高 hang 检测的准确度

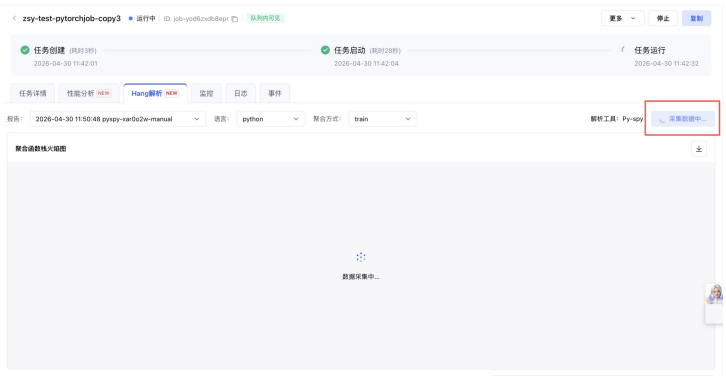


任务启动运行后，会实时检测 hang。

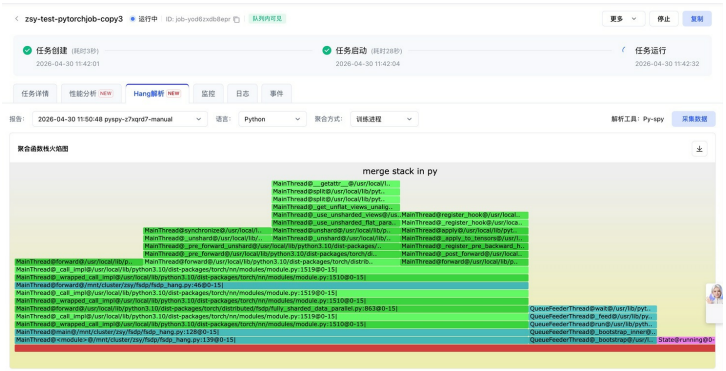
hang 分析

在任务详情页面，点击进入 **Hang 解析** 标签页，即可查看训练任务hang 的分析报告

1. 若您在创建任务时 **已开启 Hang 检测**，系统检测到任务挂起后，将 **自动触发指标采集**，并生成聚合函数栈火焰图。
2. 同时本功能 **支持手动触发采集**，无需提前开启 Hang 检测，当你发现任务 Hang 时可手动发起诊断分析



数据采集完成后，会自动生成聚合函数栈火焰图，支持下载。



视图使用说明

1. 火焰图文件都合并了所有rank的数据。

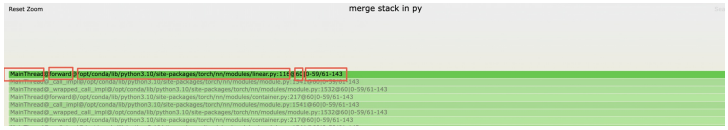
2. 栈帧格式：

```
1栈帧格式：
2<ThreadName>@<FuncName>@<CodePath>@<Ranks In There>|<Ranks Not In There>
```

   - `ThreadName`：线程名，采集不到会显示unknown
   - `FuncName`：函数名，如果获取不到会显示unknown
   - `CodePath`：函数执行路径
   - `Ranks In There`：哪些rank的调用栈执行到了这个位置
   - `Ranks Not In There`：哪些rank的调用栈没有执行到这个位置

3. 为了提升视图效果， `Ranks In There` 和 `Ranks Not In There` 的rank列表呈现逻辑为：
   - 压缩连续的rank号，格式为：开始rank号-结束rank号。例如0-5代表 [rank-0,rank-1,rank-2,rank-3,rank-4]
   - 隔离非连续的rank号，格式为：第1组rank/第2组rank。例如60-62/65代表[rank-60,rank-61,rank-62,rank-65]

4. 火焰图示例图如下：
   - rank-60的主线程（MainThread) **执行** 到了/opt/conda/.../linear.py:116处的forward函数
   - 0-59号rank和61-143号rank的主线程（Main Thread） **没有执行** 到/opt/conda/.../linear.py:116处的forward函数



分析逻辑

先看整体：所有 Rank 的调用栈是否一致

- 一致 → **全局 Hang**（代码 / 通信问题）
- 不一致 → **局部节点异常**（单 Rank / 单节点先 Hang）

再看栈顶函数

- 卡在 **业务 / 数据加载 / IO / 计算函数** → 应用层代码问题
- 卡在 **通信函数（all_reduce 等）** → 通信等待或全局阻塞

最后定位异常 Rank

- 栈与其他 Rank 明显不同的 → **故障节点**
- 其他 Rank 都在等它通信 → 根因就是这个异常 Rank

排查实践

Case1 局部hang

1. **场景描述**

节点（某个rank或pod内的一组rank）由于发生了非通信问题先hang，进而扩散，导致全局hang。

2. **分析目标**

分析出异常节点，具体是哪一个rank或哪一组rank。

3. **分析逻辑**

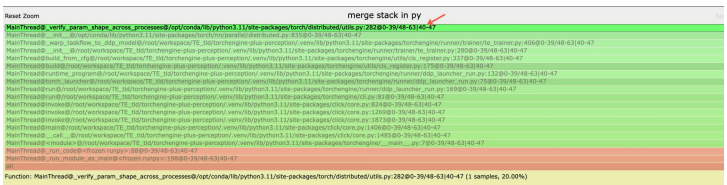
基本原理：异常rank的调用栈与其它rank的调用栈理论存在明显差异。因为异常rank是hang在了非通信问题，所以异常rank的调用栈理论停滞在非通信函数。而其它rank的hang是由于等待同异常rank通信，所以其它rank调用栈理论停滞在通信函数。

4. **经典案例**

规模为8机64卡的训练任务，判断处于 hang状态

- 查看python栈聚合函数火焰图，观察到栈顶函数，机器A的所Rank（rank40-47）调用栈停滞函数位置明显与其它rank存在差异
- 进入该训练实例，采集训练进程，发现 `Process 23x is disk-sleep`，磁盘处于休眠状态，I/O 阻塞导致了训练进程 hang 住。

可以结合节点的监控，进一步排查磁盘的IO 性能



```
Dumping on all gpu process:   0%|          | 0/8 [00:00<?, ?it/s]Process 236 is disk-sleep
Process 233 is disk-sleep
Process 231 is disk-sleep
Process 232 is disk-sleep
Process 237 is disk-sleep
Process 235 is disk-sleep
Process 230 is disk-sleep
Process 234 is disk-sleep
Dumping on all gpu process: 100%|##########| 8/8 [00:00<00:00, 53.73it/s]
Dumping on all gpu process:   0%|          | 0/10 [00:00<?, ?it/s]Process 234 is disk-sleep
Process 236 is disk-sleep
Process 231 is disk-sleep
Process 233 is disk-sleep
Process 232 is disk-sleep
Process 235 is disk-sleep
Process 230 is disk-sleep
Process 237 is disk-sleep
Dumping on all gpu process: 100%|##########| 10/10 [00:00<00:00, 53.28it/s]
```

case2 全局hang

1. **场景描述**

用户代码编写问题导致的全局hang

---

2. **分析目标**

分析出异常函数

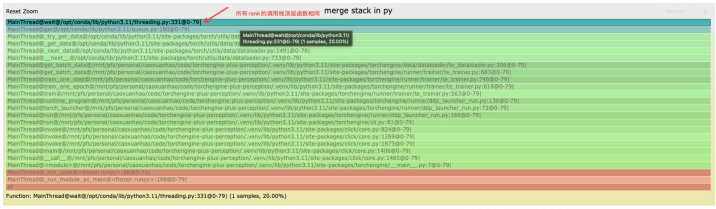
3. **分析逻辑**

由于是代码问题导致的全局hang，理论上所有rank调用栈都停滞在相同的函数。如果该函数为非通信函数（例如dataloader相关），可以推断是由用户代码导致

4. **经典案例**

10机分布式训练任务，判断处于 hang 状态

- 查看Python火焰图，明显看到所有rank的调用栈顶都停滞在相同的函数。该函数为dataloader相关函数，因此怀疑dataloader代码编写存在问题。需要进一步排查dataloader的函数



使用百舸智能助手进行排队任务的诊断排查

概述

百舸智能助手是百舸 AI 计算团队提供的基于大模型的全新助手功能。当前百舸智能助手为您提供训练任务排队场景的智能诊断能力，旨在帮助客户分析训练任务正在排队等待的可能原因，并给出指导建议。

基于百舸训练可观测体系的监控指标，结合百舸在大模型领域积累的专家知识经验，通过大语言模型的分析推理能力，进行任务排队原因的根因定位，提供更智能的产品使用体验。

您可以在任务列表页面针对状态为 **排队中** 的任务，点击 **【智能诊断】**，唤起百舸智能助手，快速诊断分析任务排队的可能原因。

前提条件

首次使用百舸智能助手，需要完成服务授权操作，允许百舸助手访问资源池内节点的系统日志以及任务元数据、事件等相关数据。

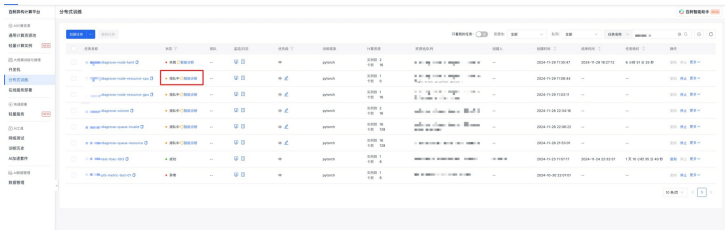
使用限制

- 调度器版本在 1.7.27 以上。
- 当前任务排队超过 10 min 时才可以点击 **【智能诊断】** 按钮。
- 当前任务如果在 FIFO 队列中，只有队首任务才可以点击 **【智能诊断】** 按钮。

操作步骤

以下是在百舸控制台进行训练任务排队中智能诊断的步骤：

1. 登录 百舸AI计算平台AIHC控制台。
2. 在左侧菜单栏选择“分布式训练”，进入任务列表页面，当任务处于“排队中”状态超过10min后，可以点击 **智能诊断**。



3. 点击目标任务操作列中的 **【智能诊断】** 后，页面会打开诊断浮窗。诊断结果包含任务基本信息、队列额度诊断、节点可用诊断、资源余量诊断 4 类信息，每个一级诊断项下会展示对应的二级诊断项。
4. 在诊断浮窗中展开存在异常的诊断项，可查看推荐操作、帮助文档和详细原因。您可以根据推荐操作和帮助文档处理任务排队问题，也可以继续展开详细原因定位具体配置或资源约束。

详细诊断项列表

一级分类	二级子分类	详细说明	针对场景
队列额度诊断	资源池队列配额诊断	针对当前等待原因，提供提高队列设置配额、降低任务申请配额等推荐操作和详细方法，供客户选择。	队列配额管理等问题
队列额度诊断	资源池队列其他诊断	针对当前等待原因，提供选择正确队列配置等推荐操作和详细方法，供客户选择。	父队列非法提交、错误队列提交等问题
节点可用诊断	节点可用性诊断	针对当前等待原因，提供故障自愈功能等推荐操作和详细方法，供客户选择。	节点硬件故障、节点封锁与管理、任务亲和性与反亲和性调度、节点 NodeSelector 等问题
节点可用诊断	节点污点容忍诊断	针对当前等待原因，提供节点污点管理等推荐操作和详细方法，供客户选择。	节点物理队列、节点系统打污点、节点用户打污点等问题
资源余量诊断	存储资源挂载诊断	针对当前等待原因，提供检查挂载点、检查挂载配置信息等推荐操作和详细方法，供客户选择。	PV/PVC 挂载点、不存在配置等问题
资源余量诊断	EHC、RDMA 网卡诊断	针对当前等待原因，提供检查 RDMA 资源配置、满足 TOR 亲和性调度策略、处理前序任务等推荐操作和详细方法，供客户选择。	非法 RDMA 申请、无资源申请、跨 EHC 调度、TOR 策略等问题
资源余量诊断	GPU 资源诊断	针对当前等待原因，提供整理集群资源碎片、处理前序任务和调整任务优先级等推荐操作和详细方法，供客户选择。	集群资源碎片、集群资源不足、资源描述符混用等问题
资源余量诊断	单节点 Pod 数上限诊断	针对当前等待原因，提供删除冗余 Pod 等推荐操作和详细方法，供客户选择。	可用 Pod 余量不足
资源余量诊断	其他资源诊断	针对当前等待原因，提供处理前序任务、调整任务优先级等推荐操作和详细方法，供客户选择。	子网 IP 不足、CPU 核数、内存大小、存储空间等问题

使用百舸智能助手进行失败任务的故障排查

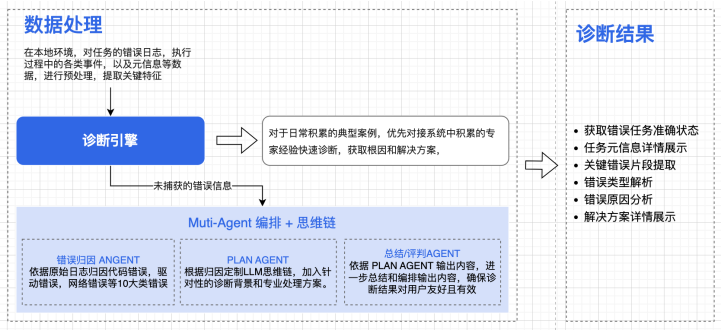
概述

本文档介绍百舸智能助手对训练失败任务的智能诊断能力。百舸智能助手是百舸 AI 计算团队提供的基于大模型的助手功能，当前支持在训练任务失败场景中分析可能原因，并给出处理建议。

更多能力正在加速集成中...

失败任务故障排查

基于百舸训练可观测体系的监控指标，结合百舸在大模型领域积累的专家知识经验，通过大语言模型的分析推理能力，进行任务失败问题的根因定位，提供了更智能的产品使用体验。



您可以在任务列表页面针对失败的任务，通过点击 **【智能诊断】**，唤起百舸智能助手，快速诊断分析任务失败的可能原因。

前提条件

- 首次使用百舸智能助手时，需要完成服务授权操作，允许百舸助手访问资源池内节点的系统日志以及任务元数据、事件等相关数据。

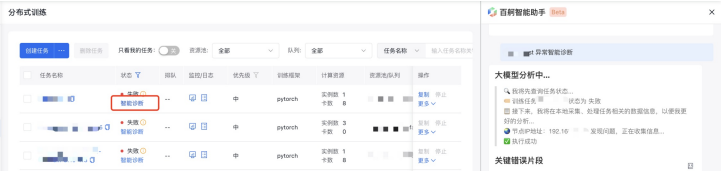
使用限制

- 仅支持失败任务的诊断。
- 仅支持 PyTorch 框架的任务。

操作步骤

以下是在百舸控制台进行训练任务异常诊断的步骤：

1. 登录 百舸AI计算平台AIHC控制台。
2. 进入 **分布式训练** 列表页面，选择希望诊断的失败任务。
3. 在任务状态栏下，点击 **智能诊断** 后，唤起百舸智能助手 快速发起诊断。输出关键的错误片段、可能的问题原因以及解决方案。



4. 诊断完成后，您可以对诊断效果进行评价。若没有解决问题，您也可以直接在诊断页面快速发起工单获取进一步的支持。

BCCL：百度自研高性能集合通信库

概述

本文档介绍 BCCL（Baidu Collective Communication Library）的主要特性和使用方法。BCCL 是百度智能云推出的面向大模型训练场景优化的集合通信库，基于开源 NCCL 进行了功能扩展和能力增强，针对大模型训练场景在可观测性、故障诊断、稳定性等方面进行优化，进一步提升集合通信库的可运维能力。

前提条件

- 已具备使用 百舸 AI 计算平台 AIHC 控制台 的账号和权限。
- 需要创建多机分布式训练任务。

- 训练镜像需满足 BCCL 兼容性要求。启用 BCCL 相关能力后，平台会对训练镜像 OS 和 BCCL 版本进行兼容性验证。

BCCL 增强的特性

- 支持集合通信带宽实时统计功能，为故障诊断排除、训练性能调优等提供数据支撑。
- 支持集合通信 hang 时故障诊断能力，快速进行任务的异常检测。
- 网络故障容错能力增强，针对偶发性异常故障场景，如单端口的偶发 updown，增加相应重试重传超次机制，提升训练任务的健壮性。

使用 BCCL

为了便于用户快捷使用 BCCL，百舸平台支持用户在创建训练任务时使用 BCCL。

使用说明

1. 仅支持多机的分布式训练任务开启。
2. 启用此能力后，平台会对训练镜像 OS 和 BCCL 的版本进行兼容性验证。如版本兼容验证通过，会在任务训练中自动安装并使用 BCCL 通信库；否则不做处理，使用训练镜像中指定的通信库。您可以在训练任务事件或任务详情页面，获取 BCCL 是否成功安装。
3. 当前控制台创建任务页面中，BCCL 关联配置位于 **容错与诊断** 模块的 **任务 hang 检测**。开启后，如果训练镜像中集成了 BCCL，任务 hang 时会同步在日志中记录集合通信状态信息，为异常诊断提供数据支持。

使用步骤

1. 登录 百舸AI计算平台AIHC控制台。
2. 进入 **分布式训练** 列表页面，点击 **创建任务**。
3. 在 **创建训练任务** > **BCCL通信库** 模块，开启使用BCCL通信库即可



> 支持通过 Open API 开启，详见：创建训练任务

推荐环境变量配置

环境变量	说明
NCCL_DEBUG=INFO	NCCL 日志级别，推荐值为 INFO。
NCCL_DEBUG_SUBSYS=INIT,ENV,GRAPH	NCCL 日志子系统配置。
NCCL_DEBUG_FILE=path/to/nccl.%h.%p	NCCL 默认日志路径。
BCCL_ERROR_FILE=path/to/err.nccl.%h.%p	将 NCCL ERROR 日志重定向到新路径，以便快速检索分析。
BCCL_IB_CONNECT_RETRY_CNT=15	增强任务初始化时网络连接容错能力，推荐取值为 10 ~ 20。
BCCL_BUS_BW_CALCULATE_MODE=None/Verbose/Agg	带宽统计功能开关。None 表示关闭带宽统计功能；Verbose 表示打印带宽统计详细日志，每个集合通信操作对应一行日志；Agg 表示打印带宽统计聚合日志，需配合 BCCL_BUS_BW_CALCULATE_AGG_INTERVAL 使用。默认值为 None。
BCCL_BUS_BW_CALCULATE_AGG_INTERVAL=15	带宽统计聚合功能的统计间隔，表示统计指定间隔内的集合通信带宽情况。默认值为 15s。
BCCL_PROFILING_FILE=path/to/profiling.nccl.%h.%p	不配置该环境变量时，带宽统计日志默认打印到 stdout。
BCCL_TRACE_HANG_ENABLE=1	TraceHang 功能开关。默认值为 0。

启用 BCCL_TRACE_HANG_ENABLE=1 后，任务 hang 时，可通过以下命令触发 TraceHang 日志信息打印：

```
find /var/run/ -type s -name "bccl_socket_*" -exec sh -c 'echo "TraceHang" | nc -U {}' \;
```

测试工具

bccl_perf 是基于 nccl-tests 扩展的集合通信测试工具，提供更丰富的测试语义。

参数	必填	说明
-O <collective_type>	是	指定需测试的通信操作，如 allReduce、allGather、reduce、broadcast、reduceScatter、alltoall、sendRecv、gather、scatter。
--split_mode	否	拆分多通信组测试，默认不开启。
--split_step	是	多通信组拆分策略，表示同一通信组 GPU rank 步长，默认为 1。step 为 8 时，GPU rank 0/8/16... 为一个通信组，GPU rank 1/9/17... 为一个通信组，以此类推。
-b	是	测试所用最小 message size，与 nccl-tests 保持一致。
-e	是	测试所用最大 message size，与 nccl-tests 保持一致。
-i	是	最小到最大 message size 的增长步长，与 nccl-tests 保持一致。
-f	是	最小到最大 message size 增长的步长因子，与 nccl-tests 保持一致。
-n	是	测试迭代次数，与 nccl-tests 保持一致。
-w <warmup_iters>	是	warmup 迭代次数，与 nccl-tests 保持一致。
-t	是	每个进程启动的线程数，与 nccl-tests 保持一致。
-g	是	每个线程管理的 GPU 数，与 nccl-tests 保持一致。
-z	是	阻塞 NCCL 集合通信，确保通信组所有进程均完成集合通信操作，与 nccl-tests 保持一致。

集合通信带宽统计

百舸平台支持实时集合通信带宽统计的功能，可以在训练过程中对集合通信性能进行实时观测，准确地展示集合通信在不同阶段的性能表现，为故障诊断排除、训练性能调优等提供数据支撑。

百舸平台已经默认集成集合通信带宽的能力，详见：训练集合通信带宽监控。

hang 诊断（TraceHang）

在大规模分布式训练中，训练任务可能会因为硬件、软件等原因而出现进程挂起（hang）的情况。此类问题的排查难点在于，无法从训练进程状态、日志上获取有效的诊断信息，且该场景一般不会立刻发生，任务可以正常启动并训练，但是在训练超过一定时间后，可能是几个小时或者数天，突然 hang 住。排查时很难稳定复现该故障，导致排查难度进一步提高。

为了应对这一挑战，BCCL 提供了近乎无损的方式实现训练 hang 场景的在线诊断，通过记录集合通信内部的通信状态，综合分析判断是否有节点出现了问题。

使用说明

1. TraceHang 开启后会额外占用约 2 MB 的 GPU 显存，这部分显存以统一内存（unified memory）进行分配。训练任务将显存打满时，不建议开启 TraceHang。如果训练任务将 GPU 显存打满，TraceHang 在访问统一内存时，会因统一内存页无法调入 GPU 显存，出现非法内存访问，即 `CUDA error: an illegal memory access was encountered`。
2. TraceHang 开启后，请不要再启动其他占用 GPU 显存的任务。原因：模型训练时，大部分时间在计算，通信占比很小，TraceHang 申请的统一内存只有在通信时使用。统一内存在不使用时，会被调出 GPU 显存，使用时再调入。如果后台启动其他 GPU 任务，可能会导致 GPU 显存打满，而显存打满后，统一内存页无法调入 GPU 显存，访问该内存空间时出现非法内存访问，即 `CUDA error: an illegal memory access was encountered`。

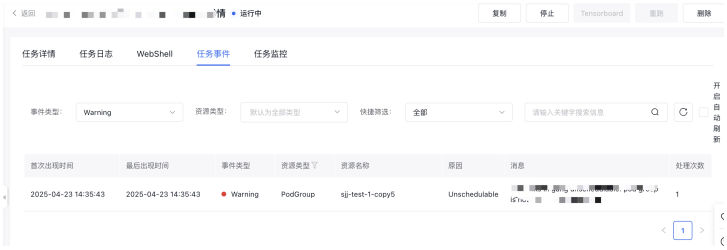
使用步骤

目前百舸平台已经自动集成BCCL Tracehang的能力，下面将介绍使用步骤。

1. 登录 百舸AI计算平台AIHC控制台。
2. 进入 **分布式训练** 列表页面，点击 **创建任务**。
3. 在 **创建训练任务** > **容错&诊断** 模块，开启任务hang检测，您可以自行配置发起hang检测的阈值。



4. 当训练任务达到您定义的hang的阈值，百舸平台会自动触发BCCL Tracehang。您可以在任务事件中，查询训练任务hang的诊断记录。



> 备注： 目前 BCCL TraceHang 的能力，无论是否诊断出问题节点，当前阶段都不会主动中止您的训练任务。您需要手动进一步处理。

FlashCkpt：大模型训练高性能ckpt保存

FlashCkpt 是百舸 AI 计算平台为 PyTorch 大模型训练场景而开发的高性能 Checkpoint 框架，实现接近 0 开销的模型状态保存。本文介绍如何使用百舸**FlashCkpt** 的能力，提升 Checkpoint 保存性能。

背景信息

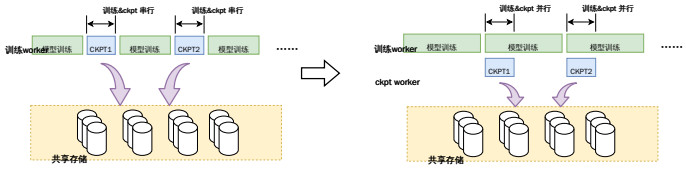
大模型训练随着参数量和数据规模的增长，受限于单机 GPU 内存容量限制，通常会使用分布式训练框架进行训练。但是分布式训练中可能会遇到单点节点故障、系统问题等难以预知的异常问题，造成训练任务中断。

在实际训练过程中，通常会开启定期 Checkpoint 机制来保存和恢复进度，尽量降低训练中断带来的算力浪费，但 Checkpoint 本身的耗时与模型大小成正比。对于百亿、千亿参数的大模型，单次 Checkpoint 的保存时间开销通常在几分钟到十几分钟之间。Checkpoint 间隔时间较短会使训练任务频繁暂停，GPU 算力严重浪费；Checkpoint 间隔时间较长，一旦发生中断，之前损失的迭代次数在恢复时需要重新计算。以 1 千卡为例，假设每 4 小时 Checkpoint 一次，单次保存耗时 10 分钟，那么每天因 Checkpoint 损失的 GPU 算力接近 1000 卡时。

因此，在大规模分布式训练时，需要一种低成本、可靠的方法来保存最新的 Checkpoint，降低时间和资源浪费，提升 GPU 的有效训练时长。

方案介绍

FlashCkpt 核心思路： Checkpoint 保存与模型训练过程重叠，降低 Checkpoint 保存占用 GPU 训练的有效时间。



收益

分别针对 7B、70B、130B 三种参数规模的模型训练进行对比测试，百舸 FlashCkpt 相比于原生 Ckpt 方案，单次保存耗时最高减少 93% 到 99%。

模型参数	ckpt 文件大小	训练环境配置	原生 ckpt 单次保存耗时（秒）	百舸 FlashCkpt 单次保存耗时（秒）
7B	88 GB	模型切分：1 TP 1 PP 8 DP 机器配置：NVIDIA A800-SMX4-80G，1 node，8 GPUs	35	**2.5**
70B	899 GB	模型切分：2 TP 8 PP 2 DP 机器配置：NVIDIA A800-SMX4-80G，4 node，32 GPUs	143	**4**
130B	1702 GB	模型切分：4 TP 6 PP 2 DP 机器配置：NVIDIA A800-SMX4-80G，6 node，48 GPUs	262	**3.8**

> 说明：
>
> 1. 训练镜像：`registry.baidubce.com/cce-ai-native/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.3.0_release`
> 2. 以上相同参数的模型，在 ckpt 保存耗时的对比测试中，采用相同的共享存储，排除存储性能的影响。

使用说明

使用限制

1. 内存占用：单机所有 GPU 卡的 Checkpoint 会优先保存在内存中，因此对于千亿参数规模模型，单机建议预留 200 GB 以上内存。

2. 框架支持：当前仅支持 Megatron 训练框架（<= 23.04 版本），DeepSpeed 支持中。

使用方式

提供以下两种使用方式：

- 通过 AIAK 加速镜像使用：适用于在训练中直接使用 AIAK-Training 加速镜像的用户。
- 通过代码包使用：适用于自定义训练镜像的用户。

通过 AIAK 镜像使用

FlashCkpt 的能力已经默认集成到 AIAK-Training 加速镜像 v1.2.3.0 及更高版本中，通过在启动命令中增加 `--enable-accelerate-checkpoint` 一键开启 FlashCkpt 加速。

1. AIAK-Training 加速镜像 v1.2.3.0 版本：

```
registry.baidubce.com/cce-ai-native/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.3.0_release  # NVIDIA A800 机型适用
registry.baidubce.com/cce-ai-native/aiak-megatron:ubuntu22.04-cu12.2-torch2.1.0-py310_v1.2.3.0_release  # NVIDIA H800 机型适用
```

2. 启动命令示例：

```
OTHER_ARGS="--data-path $DATA_PATH \
        --data-impl mmap \
        --split 949,50,1 \
        --save $CHECKPOINT_SAVE_PATH \
        --load $CHECKPOINT_LOAD_PATH \
        --enable-accelerate-checkpoint \
        --tensorboard-dir ${TENSORBOARD_PATH} \
        --distributed-backend nccl"
```

通过代码包使用

1. 下载加速包。

按容器 base 镜像选择对应加速包下载：

```
wget https://cce-ai-aihc.bj.bcebos.com/Checkpoint/AIDK_CentOS7-0.1.0-py3-none-any.whl
wget https://cce-ai-aihc.bj.bcebos.com/Checkpoint/AIDK_Ubuntu20-0.1.0-py3-none-any.whl
```

2. 安装加速包。

以下以 `AIDK_Ubuntu20-0.1.0-py3-none-any.whl` 为例：

```
pip3 install AIDK_Ubuntu20-0.1.0-py3-none-any.whl
```

3. 导入加速包，只需要修改 2 行代码。

- 替换 Megatron 的 Checkpoint 接口。
- 在最后一次 Checkpoint 结束后，调用 `finish_checkpoint_process()` 通知 Checkpoint，确保最后一次 Checkpoint 正常结束。

```
diff --git a/megatron/training.py b/megatron/training.py
index 9f5ca54..d3e5c6c 100644
--- a/megatron/training.py
+++ b/megatron/training.py
@@ -23,8 +23,11 @@ from megatron import update_num_microbatches
 from megatron.core import mpu, tensor_parallel
 from megatron import print_rank_0
 from megatron import print_rank_last
-from megatron.checkpointing import load_checkpoint, load_model_checkpoint_for_zero, load_optim_checkpoint_for_zero
-from megatron.checkpointing import save_checkpoint
+from aidk.checkpoint.megatron import (load_model_checkpoint_for_zero,
+                                      load_optim_checkpoint_for_zero,
+                                      load_checkpoint,
+                                      save_checkpoint,
+                                      finish_checkpoint_process)
 from megatron.model import Float16Module
 from megatron.model import GPTModel
 from megatron.core.enums import ModelType
@@ -166,6 +169,8 @@ def pretrain(train_valid_test_dataset_provider,
     if args.save and iteration != 0:
         save_checkpoint(iteration, model, optimizer, opt_param_scheduler)

+    finish_checkpoint_process()
+
     if args.do_test:
         # Run on test data.
         prefix = 'the end of training for test data'
```

需要替换的导入代码如下：

```
from aidk.checkpoint.megatron.checkpointing import (
  load_model_checkpoint_for_zero,
  load_optim_checkpoint_for_zero,
  load_checkpoint,
  save_checkpoint,
  finish_checkpoint_process,
)
```

在最后一次 Checkpoint 结束后调用：

```
finish_checkpoint_process()
```

观察效果

任务启动后，可以通过查询 rank0 的日志获取 ckpt 信息。

```
 iteration        4/       6 | consumed samples:          512 | elapsed time per iteration (ms): 41293.1 | learning rate: 3.750E-07 | global batch size:   128 | lm loss: 1.565808E+00 |
loss scale: 16.0 | grad norm: 1.457 | number of skipped iterations:   0 | number of nan iterations:   0 |
 iteration        5/       6 | consumed samples:          640 | elapsed time per iteration (ms): 14536.2 | learning rate: 4.687E-07 | global batch size:   128 | lm loss: 1.601635E+00 |
loss scale: 16.0 | grad norm: 0.979 | number of skipped iterations:   0 | number of nan iterations:   0 |
Successfully saved checkpoints to storage at iteration       5 to /mnt/cluster/dingtao/llama2/megatron_llama2_7b_checkpoint_tp1_pp1_dp8_zero1_fp16_test/
 iteration        6/       6 | consumed samples:          768 | elapsed time per iteration (ms): 14565.2 | learning rate: 5.625E-07 | global batch size:   128 | lm loss: 1.587495E+00 |
loss scale: 16.0 | grad norm: 0.781 | number of skipped iterations:   0 | number of nan iterations:   0 |
saving checkpoint at iteration       6 to /mnt/cluster/dingtao/llama2/megatron_llama2_7b_checkpoint_tp1_pp1_dp8_zero1_fp16_test/
Start to save in memory checkpoint at iteration 6.
Successfully saved in memory checkpoint at iteration 6.
Start to save checkpoints to storage 6.
async save checkpoint at 6, memory used 383.027060917969 GB, start timestamp 1698662497.9801579, end timestamp 1698662499.631482, cost time 1.651324033737183 seconds
(min, max) time across ranks (ms):
    save-checkpoint ................................: (1667.33, 1667.99)
[after training is done] datetime: 2023-10-30 10:41:39
```

字段	解释
async save checkpoint at 6	当前 Checkpoint 采用异步 Checkpoint，保存的是 iteration 6。
memory used	当前内存使用情况（系统整体内存使用情况，仅供参考）。
start timestamp	本次 Checkpoint 的开始时间戳。
end timestamp	本次 Checkpoint 的结束时间戳（不包括 Checkpoint 保存到存储的时间）。
cost time	本次 Checkpoint 接口调用的总耗时（不包括 Checkpoint 保存到存储的时间）。

通用环境变量列表

提交分布式训练任务时，百舸平台会自动注入多个通用环境变量，便于您在代码中直接使用。

本文将为您介绍百舸平台默认提供的环境变量列表信息。

公共环境变量

高性能网络环境变量

高性能网络环境变量，详见 平台预置NCCL环境变量。

任务通用环境变量

环境变量名	说明
AIHC_JOB_NAME	任务的名称。
AIHC_TENSORBOARD_LOG_PATH	Tensorboard 日志路径（仅开启了 Tensorboard 的任务会默认注入）。

PyTorch 环境变量

在 PyTorch 分布式训练框架中，Master 和 Worker 扮演不同的角色并需要相互建立连接以进行通信。百舸平台为 PyTorch 训练任务默认注入通用环境变量，用于关键信息的同步。

环境变量名	说明
MASTER_ADDR	Master 节点地址，默认为 {任务名称}-master-0。
MASTER_PORT	所有 Worker 进行通信协商的端口号，默认为 23456。
RANK	节点的 Index。例如，若提交一个包含 1 个 Master 和 2 个 Worker 的作业，则 Master 上设置的 RANK 为 0，Worker-0 设置的 RANK 为 1，Worker-1 设置的 RANK 为 2。
WORLD_SIZE	训练的节点总数。例如，若提交一个包含 1 个 Master 和 2 个 Worker 的作业，则 WORLD_SIZE 设置为 3。
NPROC_PER_NODE	每个 Worker 节点的 GPU 卡数。例如，该 Worker 节点申请为 8 卡，则该 Worker 节点的 NPROC_PER_NODE 为 8。

（注：以上是平台注入的环境变量，只要选择 PyTorch 类型任务，就会自动注入，和框架无关。除此之外，不同框架可能还会自己注入一些环境变量，但和框架及启动方式有关，可以参考"训练框架内部注入的环境变量"一节。）

多机任务启动示例

torchrun 启动：

```
torchrun \
--nproc_per_node "$NPROC_PER_NODE" \
--nnodes "$WORLD_SIZE" \
--node_rank "$RANK" \
--master_addr "$MASTER_ADDR" \
--master_port "$MASTER_PORT" \
your_training_script.py
```

accelerate launch 启动：

请确保训练镜像已安装 accelerate。num_processes 为训练总进程数，计算方式为 WORLD_SIZE * NPROC_PER_NODE。

```
num_processes=$((WORLD_SIZE * NPROC_PER_NODE))
echo "$num_processes"
accelerate launch --same_network --multi_gpu --gpu_ids all \
--num_machines "$WORLD_SIZE" --num_processes "$num_processes" \
--machine_rank "$RANK" --main_process_ip "$MASTER_ADDR" \
--main_process_port "$MASTER_PORT" \
your_training_script.py
```

TensorFlow 环境变量

TensorFlow 分布式作业通过 TF_CONFIG 环境变量来构建分布式的网络拓扑信息。百舸为 TensorFlow 训练配置的通用环境变量如下：

环境变量名	说明
TF_CONFIG	TensorFlow 分布式作业的集群拓扑与当前任务信息。

TF_CONFIG 示例：

```
{
"cluster": {
"ps": ["tf-dist-mnist-xxxxxx-ps-0.default.svc:2222"],
"worker": ["tf-dist-mnist-xxxxxx-worker-0.default.svc:2222"]
},
"task": {
"type": "ps",
"index": 0
},
"environment": "cloud"
}
```

常见训练框架内部注入的环境变量

环境变量名	说明
LOCAL_RANK	进程在本机的局部索引号。由 PyTorch DDP 框架自动注入，需要以 torchrun 或 python -m torch.distributed.launch 方式启动才会注入。若未注入，请检查启动命令。

在线服务部署

在线服务部署

概述

本文档介绍如何在百舸 AI 计算平台通过控制台部署和管理在线服务。百舸平台提供灵活的部署方式，您可以将通过 Docker 构建的镜像部署为推理服务，并将模型文件或代码挂载到服务实例中。

准备工作

1. 登录百舸AI计算平台，在AI计算资源池中创建全托管资源池或自运维资源池，准备算力资源；

2. 准备部署在线服务所需的信息：

- 服务镜像地址，例如： registry-vpc.cn-shanghai.aliyuncs.com/xxx/yyy:zzz。
- 镜像的启动运行命令，例如： /data/eas/ENV/bin/python /data/eas/app.py。
- 镜像中进程监听的网络端口号，例如： 8000。

标准服务

部署服务

1. 进入在线服务部署列表

登录百舸 AI 计算平台后，选择 **大规模训练与推理->在线服务部署**，进入在线服务部署列表页。列表页默认展示 **标准服务**，可查看服务名称、服务状态、资源规格、资源池、队列、创建时间等信息。

2. 打开部署服务页面

在 **在线服务** 页签的 **标准服务** 列表中，单击 **【部署服务】** 进入标准服务创建页面。

3. 填写部署服务参数

在创建页面按业务需要配置基础信息、资源配置、环境配置、流量接入和高级配置。

4. 提交部署

确认所有参数后，单击 **【确定】** 发起部署。

基础信息

参数	说明
服务名称	支持小写字母、数字和 -，必须以小写字母开头、以小写字母或数字结尾，长度为 1-50；服务名称不可重复，创建后不支持修改。
模型配置	可选择 **自定义模型** 或 **公共模型**。平台支持部署自定义模型，需使用自定义镜像或平台提供的预置镜像，并自行上传模型权重；也支持主流开源模型权重，无需自行下载模型权重，支持 vLLM、SGLang、自研 AIAK 加速推理引擎，并针对不同模型提供单机、分布式、PD 分离多种部署模板，提升部署效率。
服务类型	选择 **标准服务**。

资源配置

参数	说明
资源池类型	选择在线服务需要部署到的资源池。**自运维资源池** 需要先创建资源池并购买用于 AI 推理的计算资源；**全托管资源池** 为免运维资源池，您无需运维和管理资源池及算力资源，由平台保障资源池稳定性。
资源队列	选择服务使用的资源队列。
资源规格	选择部署服务所需的加速芯片类型和卡数、CPU、内存；加速芯片类型选择 **不使用加速芯片** 时部署 CPU 服务。
分布式推理	模型参数量较大且无法单台机器推理时，可以开启分布式推理，将单实例部署在多台机器。
单实例 Pod 数	每个副本中的 Pod 数量，默认值为 2，可按需修改。
RDMA	分布式推理或 PD 分离部署时，建议开启 RDMA 优化跨节点之间的数据传输。算力资源需支持 RDMA，否则可能会调度失败。
实例数	设置服务实例数量。
优先级	支持设置在线服务优先级。资源池开启抢占功能时，平台会根据任务抢占模式进行调度，高优先级任务可抢占低优先级任务的资源。

环境配置

参数	说明
镜像地址	支持选择百舸预置镜像，或从百度云 CCR 镜像仓库、自建镜像仓库拉取镜像；如需私有镜像仓库授权，请输入账号密码。
启动命令	填写镜像的启动运行命令。
环境变量	将被注入到容器实例中。变量名仅支持大小写字母、数字、下划线，且不能以数字作为开头。
共享内存	设置容器共享内存大小。
存储挂载	设置模型权重文件存储的源路径和容器挂载的目标路径。
健康检查	可通过健康检查自动检测并恢复异常状态的容器，确保只有健康实例接收流量。支持存活探针、就绪探针、启动探针；检查方法包括 HTTP 请求检查、TCP 连接检查和自定义执行命令。

流量接入

参数	说明
负载均衡类型	支持 AI 原生网关（7 层）和传输层负载均衡（4 层）两种接入方式。AI 原生网关和传输层负载均衡的调用方式不同，详见 API调用示例。
AI 原生网关	为服务提供鉴权机制，提供应用层监控指标采集、消费者鉴权、限流等功能。可单击 **【创建 AI 原生网关】** 创建网关实例，参考 创建网关实例。使用网关部署服务支持设置负载均衡策略，服务部署成功后可在服务监控中查看流量监控。
传输层负载均衡	提供更底层的流量转发能力，无应用层解析开销，转发效率更高。使用该方式接入流量时，需要选择 BLB 负载均衡实例；如需通过公网调用服务，请选择支持公网的 BLB 实例。仅支持绑定应用型实例，多个服务可以绑定同一个 BLB 实例。
端口	使用 AI 原生网关时，填写容器内监听端口，外部调用端口默认为容器内监听端口；使用传输层负载均衡时，复用同一 BLB 实例的服务外部请求调用端口不能重复，可以使用自动分配端口功能生成端口。

> **注意：** 使用旧版本网关部署的服务会影响后续新功能的使用，请联系技术支持人员升级至新版本，升级不会变更服务调用地址、鉴权信息。

高级配置

参数	说明
服务告警	支持对长时间未部署成功的实例进行监控告警，开启后可以设置告警和通知策略。
网关认证鉴权	仅支持使用 AI 原生网关部署的服务开启网关认证鉴权，关闭后将不提供鉴权功能。
最小实例数	AI Job 调度器开启任务抢占模式时，可设置当前服务至少保留不被抢占的最小副本数。
优雅退出时间	实例退出前将等待一段时间处理已经接收的请求，以减少实例变更带来的请求报错；无特殊需求建议使用默认值 30s。
碎片迁移	开启后，资源池治理碎片资源时此服务将不再接收新请求，并在优雅退出时间后自动删除实例并重建；关闭后，碎片整理时将不会迁移此服务。
滚动更新	支持为服务设置滚动更新策略，需要输入最大不可用和最大超量实例数比例。最大不可用表示滚动更新过程中不可用实例数量占预期实例数的百分比，范围 1%-100%；最大超量表示滚动更新过程中超出预期的实例数量占预期实例数的百分比，范围 1%-100%。

升级服务

1. 进入标准服务列表

登录百舸 AI 计算平台后，选择 **大规模训练与推理->在线服务部署**，在 **在线服务** 页签下选择 **标准服务**。确认目标服务已部署成功，并处于可操作状态。

2. 打开升级服务页面

在目标标准服务所在行的 **操作** 列中，单击 **【升级服务】** 进入服务配置编辑页面。可按需修改镜像版本、存储挂载、健康检查、滚动更新策略等配置。

3. 提交升级

完成配置修改后，在变更描述中填写本次升级内容，单击 **【确定】** 提交升级。提交后返回服务列表，等待服务状态恢复为运行中。

4. 查看版本变更记录

升级完成后，进入服务详情页，查看服务的版本变更记录，确认本次升级记录已生成。



手动扩缩容

1. 进入标准服务列表

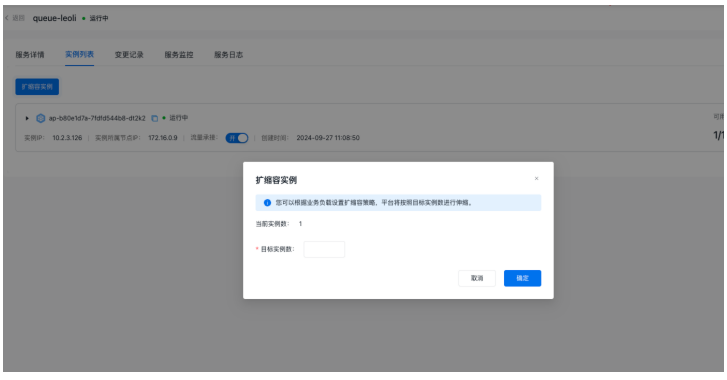
登录百舸 AI 计算平台后，选择 **大规模训练与推理->在线服务部署**，在 **在线服务** 页签下选择 **标准服务**。

2. 调整服务实例数

服务部署成功后，如果服务流量发生变化，可对目标服务执行手动扩缩容，调整实例数以满足流量负载要求。

3. 确认扩缩容结果

提交扩缩容后，返回服务列表或服务详情页，确认服务状态、实例数量和实例运行状态符合预期。



实例摘流

当某个实例出现故障或性能问题，为了避免服务中断，可以通过实例摘流来平滑过渡。开启实例摘流后，需要等待几分钟生效。

流量接入

服务部署成功后，您可以在服务列表查看调用信息。在服务详情中修改流量接入的负载均衡实例，负载均衡类型不支持变更。

删除保护

服务部署成功后，支持开启删除保护，防止在线服务被误操作删除。

设置标签

服务部署成功后，支持为服务设置标签，用于服务筛选和统计。



多角色服务组

用于实现 PD 分离或更多角色服务的关联，负责部署多角色之间有复杂调度逻辑且需要统一流量入口的服务。

> - PD 分离必须从多角色服务组入口部署。
> - 本期多角色服务组仅支持部署 Prefill 和 Decode 两个角色的服务。
> - 多角色服务组仅支持使用 AI 原生网关接入流量。
> - 服务组中的服务不能单独调用。

部署多角色服务组

1. 登录百舸AI计算平台，点击左侧导航栏中的 **在线服务部署** 服务进入列表页面。

2. 选择 **多角色服务组**，单击 **部署多角色服务组** 进入创建页面。

3. 在创建页面填写相关参数，具体参数如下：

基础信息

参数	说明
服务名称	支持小写字母、数字和 -，必须以小写字母开头、以小写字母或数字结尾，长度为 1-50；服务名称不可重复。
模型配置	选择需要部署的模型配置。
服务类型	选择 **多角色服务**。多角色服务仅支持使用 SGLang 推理引擎部署。

服务配置

参数	说明
资源池类型	选择在线服务需要部署到的资源池，仅支持全托管资源池。全托管资源池为免运维资源池，您无需运维和管理资源池及算力资源，由平台保障资源池稳定性。
服务角色	仅支持 Prefill 和 Decode 两种角色，且固定角色名不能修改。
资源队列	同一服务组中的多角色服务不能跨资源池，可以跨资源队列。
资源规格	选择部署服务所需的加速芯片类型和卡数、CPU、内存；加速芯片类型选择 **不使用加速芯片** 时部署 CPU 服务。
实例数	分别设置 Prefill 和 Decode 角色的实例数量。
分布式推理	根据模型规模和部署方式决定是否开启。
单实例 Pod 数	性能最优推荐 Prefill 和 Decode 服务单实例 Pod 数为 4，即分别部署到 4 台机器。
RDMA	分布式推理或 PD 分离部署时，建议开启 RDMA 优化跨节点之间的数据传输。算力资源需支持 RDMA，否则可能会调度失败。
KV Cache 感知调度	开启后平台会感知 Prefill 节点的缓存命中率，将请求调度到命中率高的实例，提升计算效率并降低首 Token 延迟。
镜像地址	支持选择百舸预置镜像，或从百度云 CCR 镜像仓库、自建镜像仓库拉取镜像；如需私有镜像仓库授权，请输入账号密码。
端口	支持自定义监听和调用端口，开启公网访问后会自动暴露此端口。如需根据业务逻辑设置自定义监控指标，对应的监控端口用途请选择 **Metrics**。8001、8002 端口已被系统占用，为避免服务因端口冲突而无法正常完成启动，请勿设置为 8001、8002。
启动命令	填写镜像的启动运行命令。
环境变量	将被注入到容器实例中。变量名仅支持大小写字母、数字、下划线，且不能以数字作为开头。
共享内存	设置共享内存大小。
存储挂载	设置模型权重文件存储的源路径和容器挂载的目标路径。
健康检查	可通过健康检查自动检测并恢复异常状态的容器，确保只有健康实例接收流量。支持存活探针、就绪探针、启动探针；检查方法包括 HTTP 请求检查、TCP 连接检查和自定义执行命令。
优先级	支持设置在线服务优先级。资源池开启抢占功能时，平台会根据任务抢占模式进行调度，高优先级任务可抢占低优先级任务的资源。
最小实例数	AI Job 调度器开启任务抢占模式时，可设置当前服务至少保留不被抢占的最小副本数。
优雅退出时间	实例退出前将等待一段时间处理已经接收的请求，以减少实例变更带来的请求报错；无特殊需求建议使用默认值 30s。
碎片迁移	开启后，资源池治理碎片资源时此服务将不再接收新请求，并在优雅退出时间后自动删除实例并重建；关闭后，碎片整理时将不会迁移此服务。
滚动更新	支持为服务设置滚动更新策略，需要输入最大不可用和最大超量实例数比例。最大不可用表示滚动更新过程中不可用实例数量占预期实例数的百分比，范围 1%-100%；最大超量表示滚动更新过程中超出预期的实例数量占预期实例数的百分比，范围 1%-100%。

4. 配置流量接入

在页面下方的 **流量接入** 区域选择 **AI 原生网关**。如当前没有可用网关，可单击 **【创建 AI 原生网关】**，参考 创建网关实例 创建网关实例；创建的网关需与服务所属资源池在同一 VPC，且仅支持绑定运行中状态的网关实例。

选择网关后，可配置 **负载均衡策略**，控制台提供 **轮询**、**最小连接数**、**随机** 三种策略。确认 Prefill 和 Decode 两个角色配置、网关及高级配置无误后，单击 **【确定】** 提交部署。

参数	说明
AI 原生网关	多角色服务组仅支持使用 AI 原生网关接入流量。使用网关部署服务支持设置负载均衡策略，服务部署成功后可在服务监控中查看流量监控。
负载均衡策略	支持 **轮询**、**最小连接数**、**随机**。
网关认证鉴权	仅支持使用 AI 原生网关部署的服务开启网关认证鉴权，关闭后将不提供鉴权功能。

> **注意：** 使用旧版本网关部署的服务会影响后续新功能的使用，请联系技术支持人员升级至新版本，升级不会变更服务调用地址、鉴权信息。

调用信息

多角色服务组的统一入口调用地址。

流量接入

服务部署成功后，您可以在服务列表查看调用信息。在服务详情中修改流量接入的负载均衡实例，负载均衡类型不支持变更。

删除

删除服务组和服务组中的服务。

多服务流量分流

概述

流量策略可以为多个服务提供一个统一的流量入口，根据流量策略将入口流量分配给各个服务，适用于灰度发布、异构资源调度等多种业务场景。本文介绍如何创建流量策略、设置多个服务的流量比例，以及查看流量入口。

前提条件

已登录 百舸 AI 计算平台 AIHC 控制台。

已准备可用的 AI 原生网关和在线服务。创建流量策略时，AI 原生网关需与服务所属资源池在同一 VPC，且仅支持绑定“运行中”状态的网关实例。## 应用场景

灰度发布

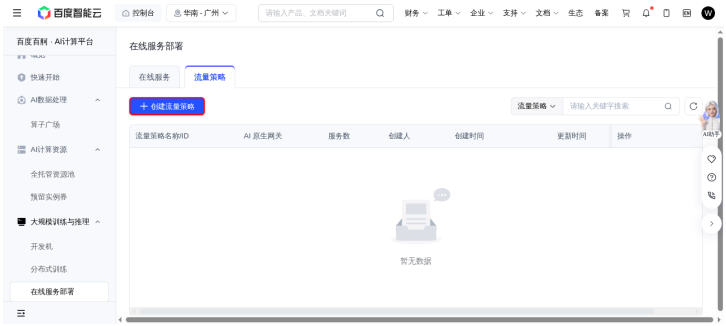
发布新版本时，可以先让新版本服务承载少部分流量，观察运行情况，若有问题可以及时调整新旧服务的流量比例。

异构资源弹性伸缩

当业务存在峰谷，对算力需求存在差异且资源紧张时，可以将多个服务部署在不同算力并加入一个流量策略，配合弹性伸缩功能，动态调整实例数及服务的流量比例。

创建流量策略

1. 进入 **大规模训练与推理->在线服务部署->流量策略** 页面。



2. 点击 **【创建流量策略】**，在弹窗中配置流量策略。



3. 配置流量策略参数。

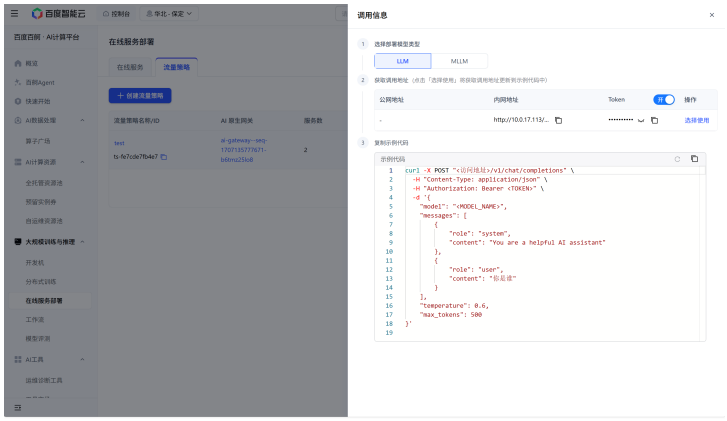
参数	必填	说明
分流策略类型	是	支持按权重分流和按实例数分流。按权重分流时，根据各服务的权重进行加权分配。按实例数分流时，根据服务的运行中实例数动态分配流量比例。多角色服务不支持按实例数分流。
流量策略名称	是	支持小写字母、数字和 -，需以小写字母开头，以小写字母或数字结尾。
AI 原生网关	是	为流量策略指定请求入口网关。添加服务时，仅可选择与网关属于同一 VPC 的服务。
流量配置	是	仅可添加与请求入口 AI 原生网关属于同一 VPC 的服务。可以添加跨资源池的多个标准服务或多角色服务。设置自定义权重方式时，需为每个服务设置端口和流量权重，平台会自动计算预计流量比例。每个服务的权重可输入范围为 1-10 的整数。

注意： 不支持添加使用调度器部署的服务。相同的服务仅支持加入一个流量策略中。

4. 点击 **【确定】** 创建流量策略。创建成功后，请求将按照策略设置的权重分流到每个服务。

查看调用信息

1. 进入 **大规模训练与推理->在线服务部署->流量策略** 页面。
2. 在流量策略列表中找到目标流量策略，在 **操作** 列点击 **【调用信息】**，查看统一流量入口的公网调用地址和 VPC 内网调用地址。



服务自动扩缩容

概述

本文档介绍百舸 AI 计算平台 AIHC 在线服务的自动扩缩容功能。如果业务负载出现显著的波峰波谷，您可以开启弹性扩缩容或定时扩缩容功能，根据业务负载变化配置扩缩容策略。平台将自动调整服务的实例数量，实现计算资源的动态管理，确保业务平稳运行并提高资源利用率。

前提条件

1. 已创建资源池、资源队列，并已部署至少一个在线服务。
2. 自运维资源池 K8s 版本为 1.24 及以上。若低于 1.24 版本，请联系技术支持人员升级后使用此功能。
3. 服务所属的资源池已关联监控实例 CProm。若当前资源池已关联监控实例，可跳过此配置。
4. 如需新建监控实例，请在 **"Prometheus 服务"** 中创建监控实例，并在 **"容器引擎 CCE"** 中将 CCE 集群接入该监控实例。若容器引擎 CCE 已接入实例，可跳过此配置。
5. 已生成监控实例 CProm 的 Token。使用弹性扩缩容功能时，平台需要获取您的 Grafana 访问地址和 Token。进入资源池关联的 CProm 监控实例详情，查看实例信息并生成 Token。

开始使用

弹性扩缩容

1. 登录 百舸 AI 计算平台 AIHC 控制台。
2. 选择 **大规模训练与推理->在线服务部署->在线服务**，进入 **在线服务** 列表。
3. 选择需要扩缩容的服务，点击服务名称进入服务详情页。
4. 点击 **自动扩缩容** 页签，开启 **弹性扩缩容** 开关。
5. 配置弹性扩缩容策略参数。

参数	描述	说明
常规指标	平台提供 GPU、CPU、内存利用率常规指标。	选择平台内置指标作为扩缩容判断依据。
自定义指标	如需根据业务逻辑设置自定义的监控指标，请参考 自定义监控指标。	适用于需要按业务指标触发扩缩容的场景。
最小实例数	弹性扩缩容可缩容到的最小 POD 实例数，可选范围为 1-1000，默认值为 1。	该值不能大于最大实例数。
最大实例数	弹性扩缩容可扩容到的最多 POD 实例数，可选范围为 1-1000，默认值为 1。	该值不能小于最小实例数。
扩容生效时长	指定从扩容指令下达开始到扩容实际生效所需的时长。可输入范围为 0-300 秒，默认值为 0 秒。	如果在此期间系统检测到业务负载恢复正常，将取消扩容指令。
缩容生效时长	指定从缩容指令下达开始到缩容实际生效所需的时长。可输入范围为 0-300 秒，默认值为 300 秒。	如果在此期间系统检测到业务负载恢复正常，将取消缩容指令，并且每隔 5 分钟检查一次，直到单实例 TPS 或 QPS 小于目标值后再开始缩容。

6. 配置完成后，点击 **【确定】** 保存。### 定时扩缩容
7. 登录 百舸 AI 计算平台 AIHC 控制台。
8. 选择 **大规模训练与推理->在线服务部署->在线服务**，进入 **在线服务** 列表。
9. 选择需要扩缩容的服务，点击服务名称进入服务详情页。
10. 点击 **自动扩缩容** 页签，找到 **定时扩缩容** 配置卡片，开启 **定时扩缩容** 开关。
11. 在 **开启定时扩缩容** 弹窗中配置定时扩缩容策略。

参数	描述	说明
目标实例数	设置定时周期内需要扩缩容的预期实例数。	平台会在触发时间到达后，将服务实例数调整到该目标值。
触发时间	设置定时执行时间，支持每天、每周、每月、Cron 表达式。	选择每天时，配置每天触发的具体时间；选择每周时，配置星期和触发时间；选择每月时，配置每月日期和触发时间；选择 Cron 表达式时，分别填写秒、分钟、小时、日、月、星期字段。

最多可添加 5 条定时策略。配置完成后，点击 **【确定】** 保存。

Cron 表达式

参数说明

名称	可用值	可输入特殊符号	说明
秒	0-59	`*/,-`	指定触发时间中的秒字段。
分钟	0-59	`*/,-`	指定触发时间中的分钟字段。
小时	0-23	`*/,-`	指定触发时间中的小时字段。
日	1-31	`*/,-?`	指定触发时间中的日期字段。
月	1-12	`*/,-`	指定触发时间中的月份字段。
星期	0-6	`*/,-?`	指定触发时间中的星期字段。

特殊字符说明

- 星号（`*`）表示表达式能匹配该字段的所有值。如在第 5 个字段使用星号（month），表示每个月。
- 斜线（`/`）表示增长间隔。如第 1 个字段（minutes）值是 `3-59/15`，表示每小时的第 3 分钟开始执行一次，之后每隔 15 分钟执行一次，即在 3、18、33、48 这些时间点执行，也可以表示为 `3/15`。
- 逗号（`,`）用于枚举值。如第 6 个字段值是 `MON,WED,FRI`，表示星期一、星期三、星期五执行。
- 连字号（`-`）表示一个范围。如第 3 个字段的值为 `9-17`，表示 9 点到 17 点之间每个小时执行一次。
- 问号（`?`）只用于日（Day of month）和星期（Day of week），表示不指定值，可用于代替 `*`。

Cron举例说明

- 每隔 5 秒执行一次：`/5 * * * ?`
- 每隔 1 分钟执行一次：`0 /1 * * ?`
- 每天凌晨 1 点执行一次：`0 0 1 * * ?`
- 每月 1 号凌晨 1 点执行一次：`0 0 1 1 * ?`
- 在 26 分、29 分、33 分执行一次：`0 26,29,33 * * * ?`
- 每天的 0 点、13 点、18 点、21 点都执行一次：`0 0 0,13,18,21 * * ?`

自定义监控及扩缩容指标

概述

本文档介绍如何根据业务逻辑设置自定义监控指标。您需要在代码中配置定期上报自定义指标的逻辑，通过百度云监控服务配置采集任务收集监控数据，并配置监控大盘，以便在平台查看自定义监控指标。

前提条件

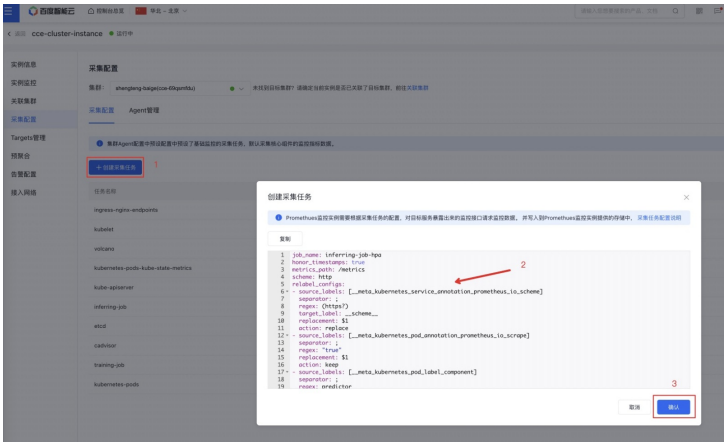
- 已在 **百舸平台->自运维资源池或全托管资源池->资源池详情** 中，为在线服务所属资源池关联接入 CProm 监控实例。若资源池已关联监控实例，可跳过此步骤。
- 部署自定义服务时，已在高级配置中开启自定义监控指标，并填写监控端口。
- 已准备 Grafana 账号和密码。若未配置或需要修改密码，可在 Grafana 服务详情页设置。

配置采集任务

采集任务（Job）是一组 Target 的配置集合，用于定义抓取间隔、访问限制等作用于一组 Target 的抓取行为。Prometheus 监控服务默认预置常用基础监控的采集任务，也支持自定义采集任务。

操作步骤

1. 在百舸控制台进入 **百舸控制台->平台管理->报表&监控**，在资源池下拉框中选择已关联 CProm 监控实例的资源池。资源池选定后，通过资源池关联的 Prometheus 监控实例进入实例详情页。
2. 在 Prometheus 监控实例详情页打开 **【采集配置】**，点击 **【创建采集任务】**。采集任务配置可参考 https://cloud.baidu.com/doc/CProm/s/Ml0qn8ej1。创建任务时，按业务实际暴露的指标路径填写 `metrics_path`。



采集任务配置说明

采集任务配置包括采集时间间隔、采集超时时间、采集协议、采集指标的端口号等。一个任务支持多组 Job 配置。

```yaml
scrape_configs:
  job_name: kube-state-metrics # 必选，Job 的名称
  scrape_interval: 30s # 可选，该 Job 的采集时间间隔，优先级高于全局配置
  scrape_timeout: 30s # 可选，该 Job 的采集超时时间，优先级高于全局配置
  metrics_path: /metrics # 必选，采集指标的 URI 路径，默认为 /metrics
  honor_labels: true # 可选，标签冲突处理策略，配置为 true 表示保留标签；配置为 false 表示重命名标签
  honor_timestamps: true # 可选，是否使用指标中的时间戳
  scheme: http # 必选，采集协议，支持 http 和 https ，默认为 http
  params: # 可选，http 协议参数，仅 scheme 为 http 时有效
    p1: ["v1", "v2"]
  tls_config: # 可选，https 协议时的 TLS 配置，仅 scheme 为 https 时有效
    ca: MIICXwIBAA*** # 配置 CA 证书
    cert: MIICjzCCAfg*** # 配置 TLS 证书
    key: MIICXQIBAA*** # 配置 TLS 密钥
    server_name: tls-server-demo # 配置 TLS 服务器名称
    insecure_skip_verify: true # 是否禁用 https 服务器证书验证
    min_version: TLS12 # TLS 最小支持的版本号，取值：TLS10 (TLS 1.0), TLS11 (TLS 1.1), TLS12 (TLS 1.2), TLS13 (TLS 1.3)
  headers: # 可选，http 请求中的自定义 header
    - a
    - b
  kubernetes_sd_configs:
    - role: pod # K8s 服务发现配置
  relabel_configs: # 可选，配置指标的 relabel，可配置多个relabel，按顺序执行多个 relabel 规则。如没有需求，可省略
    - action: replace # 标签匹配后的操作，包括：replace（替换）、keep（保留）和 drop（丢弃）
      source_labels: [ xxxxxx ] # 从原始 labels 中取哪些 label 的值进行 relabel
      separator: ; # 定义 regex 中的分隔符，默认为 ;
      target_label: xxxxxx # 当 action 为 replace 时，需要添加或重置的标签名称
      regex: (.*) # 对 source_labels 对应的值进行正则匹配
      modulus: 123 # 源标签 Hash 值的模数
      replacement: $1
  metric_relabel_configs: # 可选，配置指标抓取之后的 relabel。如没有需求，可省略
    - action: replace # 标签匹配后的操作，包括：replace（替换）、keep（保留）和 drop（丢弃）
      source_labels: [ xxxxxx ] # 从原始 labels 中取哪些 label 的值进行 relabel
      separator: ; # 定义 regex 中的分隔符，默认为 ;
      target_label: xxxxxx # 当 action 为 replace 时，需要添加或重置的标签名称
      regex: (.*) # 对 source_labels 对应的值进行正则匹配
      modulus: 123 # 源标签 Hash 值的模数
      replacement: $1
  sample_limit: 1024 # 可选，采集器每次采集的样本数量，取值为 0 表示无限制
```

采集任务 YAML 内容如下，除 `metrics_path` 外，其他字段不要修改：

```yaml
 job_name: aihc-pom-job
honor_timestamps: true
metrics_path: /metrics
scheme: http
relabel_configs:
  - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scheme]
    separator: ;
    regex: (https?)
    target_label: __scheme__
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_pod_label_component]
    separator: ;
    regex: aihcpom
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
    separator: ;
    regex: "true"
    replacement: $1
    action: keep
  - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
    separator: ;
    regex: (.+)
    target_label: __metrics_path__
    replacement: $1
    action: replace
  - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
    separator: ;
    regex: ([^:]+)(?::\d+)?;(\d+)
    target_label: __address__
    replacement: $1:$2
    action: replace
  - source_labels: [__meta_kubernetes_namespace]
    separator: ;
    action: replace
    target_label: kubernetes_namespace
  - source_labels: [__meta_kubernetes_pod_name]
    separator: ;
    action: replace
    target_label: kubernetes_pod_name
  - separator: ;
    regex: __meta_kubernetes_pod_label_(.+)
    replacement: $1
    action: labelmap
  - separator: ;
    regex: (.+)_revision_hash|(.+)_template_generation
    replacement: $1
    action: labeldrop
kubernetes_sd_configs:
  - role: pod
```

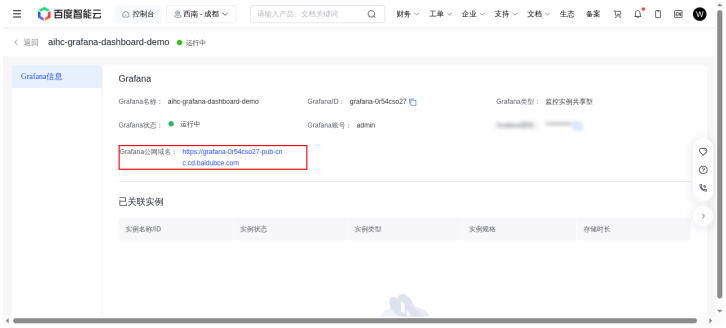
配置监控大盘

操作步骤

1. 进入 **Prometheus 监控->Grafana 服务**，确认 Grafana 服务状态为"运行中"，并查看 Grafana 公网域名。为 CProm 监控实例绑定 Grafana 时，进入监控实例创建或编辑页面，打开 **Grafana 服务** 开关，选择已有 Grafana 实例后提交。绑定完成后，可在 Grafana 详情页的 **已关联实例** 区域查看关联的监控实例。



2. 在 Grafana 详情页点击 **Grafana 公网域名**，进入 Grafana 登录页。使用页面显示的 Grafana 账号和已配置的密码登录。若未配置或需要修改密码，点击 **Grafana 密码** 后的编辑图标，填写新密码并确认；系统会要求完成短信验证码安全验证，验证通过后新密码生效。



3. 登录 Grafana 后，进入 **Grafana->Dashboards/Manage** 或 **Grafana->Create Dashboard**，创建 Folder 或新建 Dashboard，点击 **【Add an empty panel】**。在 Panel 编辑页面选择对应数据源、监控实例和指标查询条件，确认图表有数据后保存 Panel 和 Dashboard。

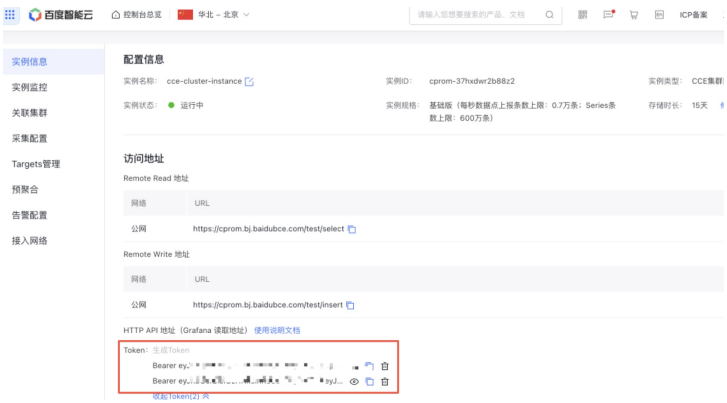
4. **查看监控**



自定义扩缩容指标

1. **生成Token**

进入资源池关联的Cprom监控实例详情，查看实例信息生成Token。基于GPU利用率及自定义业务指标进行弹性扩缩容，平台需要获取您的Grafana访问地址和Token。



2. **编写查询语句并调试**

使用 Grafana 公网域名打开 Grafana，输入 Grafana 账号和密码登录。登录后进入 **Grafana->Explore**，选择 Prometheus 数据源，编写并执行 PromQL 查询语句。查询结果必须返回向量或标量的单元素响应，且结果应为整个服务的数值求和。

查询语句模板：

```
sum(rate(vllm:prompt_tokens_total{kubernetes_namespace="<服务所在命名空间>",pom_aihc_baidubce_com_app_name="<服务名称>"}))
```

默认采集规则会对采集指标添加以下 Label，可用于实现更高级的弹性扩缩容，例如服务 B 基于服务 A 的指标进行扩缩容。

参数	说明
app	服务的唯一 ID，可在实例列表中查看。
clusterID	集群 ID。
component	组件名称，默认为 aihcpom。
instance	实例的 IP 和端口。
kubernetes_namespace	K8s 命名空间，自定义部署服务的默认命名空间为 aihc-pom。
kubernetes_pod_name	实例名称。
pom_aihc_baidubce_com_app_name	配置的服务名称。

3. **添加扩缩容策略**

服务需设置一个Metrics用途的端口，在弹性扩缩容功能中添加自定义指标。



日志查询及采集

概述

本文档介绍在百舸自定义部署场景中，使用日志服务 BLS 采集非标准输出日志的方法。平台默认采集标准输出（Stdout）方式的日志，服务部署成功后可在服务详情中查看标准输出日志。对于非标准输出日志，可使用日志服务 BLS 的收集器、日志集和传输任务完成采集，并在 BLS 中查询日志。

安装收集器

通过 **百度云控制台->日志服务->收集器->收集器安装** 进入收集器安装页面。收集器安装页面用于创建收集器认证 Token，并获取主机安装收集器所需的安装命令。更多信息请参考 安装收集器。

创建日志集

日志集是日志数据的存储单元。您可以使用采集器 Agent 或 API 方式把日志数据写入到指定的日志集中。

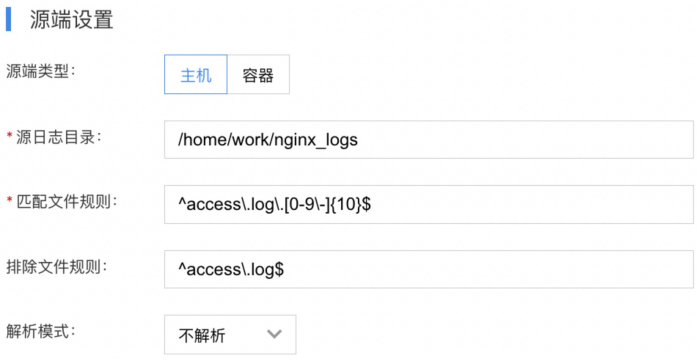
通过 **百度云控制台>日志管理>日志集**，创建日志集。 创建日志集

配置传输任务

采集主机日志文件

当容器挂载了主机文件，并将日志写入该文件时，可以使用本节方式将该日志持久化保存。以收集 /home/work/nginx_logs 目录下持续产生的日志为例，介绍创建传输任务的具体操作：

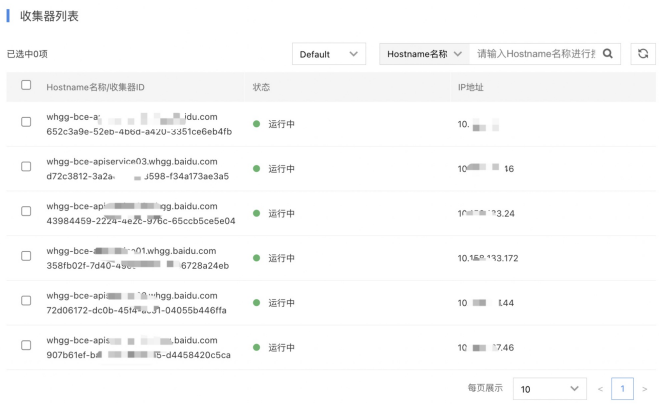
1. 在日志服务 BLS 控制台左侧导航中，选择 **日志管理->日志接入**，在 **传输任务** 页签中单击 **【创建传输任务】**。
2. 在 **任务信息** 区域填写任务名称，不添加标签。
3. 在“源端设置”区，选择“主机”作为源端类型，并配置如下参数：
   - 源日志目录：输入：/home/work/nginx_logs。
   - 匹配文件规则：根据样例日志文件名格式输入：^access.log.[0-9-]{10}$。
   - 排除文件规则：需排除access.log，因此输入：^access.log$。



4. 在“目的端设置”区，选择“日志集”作为目的端，并配置如下参数：
   - 日志集：选择已创建的日志集；关于日志集功能，请参考 日志集。
   - 传输速率：默认10MB/s，支持速率范围为 [1-100] MB/s。



5. 在收集器列表中，添加需部署该传输任务的收集器。



6. 单击 **【确认】**，完成日志传输任务创建。传输任务创建后，最晚 3 分钟生效。

采集容器日志文件

当日志仅写入容器内部的某个文件时，可以使用本节提供的步骤进行日志持久化。和主机日志文件采集不同的是，容器采集支持根据环境变量进行过滤，例如添加 `AIHC_POM=true` 环境变量后，该传输任务将只作用在拥有此环境变量的容器上。

1. 在日志服务 BLS 控制台进入 **日志管理->日志接入**，在 **传输任务** 页签中单击 **【创建传输任务】**。
2. 在 **源端设置** 中，将 **源端类型** 切换为 **容器**，将 **日志类型** 选择为 **容器内部日志**。按实际容器内日志路径填写 **源日志目录**，并配置 **匹配文件规则**、**排除文件规则**。
3. 在 **环境变量白名单** 中单击 **【添加白名单】**，将 **LabelKey** 填写为 `AIHC_POM`，将 **LabelValue** 填写为 `true`。该白名单用于让传输任务仅作用于带有 `AIHC_POM=true` 环境变量的容器。



4. 在百舸控制台进入 **大规模训练与推理->在线服务部署**，单击 **【部署服务】**。
5. 在 **环境配置** 区域找到 **环境变量**，单击 **【添加环境变量】**，将变量名填写为 `AIHC_POM`，变量值填写为 `true`。
6. 服务部署后，包含 `AIHC_POM=true` 环境变量的容器会被 BLS 容器日志传输任务匹配，从而采集容器内部符合规则的日志文件。



AI调用示例

概述

通过在线服务部署的自定义服务支持通过公网或 VPC 内地址调用两种方式。本文介绍如何调用百舸平台部署的在线服务。

调用方式

部署服务时，您可以选择网络接入方式。默认的云原生 AI 网关相对 BLB 负载均衡具备服务鉴权能力，并提供流量可观测监控大盘。如果您需要部署 WebUI、非 HTTP 网络协议，以及为服务提供独享网络接入时，请关闭云原生 AI 网关切换到 BLB 负载均衡。

公网访问

您可以直接通过公共网络访问部署在线服务，部署服务时需开启公网访问。为满足安全隔离和访问控制的要求，您可以配置 AI 网关。通过 AI 网关，您可以将请求转发至在线服务部署，从而确保客户端与服务器之间实现稳定且可靠的网络连接。

VPC 内访问

同一 Region 的两个 VPC 网络支持建立 VPC 连接，您可以访问同一地域中部署在资源池中的服务。

API调用说明

在百舸平台部署在线服务后，系统会自动生成调用服务地址。您可以在服务列表页或详情页查看调用信息，通过该调用信息可以进行调用测试。

AI网关

AI 网关集成 Token 鉴权机制，显著增强了服务的安全性。所有未经授权的请求将被网关自动拦截，能有效保护推理服务免受未授权访问。

使用 curl 访问

在 URL 后需拼接镜像内服务的路由路径 `/v1/chat/completions`。

```
curl --location '<访问地址>/v1/chat/completions' \
--header 'Content-Type: application/json' \
--header 'Authorization: <TOKEN>' \
--data '{
    "model": "deeepseek-r1",
    "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}],
    "max_tokens": 1024,
    "temperature": 0
}'
```

使用 Python 访问

```
from openai import OpenAI

def get_response():
    client = OpenAI(
        api_key="<TOKEN>",  # 需要您将 Token 的 "Bearer" 去掉，该 SDK 会自动补齐
        base_url="<访问地址>",
    )

    completion = client.chat.completions.create(
        model="deeepseek-r1",
        messages=[{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}]
    )

    print(completion.model_dump_json())

if __name__ == '__main__':
    get_response()
```

BLB负载均衡

使用 curl 访问

```
curl --location '<访问地址>/v1/chat/completions' \
--header 'Content-Type: application/json' \
-d '{
  "model": "deeepseek-r1",
  "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}]
}'
```

使用 Python 访问

```
from openai import OpenAI

def get_response():
    client = OpenAI(
        api_key="",  # BLB 负载均衡无需填写 api_key，可以按照您的镜像配置
        base_url="<访问地址>",
    )

    completion = client.chat.completions.create(
        model="deeepseek-r1",
        messages=[{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}]
    )

    print(completion.model_dump_json())

if __name__ == '__main__':
    get_response()
```

推理参数说明

概述

推理参数本文为您介绍 AIAK-inference 适用的推理参数详情。建议您在使用 AIAK-inference 进行模型推理加速前，先阅读本文档。

量化参数

参数名	取值类型	是否必选	默认值	可选值	描述
-i	String	是	-	-	原始模型权重输入路径。
-o	String	是	-	-	量化后的模型权重输出路径。
-quant_type	String	是	-	weight_only_int8、smooth_quant、awq、gptq、squeezellm	量化算法。
-tp	Integer	是	-	1、2、4、8	服务部署的 GPU 卡数。
-t	String	是	-	fp16、bf16	指定非量化的部分存储类型。
-sq	Float	否	Llama 默认为 0.8，GLM-130 为 0.75	取值范围为 0 到 1 之间的小数	指定 smooth_quant 的量化 smoother 参数。
-token	String	否	-	-	特殊需求时，指定 smooth_quant 量化需要的 token_ids 路径。
--multi-query-mode	Boolean	否	False	-	是否使用 multi-query attention，适用于 smooth_quant。

算法模型支持

算法名称	模型支持
weight_only_int8	llama-7b-hf、llama-13b-hf、llama-65b-hf、llama-2-7b-hf、llama-2-13b-hf、llama-2-70b-hf、baichuan1-7b/13b
smooth_quant	llama-7b-hf、llama-13b-hf、llama-65b-hf、llama-2-7b-hf、llama-2-13b-hf、llama-2-70b-hf、baichuan7b、baichuan13b、qwen1.5-14b、qwen1.5-72b。非量化数据类型目前 bf16 会转换成 fp16。
gptq	qwen1.5-14b、qwn1.5-72b、qwen-14b、qwen-72b

推理参数

参数名	取值类型	是否必选	默认值	可选值	描述
--model_name	String	是	-	llama、llama2、chatglm2、baichuan、glm、qwen1、qwen1.5、InternLM2-20B-chat	用于识别模型的名称。
--num_gpus	Integer	是	1	-	使用的 GPU 数量。
--ckpt_path	String	是	-	-	Checkpoint 的路径。
--data_type	String	是	-	fp16、fp32	数据类型。
--batch_size	Integer	否	8	-	批处理的大小。
--tokenizer_path	String	否	ckpt_path	-	Tokenizer 的路径。
--extension_path	String	否	/dev/	-	扩展路径。使用默认值时，表示没有扩展。
--quant_mode	String	否	-	weight_only_int8、smooth_quant、awq、gptq、squeezellm	量化模式，可以不开启或启用量化。
--enforce_eager	Boolean	否	True	-	强制使用 eager-mode PyTorch。默认值为 True；如果设置为 False，将混合使用 CUDA Graph 与 eager mode。
--gpu_memory_utilization	Float	否	0.99	-	模型推理过程显存使用比例。如果开启 CUDA Graph，可能需要降低显存使用比例。
--grpc_port	Integer	否	8001	-	Triton 服务器的 gRPC 端口。
--http_port	Integer	否	8000	-	Triton 服务器的 HTTP 端口。
--metrics_port	Integer	否	8002	-	Triton 服务器的指标端口。
--log_verbose	Boolean	否	False	-	如果设置为 true，则启用详细的日志记录。
--kv_cache_dtype	String	否	auto	auto、fp8_e5m2	复用 KV cache 提高推理效率。默认为 auto，表示不开启；如需开启 KV cache，请设置为 fp8_e5m2。
--no_prompts	Boolean	否	False	-	输出内容可选是否包含输入的 prompt。默认为 False，输出内容包含输入的 prompt；如需过滤 prompt，请设置为 True。
--enable_decouple	Boolean	否	True	-	开启流式返回。
--task_type	String	是	causal_lm	causal_lm、sequence_classification	模型类型，生成式默认使用 causal_lm，判别式使用 sequence_classification。
--spec_dec_type	String	否	none	none、medusa	设置投机采样模式，目前仅支持 medusa 或 none。
--draft_model	String	否	-	-	配合投机采样的模型路径。对于 Medusa，是训练后的模型路径。当 --spec_dec_type 不为 none 时必填。
--propose_cnt	Integer	否	-	-	投机采样数量，数量越大，采样命中率越高，但单次计算时间越长，可以根据业务情况调整。对于 Medusa 模式，这里是 3 个 Integer 组成的列表，推荐值为 1,3,4。当 --spec_dec_type 不为 none 时必填。
--max_num_seqs	Integer	否	none	正整数或 none	在一次迭代中可以处理的最大序列数量。
--max_num_batched_tokens	Integer	否	none	正整数或 none	在一次迭代中可以处理的最大 token 数量。
--max_model_len	Integer	否	none	正整数或 none	序列的最大长度，包括提示和生成的文本。

服务性能压测

概述

本文档介绍如何在百舸平台对已部署的在线推理服务进行性能压测，包括创建压测任务、管理压测任务、查看压测详情与结果报表，以及实时监控压测过程中的性能指标变化趋势。

性能压测功能面向在线推理服务上线前的性能评估场景，通过模拟不同负载下的并发请求，帮助用户发现服务性能瓶颈、评估服务承载能力并保障 SLA。当前版本仅支持对标准服务中的 LLM 和 VLM 模型进行性能压测。

前提条件

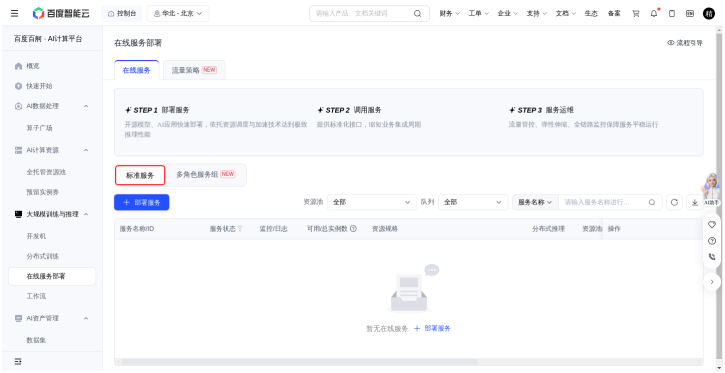
- 已在百舸平台完成标准服务的创建，且目标服务处于 **运行中** 状态。
- 平台提供公共数据集，若需使用自定义数据集，相关数据文件已上传至 BOS，并已记录具体文件路径。

创建压测任务

步骤 1：进入性能压测 Tab

登录百舸控制台，在导航栏依次选择 **大规模训练与推理** -> **在线服务部署**，进入「在线服务部署」页面。在顶部切换栏选择 **在线服务**。

在服务列表中找到目标标准服务（服务状态须为「运行中」），点击该服务的 **服务名称**，进入服务详情页。在服务详情页顶部 Tab 栏中点击 **【性能压测】** Tab，进入压测任务列表。



> **说明：** 仅「运行中」状态的服务支持创建压测任务。若当前无可用服务，需先通过 **【部署服务】** 完成服务创建，并等待其进入运行中状态。

步骤 2：创建压测任务

进入 **【性能压测】** Tab 后，点击 **【创建压测任务】** 按钮，弹出创建表单。

> **说明：** 若目标服务不处于运行中状态，**【创建压测任务】** 按钮置灰不可点击，鼠标悬停时提示"仅支持对运行中的服务进行压测"。

步骤 3：填写基本信息

在创建表单的「基本信息」区域，依次填写以下字段：

参数	必填	说明
压测任务名称	是	支持小写字母、数字及 -，须以小写字母开头，以小写字母或数字结尾；同一服务下不可重名。
资源规格	是	选择资源池/队列，填写所需 CPU 核数与内存容量。该资源在资源统计视图中归属「训练任务」类型。
模型类型	是	可选 **LLM** 或 **VLM**。不同模型类型对应不同的预置数据集和压测方法。
模型 ID	是	目前仅支持从 **魔搭** 加载 Tokenizer。最多 200 个字符，支持大小写字母、数字及特殊字符。
模型名称	是	即压测请求参数中的 `model name` 字段值。最多 100 个字符，支持大小写字母、数字及特殊字符。

步骤 4：压测配置

在「压测配置」区域的「压测地址」部分，依次完成以下配置：

1. **端口号**：若目标服务配置了多个端口，在下拉列表中选择需要压测的端口号。
2. **URL 类型**：选择 **内网地址** 或 **公网地址**。
   - 压测工具与服务跨 VPC 部署时，只能选择公网地址。
   - 选择公网地址（BLB 接入服务）时，需为任务所属资源池配置 NAT 网关，并在 BLB 实例安全组入站规则中放行 NAT 网关关联的公网 IP。
   - 选择公网地址（云原生 AI 网关接入服务）时，同样需为任务所属资源池配置 NAT 网关。
3. **接口**：选择压测接口，支持 `/v1/chat/completions`（Chat 接口）和 `/v1/completions`（Completions 接口）。

> **说明：** 与推理服务所在资源池同一个 VPC 下的资源池部署压测任务可以选择外网和内网。与推理服务所在资源池不在同一个 VPC 下的资源池部署压测任务只能选择外网。

步骤 5：配置压测数据集

在「压测数据集」区域选择数据集类型。数据集类型必填，默认选择「公共数据集」。

公共数据集：

- 模型类型为 LLM 时，默认预置 **ShareGPT_V3**，默认格式为 openqa。
- 模型类型为 VLM 时，默认预置 **flickr8k**。

自定义数据集（仅 LLM 支持）：

1. 选择存储类型：**BOS**。
2. 选择 Bucket，填写具体文件的存储路径。
3. 选择 **数据集格式**：
   - **openqa**：问答对格式。
   - **line_by_line**：逐行格式。

> **说明：** VLM 模型不支持自定义数据集，仅可使用平台预置数据集。

步骤 6：配置其他压测参数

继续填写以下压测参数：

参数	必填	说明
API Key	否	压测调用服务的鉴权 Token。服务已开启平台鉴权时无需填写；使用云原生网关接入时会自动填入网关 Token。
上下文长度	否	控制输入/输出 Token 的上下限，填写最小/最大输入输出长度，范围为 1–10,000,000。
单个请求样本数	是	每轮请求携带的样本数。总请求样本数 = 单个请求样本数 × 并发数。
起始并发数	是	压测起始阶段的并发请求数，取值范围为 1–50。
最大并发数	是	压测允许达到的最大并发请求数，取值范围为 1–50。
每阶段增加并发数	是	每个压测阶段递增的并发数，取值范围为 1–50。
Header	否	自定义 HTTP 请求头，最多可添加 20 个。

步骤 7：提交创建

确认所有参数填写无误后，点击 **【确认】** 按钮，完成压测任务创建。任务创建成功后，将出现在 **【性能压测】** Tab 的任务列表中，初始状态为「创建中」或「排队中」。

管理压测任务

查看压测任务列表

在百舸控制台左侧导航栏中，依次选择 **大规模训练与推理->在线服务部署**，进入「在线服务部署」页面。页面默认位于 **在线服务** Tab，并选中 **标准服务** 类型。

点击目标标准服务的 **服务名称**，进入服务详情页，点击顶部 Tab 栏中的 **【性能压测】** Tab。

页面展示压测任务列表，包含以下字段：

字段	说明
压测任务名称/ID	任务标识，同一服务内唯一。
状态	排队中、创建中、创建失败、压测中、停止中、已停止、已完成、失败。
模型类型	LLM 或 VLM。
模型 ID	加载 Tokenizer 所使用的模型 ID。
资源池名称/ID	运行该压测任务所使用的资源池。
队列名称/ID	所属资源队列。
资源规格	CPU/内存规格。
创建时间	任务创建时间。
操作	复制、停止（压测中状态）、删除（终态）。

复制压测任务

在压测任务列表中，找到需要复制的目标任务，点击操作列中的 **【复制】**。

页面跳转至创建压测任务表单，并自动填充被复制任务的全部参数（任务名称、资源规格、模型类型、模型 ID、压测配置等）。按需修改参数后，点击 **【提交】** 即完成新任务的创建。



停止压测任务

在压测任务列表中，找到状态为「压测中」的任务，点击操作列中的 **【停止】**。

弹出二次确认弹窗，确认后任务状态切换为「停止中」，稍后变更为「已停止」，压测工具 Pod 随之释放资源。

注意：停止操作不可逆，当前压测进度将中止，但已产出的压测结果数据不会被删除。

删除压测任务

终态任务（创建失败 / 失败 / 已停止 / 已完成）： 在对应任务行点击操作列中的 **【删除】**，弹出二次确认弹窗，确认后完成删除。

压测中状态任务： 须先执行「停止」操作，待任务进入「已停止」终态后，再执行删除。

删除服务时的联动说明： 若服务下仍存在执行中的压测任务，删除服务时会弹窗提示"有压测任务正在执行，删除后将停止压测并删除任务和测试结果"，确认后平台同步停止并删除所有关联压测任务。

注意： 压测任务删除后结果数据不可恢复，请在删除前确认已记录或保存所需压测结果。

查看压测任务详情

详情面板默认展示 **基本信息** 区块，内容分三个部分：

- **基本信息：** 压测任务名称、资源规格、模型类型、模型 ID、模型名称。
- **压测配置：**
  - **压测地址：** 服务推理接口完整 URL（含协议、主机、端口、API 路径）。
  - **压测结果存储路径：** BOS 格式路径（如 `bos://<bucket>/<prefix>/`），压测完成后结果文件存储于此。
  - **压测数据集：** 数据集类型（公共数据集 / 自定义数据集）及数据来源路径。
  - **API Key：** 调用压测服务时使用的认证密钥。
  - **上下文长度：** 输入/输出的最小和最大 Token 数范围。
  - **并发数：** 起始并发数、每阶段增加并发数、最大并发数。
  - **单个请求样本数：** 每次请求携带的样本数量。
- **关联任务：** 与该压测任务关联的训练任务名称，点击可跳转至对应训练任务详情。



查看压测结果

在压测任务列表中选择状态为「已完成」的任务，点击任务名称进入详情页。详情页包含 **基本信息** 和 **压测结果** 页签。点击 **压测结果** 后，页面以表格展示不同并发阶段的压测指标，包含以下列：

指标列名	说明
Average input tokens per request	平均输入 token 数/请求。
Average inter-token latency (s)	平均 token 间延迟。
Average latency (s)	平均延迟。
Average output tokens per request	平均输出 token 数/请求。
Average time per output token (s)	平均每个输出 token 耗时。
Average time to first token (s)	首 token 平均响应时间。
Failed requests	失败请求数。
Number of concurrency	并发数。
Output token throughput (tok/s)	输出 token 吞吐量。
Request throughput (req/s)	请求吞吐量。
Succeed requests	成功请求数。
Time taken for tests (s)	测试总耗时。
Total requests	总请求数。
Total token throughput (tok/s)	总 token 吞吐量。

下载压测报告

压测结果已存储在平台提供公共BOS中，您可以在压测任务详情中查看BOS地址，下载压测报告文件。路径以 `bos://` 开头。点击路径右侧的复制图标，可将完整 BOS 路径复制到剪贴板。



在线调试

概述

此服务为已部署的模型提供可视化调试界面，可在界面发送请求，实时查看响应结果，快速验证服务接口连通性与响应效果。

> **说明：** 目前仅支持通过内网地址进行在线调试。

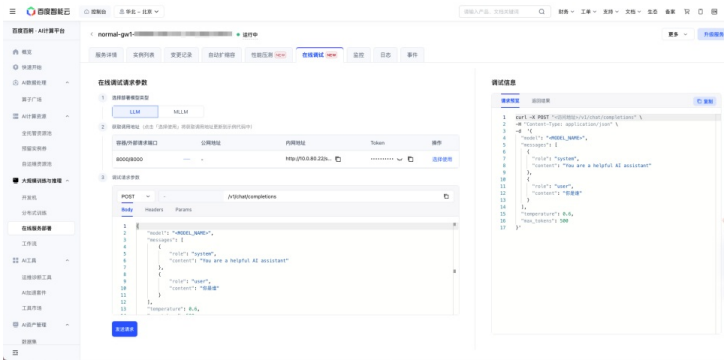
前提条件

已创建并部署运行中的在线服务。

使用步骤

步骤 1：进入在线调试页面

在控制台左侧导航栏选择 **【在线服务部署】**，点击需要调试的服务名称进入详情页面，点击 **【在线调试】** 标签。



步骤 2：选择服务并填写调用信息

选择部署模型类型：

- 点击 **【LLM】**，适用于语言模型。
- 点击 **【MLLM】**，适用于多模态语言模型。

查看调用地址表格，点击操作列的 **【选择使用】**，将端口地址自动填入 **请求地址**。

步骤 3：配置请求方法与请求体

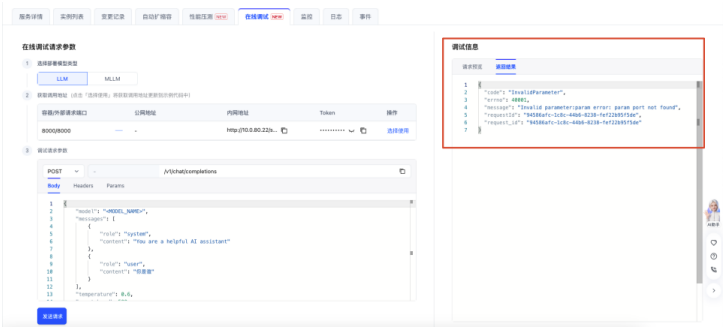
配置请求信息如下，配置完成后，点击 **【发送请求】**。

参数	必填	说明
请求方法	是	默认使用 POST。
请求路径	是	默认使用 /v1/chat/completions。
Body	是	在 Monaco 编辑器中配置请求体。编辑器预置包含 model、messages、temperature、max_tokens 的 JSON 内容，将 <MODEL_NAME> 替换为实际模型名称。
Headers	否	添加请求头。
Params	否	添加查询参数。

页面「调试信息」区域的 **请求预览** 标签实时展示当前配置的完整 curl 命令。点击 **【复制】** 可复制命令，修改请求参数后预览内容同步更新。

步骤 4：查看返回结果

请求发出后，切换至 **返回结果** 标签查看响应。响应体在 Monaco 编辑器中以 JSON 格式展示。



健康检查

概述

本文档介绍推理服务健康检查功能的作用、探针策略、健康检测方法、参数含义，以及在创建在线服务时配置健康检查的方法。健康检查可以自动检测并恢复异常状态的容器，确保只有健康的实例接收流量，并避免向不健康的实例分配资源。

前提条件

- 已登录百度百舸 · AI 计算平台控制台。
- 已具备创建在线服务的权限。

探针策略

推理服务支持存活探针、就绪探针、启动探针策略配置。

探针类型	说明
存活探针	定期执行，用于检测容器是否处于不健康状态。
就绪探针	定期执行，检查容器是否准备好接收请求。
启动探针	用于判断容器何时开始启动，可以控制容器在启动成功后再进行存活性和就绪性检查。可以使用该探针对慢启动容器进行存活性检测，避免容器在启动运行之前被 kill。

支持的健康检测方法

健康检测方法	说明
HTTP 请求检查	通过容器的 IP 地址、端口号及路径调用 HTTP Get 方法，如果响应的状态码大于等于 200 且小于 400，则容器健康。
TCP 连接检查	通过容器的 IP 地址和端口号执行 TCP 检查，如果能够建立 TCP 连接，则容器健康。
自定义执行命令	在容器中执行指定命令，如果执行成功后退出码为 0，则健康检查成功。

健康检查参数表

健康检查功能默认关闭，您可以在创建服务时，通过配置健康检查参数开启健康检查功能。

参数	说明
调用路径	仅选择方法为 **HTTP 请求检查** 时配置。填写 HTTP Server 访问地址。
端口号	仅选择方法为 **HTTP 请求检查** 或 **TCP 连接检查** 时配置。填写服务实际监听的端口号。
运行命令	仅选择方法为 **自定义执行命令** 时配置。用于检测服务运行状态的命令。系统将定期执行该命令，当命令执行成功时认为服务健康；执行失败则认为健康检查未通过。
初始化检查延时	服务启动后，延迟多长时间开始进行第一次健康检查，单位为秒。用于避免服务尚未完全启动时被误判为不健康，默认值为 **0**。
轮询检查时间间隔	两次健康检查之间的时间间隔，单位为秒，默认值为 **10 秒**。检查频率过高可能增加服务运行开销，频率过低则可能无法及时发现服务异常。
检查超时时长	单次健康检查允许的最长执行时间，单位为秒，默认值为 **1 秒**。超过该时间仍未返回检查结果，则本次健康检查视为失败。
检测成功认定次数	处于失败状态时，健康检查连续成功可被认定为成功的次数，默认 **1 次**。
检测失败认定次数	处于成功状态时，健康检查连续失败可被认定为失败的次数，就绪探针默认为 **3 次**，存活探针和启动探针默认为 **1 次**。

探针开启后回显配置信息。



滚动更新与优雅退出

概述

本文档介绍推理服务的滚动更新和优雅退出配置。滚动更新是在扩缩容、更新服务过程中逐步替换旧实例为新实例的发布策略，保证服务更新过程中业务不中断。优雅退出是滚动更新过程中的安全收尾机制，确保实例在关闭前完成存量请求的处理，避免用户请求被强制中断。

前提条件

- 已登录百度智能云控制台，并进入百度百舸·AI 计算平台。
- 已具备部署在线服务的操作权限。

滚动更新

推理服务支持滚动更新策略，创建服务时可在高级配置中配置滚动更新参数。



参数说明如下。

参数	说明
最大不可用	滚动更新过程中不可用实例的数量占预期实例数的百分比，取值范围为 0%-100%，默认值为 25%。该参数会影响更新过程中保留的可用实例数量，建议根据实际业务负载设置。
最大超量	滚动更新过程中超出预期的实例数量占预期实例数的百分比，取值范围为 0%-100%，默认值为 25%。该参数会影响更新过程中可额外创建的实例数量，建议结合资源容量和服务预热情况设置。

> **重要：** 建议根据实际负载和预热情况调整参数。

优雅退出

实例退出前将等待一段时间来处理已经接收的请求，以减少实例变更带来的请求报错。



参数说明如下。

参数	说明
优雅退出时间	实例在收到关闭指令后预留一段让实例处理完已接收请求的时间，单位为秒，取值需大于 0 秒，默认值为 30 秒。该参数会影响服务稳定性和服务更新速度，无特殊需求建议使用默认值。

> **重要：** 参数值过小可能会影响服务稳定性，过大可能会导致服务更新速度过慢，无特殊需求建议使用默认值。

工作流

概述

概述

工作流是在百舸平台实现端到端任务编排与自动化的核心模块，旨在帮助算法工程师高效地组织、管理和执行复杂的机器学习工作流，打通从数据准备、模型开发到模型部署的全生命周期环节。

通过工作流模块，用户可以将碎片化的机器学习步骤整合为标准化的流水线，从而提升研发效率与模型交付质量。

核心功能

基于 DAG 的可视化任务编排

工作流采用有向无环图（DAG）和基于步骤的流水线机制进行构建。在工作流中，每个节点代表一个具体任务，如数据预处理、模型训练、模型评估等；任务之间的连线用于明确任务的执行顺序和数据依赖关系。

平台提供灵活的建模方式：

- **YAML 方式编排：** 算法工程师可以使用平台提供的 YAML 模板和示例，通过编写配置文件自定义控制工作流的执行逻辑。
- **可视化预览：** YAML 配置可通过可视化界面展示，帮助用户清晰查看任务的执行顺序和结构关系。

全场景的流水线覆盖

工作流模块能够支撑开发者在百舸平台实现模型开发运营一体化，覆盖以下两大核心流水线场景：

- **数据处理流水线：** 编排数据采集、数据探索、数据处理、数据增强以及特征工程等数据加工作业。

- **机器学习流水线：** 编排算法选择、模型构建、模型训练、模型评估、模型部署以及上线后的模型监控等全流程任务。

多种触发机制

依托于底层工作流调度引擎，工作流不仅支持用户手动直接触发，还支持定时触发，能够满足各类自动化生产需求。

细粒度的异构算力资源分配

在模型训练的完整工作流中，不同环节对计算资源的要求存在显著差异。例如，数据处理环节主要依赖 CPU，而模型训练环节则需要 GPU 算力。开发者可通过为不同任务分配合理的资源，并使用工作流串联任务，实现资源利用的最大化。

实验追踪与组件化复用

通过使用工作流，用户不仅可以沉淀自定义算子和任务模板，还能通过重用现有流水线快速创建端到端解决方案，避免每次从零开始构建，降低研发门槛。

创建工作流

概述

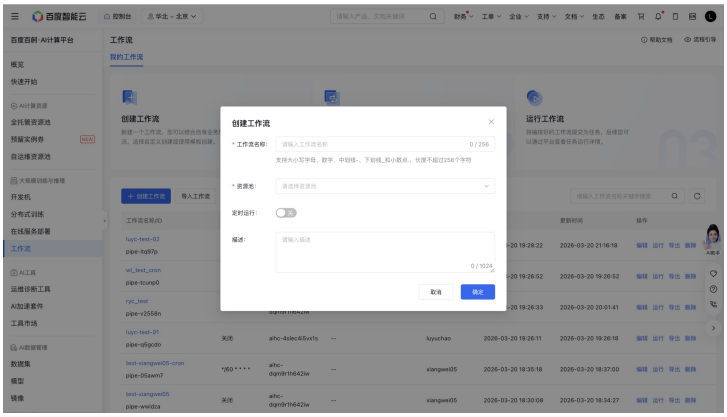
本文档介绍在百舸 AI 计算平台 AIHC 控制台创建、编辑并运行工作流的操作方法。百舸工作流提供数据处理、训练、推理部署等业务场景中需要组件化编排的自动化流程能力。平台提供了基础的任务模板和工作流模板示例，开发者可使用工作流模板开发符合业务需求的业务工作流。

前提条件

1. 加入白名单：百舸工作流服务当前处于邀测阶段，暂时以客户白名单方式提供，可通过发起工单、联系平台运营来开通工作流功能。
2. 角色权限：至少拥有百舸平台开发者权限（AIHCDevelopPolicy），如果没有需联系所使用账号的管理员进行授权。
3. 计算资源及权限：在使用工作流前，需要已经创建有百舸全托管资源池，新建资源池参考 创建全托管资源池。运行工作流前，需确保所选资源池下存在可用队列。

创建步骤

1. 登录 百舸 AI 计算平台 AIHC 控制台。
2. 在控制台导航中选择 ** 下的 **，进入工作流模块首页。
3. 点击页面的【创建工作流】按钮，填写表单信息：



参数说明工作流名称必填，填写任务的名称。

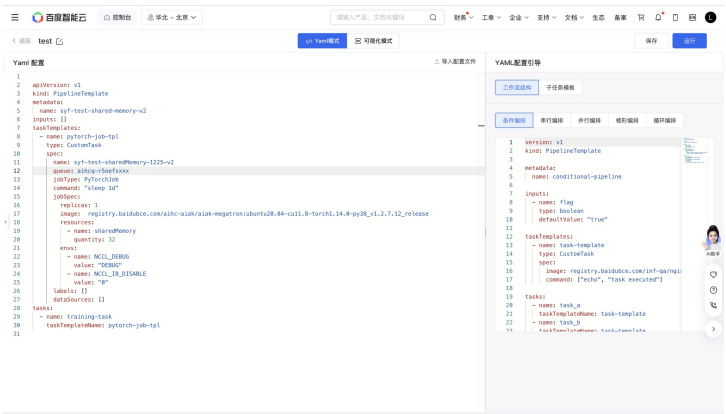
支持大小写字母、数字、中划线-、下划线_和小数点.，长度不超过256个字符资源池必选，选择一个资源池作为运行工作流的计算资源

当前仅支持全托管资源池定时运行默认关闭，开启定时运行时，工作流会按设定的周期自动运行Cron表达式当周期运行开启时，使用Cron语言配置工作流执行的循环周期

Cron语法见 https://en.wikipedia.org/wiki/Cron描述选填，自定义描述工作流信息

4. 点击【确定】，创建成功后自动进入工作流编辑页面，继续完成工作流的任务参数设置、编排之后【保存】配置。

> 创建成功后默认配置中，只需要修改queue参数为当前工作流绑定资源池下的任意有资源的队列ID即可运行你的第一个工作流任务。



5. 编辑工作流后点击【运行】工作流，即可立即执行一次工作流任务。

下一步操作

对工作流有初步了解后，你需要进一步学习百舸工作流任务节点以及工作流结构的配置方法来构建自己的业务流。

- 使用 YAML 定义工作流
- 运行工作流

使用YAML定义工作流

百舸工作流使用 YAML 格式定义。您可以在一份 YAML 配置文件中声明工作流的输入参数、各节点的任务配置，以及节点之间的串行、并行、菱形等依赖关系。工作流中的每个节点在运行时都会创建一个分布式训练任务，工作流中称为子任务。

基本结构

工作流 YAML 由以下三部分组成：

- 全局配置：对整个工作流生效的配置项，包括 `version`、`kind`、`inputs` 等。
- 任务模板（taskTemplates）：子任务的配置模板。模板中的 `spec` 与创建训练任务接口的请求参数相同。
- 任务编排（tasks）：工作流中的节点列表。每个节点引用一个任务模板，并定义该节点的输入值和依赖关系。

```
 version: v1
kind: PipelineTemplate
inputs:        # 全局输入参数，可选
  ...
taskTemplates:   # 任务模板
  ...
tasks:         # 任务编排
  ...
```

全局字段

字段	是否必选	说明
version	是	固定为 `v1`
kind	是	固定为 PipelineTemplate
inputs	否	全局输入参数列表
taskTemplates	是	任务模板列表
tasks	是	节点列表

任务模板字段

字段	是否必选	说明
name	是	模板名称，在工作流内唯一
type	是	固定为 CustomTask
inputs	否	模板输入参数列表
output	否	模板输出参数列表
spec	是	子任务配置

节点字段

字段	是否必选	说明
name	是	节点名称，在工作流内唯一
taskTemplateName	是	引用的任务模板名称
dependencies	否	上游节点名称列表
inputs	否	为模板输入参数赋值
when	否	执行条件表达式

> **注意：**
>
> - 节点名称建议仅使用小写字母、数字和中划线（-），不要使用下划线（_）。

子任务配置（spec）

`spec` 与创建训练任务接口的请求参数相同，常用字段如下：

字段	是否必选	说明
queue	是	队列 ID
jobType	否	分布式框架类型，默认值为 PyTorchJob
command	是	启动命令
jobSpec	是	实例配置，格式由 jobType 决定
priority	否	调度优先级：high、normal、low
datasources	否	存储挂载配置
labels	否	任务标签

`jobType` 支持 PyTorchJob、TFJob、MPIJob、RayJob，`jobSpec` 的格式随之变化：

- PyTorchJob：`jobSpec` 为单个 JobSpec 对象，直接配置 `replicas`、`image`、`resources`、`envs` 等字段。
- RayJob：`jobSpec` 为以角色名为键的 Map，必须包含 `Head` 角色，每个角色下配置 JobSpec 字段。详情请参见下文“示例：创建 RayJob 类型的节点”。
- TFJob、MPIJob：同样按角色组织，可选角色详情请参见创建训练任务。

JobSpec、Resource 等结构的字段说明，详情请参见数据结构。

配置 spec 时请注意：

- 环境变量 `envs` 配置在 `jobSpec` 中（RayJob 配置在各角色下），格式为 `name`、`value` 列表。`value` 为字符串类型，取值为 `true` 或数字时请加引号，例如 `value: "true"`。
- 任务标签 `labels` 须使用 `key`、`value` 列表格式，不支持 `key: value` 映射格式。

- 存储挂载 `datasources` 支持的存储类型及写法，详情请参见创建训练任务中的数据源挂载示例。

```
spec:
  queue: aihcq-xxxx
  jobType: PyTorchJob
  command: python train.py
  jobSpec:
    replicas: 1
    image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
    resources:
    - name: cpu
      quantity: 4
    - name: memory
      quantity: 16
    envs:                    # 列表格式
    - name: LOG_LEVEL
      value: "info"
  labels:                    # 列表格式
  - key: project
    value: demo
  datasources:
  - type: pfs
    name: pfs-xxxx
    mountPath: /mnt/pfs
```

示例：创建单节点工作流

> **说明：**
>
> - 示例中的队列 ID、存储 ID 等参数，请替换为工作流所在资源池下的实际值。
> - 为便于说明，示例中的任务模板仅包含必要参数。完整参数详情请参见创建训练任务。

以下工作流包含一个节点，该节点的子任务休眠 30 秒后自动结束。

1. 定义任务模板。

```
  taskTemplates:
  - name: hello-world-temp
    type: CustomTask
    spec:                    # 与创建训练任务接口参数相同
      queue: aihcq-xxxx      # 替换为实际的队列 ID
      jobType: PyTorchJob
      command: /bin/bash -c "echo 'I am a aihc-workflow test job' && sleep 30s"
      jobSpec:
        replicas: 1
        image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
        envs:
        - name: NCCL_DEBUG
          value: DEBUG
      datasources: []        # 存储挂载
```

2. 编排节点，引用上一步定义的任务模板。

```
  tasks:
  - name: hello-world                # 节点名称
    taskTemplateName: hello-world-temp  # 引用的任务模板名称
```

完整 YAML 如下：

```
  version: v1
kind: PipelineTemplate
taskTemplates:
- name: hello-world-temp
  type: CustomTask
  spec:
    queue: aihcq-xxxx
    jobType: PyTorchJob
    command: /bin/bash -c "echo 'I am a aihc-workflow test job' && sleep 30s"
    jobSpec:
      replicas: 1
      image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
      envs:
      - name: NCCL_DEBUG
        value: DEBUG
    datasources: []
tasks:
- name: hello-world
  taskTemplateName: hello-world-temp
```

示例：创建 RayJob 类型的节点

将 `spec.jobType` 设置为 `RayJob` 后，节点的子任务以 Ray 作业运行。平台会为每个子任务创建一个独立的 Ray 集群，包含 1 个 Head 节点和若干 Worker 节点；集群就绪后，平台自动启动 Submitter 节点，将 `command` 作为 entrypoint 提交到集群。子任务结束后，Ray 集群自动释放。Ray 任务的更多说明，详情请参见快速提交Ray任务。

RayJob 与 PyTorchJob 的配置差异如下：

配置项	PyTorchJob	RayJob
jobType	PyTorchJob（默认值）	RayJob
jobSpec	单个 JobSpec 对象	按角色组织，须包含 Head
command	各实例执行的命令	Ray 作业的 entrypoint

配置说明

- 角色：Head 为必选角色，replicas 固定为 1。Worker 角色名称可自定义，也可以配置多组 Worker，例如 CpuWorker 和 GpuWorker 分别使用不同资源。
- 启动命令：直接填写入口命令，例如 python3 /mnt/pfs/code/step1.py，无需填写 ray start 或 ray job submit。
- 镜像：镜像中须包含 ray[default]，平台已验证 Ray 2.7～2.49 版本。建议 Head 与 Worker 使用同一镜像。
- 资源：resources 中的 name 可设置为 cpu（单位：核）、memory（单位：GB）、sharedMemory（单位：GB）；GPU 资源按型号填写资源名，例如 baidu.com/a800_80g_cgpu。纯 CPU 任务无需配置 GPU。

RayJob 还支持以下可选字段：

字段	说明
preInitCommand	Ray 前置执行命令
advancedSettings.runtimeEnv	Ray 运行时环境，JSON 字符串
advancedSettings.submitterBackoffLimit	Submitter 最大重试次数，默认值为 0
advancedSettings.redisInstanceId	云数据库 Redis 实例 ID，用于 Head 节点容错

开启 Head 节点容错时，Redis 实例须与资源池处于同一 VPC，且 submitterBackoffLimit 须大于 0。账号与密码通过 redisUsername、redisPassword 配置，详情请参见数据结构中的 AdvancedSettings。

> **注意：**
>
> - RayJob 不支持 TensorBoard 和任务保留时间（retentionPeriod），告警仅支持任务状态类告警。
> - RayJob 不支持平台容错与诊断，支持 Ray 原生的自动重试。

运行后，您可以在运行记录的子任务列表中查看日志：作业输出在 Submitter 节点的标准输出中，Ray 集群系统日志在 Head、Worker 节点中。

完整 YAML

以下工作流包含两个串行执行的 RayJob 节点，镜像、队列和 Worker 实例数通过全局参数统一配置。

```yaml
  version: v1
kind: PipelineTemplate
inputs:
- name: queue_id
  type: string
  hint: 队列 ID
  defaultValue: aihcq-xxxx          # 替换为实际的队列 ID
- name: image
  type: string
  hint: 包含 ray[default] 的镜像
  defaultValue: <your-ray-image>    # 替换为百舸预置 Ray 镜像或自定义镜像
- name: worker_replicas
  type: int
  hint: Worker 实例数
  defaultValue: 2
taskTemplates:
- name: ray-step
  type: CustomTask
  inputs:
  - name: script
    type: string
    hint: 入口脚本路径
  spec:
    queue: '{{pipeline.parameters.queue_id}}'
    jobType: RayJob
    command: python3 {{inputs.parameters.script}}   # 作为 Ray 作业的 entrypoint
    jobSpec:
      Head:                     # 必选，replicas 固定为 1
        replicas: 1
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 4
        - name: memory
          quantity: 16
      Worker:                   # 角色名称可自定义，可配置多组
        replicas: '{{pipeline.parameters.worker_replicas}}'
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 8
        - name: memory
          quantity: 32
        - name: sharedMemory
          quantity: 8
        # 如需 GPU，追加对应型号的资源，例如：
        # - name: baidu.com/a800_80g_cgpu
        #   quantity: 1
        envs:
        - name: STAGE
          value: "demo"
    advancedSettings:           # 可选
      submitterBackoffLimit: 1
    datasources:
    - type: pfs
      name: pfs-xxxx            # 替换为资源池绑定的 PFS 实例 ID
      mountPath: /mnt/pfs
tasks:
- name: step-1
  taskTemplateName: ray-step
  inputs:
  - name: script
    value: /mnt/pfs/code/step1.py
- name: step-2
  taskTemplateName: ray-step
  dependencies:
  - step-1
  inputs:
  - name: script
    value: /mnt/pfs/code/step2.py
```

包含多个 RayJob 节点的完整数据管线，详情可参见示例：人类演示数据视觉预处理与质量清洗。您也可以在控制台的**工作流 > 工作流模板**页面，基于预置模板创建工作流。

编排节点依赖关系

节点通过 `dependencies` 声明上游节点，工作流支持以下三种运行结构：

- 串行：节点依次执行，上游节点的子任务结束后才启动下游节点。
- 并行：未声明依赖关系的节点同时运行。
- 菱形：上游节点完成后，多个节点并行执行；这些节点全部完成后，再执行下游节点。

> **注意：** 依赖关系不能成环，节点也不能依赖自身。

以下示例中的任务模板 `demo` 均可替换为前文的任意模板。

串行

task-a、task-b、task-c 三个节点依次执行：

```
 version: v1
kind: PipelineTemplate
taskTemplates:
- name: demo
  type: CustomTask
  spec:
    ...
tasks:
- name: task-a
  taskTemplateName: demo
- name: task-b
  taskTemplateName: demo
  dependencies:            # 上游节点
  - task-a                 # task-a 结束后开始执行
- name: task-c
  taskTemplateName: demo
  dependencies:
  - task-b

##### +--------+
##### |  task-a |
##### +----+----+
##### |
##### +----v----+
##### |  task-b |
##### +----+----+
##### |
##### +----v----+
##### |  task-c |
##### +--------+
```

并行

task-a、task-b、task-c 三个节点同时执行：

```
 version: v1
kind: PipelineTemplate
taskTemplates:
- name: demo
  type: CustomTask
  spec:
    ...
tasks:
- name: task-a
  taskTemplateName: demo
- name: task-b
  taskTemplateName: demo
- name: task-c
  taskTemplateName: demo

##### +--------+   +--------+   +--------+
##### | task-a |   | task-b |   | task-c |
##### +--------+   +--------+   +--------+
```

菱形

task-a 完成后并行执行 task-b 和 task-c，两者都完成后执行 task-d：

```
 version: v1
kind: PipelineTemplate
taskTemplates:
- name: demo
  type: CustomTask
  spec:
    ...
tasks:
- name: task-a
  taskTemplateName: demo
- name: task-b
  taskTemplateName: demo
  dependencies:
  - task-a
- name: task-c
  taskTemplateName: demo
  dependencies:
  - task-a
- name: task-d
  taskTemplateName: demo
  dependencies:
  - task-b
  - task-c

##### +--------+
##### | task-a |
##### +--------+
##### /      \
##### +--------+   +--------+
##### | task-b |   | task-c |
##### +--------+   +--------+
##### \      /
##### +--------+
##### | task-d |
##### +--------+
```

配置输入参数

多个节点引用同一个任务模板、仅部分参数不同时，可以在模板中声明输入参数，再由各节点分别传值；需要统一调整的值，可以定义为全局输入参数，在模板或节点中引用。

> **说明：** 简单工作流也可以让任务模板与节点一一对应，直接在模板的 `spec` 中写入参数值。

输入参数包含以下字段：

字段	是否必选	说明
name	是	参数名称，以字母或下划线开头
type	否	参数类型：string、int、float、bool
hint	否	参数描述
defaultValue	否	默认值

步骤一：定义全局输入参数

在顶层 `inputs` 中定义参数。任务模板和节点中可以通过 `{{pipeline.parameters.<参数名>}}` 引用。

```yaml
inputs:
- name: image_url        # 参数名称
  type: string           # 参数类型
  hint: 镜像地址          # 参数描述
  defaultValue: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
- name: command
  type: string
  hint: 启动命令
  defaultValue: echo "This is a globally defined command."
```

步骤二：定义任务模板输入参数

在任务模板的 `inputs` 中声明参数，在 `spec` 中通过 `{{inputs.parameters.<参数名>}}` 引用。

```yaml
taskTemplates:
- name: task-demo
  type: CustomTask
  inputs:
  - name: queue_id
    type: string
    hint: 队列 ID
    defaultValue: aihcq-xxxx     # 默认值
  - name: image_url
    type: string
    hint: 镜像地址
  - name: command
    type: string
    hint: 启动命令
  spec:
    queue: '{{inputs.parameters.queue_id}}'
    jobType: PyTorchJob
    command: '{{inputs.parameters.command}}'
    jobSpec:
      replicas: 1
      image: '{{inputs.parameters.image_url}}'
    datasources: []
```

步骤三：在节点中为参数赋值

在节点的 `inputs` 中为模板参数赋值，取值可以引用全局参数，也可以是常量。未赋值的参数使用模板中的默认值。

```yaml
tasks:
- name: task-a
  taskTemplateName: task-demo
  inputs:
  - name: image_url
    value: '{{pipeline.parameters.image_url}}'   # 引用全局参数
  - name: command
    value: '{{pipeline.parameters.command}}'
  # 未赋值的 queue_id 使用模板默认值
- name: task-b
  taskTemplateName: task-demo
  dependencies:
  - task-a
  inputs:
  - name: image_url
    value: '{{pipeline.parameters.image_url}}'
  - name: command
    value: echo "This is a task b's command."   # 使用常量
  - name: queue_id
    value: aihcq-yyyy                          # 覆盖模板默认值
```

完整 YAML

```yaml
version: v1
kind: PipelineTemplate
inputs:
- name: image_url
  type: string
  hint: 镜像地址
  defaultValue: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
- name: command
  type: string
  hint: 启动命令
  defaultValue: echo "This is a globally defined command."
taskTemplates:
- name: task-demo
  type: CustomTask
  inputs:
  - name: queue_id
    type: string
    hint: 队列 ID
    defaultValue: aihcq-xxxx
  - name: image_url
    type: string
    hint: 镜像地址
  - name: command
    type: string
    hint: 启动命令
  spec:
    queue: '{{inputs.parameters.queue_id}}'
    jobType: PyTorchJob
    command: '{{inputs.parameters.command}}'
    jobSpec:
      replicas: 1
      image: '{{inputs.parameters.image_url}}'
      envs:
      - name: NCCL_DEBUG
        value: DEBUG
    datasources: []
tasks:
- name: task-a
  taskTemplateName: task-demo
  inputs:
  - name: image_url
    value: '{{pipeline.parameters.image_url}}'
  - name: command
    value: '{{pipeline.parameters.command}}'
- name: task-b
  taskTemplateName: task-demo
  dependencies:
  - task-a
  inputs:
  - name: image_url
    value: '{{pipeline.parameters.image_url}}'
  - name: command
    value: echo "This is a task b's command."
  - name: queue_id
    value: aihcq-yyyy
```

配置输出参数

任务模板可以通过 `output` 声明输出参数：上游节点的子任务将值写入文件，下游节点引用该输出作为输入参数，从而在节点之间传递数据。

> **注意：**
>
> - 输出值从文件中读取，valueFrom.path 须位于对象存储 BOS 的挂载路径下，暂不支持其他存储类型及容器系统路径。
> - 声明了 `output` 的任务模板，BOS 挂载须配置在 dataSources 字段中。
> - 输出值为文件的全部内容。写入时建议使用 `echo -n`，避免末尾换行符影响下游的取值和条件判断。
> - 输出参数只能在 `tasks` 中引用，且只能由下游节点引用上游节点的输出。
> - 被引用输出的节点，名称中不能包含下划线（_）。

步骤一：定义任务模板输出参数

字段	是否必选	说明
name	是	输出参数名称
valueFrom.path	是	输出文件路径，须位于 BOS 挂载路径下

```
taskTemplates:
- name: template-a
  type: CustomTask
  outputs:                    # 输出参数
  - name: version
    valueFrom:
      path: /mnt/bos/version    # BOS 挂载路径下的文件
  spec:
    command: echo -n "v2.1.0" > /mnt/bos/version
    ...
    dataSources:
    - type: bos
      sourcePath: my-bucket/datasets/
      mountPath: /mnt/bos
- name: template-b
  type: CustomTask
  inputs:
  - name: input_version
    type: string
    hint: 版本号
  spec:
    command: echo '{{inputs.parameters.input_version}}'
    ...
```

步骤二：在下游节点中引用输出

在下游节点的 `inputs` 中，通过 {{tasks.<上游节点名称>.outputs.parameters.<输出参数名称>}} 引用上游节点的输出：

```
tasks:
- name: task-a
  taskTemplateName: template-a
- name: task-b
  taskTemplateName: template-b
  dependencies:
  - task-a
  inputs:
  - name: input_version
    value: '{{tasks.task-a.outputs.parameters.version}}'
```

完整 YAML

```
version: v1
kind: PipelineTemplate
taskTemplates:
- name: template-a
  type: CustomTask
  outputs:
  - name: version
    valueFrom:
      path: /mnt/bos/version
  spec:
    queue: aihcq-xxxx
    jobType: PyTorchJob
    command: echo -n "v2.1.0" > /mnt/bos/version
    jobSpec:
      replicas: 1
      image: registry.baidubce.com/aihc-aiak/aiak-megatron-ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
    dataSources:
    - type: bos
      sourcePath: my-bucket/datasets/
      mountPath: /mnt/bos
- name: template-b
  type: CustomTask
  inputs:
  - name: input_version
    type: string
    hint: 版本号
  spec:
    queue: aihcq-xxxx
    jobType: PyTorchJob
    command: echo '{{inputs.parameters.input_version}}'
    jobSpec:
      replicas: 1
      image: registry.baidubce.com/aihc-aiak/aiak-megatron-ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
tasks:
- name: task-a
  taskTemplateName: template-a
- name: task-b
  taskTemplateName: template-b
  dependencies:
  - task-a
  inputs:
  - name: input_version
    value: '{{tasks.task-a.outputs.parameters.version}}'
```

配置条件执行

节点可以通过 `when` 字段配置执行条件。上游节点全部结束后，平台计算该表达式：结果为真时执行节点，否则跳过该节点，运行记录中显示为"已跳过"。被跳过的节点视为已满足依赖，其下游节点会继续执行。`when` 通常与全局参数或上游输出配合使用，实现分支流程。

`when` 表达式中的引用写法如下：

引用对象	写法
全局输入参数	{{pipeline.inputs.<参数名称>}}
上游节点输出	{{tasks.<上游节点名称>.outputs.<输出参数名称>}}

> **注意：** 在 `when` 中引用上游输出时，无需写 `parameters.`，与 `inputs` 中的引用写法不同。

示例：全局参数 `flag` 为 true，且 task-a 输出的 `version` 为 v2.1.0 时，执行 task-b。

```
- name: task-b
  taskTemplateName: template-b
  dependencies:
  - task-a
  when: '{{pipeline.inputs.flag}} == true && {{tasks.task-a.outputs.version}} == v2.1.0'
```

完整 YAML

```
  version: v1
kind: PipelineTemplate
inputs:
- name: flag
  type: bool
  hint: 是否执行 task-b
  defaultValue: true
taskTemplates:
- name: template-a
  type: CustomTask
  outputs:
  - name: version
    valueFrom:
      path: /mnt/bos/version
  spec:
    queue: aihcq-xxxx
    jobType: PyTorchJob
    command: echo -n "v2.1.0" > /mnt/bos/version
    jobSpec:
      replicas: 1
      image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
    dataSources:
    - type: bos
      sourcePath: my-bucket/datasets/
      mountPath: /mnt/bos
- name: template-b
  type: CustomTask
  inputs:
  - name: input_version
    type: string
    hint: 版本号
  spec:
    queue: aihcq-xxxx
    jobType: PyTorchJob
    command: echo '{{inputs.parameters.input_version}}'
    jobSpec:
      replicas: 1
      image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
tasks:
- name: task-a
  taskTemplateName: template-a
- name: task-b
  taskTemplateName: template-b
  dependencies:
  - task-a
  inputs:
  - name: input_version
    value: '{{tasks.task-a.outputs.parameters.version}}'
  # flag 为 true 且上游输出的版本号为 v2.1.0 时执行，否则跳过
  when: '{{pipeline.inputs.flag}} == true && {{tasks.task-a.outputs.version}} == v2.1.0'
```

附录：变量引用

写法	适用位置
{{pipeline.parameters.<名称>}}	任务模板的 spec、inputs 默认值；节点的 inputs
{{inputs.parameters.<名称>}}	任务模板的 spec
{{tasks.<节点>.outputs.parameters.<名称>}}	节点的 inputs
{{pipeline.inputs.<名称>}}	节点的 when
{{tasks.<节点>.outputs.<名称>}}	节点的 when

取值时机说明：

- 全局参数和模板参数在工作流提交运行时替换为实际值。
- 上游输出在运行到引用它的节点时才取值。
- 字段值仅包含一个变量时（例如 `replicas: '{{pipeline.parameters.worker_replicas}}'`），替换后保留参数的原始类型，int 类型参数替换后仍为数字。

相关文档

- 创建训练任务
- 数据结构
- 快速提交Ray任务
- 人类演示数据视觉预处理与质量清洗

运行工作流

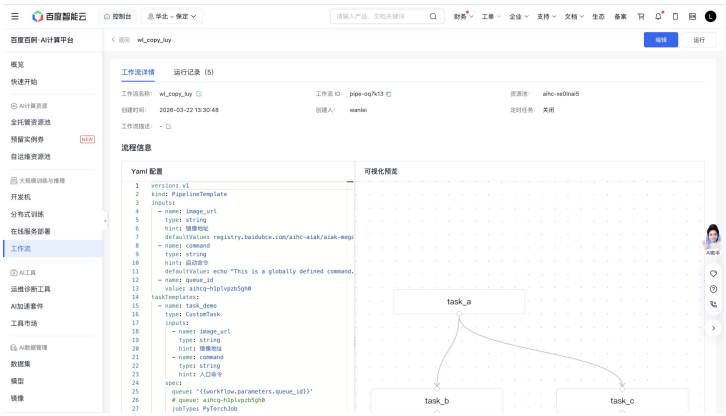
概述

本文档介绍如何通过控制台运行工作流，以及在工作流因资源不足、平台故障等非工作流参数配置原因运行失败时，如何对失败运行记录进行重跑。

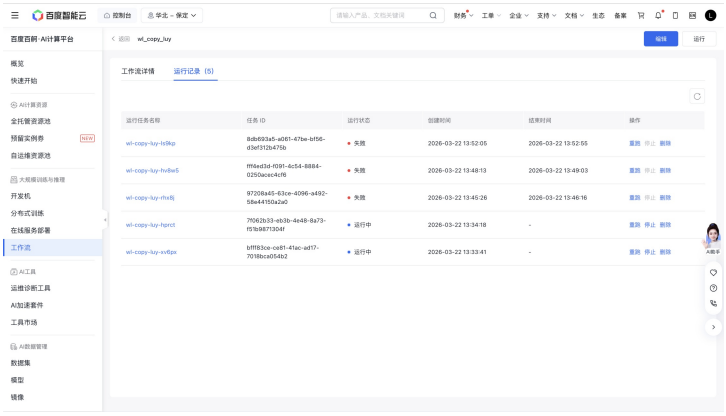
前提条件

- 已登录百度智能云控制台，并进入百度百舸 · AI 计算平台。
- 已创建可运行的工作流。
- 如需重跑工作流，目标工作流的 **运行记录** 页签中需存在失败状态的运行记录。## 运行工作流

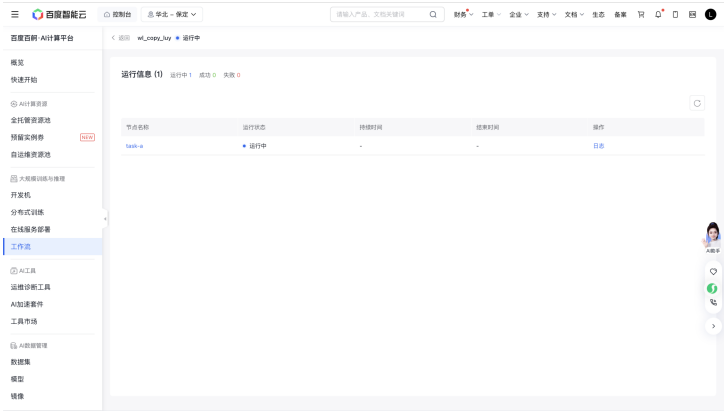
1. 在控制台进入 **百度百舸 · AI 计算平台->工作流**，打开目标工作流详情页，单击 **【运行】**，提交运行。



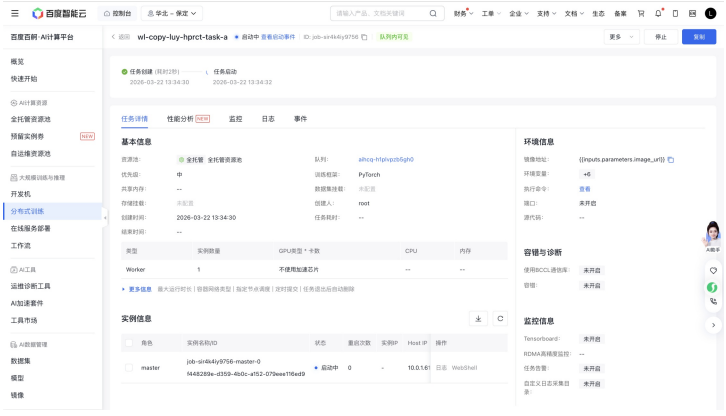
2. 运行提交后，在 **【运行记录】** 中查看当前工作流的全部任务实例，找到本次运行的状态和子任务详情。



3. 进入运行记录详情页，查看工作流各子任务运行状态。



4. 单击 **【任务名称】** 或 **【日志】**，跳转到分布式训练任务的详情页查看任务详细信息。

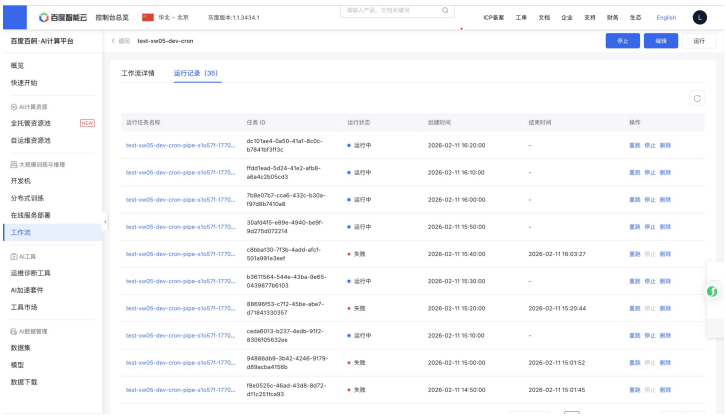


重跑

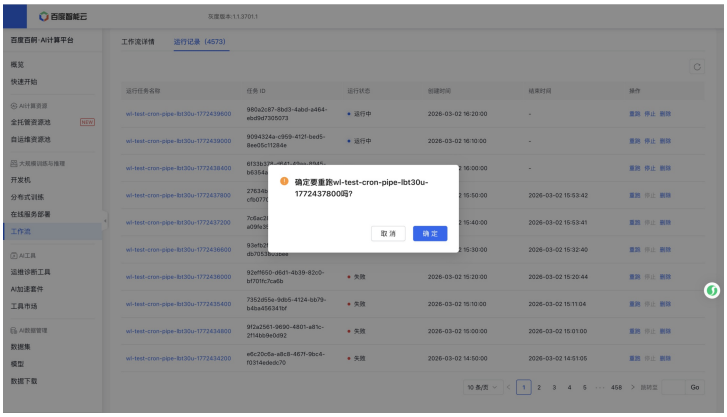
工作流的运行记录级重跑功能，旨在解决工作流因资源不足、平台故障等非工作流参数配置原因导致的运行失败时快速重新运行问题。

按照以下步骤，从失败的工作流记录重跑一个工作流任务。

1. 在控制台进入 **百度百舸 · AI 计算平台->工作流**，打开目标工作流详情页后切换到 **【运行记录】** 页签。找到状态为失败的运行记录，在该记录的 **【操作】** 列单击 **【重跑】**。



2. 在弹出的 **【确认】** 对话框中单击 **【确定】**。系统会立即基于该失败运行记录创建一条新的运行记录，可在 **【运行记录】** 列表中查看新任务状态。



开发任务模板

此文档讲解如何使用分布式训练任务开发自己的业务任务模板。

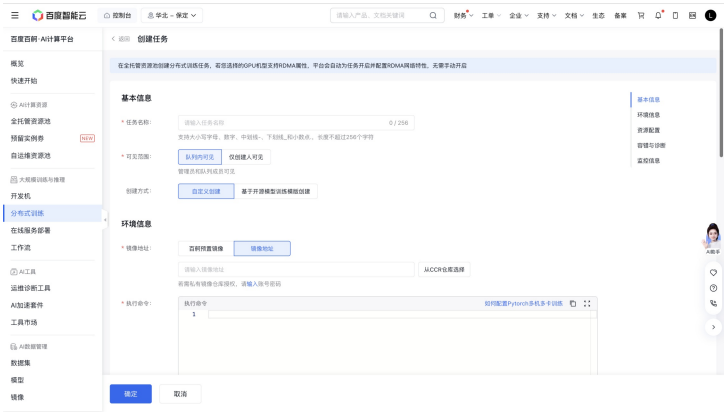
在进行工作流编排前，开发者需要先开发单个任务，将工作流中的每个单个任务在分布式训练任务中进行提交验证，一旦单个任务顺利跑通，继续将任务制作为任务模板即可在工作流中被复用，沉淀为AI资产。

任务模板开发

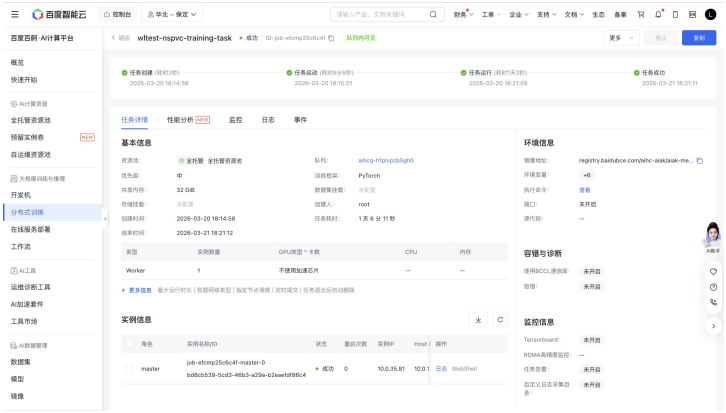
工作流的CustomTask对应百舸的分布式训练任务，所以百舸CustomTask类型的任务模板支持的参数与分布式训练任务完全一致。

在分布式训练任务中跑通任务

1. 任务代码开发完成后，进入百舸控制台，点击左侧【分布式训练】，点击页面的【创建任务】填写表单验证任务。创建训练任务详细步骤参考 创建训练任务



2. 任务跑通后，将任务参数信息整理为工作流任务基础模板，即将创建任务时填写的表单项使用接口参数方式表达，任务参数可以在任务详情页查看。



1. 以下模板主要以数据处理场景进行演示
2. 完整的任务参数模板可参考 任务模板示例/固定参数任务模板

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: custom-task-demo
    type: CustomTask
    spec:
      queue: aihcq-h1plvpzb5gh0
      jobType: PyTorchJob
      command: sleep 5s
      priority: normal
      jobSpec:
        image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
        replicas: 1
        resources: []
        envs:
          - name: SOURCE_DATA_DIR
            value: bos://my-bucket/datasets/source
          - name: TARGET_DATA_DIR
            value: bos://my-bucket/datasets/target
      datasources:
        - type: pfs
          name: pfs-pxE6jz
          sourcePath: /
          mountPath: /mnt/cluster
        - type: bos
          name: ""
          sourcePath: my-bucket/datasets
          mountPath: /mnt/bos/data
tasks:
  - name: task_a
    taskTemplateName: custom-task-demo
```

设计任务模板参数 将任务中可以外部传入的参数设置为input参数，推荐的设计原则：

1. 如果不同任务使用的资源量不同，将资源参数提取为input
2. 所有的任务逻辑控制参数，放置在envs中传入

上述示例中我们将环境变量中定义的数据目录（SOURCE_DATA_DIR、TARGET_DATA_DIR）、执行命令（command）、BOS挂载地址设置为模板的可传入参数

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: custom-task-demo
    type: CustomTask
    inputs:
      - name: command # 参数名
        type: string  # 参数类
        hint: 执行命令  # 参数描述
      - name: source_data_dir # 参数名
        type: string # 参数类型
        hint: 源数据目录 # 参数描述
      - name: target_data_dir # 参数名
        type: string # 参数类型
        hint: 目标数据目录 # 参数描述
      - name: bos_dir # 参数名
        type: string # 参数类型
        hint: BOS存储地址 # 参数描述
    spec:
      queue: aihcq-h1plvpzb5gh0
      jobType: PyTorchJob
      command: '{{inputs.parameters.command}}'
      jobSpec:
        replicas: 1
        image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
        resources: []
        envs:
          - name: SOURCE_DATA_DIR
            value: '{{inputs.parameters.source_data_dir}}'
          - name: TARGET_DATA_DIR
            value: '{{inputs.parameters.target_data_dir}}'
      labels: []
      datasources:
        - type: bos
          name: ""
          sourcePath: '{{inputs.parameters.bos_dir}}'
          mountPath: /mnt/bos/data
tasks:
  - name: task_a
    taskTemplateName: custom-task-demo
    inputs:
      - name: command
        value: echo "This is a defined command."
      - name: source_data_dir
        value: my-bucket/datasets/source
      - name: target_data_dir
        value: my-bucket/datasets/target
      - name: bos_dir
        value: my-bucket/datasets
```

在工作流中使用任务模板

以下示例设置job-1打印 `This is job_a.` ，job-2设置打印 `This is job_b.`

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: custom-task-demo
    type: CustomTask
    inputs:
      - name: command # 参数名
        type: string  # 参数类
        hint: 执行命令  # 参数描述
      - name: source_data_dir # 参数名
        type: string # 参数类型
        hint: 源数据目录 # 参数描述
      - name: target_data_dir # 参数名
        type: string # 参数类型
        hint: 目标数据目录 # 参数描述
      - name: bos_dir # 参数名
        type: string # 参数类型
        hint: BOS存储地址 # 参数描述
    spec:
      queue: aihcq-h1plvpzb5gh0
      jobType: PyTorchJob
      command: '{{inputs.parameters.command}}'
      jobSpec:
        replicas: 1
        image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
        resources: []
        envs:
          - name: SOURCE_DATA_DIR
            value: '{{inputs.parameters.source_data_dir}}'
          - name: TARGET_DATA_DIR
            value: '{{inputs.parameters.target_data_dir}}'
      labels: []
      datasources:
        - type: bos
          name: ""
          sourcePath: '{{inputs.parameters.bos_dir}}'
          mountPath: /mnt/bos/data
tasks:
  - name: task_a
    taskTemplateName: custom-task-demo
    inputs:
      - name: command
        value: echo "This is job_a."
      - name: source_data_dir
        value: bos://my-bucket/datasets/source/a
      - name: target_data_dir
        value: bos://my-bucket/datasets/target/a
      - name: bos_dir
        value: my-bucket/datasets/a
  - name: task_b
    taskTemplateName: custom-task-demo
    inputs:
      - name: command
        value: echo "This is job_b."
      - name: source_data_dir
        value: bos://my-bucket/datasets/source/b
      - name: target_data_dir
        value: bos://my-bucket/datasets/target/b
      - name: bos_dir
        value: my-bucket/datasets/b
    dependencies:
      - task_a
```

以上我们讲解了如何开发一个简单的任务模板，更多高级的任务参数你可以继续看后续任务示例模板。

任务模板示例

以下提供通用任务模板供参考,实际使用时可以根据任务参数需求自行删减参数。

通用

通用任务模板

参数说明：

参数名称	说明
command	任务的执行命令
source_data_dir	源数据集的目录
target_data_dir	数据集的保存目录
bos_dir	挂载的BOS目录

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: custom-task-demo
    type: CustomTask
    inputs:
      - name: command
        type: string
        hint: 执行命令
      - name: source_data_dir
        type: string
        hint: 源数据目录
      - name: target_data_dir
        type: string
        hint: 目标数据目录
      - name: bos_dir
        type: string
        hint: 挂载的BOS目录
    spec:
      queue: aihcq-h1plvpzb5gh0
      jobType: PyTorchJob
      command: '{{inputs.parameters.command}}'
      jobSpec:
        replicas: 1
        image: >-
          registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
        resources: []
        envs:
          - name: SOURCE_DATA_DIR
            value: '{{inputs.parameters.source_data_dir}}'
          - name: TARGET_DATA_DIR
            value: '{{inputs.parameters.target_data_dir}}'
      labels: []
      datasources:
        - type: bos
          name: ''
          sourcePath: '{{inputs.parameters.bos_dir}}'
          mountPath: /mnt/bos
tasks:
  - name: task_a
    taskTemplateName: custom-task-demo
    inputs:
      - name: command
        value: echo "This is job_a."
      - name: source_data_dir
        value: bos://my-bucket/datasets/source
      - name: target_data_dir
        value: bos://my-bucket/datasets/target
      - name: bos_dir
        value: my-bucket/datasets
```

分布式训练任务模板

以下是完整的分布式训练任务参数，参数说明参考 分布式训练任务的OpenAPI接口。

开发者可以基于完整参数模板开发自己训练、数据处理等业务任务模板。

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: custom-task-template
    type: CustomTask
    spec:
      queue: aihcq-xxxxx
      jobType: PyTorchJob
      command: sleep 30s
      priority: normal
      enableBccl: false
      faultTolerance: true
      faultToleranceArgs: --enable-replace=true --enable-hang-detection=true
        --hang-detection-log-timeout-minutes=7
        --hang-detection-startup-toleration-minutes=15
        --hang-detection-stack-timeout-minutes=3 --max-num-of-unconditional-retry=2
        --custom-log-patterns=timeout1 --custom-log-patterns=timeout2
      retentionPeriod: 1d
      jobSpec:
        image: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
        imageConfig:
          username: your-registry-username
          password: your-registry-password
        replicas: 2
        resources:
          - name: baidu.com/a800_80g_cgpu
            quantity: 8
          - name: cpu
            quantity: 96
          - name: memory
            quantity: 512
          - name: sharedMemory
            quantity: 64
        envs:
          - name: NCCL_DEBUG
            value: INFO
          - name: NCCL_IB_DISABLE
            value: "0"
          - name: CUDA_VISIBLE_DEVICES
            value: 0,1,2,3,4,5,6,7
        enableRDMA: true
```

```
      hostNetwork: false
    labels:
      - key: project
        value: llm-training
      - key: team
        value: ai-platform
    datasources:
      - type: pfs
        name: pfs-pxE6jz
        sourcePath: /
        mountPath: /mnt/cluster
        options:
          readOnly: false
      - type: hostPath
        name: host-data
        sourcePath: /data/shared
        mountPath: /mnt/host-data
        options:
          readOnly: true
      - type: bos
        name: ""
        sourcePath: my-bucket/datasets/
        mountPath: /mnt/bos-data
        options:
          readOnly: true
      - type: cfs
        name: cfs-instance-id
        sourcePath: /
        mountPath: /mnt/cfs-data
        options:
          readOnly: false
      - type: rapidfs
        name: rapidfs-instance-id
        sourcePath: /
        mountPath: /mnt/rapidfs-data
        options:
          readOnly: false
      - type: dataset
        name: dataset-id
        sourcePath: /
        mountPath: /mnt/dataset
        options:
          readOnly: true
    tensorboardConfig:
      enable: true
      logPath: /mnt/cluster/tensorboard-logs
    alertConfig:
      instanceId: your-cluster-monitor-instance-id
      alertItems:
        - jobRunning
        - jobFT
        - nodeFT
        - jobFailed
        - jobSucceed
        - jobHang
      for: 0m
      notifyRuleId: notify-xxxxxxxx
tasks:
- name: job-3
  taskTemplateName: custom-task-template
- name: job-2
  taskTemplateName: custom-task-template
```

> 在业务实践中可以使用"固定参数+自定义传参"的混合方式将常用的数据处理任务、训练任务、测评任务等固定为任务模板在不同的工作流中复用

数据集下载

从魔搭下载数据集

参数说明：

参数名称	说明
queue_id	队列ID，示例：aihcq-h1plvp
dataset_name	魔搭社区的数据集名称，示例：liucong/Chinese-DeepSeek-R1-Distill-data-110k
target_data_dir	数据集的保存目录，示例：my-bucket/datasets

yaml模板：

```yaml
version: v1
kind: PipelineTemplate

taskTemplates:
  - name: template-dataset-download-modelscope
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
      - name: dataset_name
        type: string
        hint: 数据集名称
      - name: target_data_dir
        type: string
        hint: 数据集在BOS的存储目录
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: |
        #!/bin/sh

        # 检查是否已安装 modelscope
        if ! command -v modelscope >/dev/null 2>&1; then
            echo "modelscope 未安装，正在安装..."
            pip install --user modelscope
            export PATH="$HOME/.local/bin:$PATH"
        fi

        # 使用 modelscope CLI 下载数据集
        echo "正在下载数据集 $DATASET_NAME..."
        modelscope download --dataset "$DATASET_NAME" --revision master --local_dir /mnt/bos/data

        if [ $? -eq 0 ]; then
            echo "数据集已成功下载至: /mnt/bos/data"
        else
            echo "下载失败！请检查网络、权限或数据集是否存在。"
            exit 1
        fi
      jobSpec:
        image: registry.baidubce.com/aihcp-public/pytorch-2.7.0-cu12.8.61-py3.12-ubuntu24.04
        replicas: 1
        envs:
          - name: DATASET_NAME
            value: '{{inputs.parameters.dataset_name}}'
      datasources:
        - type: bos
          name: ""
          sourcePath: '{{inputs.parameters.target_data_dir}}'
          mountPath: /mnt/bos/data

tasks:
  - name: job-1
    taskTemplateName: template-dataset-download-modelscope
    inputs:
      - name: queue_id
        value: aihcq-h1plvp
      - name: dataset_name
        value: liucong/Chinese-DeepSeek-R1-Distill-data-110k
      - name: target_data_dir
        value: my-bucket/datasets
```

从HuggingFace下载数据集

从HuggingFace下载数据集，需要自行保证集群可以正常访问HuggingFace

参数说明：

参数名称	说明
queue_id	队列ID，示例：aihcq-h1plvp
dataset_name	魔搭社区的数据集名称，示例：liucong/Chinese-DeepSeek-R1-Distill-data-110k
target_data_dir	数据集的保存目录，示例：my-bucket/datasets

yaml模板：

```
version: v1
kind: PipelineTemplate

taskTemplates:
  - name: template-dataset-download-huggingface
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
      - name: dataset_name
        type: string
        hint: 数据集名称
      - name: target_data_dir
        type: string
        hint: 数据集在BOS的存储目录
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: |
        #!/bin/sh

        # 构建本地目标路径
        LOCAL_DIR="/mnt/bos/data/$DATASET_NAME"
        mkdir -p "$LOCAL_DIR"

        # 检查是否已存在（简单判断）
        if [ -f "$LOCAL_DIR/.hf_download_complete" ]; then
            echo "数据集似乎已下载（检测到 .hf_download_complete 标记），跳过。"
            exit 0
        fi

        # 安装 huggingface_hub（如果未安装）
        if ! python3 -c "import huggingface_hub" &> /dev/null; then
            echo "正在安装 huggingface_hub..."
            pip install --quiet huggingface_hub
        fi

        # 创建临时 Python 脚本
        PY_SCRIPT=$(cat <<EOF
        from huggingface_hub import snapshot_download
        import os

        local_dir = os.environ['LOCAL_DIR']
        repo_id = os.environ['DATASET_NAME']

        print(f"正在从 Hugging Face 下载数据集: {repo_id}")
        snapshot_download(
            repo_id=repo_id,
            local_dir=local_dir,
            repo_type="dataset",
            max_workers=8
        )
        print("下载完成！")
        EOF
        )

        # 导出环境变量供 Python 使用
        export LOCAL_DIR="$LOCAL_DIR"
        export DATASET_NAME="$DATASET_NAME"

        # 执行下载
        echo "开始从 Hugging Face Hub 下载数据集..."
        python3 -c "$PY_SCRIPT"

        # 检查是否成功并创建标记文件
        if [ $? -eq 0 ]; then
            touch "$LOCAL_DIR/.hf_download_complete"
            echo "数据集已保存至: $LOCAL_DIR"
        else
            echo "下载失败。请检查数据集 ID、网络或访问权限。"
            exit 1
        fi
      jobSpec:
        image: registry.baidubce.com/aihcp-public/pytorch-2.7.0-cu12.8.61-py3.12-ubuntu24.04
        replicas: 1
        envs:
          - name: DATASET_NAME
            value: '{{inputs.parameters.dataset_name}}'
        datasources:
          - type: bos
            name: ""
            sourcePath: '{{inputs.parameters.target_data_dir}}'
            mountPath: /mnt/bos/data

tasks:
  - name: job-1
    taskTemplateName: template-dataset-download-huggingface
    inputs:
      - name: queue_id
        value: aihcq-h1plvp
      - name: dataset_name
        value: liucong/Chinese-DeepSeek-R1-Distill-data-110k
      - name: target_data_dir
        value: my-bucket/datasets
```

数据转储

从BOS拉取数据到PFS

可用于从BOS动态加载冷数据到PFS上用于训练、数据处理等

参数说明：

参数名称	说明
queue_id	队列ID，示例：aihcq-h1plvp
bos_path	BOS源路径，示例：bos://my-bucket/datasets/
bos_ak	访问BOS的ak
bos_sk	访问BOS的sk
pfs_path	PFS源路径，示例：/datasets/my-dataset
pfs_id	PFS实例ID

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: bos-to-pfs
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
      - name: bos_path
        type: string
        hint: 数据集的BOS路径
      - name: bos_ak
        type: string
        hint: BOS存储桶的访问ak
      - name: bos_sk
        type: string
        hint: BOS存储桶的访问sk
      - name: pfs_path
        type: string
        hint: PFS源路径
      - name: pfs_id
        type: string
        hint: PFS实例ID
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: |
        WORK_DIR="$(pwd)"
        ZIP_URL="https://doc.bce.baidu.com/bos-optimization/mac-bcecmd-0.5.10.zip"
        ZIP_FILE="mac-bcecmd-0.5.10.zip"
        EXTRACT_DIR="mac-bcecmd-0.5.10"
        echo "正在下载 bcecmd 工具..."
        curl -LO "$ZIP_URL"
        if [ $? -ne 0 ]; then
            echo "下载失败！请检查网络或 URL。"
            exit 1
        fi
        echo "正在解压..."
        unzip -o "$ZIP_FILE"
        if [ $? -ne 0 ]; then
            echo "解压失败！"
            exit 1
        fi
        cd "$EXTRACT_DIR" || { echo " 无法进入目录 $EXTRACT_DIR"; exit 1; }
        echo " 正在创建 credentials 文件..."
        mkdir -p /root/.go-bcecli
        cat > ~/.go-bcecli/credentials <<EOF
        [Defaults]
        Ak = "$BOS_AK"
        Sk = "$BOS_SK"
        EOF
        chmod 600 ~/.go-bcecli/credentials
        echo "bcecmd工具安装及配置完成,开始下载数据"
        ./bcecmd bos sync {{inputs.parameters.bos_path}} /mnt/cluster/dataset
        echo "数据集下载成功，已保存到PFS的{{inputs.parameters.pfs_path}}路径"
      jobSpec:
        image: registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04
        replicas: 1
        envs:
          - name: BOS_AK
            value: '{{inputs.parameters.bos_ak}}'
          - name: BOS_SK
            value: '{{inputs.parameters.bos_sk}}'
      datasources:
        - type: pfs
          name: '{{inputs.parameters.pfs_id}}'
          sourcePath: '{{inputs.parameters.pfs_path}}'
          mountPath: /mnt/cluster/dataset
tasks:
  - name: sync-bos-to-pfs
    taskTemplateName: bos-to-pfs
    inputs:
      - name: queue_id
        value: aihcq-h1plvpzb5gh0
      - name: bos_path
        value: bos://my-bucket/datasets/
      - name: bos_ak
        value: <你的sk>
      - name: bos_sk
        value: <你的sk>
      - name: pfs_path
        value: /datasets/my-dataset
      - name: pfs_id
        value: pfs-xxxx
```

备份PFS数据到BOS

可用于动态备份PFS上的数据到BOS存储，释放PFS存储空间

参数说明：

参数名称	说明
queue_id	队列ID，示例：aihcq-h1plvp
bos_path	BOS源路径，示例：bos://my-bucket/datasets/
bos_ak	访问BOS的ak
bos_sk	访问BOS的sk
pfs_path	PFS源路径，示例：/datasets/my-dataset
pfs_id	PFS实例ID

```yaml
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: pfs-to-bos
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
      - name: bos_path
        type: string
        hint: 数据集的BOS路径
      - name: bos_ak
        type: string
        hint: BOS存储桶的访问ak
      - name: bos_sk
        type: string
        hint: BOS存储桶的访问sk
      - name: pfs_path
        type: string
        hint: PFS源路径
      - name: pfs_id
        type: string
        hint: PFS实例ID
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: |
        WORK_DIR="$(pwd)"
        ZIP_URL="https://doc.bce.baidu.com/bos-optimization/mac-bcecmd-0.5.10.zip"
        ZIP_FILE="mac-bcecmd-0.5.10.zip"
        EXTRACT_DIR="mac-bcecmd-0.5.10"
        echo "正在下载 bcecmd 工具..."
        curl -LO "$ZIP_URL"
        if [ $? -ne 0 ]; then
            echo "下载失败！请检查网络或 URL。"
            exit 1
        fi
        echo "正在解压..."
        unzip -o "$ZIP_FILE"
        if [ $? -ne 0 ]; then
            echo "解压失败！"
            exit 1
        fi
        cd "$EXTRACT_DIR" || { echo " 无法进入目录 $EXTRACT_DIR"; exit 1; }
        echo " 正在创建 credentials 文件..."
        mkdir -p /root/.go-bcecli
        cat > ~/.go-bcecli/credentials <<EOF
        [Defaults]
        Ak = "$BOS_AK"
        Sk = "$BOS_SK"
        EOF
        chmod 600 ~/.go-bcecli/credentials
        echo "bcecmd工具安装及配置完成,开始上传数据"
        ./bcecmd bos sync /mnt/cluster/dataset {{inputs.parameters.bos_path}}
        echo "数据集上传成功，已保存到BOS的{{inputs.parameters.bos_path}}路径"
      jobSpec:
        image: registry.baidubce.com/inference/aibox-ubuntu-v2.0-22.04
        replicas: 1
        envs:
          - name: BOS_AK
            value: '{{inputs.parameters.bos_ak}}'
          - name: BOS_SK
            value: '{{inputs.parameters.bos_sk}}'
      datasources:
        - type: pfs
          name: '{{inputs.parameters.pfs_id}}'
          sourcePath: '{{inputs.parameters.pfs_path}}'
          mountPath: /mnt/cluster/dataset
tasks:
  - name: sync-pfs-to-bos
    taskTemplateName: pfs-to-bos
    inputs:
      - name: queue_id
        value: aihcq-h1plvpzb5gh0
      - name: bos_path
        value: bos://my-bucket/datasets/
      - name: bos_ak
        value: <你的sk>
      - name: bos_sk
        value: <你的sk>
      - name: pfs_path
        value: /datasets/my-dataset
      - name: pfs_id
        value: pfs-xxxx
```

清理PFS存储的数据

清理PFS上的数据，释放存储空间，如训练结束后自动清理数据集

参数说明：

参数名称	说明
queue_id	队列ID
pfs_path	PFS源路径，示例：/datasets/my-dataset
pfs_id	PFS实例ID

```
version: v1
kind: PipelineTemplate
taskTemplates:
  - name: pfs-remove
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
      - name: pfs_id
        type: string
        hint: PFS实例ID
      - name: pfs_path
        type: string
        hint: PFS源路径
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: |
        rm -r /mnt/cluster/dataset
        echo "数据集删除成功，已删除PFS的{{inputs.parameters.pfs_path}}路径上的数据"
      priority: normal
      jobSpec:
        image: registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04
        replicas: 1
      datasources:
        - type: pfs
          name: '{{inputs.parameters.pfs_id}}'
          sourcePath: '{{inputs.parameters.pfs_path}}'
          mountPath: /mnt/cluster/dataset
tasks:
  - name: pfs-file-remove
    taskTemplateName: pfs-remove
    inputs:
      - name: queue_id
        value: aihcq-h1plvpzb5gh0
      - name: pfs_path
        value: /datasets/my-dataset
      - name: pfs_id
        value: pfs-xxxx
```

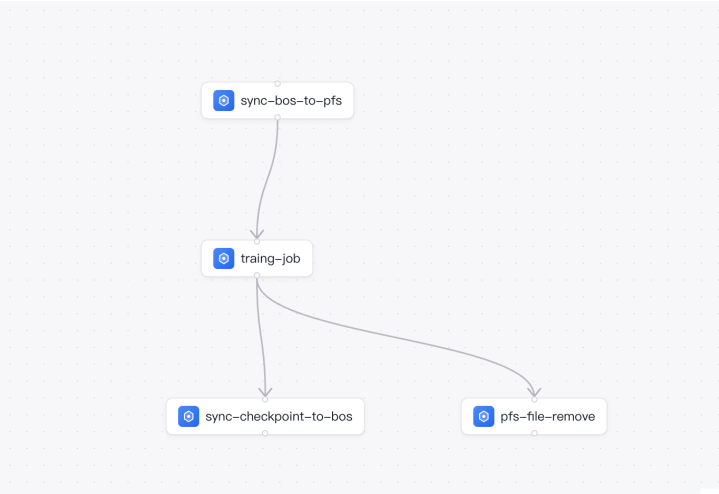
> 更多模板持续更新中...

工作流模板示例

此文档涵盖典型业务场景工作流模板，开发者可以使用这些模板快速修改应用的自己的工作流当中。

数据集动态加载

以下工作流实现BOS冷数据动态加载至PFS并在训练后自动清理。



简介

背景 在模型训练场景中，由于训练程序需要极高的磁盘I/O吞吐，通常建议将训练数据集放置在并行文件存储PFS中以保障训练效率。而PFS的存储成本相对较高，不适合作为数据的长期持久化存储。因此，企业通常将海量数据集（冷数据）存放在成本更低的对象存储（BOS）中。

痛点

1. **人工介入成本高**：算法工程师在每次启动训练前，需要手动启动数据搬运任务将BOS的数据拷贝到PFS，待拷贝完成后再手动启动训练任务。
2. **资源浪费严重**：训练任务结束后，经常会忘记清理PFS上的数据集，导致昂贵的高性能存储空间被无效占用，产生不必要的成本。

方案说明

借助百舸**工作流，我们可以构建一个自动化的三步串行流水线**：

1. **数据预热节点**：挂载PFS，利用CPU节点将BOS中的冷数据自动拷贝至PFS中。
2. **模型训练节点**：依赖数据预热节点，待数据拷贝完成后，挂载数据预热节点相同PFS源路径执行模型训练任务。
3. **checkpoint存档及数据清理（并行执行）：**
    1. **checkpoint存档节点**：将训练得到的权重上传到bos存档。
    2. **数据清理节点**：依赖模型训练节点，待训练完成后，清理PFS上的数据集缓存，从而释放高性能存储空间。

YAML 配置

以下是该最佳实践的完整工作流 YAML 配置。您可以将其直接复制并根据实际环境（如队列ID、PFS实例ID、BOS路径）进行替换使用。

```
 version: v1
kind: PipelineTemplate
taskTemplates:
  - name: bos-to-pfs
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
      - name: bos_path
        type: string
        hint: 数据集的BOS路径
      - name: bos_ak
        type: string
        hint: BOS存储桶的访问ak
      - name: bos_sk
        type: string
        hint: BOS存储桶的访问sk
      - name: pfs_path
        type: string
        hint: PFS源路径
      - name: pfs_id
        type: string
        hint: PFS实例ID
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: >
        WORK_DIR="$(pwd)"

        ZIP_URL="https://doc.bce.baidu.com/bos-optimization/mac-bcecmd-0.5.10.zip"

        ZIP_FILE="mac-bcecmd-0.5.10.zip"

        EXTRACT_DIR="mac-bcecmd-0.5.10"

        echo "正在下载 bcecmd 工具..."

        curl -LO "$ZIP_URL"

        if [ $? -ne 0 ]; then
            echo "下载失败！请检查网络或 URL。"
            exit 1
        fi

        echo "正在解压..."

        unzip -o "$ZIP_FILE"

        if [ $? -ne 0 ]; then
            echo "解压失败！"
            exit 1
        fi

        cd "$EXTRACT_DIR" || { echo " 无法进入目录 $EXTRACT_DIR"; exit 1; }

        echo " 正在创建 credentials 文件..."

        mkdir -p /root/.go-bcecli

        cat > ~/.go-bcecli/credentials <<EOF

        [Defaults]

        Ak = "$BOS_AK"

        Sk = "$BOS_SK"

        EOF

        chmod 600 ~/.go-bcecli/credentials

        echo "bcecmd工具安装及配置完成,开始下载数据"

        ./bcecmd bos sync {{inputs.parameters.bos_path}} /mnt/cluster/dataset

        echo "数据集下载成功，已保存到PFS的{{inputs.parameters.pfs_path}}路径"
      jobSpec:
        image: registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04
        replicas: 1
        envs:
          - name: BOS_AK
```

```
        value: '{{inputs.parameters.bos_ak}}'
      - name: BOS_SK
        value: '{{inputs.parameters.bos_sk}}'
    datasources:
      - type: pfs
        name: '{{inputs.parameters.pfs_id}}'
        sourcePath: '{{inputs.parameters.pfs_path}}'
        mountPath: /mnt/cluster/dataset
- name: pfs-to-bos
  type: CustomTask
  inputs:
    - name: queue_id
      type: string
      hint: 队列ID
    - name: bos_path
      type: string
      hint: 数据集的BOS路径
    - name: bos_ak
      type: string
      hint: BOS存储桶的访问ak
    - name: bos_sk
      type: string
      hint: BOS存储桶的访问sk
    - name: pfs_path
      type: string
      hint: PFS源路径
    - name: pfs_id
      type: string
      hint: PFS实例ID
  spec:
    queue: '{{inputs.parameters.queue_id}}'
    jobType: PyTorchJob
    command: >
      WORK_DIR="$(pwd)"

      ZIP_URL="https://doc.bce.baidu.com/bos-optimization/mac-bcecmd-0.5.10.zip"

      ZIP_FILE="mac-bcecmd-0.5.10.zip"

      EXTRACT_DIR="mac-bcecmd-0.5.10"

      echo "正在下载 bcecmd 工具..."

      curl -LO "$ZIP_URL"

      if [ $? -ne 0 ]; then
          echo "下载失败！请检查网络或 URL。"
          exit 1
      fi

      echo "正在解压..."

      unzip -o "$ZIP_FILE"

      if [ $? -ne 0 ]; then
          echo "解压失败！"
          exit 1
      fi

      cd "$EXTRACT_DIR" || { echo " 无法进入目录 $EXTRACT_DIR"; exit 1; }

      echo " 正在创建 credentials 文件..."

      mkdir -p /root/.go-bcecli

      cat > ~/.go-bcecli/credentials <<EOF

      [Defaults]

      Ak = "$BOS_AK"

      Sk = "$BOS_SK"

      EOF

      chmod 600 ~/.go-bcecli/credentials

      echo "bcecmd工具安装及配置完成,开始上传数据"

      ./bcecmd bos sync /mnt/cluster/dataset {{inputs.parameters.bos_path}}

      echo "数据集上传成功，已保存到BOS的{{inputs.parameters.bos_path}}路径"
    jobSpec:
      image: registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04
      replicas: 1
      envs:
        - name: BOS_AK
          value: '{{inputs.parameters.bos_ak}}'
        - name: BOS_SK
          value: '{{inputs.parameters.bos_sk}}'
    datasources:
      - type: pfs
        name: '{{inputs.parameters.pfs_id}}'
        sourcePath: '{{inputs.parameters.pfs_path}}'
        mountPath: /mnt/cluster/dataset
- name: custom-task-template
  type: CustomTask
  spec:
    queue: aihcq-xxxxx
    jobType: PyTorchJob
```

```
    jobType: PyTorchJob
    command: sleep 30s
    priority: normal
    enableBccl: false
    faultTolerance: true
    faultToleranceArgs: >-
      --enable-replace=true --enable-hang-detection=true
      --hang-detection-log-timeout-minutes=7
      --hang-detection-startup-toleration-minutes=15
      --hang-detection-stack-timeout-minutes=3
      --max-num-of-unconditional-retry=2 --custom-log-patterns=timeout1
      --custom-log-patterns=timeout2
    retentionPeriod: 1d
    jobSpec:
      image: >-
        registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release
      imageConfig:
        username: your-registry-username
        password: your-registry-password
      replicas: 2
      resources:
        - name: baidu.com/a800_80g_cgpu
          quantity: 8
        - name: cpu
          quantity: 96
        - name: memory
          quantity: 512
        - name: sharedMemory
          quantity: 64
      envs:
        - name: NCCL_DEBUG
          value: INFO
        - name: NCCL_IB_DISABLE
          value: '0'
        - name: CUDA_VISIBLE_DEVICES
          value: 0,1,2,3,4,5,6,7
      enableRDMA: true
      hostNetwork: false
    labels:
      - key: project
        value: llm-training
      - key: team
        value: ai-platform
    datasources:
      - type: pfs
        name: pfs-pxE6jz
        sourcePath: /
        mountPath: /mnt/cluster
        options:
          readOnly: false
      - type: hostPath
        name: host-data
        sourcePath: /data/shared
        mountPath: /mnt/host-data
        options:
          readOnly: true
      - type: bos
        name: ''
        sourcePath: bos://my-bucket/datasets/
        mountPath: /mnt/bos-data
        options:
          readOnly: true
      - type: cfs
        name: cfs-instance-id
        sourcePath: /
        mountPath: /mnt/cfs-data
        options:
          readOnly: false
      - type: rapidfs
        name: rapidfs-instance-id
        sourcePath: /
        mountPath: /mnt/rapidfs-data
        options:
          readOnly: false
      - type: dataset
        name: dataset-id
        sourcePath: /
        mountPath: /mnt/dataset
        options:
          readOnly: true
    tensorboardConfig:
      enable: true
      logPath: /mnt/cluster/tensorboard-logs
    alertConfig:
      instanceId: your-cluster-monitor-instance-id
      alertItems:
        - jobRunning
        - jobFT
        - nodeFT
        - jobFailed
        - jobSucceed
        - jobHang
      for: 0m
      notifyRuleId: notify-xxxxxxxx
  - name: pfs-remove
    type: CustomTask
    inputs:
      - name: queue_id
        type: string
        hint: 队列ID
```

```
      - name: pfs_id
        type: string
        hint: PFS实例ID
      - name: pfs_path
        type: string
        hint: PFS源路径
    spec:
      queue: '{{inputs.parameters.queue_id}}'
      jobType: PyTorchJob
      command: |
        rm -r /mnt/cluster/dataset
        echo "数据集删除成功，已删除PFS的{{inputs.parameters.pfs_path}}路径上的数据"
      priority: normal
      jobSpec:
        image: registry.baidubce.com/inference/aibox-ubuntu-v2.0-22.04
        replicas: 1
      datasources:
        - type: pfs
          name: '{{inputs.parameters.pfs_id}}'
          sourcePath: '{{inputs.parameters.pfs_path}}'
          mountPath: /mnt/cluster/dataset
  tasks:
    - name: sync-bos-to-pfs
      taskTemplateName: bos-to-pfs
      inputs:
        - name: queue_id
          value: aihcq-h1plvpzb5gh0
        - name: bos_path
          value: bos://my-bucket/datasets/
        - name: bos_ak
          value: <你的sk>
        - name: bos_sk
          value: <你的sk>
        - name: pfs_path
          value: /datasets/my-dataset
        - name: pfs_id
          value: pfs-xxxx
    - name: traing-job
      taskTemplateName: custom-task-template
      inputs:
        - name: queue_id
          value: aihcq-h1plvpzb5gh0
        - name: bos_path
          value: bos://my-bucket/datasets/
        - name: bos_ak
          value: <你的sk>
        - name: bos_sk
          value: <你的sk>
        - name: pfs_path
          value: /datasets/my-dataset
        - name: pfs_id
          value: pfs-xxxx
      dependencies:
        - sync-bos-to-pfs
    - name: sync-checkpoint-to-bos
      taskTemplateName: pfs-to-bos
      inputs:
        - name: queue_id
          value: aihcq-h1plvpzb5gh0
        - name: bos_path
          value: bos://my-bucket/checkpoints/
        - name: bos_ak
          value: <你的sk>
        - name: bos_sk
          value: <你的sk>
        - name: pfs_path
          value: /checkpoints/my-checkpoint
        - name: pfs_id
          value: pfs-xxxx
      dependencies:
        - traing-job
    - name: pfs-file-remove
      taskTemplateName: pfs-remove
      inputs:
        - name: queue_id
          value: aihcq-h1plvpzb5gh0
        - name: pfs_path
          value: /datasets/my-dataset
        - name: pfs_id
          value: pfs-xxxx
      dependencies:
        - traing-job
```

预置工作流模板

人类演示数据视觉预处理与质量清洗

概述 用于在进入昂贵的 GPU 感知之前，对海量原始视频做质量清洗，剔除低质片段，降低下游算力浪费。

- **输入**：原始演示视频（含相机内参与畸变系数）
- **输出**：去畸变 + 场景切分后，通过清晰度/运动量/时长过滤的高质量片段清单（JSONL 格式，供 P2 消费）
- **业务价值**：GPU 感知成本高，前置 CPU 清洗可显著提升闭环整体吞吐与数据质量

处理流程

```
原始视频 → S1 去畸变 → S2 按场景切分 → S3 元数据提取 → S4 清晰度评分 → S5 运动评分 → S6 质量过滤 → 高质量片段清单
```

算子映射 |步骤|算子|说明| |-|-|-| |S1| VideoUndistort|按相机内参+畸变系数做镜头去畸变，输出校正视频（可选，no_undistort=true 跳过）| |S2|VideoSplitByScene|ContentDetector 按画面内容突变将视频切成多段 clip| |S3|VideoExtractMetadata|ffprobe 提取时长/分辨率/fps 等元信息| |S4|VideoSharpness|Laplacian 方差清晰度评分（数值越大越清晰）| |S5|VideoMotionScore|Farneback 光流运动量评分（standardized 0~1）| |S6|Filter|按清晰度/运动/时长阈值过滤，保留高质量片段|

数据流向

```
input_file (manifest.jsonl)
→ S1 → tmp_dir/s1_undistort.jsonl
→ S2 → tmp_dir/s2_split_scene.jsonl
→ S3 → tmp_dir/s3_metadata.jsonl
→ S4 → tmp_dir/s4_sharpness.jsonl
→ S5 → tmp_dir/s5_motion.jsonl
→ S6 → output_file (clips.jsonl)
```

前置条件 |资源|说明| |-|-| |AIHC 队列|需要有可用的计算队列（queue_id）| |PFS 存储|用于存放中间数据和最终输出（pfs_id）| |BOS Bucket|用于挂载算子脚本和输入数据，默认 aihc-rdw-bj/aihc-daft| |镜像|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04|

全局参数说明 |参数名|类型|必填|默认值|说明| |-|-|-|-|-| | image|string|**是**|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04|镜像地址（含 tag），不填则使用默认值| |queue_id|string|**是**|—|AIHC 队列 ID| |priority|string|否|high|任务优先级| |pfs_id|string|**是**|—|PFS 存储 ID| |pfs_source_path|string|否|/|PFS sourcePath| |pfs_mount_path|string|否|/mnt/pfs|PFS 挂载点（中间数据与最终输出存储）| |bos_source_path|string|否|aihc-rdw-bj/aihc-daft|BOS sourcePath（脚本+输入数据统一 bucket）| |bos_mount_path|string|否|/opt/aihc|BOS 挂载点| |input_file|string|**是**|/opt/aihc/data/embodied_pipelines/input_data/p1/manifest.json|输入文件绝对路径（BOS 上）| |output_file|string|**是**|—|最终输出文件路径（PFS 上）| |tmp_dir|string|**是**|—|中间数据目录（6 个 step 共用，存于 PFS）| |no_undistort|string|否|false|是否跳过去畸变（true/false）| |head_cpu|int|否|4|Head 节点 CPU 数| |head_memory|int|否|8|Head 节点内存（GB）| |shared_memory|int|否|8|sharedMemory 大小（GB）| |worker_replicas|int|否|1|Worker 副本数| |worker_cpu|int|否|6|Worker 节点 CPU 数| |worker_memory|int|否|12|Worker 节点内存（GB）| |worker_gpu|int|否|0|Worker 节点 GPU 数|

步骤详解 S1 去畸变（Undistort） 按相机内参和畸变系数对视频帧做镜头校正，消除桶形/枕形畸变，还原真实空间几何。支持跳过（设置 no_undistort=true）。

参数	类型	默认值	说明
camera_model	string	pinhole	相机模型类型（pinhole / fisheye）

- **输入列**：video，intrinsics，distortion
- **输出列**：新增 undistorted（去畸变后的视频路径）

S2 按场景切分（Split Scene） 使用 ContentDetector 检测画面内容突变点，将长视频切分为独立场景片段。切分后按行 explode，每行对应一个 clip。

参数	类型	默认值	说明
detector	string	ContentDetector	场景检测算法
threshold	string	27.0	检测灵敏度阈值（越小越敏感）
min_scene_len	string	15	最小场景帧数（低于此长度的片段将被合并）

- **输入列**：undistorted（或跳过去畸变时为 video）
- **输出列**：新增 clip（单个片段路径）

S3 元数据提取（Metadata） 通过 ffprobe 提取视频元信息，包括时长、分辨率、帧率、编码格式等。

参数	类型	默认值	说明
（无独有参数）	—	—	—

- **输入列**：clip
- **输出列**：新增 meta（含 duration，video_width，video_height，video_fps 等）

S4 清晰度评分（Sharpness） 对视频按固定帧率采样，计算每帧的 Laplacian 方差作为清晰度指标，输出统计值（mean/median/max/min/std）。

参数	类型	默认值	说明
method	string	laplacian	清晰度评估算法
fps	string	2.0	采样帧率（每秒采几帧用于评分）
max_frames	string	60	最大采样帧数

- **输入列**：clip
- **输出列**：新增 clarity（含 mean，median，max，min，std）

S5 运动评分（Motion） 基于 Farneback 光流算法计算视频运动量，输出标准化评分（0~1），用于识别静止/低运动片段。

参数	类型	默认值	说明
optical_flow_algorithm	string	farneback	光流算法
use_cuda	string	false	是否使用 CUDA 加速（true/false）

- **输入列**：clip
- **输出列**：新增 motion（含 standardized_score，motion_pattern，mean_score 等）

S6 质量过滤（Filter） 根据清晰度、运动量、时长三个维度的阈值过滤片段，仅保留高质量视频。

参数	类型	默认值	说明
clarity_min	string	20.0	清晰度最低阈值（clarity.mean 低于此值的片段被剔除）
motion_min	string	0.05	运动量最低阈值（motion.standardized_score 低于此值的片段被剔除）
min_duration	string	0.5	最短时长（秒），低于此值的片段被剔除

- **输入列**：clip，meta，clarity，motion
- **输出列**：clip，duration，clarity，motion（最终产物）

完整工作流 YAML 示例 以下为 P1 视觉清洗管线的完整工作流模板（6 步串行 DAG）：

```
version: v1
```

```yaml
kind: PipelineTemplate

inputs:
- name: image
  type: string
  hint: 镜像地址（含 tag）
  defaultValue: ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04
- name: queue_id  # [必填] 替换为实际的队列 ID
  type: string
  hint: AIHC 队列 ID
  defaultValue: <YOUR_QUEUE_ID>
- name: priority
  type: string
  hint: 任务优先级
  defaultValue: high
- name: pfs_id  # [必填] 替换为实际的 PFS 存储 ID
  type: string
  hint: PFS 存储 ID
  defaultValue: <YOUR_PFS_ID>
- name: pfs_source_path
  type: string
  defaultValue: /
- name: pfs_mount_path
  type: string
  hint: PFS 挂载点（中间数据与最终输出存储）
  defaultValue: /mnt/pfs
- name: bos_source_path
  type: string
  hint: BOS sourcePath（脚本+输入数据统一 bucket）
  defaultValue: aihc-rdw-bj/aihc-daft
- name: bos_mount_path
  type: string
  hint: BOS 挂载点
  defaultValue: /opt/aihc
- name: tmp_dir  # [必填] 替换为实际的中间数据目录
  type: string
  hint: 中间数据目录（6 个 step 共用，存于 PFS）
  defaultValue: /mnt/pfs/<YOUR_PATH>/tmp_data/p1
- name: input_file  # [必填] 输入 manifest 文件路径
  type: string
  hint: 输入文件绝对路径（BOS 上）
  defaultValue: /opt/aihc/data/embodied_pipelines/input_data/p1/manifest.json
- name: output_file  # [必填] 最终输出路径
  type: string
  hint: 最终输出文件路径（PFS 上）
  defaultValue: /mnt/pfs/<YOUR_PATH>/output_data/p1/clips.jsonl
- name: no_undistort
  type: string
  hint: 是否跳过去畸变 (true/false)
  defaultValue: "false"
- name: head_cpu
  type: int
  defaultValue: 4
- name: head_memory
  type: int
  defaultValue: 8
- name: shared_memory
  type: int
  defaultValue: 8
- name: worker_replicas
  type: int
  defaultValue: 1
- name: worker_cpu
  type: int
  defaultValue: 6
- name: worker_memory
  type: int
  defaultValue: 12
- name: worker_gpu
  type: int
  defaultValue: 0

taskTemplates:
###### --------- S1: 去畸变 ---------
- name: p1-1-undistort
  type: CustomTask
  inputs:
  - name: camera_model
    type: string
    defaultValue: pinhole
  spec:
    queue: '{{pipeline.parameters.queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      if [ "{{pipeline.parameters.no_undistort}}" = "true" ]; then
        echo "[P1 S1] skip undistort"; exit 0;
      fi;
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p1_1_undistort.py
      --input {{pipeline.parameters.input_file}}
      --output {{pipeline.parameters.tmp_dir}}/s1_undistort.jsonl
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --camera-model {{inputs.parameters.camera_model}}
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
```

```
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
      Worker:
        replicas: '{{pipeline.parameters.worker_replicas}}'
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.worker_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.worker_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
    datasources:
    - type: bos
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'

  ###### ---------- S2: 按场景切分 ----------
  - name: p1-2-split-scene
    type: CustomTask
    inputs:
    - name: detector
      type: string
      defaultValue: ContentDetector
    - name: threshold
      type: string
      defaultValue: "27.0"
    - name: min_scene_len
      type: string
      defaultValue: "15"
    spec:
      queue: '{{pipeline.parameters.queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        if [ "{{pipeline.parameters.no_undistort}}" = "true" ]; then
          INPUT_FILE="{{pipeline.parameters.input_file}}";
          VIDEO_COL="video";
        else
          INPUT_FILE="{{pipeline.parameters.tmp_dir}}/s1_undistort.jsonl";
          VIDEO_COL="undistorted";
        fi;
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p1_2_split_scene.py
        --input "$INPUT_FILE"
        --output {{pipeline.parameters.tmp_dir}}/s2_split_scene.jsonl
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --video-col "$VIDEO_COL"
        --detector {{inputs.parameters.detector}}
        --threshold {{inputs.parameters.threshold}}
        --min-scene-len {{inputs.parameters.min_scene_len}}
      # jobSpec 同上，省略...

  ###### ---------- S3: 提取元数据 ----------
  - name: p1-3-metadata
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p1_3_metadata.py
        --input {{pipeline.parameters.tmp_dir}}/s2_split_scene.jsonl
        --output {{pipeline.parameters.tmp_dir}}/s3_metadata.jsonl
        --tmp-dir {{pipeline.parameters.tmp_dir}}

  ###### ---------- S4: 清晰度评分 ----------
  - name: p1-4-sharpness
    type: CustomTask
    inputs:
    - name: method
      type: string
      defaultValue: laplacian
    - name: fps
      type: string
      defaultValue: "2.0"
    - name: max_frames
      type: string
      defaultValue: "60"
    spec:
      queue: '{{pipeline.parameters.queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p1_4_sharpness.py
        --input {{pipeline.parameters.tmp_dir}}/s3_metadata.jsonl
        --output {{pipeline.parameters.tmp_dir}}/s4_sharpness.jsonl
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --method {{inputs.parameters.method}}
        --fps {{inputs.parameters.fps}}
        --max-frames {{inputs.parameters.max_frames}}

  ###### ---------- S5: 运动评分 ----------
```

```
###### --------- S5: 运动评分 ---------
- name: p1-5-motion
  type: CustomTask
  inputs:
  - name: optical_flow_algorithm
    type: string
    defaultValue: farneback
  - name: use_cuda
    type: string
    defaultValue: "false"
  spec:
    queue: '{{pipeline.parameters.queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p1_5_motion.py
      --input {{pipeline.parameters.tmp_dir}}/s4_sharpness.jsonl
      --output {{pipeline.parameters.tmp_dir}}/s5_motion.jsonl
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --optical-flow-algorithm {{inputs.parameters.optical_flow_algorithm}}
      $([ "{{inputs.parameters.use_cuda}}" = "true" ] && echo "--use-cuda" || echo "")

###### --------- S6: 质量过滤 ---------
- name: p1-6-filter
  type: CustomTask
  inputs:
  - name: clarity_min
    type: string
    defaultValue: "20.0"
  - name: motion_min
    type: string
    defaultValue: "0.05"
  - name: min_duration
    type: string
    defaultValue: "0.5"
  spec:
    queue: '{{pipeline.parameters.queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p1_6_filter.py
      --input {{pipeline.parameters.tmp_dir}}/s5_motion.jsonl
      --output {{pipeline.parameters.output_file}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --clarity-min {{inputs.parameters.clarity_min}}
      --motion-min {{inputs.parameters.motion_min}}
      --min-duration {{inputs.parameters.min_duration}}

###### ============ 任务编排（串行 DAG） ============
tasks:
- name: step1-undistort
  taskTemplateName: p1-1-undistort
  inputs:
  - name: camera_model
    value: 'pinhole'

- name: step2-split-scene
  taskTemplateName: p1-2-split-scene
  dependencies:
  - step1-undistort
  inputs:
  - name: detector
    value: 'ContentDetector'
  - name: threshold
    value: '27.0'
  - name: min_scene_len
    value: '15'

- name: step3-metadata
  taskTemplateName: p1-3-metadata
  dependencies:
  - step2-split-scene

- name: step4-sharpness
  taskTemplateName: p1-4-sharpness
  dependencies:
  - step3-metadata
  inputs:
  - name: method
    value: 'laplacian'
  - name: fps
    value: '2.0'
  - name: max_frames
    value: '60'

- name: step5-motion
  taskTemplateName: p1-5-motion
  dependencies:
  - step4-sharpness
  inputs:
  - name: optical_flow_algorithm
    value: 'farneback'
  - name: use_cuda
    value: 'false'

- name: step6-filter
  taskTemplateName: p1-6-filter
  dependencies:
  - step5-motion
  inputs:
```

```
  - name: clarity_min
    value: '20.0'
  - name: motion_min
    value: '0.05'
  - name: min_duration
    value: '0.5'
```

> 上述 YAML 中 S2-S6 的 jobSpec 和 datasources 结构与 S1 相同，实际使用时请参考完整模板文件。 **任务拓扑**

```
 +------------+     +--------------+     +------------+     +-------------+     +-----------+     +----------+
| S1 Undistort| --> | S2 SplitScene | --> | S3 Metadata| --> | S4 Sharpness| --> | S5 Motion | --> | S6 Filter|
+------------+     +--------------+     +------------+     +-------------+     +-----------+     +----------+
```

6 个步骤严格串行执行，每步通过 dependencies 声明对前一步的依赖。

单步独立执行 每个步骤也提供独立的工作流模板，可以单独提交运行。适用于调试单个算子或从中间步骤恢复执行。

例如单独运行 S4 清晰度评分：

```
 version: v1
kind: PipelineTemplate

inputs:
- name: image
  type: string
  defaultValue: ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04

taskTemplates:
- name: p1-4-sharpness
  type: CustomTask
  inputs:
  - name: queue_id
    type: string
  - name: pfs_id
    type: string
  - name: input_file
    type: string
  - name: output_file
    type: string
  - name: tmp_dir
    type: string
  - name: method
    type: string
    defaultValue: laplacian
  - name: fps
    type: string
    defaultValue: "2.0"
  - name: max_frames
    type: string
    defaultValue: "60"
  spec:
    queue: '{{inputs.parameters.queue_id}}'
    priority: high
    jobType: RayJob
    command: >-
      DAFT_RUNNER=ray python3 /opt/aihc/workflow/p1_4_sharpness.py
      --input {{inputs.parameters.input_file}}
      --output {{inputs.parameters.output_file}}
      --tmp-dir {{inputs.parameters.tmp_dir}}
      --method {{inputs.parameters.method}}
      --fps {{inputs.parameters.fps}}
      --max-frames {{inputs.parameters.max_frames}}
    # jobSpec / datasources 配置同完整模板

tasks:
- name: p1-sharpness-task
  taskTemplateName: p1-4-sharpness
  inputs:
  - name: queue_id
    value: '<YOUR_QUEUE_ID>'          # 替换为实际队列 ID
  - name: pfs_id
    value: '<YOUR_PFS_ID>'            # 替换为实际 PFS ID
  - name: input_file
    value: '/mnt/pfs/<YOUR_PATH>/tmp_data/p1/s3_metadata.jsonl'
  - name: output_file
    value: '/mnt/pfs/<YOUR_PATH>/tmp_data/p1/s4_sharpness.jsonl'
  - name: tmp_dir
    value: '/mnt/pfs/<YOUR_PATH>/tmp_data/p1'
```

使用建议

- 首次使用建议设置 no_undistort=true 快速验证全链路是否跑通
- clarity_min、motion_min、min_duration 三个过滤阈值可根据实际数据质量灵活调整
- Worker 资源配置中 worker_gpu=0，P1 全程为 CPU 计算，无需 GPU 资源
- 如数据量大，可增加 worker_replicas 实现 Ray 分布式加速

异构多格式具身数据集统一升级合并

概述 将多种来源、不同格式的机器人数据集（RLDS、v2.1 等）统一转换为 LeRobot v3.0 格式，并可选进行跨本体对齐（帧率统一、动作空间归一化），最终合并为单一数据集。

- **输入**：含 .tar/.tar.gz 归档的目录（BOS 或 PFS）
- **输出**：统一的 LeRobot v3.0 格式数据集

- **业务价值**：一站式解决异构数据格式统一 + 跨机器人本体对齐问题，为多机器人联合训练提供标准化数据

处理流程

```
tar 归档目录 → S1 解压 → S2 格式路由 → ┬── S3 RLDS分支转换 ──┬→ S5 跨本体对齐(可选) → S6 统一合并 → v3.0数据集
                                       └── S4 v2.1分支转换 ──┘
```

算子映射 |步骤|算子|说明| |-|-|-| |S1| Extract|递归解压 tar/tar.gz 归档文件| |S2|Route|自动识别源格式（RLDS / v2.1），生成路由标记| |S3|ConvertRLDS|RLDS 分支：转为 v3.0 格式（含分片）| |S4|ConvertV21|v2.1 分支：转为 v3.0 格式| |S5|Align|跨本体对齐：统一帧率、动作归一化、robot_type 标签| |S6|Merge|将所有转换后的子数据集合并为统一输出|

数据流向

```
input_dir (tar archives)
→ S1 → tmp_dir/extracted/ + tmp_dir/s1_dirs.jsonl
→ S2 → tmp_dir/s2_routed.jsonl
→ S3 → output_dir/ (RLDS 分支) + tmp_dir/s3_rlds.jsonl   ──┐
→ S4 → output_dir/ (v2.1 分支) + tmp_dir/s4_v21.jsonl    ──┤
→ S5 → output_dir/ (对齐后) + tmp_dir/s5_aligned.jsonl    <─┘ (可选)
→ S6 → output_dir/ (最终合并)
```

前置条件 |资源|说明| |-|-| |AIHC 队列|需要有可用的计算队列（queue_id）| |PFS 存储|用于存放中间数据和最终输出（pfs_id）| |BOS Bucket|用于挂载算子脚本和输入数据，默认 aihc-rdw-bj/aihc-daft| |镜像|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04|

全局参数说明 |参数名|类型|必填|默认值|说明| |-|-|-|-|-| | image|string|**是**|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04|镜像地址（含 tag），不填则使用默认值| |queue_id|string|**是**|—|AIHC 队列 ID| |priority|string|否|high|任务优先级| |pfs_id|string|**是**|—|PFS 存储 ID| |pfs_source_path|string|否|/|PFS sourcePath| |pfs_mount_path|string|否|/mnt/pfs|PFS 挂载点| |bos_source_path|string|否|aihc-rdw-bj/aihc-daft|BOS sourcePath| |bos_mount_path|string|否|/opt/aihc|BOS 挂载点| |input_dir|string|**是**|/opt/aihc/data/embodied_pipelines/input_data/p5/archive_real|含 tar/tar.gz 的目录| |output_dir|string|**是**|—|统一 v3.0 输出根目录| |tmp_dir|string|**是**|—|中间数据目录| |num_splits|int|否|3|RLDS 转换分片数| |align|string|否|true|是否做跨本体对齐（true/false）| |target_fps|string|否|空|对齐目标帧率（空=取各集 fps 最小值）| |target_robot_type|string|否|aligned_generic|对齐后统一的 robot_type 标签| |normalize|string|否|meanstd|动作/状态归一化方式（none/meanstd/minmax）| |vcodec|string|否|h264|视频编码器（h264/auto/libsvtav1）| |encoder_threads|string|否|空|每个编码器实例的线程数（空=不限制）| |head_cpu|int|否|4|Head 节点 CPU 数| |head_memory|int|否|8|Head 节点内存（GB）| |shared_memory|int|否|8|sharedMemory 大小（GB）| |worker_replicas|int|否|1|Worker 副本数| |worker_cpu|int|否|6|Worker 节点 CPU 数| |worker_memory|int|否|12|Worker 节点内存（GB）|

步骤详解 S1 递归解压（Extract） 递归解压输入目录中的 tar/tar.gz 归档文件，展开为数据集子目录。

参数	类型	默认值	说明
（使用全局参数）	—	—	—

- **输入**：input_dir/（含 tar 归档）
- **输出**：tmp_dir/extracted/（解压后的目录）+ tmp_dir/s1_dirs.jsonl（目录清单）

S2 格式路由（Route） 自动识别每个解压目录的源格式（RLDS / v2.1 / 其他），生成路由标记供后续分支使用。

参数	类型	默认值	说明
（无独有参数）	—	—	—

- **输入**：tmp_dir/s1_dirs.jsonl
- **输出**：tmp_dir/s2_routed.jsonl（含 format 路由标记）

S3 RLDS 分支转换（Convert RLDS） 将路由标记为 RLDS 的数据集转为 LeRobot v3.0 格式。支持分片并行。

参数	类型	默认值	说明
num_splits	int	3	RLDS 分片数
vcodec	string	h264	视频编码器
encoder_threads	string	空	编码线程数

- **输入**：tmp_dir/s2_routed.jsonl（RLDS 子集）
- **输出**：output_dir/（RLDS 转换结果）+ tmp_dir/s3_rlds.jsonl

S4 v2.1 分支转换（Convert V21） 将路由标记为 v2.1 的数据集转为 LeRobot v3.0 格式。

参数	类型	默认值	说明
（无独有参数）	—	—	—

- **输入**：tmp_dir/s2_routed.jsonl（v2.1 子集）
- **输出**：output_dir/（v2.1 转换结果）+ tmp_dir/s4_v21.jsonl

> S3 和 S4 **并行执行**，互不依赖。 **S5 跨本体对齐（Align）— 可选** 当 align=true 时执行。将来自不同机器人的数据集做帧率对齐、动作空间归一化和 robot_type 统一。

参数	类型	默认值	说明
target_fps	string	空	目标帧率（空=取各集最小 fps）
target_robot_type	string	aligned_generic	统一的 robot_type 标签
normalize	string	meanstd	归一化方式（none/meanstd/minmax）
vcodec	string	h264	重编码视频编码器

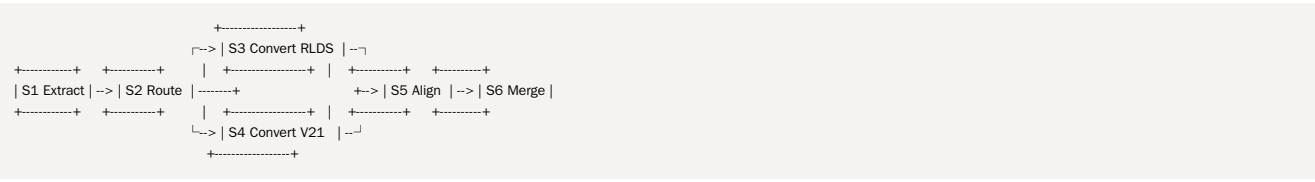
- **输入**：tmp_dir/s3_rlds.jsonl + tmp_dir/s4_v21.jsonl
- **输出**：output_dir/（对齐后数据）+ tmp_dir/s5_aligned.jsonl
- **跳过条件**：align=false 时直接跳过

S6 统一合并（Merge） 将所有转换/对齐后的子数据集合并为一个统一的 LeRobot v3.0 数据集。

参数	类型	默认值	说明
（无独有参数）	—	—	—

- **输入**：
  - `align=true`：`tmp_dir/s5_aligned.jsonl`
  - `align=false`：`tmp_dir/s3_rlds.jsonl` + `tmp_dir/s4_v21.jsonl`
- **输出**：`output_dir/`（最终合并的完整数据集）

任务拓扑



- S1 → S2：串行
- S2 → S3/S4：**并行分支**（同时执行）
- S3 + S4 → S5：汇聚（两个分支都完成后执行）
- S5 → S6：串行

使用建议

- P5 全程 CPU 计算，无需 GPU
- `align=true`（默认）适用于多机器人联合训练场景；如只是格式转换不需对齐，设置 `align=false` 可跳过 S5
- `normalize=meanstd` 最常用；`minmax` 适合已知动作空间范围的场景；`none` 跳过归一化
- `target_fps` 留空时自动取所有数据集中最小的帧率，避免上采样引入伪数据
- `num_splits` 影响 S3 的并行度和内存占用，大数据集建议增加到 4~8
- S3/S4 并行执行是 P5 独有的 DAG 结构，可有效缩短总执行时间
- 合并后的输出目录直接可被 LeRobot 训练脚本加载

RLDS、OXE 转 LeRobot

概述 将 Open X-Embodiment（OXE）标准的 RLDS 格式机器人数据集转换为 LeRobot v3.0 格式，支持分片并行转换和最终合并。

- **输入**：RLDS/OXE 格式数据集目录（如 `cmu_play_fusion`）
- **输出**：LeRobot v3.0 格式数据集（含 parquet + 可选视频）
- **业务价值**：一键迁移社区开源数据集到统一训练框架，无需手动编写转换脚本

处理流程

```
RLDS 数据集 → S1 分片转换 → S2 合并分片 → LeRobot v3.0 数据集
```

算子映射 |步骤|算子|说明| |-|-|-| |S1| `Convert`|按分片并行读取 RLDS tfrecord，转为 LeRobot v3.0 子目录| |S2|`Merge`|合并所有分片子目录为完整数据集，生成统一 metadata|

数据流向

```
input_dir (RLDS dataset)
→ S1 → tmp_dir/shards/ (分片子目录) + tmp_dir/s1_convert.jsonl
→ S2 → output_dir/ (LeRobot v3.0 dataset)
```

前置条件 |资源|说明| |-|-| |AIHC 队列|需要有可用的计算队列（queue_id）| |PFS 存储|用于存放分片中间数据和最终输出（pfs_id）| |BOS Bucket|用于挂载算子脚本和 RLDS 输入数据，默认 `aihc-rdw-bj/aihc-daft`| |镜像|`ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04`|

全局参数说明 |参数名|类型|必填|默认值|说明| |-|-|-|-|-| | `image`|string|**是**|`ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04`|镜像地址（含 tag），不填则使用默认值| |`queue_id`|string|**是**|—|AIHC 队列 ID| |`priority`|string|否|`high`|任务优先级| |`pfs_id`|string|**是**|—|PFS 存储 ID| |`pfs_source_path`|string|否|`/`|PFS sourcePath| |`pfs_mount_path`|string|否|`/mnt/pfs`|PFS 挂载点| |`bos_source_path`|string|否|`aihc-rdw-bj/aihc-daft`|BOS sourcePath| |`bos_mount_path`|string|否|`/opt/aihc`|BOS 挂载点| |`dataset`|string|**是**|`cmu_play_fusion`|OXE 数据集名| |`input_dir`|string|**是**|`/opt/aihc/data/embodied_pipelines/input_data/p4/oxe`|含 RLDS 数据集的父目录| |`tmp_dir`|string|**是**|—|中间数据目录（分片落盘 + handoff JSONL）| |`output_dir`|string|**是**|—|合并后 v3.0 数据集根目录（最终产物）| |`num_splits`|int|否|2|分片数（并行度）| |`no_videos`|string|否|`false`|是否跳过视频生成（`true`/`false`）| |`vcodec`|string|否|`h264`|视频编码器（`h264`/`auto`/`libsvtav1`）| |`encoder_threads`|string|否|空|每个编码器实例的线程数（空=不限制）| |`no_cleanup`|string|否|`false`|合并后是否保留分片子目录（`true`/`false`）| |`head_cpu`|int|否|4|Head 节点 CPU 数| |`head_memory`|int|否|20|Head 节点内存（GB）| |`shared_memory`|int|否|4|sharedMemory 大小（GB）| |`worker_replicas`|int|否|1|Worker 副本数| |`worker_cpu`|int|否|6|Worker 节点 CPU 数| |`worker_memory`|int|否|24|Worker 节点内存（GB）|

步骤详解 S1 分片转换（Convert） 将 RLDS/OXE 数据集按 `num_splits` 分片并行读取 tfrecord，转为 LeRobot v3.0 格式的子目录。每个分片独立输出一个子数据集目录。

参数	类型	默认值	说明
`dataset`	string	`cmu_play_fusion`	OXE 数据集名（用于定位数据和确定 schema）
`num_splits`	int	`2`	分片数，控制并行度和内存占用
`no_videos`	string	`false`	跳过视频编码（纯 parquet 导出）
`vcodec`	string	`h264`	视频编码器
`encoder_threads`	string	空	编码线程数

- **输入**：`input_dir/`（RLDS tfrecord 文件）
- **输出**：`tmp_dir/shards/`（分片子目录）+ `tmp_dir/s1_convert.jsonl`（分片清单）

S2 合并分片（Merge） 将 S1 输出的多个分片子目录合并为一个完整的 LeRobot v3.0 数据集。合并 parquet 索引、视频文件和 metadata.json。

参数	类型	默认值	说明
`no_cleanup`	string	`false`	合并后是否保留分片子目录（调试用）

- **输入**：`tmp_dir/s1_convert.jsonl`（分片清单）

- **输出**：output_dir/（完整 LeRobot v3.0 数据集）

任务拓扑

```
 +------------+     +----------+
| S1 Convert  | --> | S2 Merge |
|  (CPU)     |     |  (CPU)   |
+------------+     +----------+
```

2 个步骤串行执行，均为 CPU 计算，无需 GPU。

使用建议

- num_splits 应根据数据集大小和可用内存调整：
  - 小数据集（<10GB）：num_splits=1 即可
  - 大数据集（>100GB）：建议 num_splits=4~8，配合增加 worker_replicas
- P4 全程 CPU 计算，head_memory 和 worker_memory 比 P1-P3 更大（默认 20GB/24GB），因为 tfrecord 解析内存开销较高
- 设置 no_videos=true 可跳过视频编码，大幅加速转换（适合只需要动作数据的场景）
- no_cleanup=true 适用于调试：保留分片子目录方便检查中间结果
- 支持的 OXE 数据集列表取决于镜像内 oxe_dataset_configs 的注册情况

人类演示视频转LeRobot v3.0

概述 将第一人称/第三人称人类手部操作演示视频转换为 LeRobot v3.0 格式的机器人训练数据集。通过 3D 手部重建、动作解算、轨迹平滑和原子动作切分，提取出可用于模仿学习的结构化动作轨迹。

- **输入**：清洗后的高质量人类演示视频（manifest.json）
- **输出**：LeRobot v3.0 格式数据集（含动作轨迹、视频帧、元信息）
- **业务价值**：将低成本的人类手部演示视频自动转换为机器人可学习的动作数据，无需遥操作采集

处理流程

```
演示视频 → S1 场景切分 → S2 3D手部重建(GPU) → S3 动作解算 → S4 轨迹平滑 → S5 原子动作切分 → S6 导出LeRobot → v3.0数据集
```

算子映射 |步骤|算子|GPU|说明| |-|-|-|-| |S1|SplitScene|否|按场景/任务边界切分视频为独立 clip| |S2|HandRecon|**是**|使用 HaMeR/ViTPose 进行 3D 手部网格重建| |S3|HandAction|否|从 3D 手部网格解算末端位姿和夹爪状态| |S4|TrajSmooth|否|Savitzky-Golay 滤波平滑动作轨迹| |S5|AtomicSegment|否|按运动模式切分为原子动作片段| |S6|ExportLeRobot|否|导出为 LeRobot v3.0 数据集格式|

数据流向

```
input_file (manifest.json)
→ S1 → tmp_dir/s1_clips.jsonl
→ S2 → tmp_dir/s2_recon.jsonl
→ S3 → tmp_dir/s3_action.jsonl
→ S4 → tmp_dir/s4_smooth.jsonl
→ S5 → tmp_dir/s5_segments.jsonl
→ S6 → output_dir/ (LeRobot v3.0 dataset)
```

前置条件 |资源|说明| |-|-| |AIHC 队列|需要有可用的计算队列（queue_id）| |PFS 存储|用于存放中间数据和最终输出（pfs_id）| |BOS Bucket|用于挂载算子脚本、模型和输入数据，默认 aihc-rdw-bj/aihc-daft| |GPU 资源|S2 手部重建需要 GPU（默认 1 卡）| |镜像|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04| |模型|HaMeR/ViTPose 模型文件（BOS 挂载 /opt/aihc/model 下）|

全局参数说明 |参数名|类型|必填|默认值|说明| |-|-|-|-|-| | image|string|**是**|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04|镜像地址（含 tag），不填则使用默认值| |queue_id|string|**是**|—|AIHC 队列 ID| |priority|string|否|high|任务优先级| |pfs_id|string|**是**|—|PFS 存储 ID| |pfs_source_path|string|否|/|PFS sourcePath| |pfs_mount_path|string|否|/mnt/pfs|PFS 挂载点| |bos_source_path|string|否|aihc-rdw-bj/aihc-daft|BOS sourcePath（统一）| |bos_mount_path|string|否|/opt/aihc|BOS 挂载点（统一）| |input_file|string|**是**|/opt/aihc/data/embodied_pipelines/input_data/p2/manifest.json|输入文件绝对路径| |output_dir|string|**是**|—|LeRobot v3.0 数据集输出目录（最终产物）| |tmp_dir|string|**是**|—|中间数据目录（6 个 step 共用）| |task|string|否|pick and place the object|任务描述文本（写入数据集元信息）| |no_split|string|否|false|是否跳过场景切分（true/false）| |use_videos|string|否|false|导出时是否包含视频（true/false）| |vcodec|string|否|h264|视频编码器（h264/auto/libsvtav1）| |encoder_threads|string|否|空|每个编码器实例的线程数（空=不限制）| |model_dir|string|否|/opt/aihc/model|模型根目录（BOS 挂载点下的子路径）| |head_cpu|int|否|4|Head 节点 CPU 数| |head_memory|int|否|8|Head 节点内存（GB）| |shared_memory|int|否|8|sharedMemory 大小（GB）| |worker_replicas|int|否|1|Worker 副本数| |worker_cpu|int|否|6|Worker 节点 CPU 数| |worker_memory|int|否|12|Worker 节点内存（GB）| |worker_gpu|int|否|1|Worker 节点 GPU 数（仅 S2 手部重建使用）|

步骤详解 S1 场景切分（Split Scene） 将输入视频按任务边界切分为独立的 clip 片段。每个 clip 对应一次完整的操作演示。支持跳过切分（设置 no_split=true），此时整段视频作为单个 clip。

参数	类型	默认值	说明
（使用全局参数 no_split）	—	—	—

- **输入**：manifest.json（视频路径清单）
- **输出**：tmp_dir/s1_clips.jsonl（逐 clip 展开的记录）

S2 3D 手部重建（Hand Recon） GPU 使用 HaMeR + ViTPose 模型对视频逐帧进行 3D 手部网格重建，输出手部关键点和 MANO 参数。

参数	类型	默认值	说明
concurrency	int	1	GPU 推理并发数

- **输入**：tmp_dir/s1_clips.jsonl
- **输出**：tmp_dir/s2_recon.jsonl（含 3D 手部网格数据）
- **资源需求**：需要 GPU（默认 1 卡 baidu.com/rtx_rpbzzz6_96g_cgpu）

S3 动作解算（Hand Action） 从 3D 手部网格数据中解算末端执行器的位姿（位置+旋转）和夹爪开合状态，生成结构化动作序列。

参数	类型	默认值	说明
coordinate_frame	string	camera	坐标系（camera / world）
rotation_format	string	axis_angle	旋转表示格式（axis_angle / quaternion / euler）

- **输入**：`tmp_dir/s2_recon.jsonl`
- **输出**：`tmp_dir/s3_action.jsonl`（含位姿+夹爪动作序列）

S4 轨迹平滑（Trajectory Smooth） 对解算得到的动作轨迹进行 Savitzky-Golay 滤波平滑，去除抖动噪声。

参数	类型	默认值	说明
`method`	string	`savgol`	平滑算法
`window`	string	5	滑动窗口大小（奇数）
`polyorder`	string	2	多项式阶数

- **输入**：`tmp_dir/s3_action.jsonl`
- **输出**：`tmp_dir/s4_smooth.jsonl`（平滑后的轨迹）

S5 原子动作切分（Atomic Segment） 根据运动模式变化点将连续轨迹切分为原子动作片段（如"伸手"、"抓取"、"移动"、"放置"）。

参数	类型	默认值	说明
`min_segment_len`	string	3	最小片段帧数（低于此长度的片段将被合并）

- **输入**：`tmp_dir/s4_smooth.jsonl`
- **输出**：`tmp_dir/s5_segments.jsonl`（含原子动作标注）

S6 导出 LeRobot（Export LeRobot） 将处理后的数据导出为 LeRobot v3.0 格式数据集，包含动作轨迹（parquet）、视频帧（可选）和元信息。

参数	类型	默认值	说明
`repo_id`	string	`human_demo_v3`	数据集 repo_id
`fps`	string	30	目标帧率

- **输入**：`tmp_dir/s5_segments.jsonl`
- **输出**：`output_dir/`（LeRobot v3.0 数据集目录）

任务拓扑

```
  +--------------+     +---------------+     +--------------+     +--------------+     +----------------+     +----------------+
| S1 SplitScene | --> | S2 HandRecon   | --> | S3 HandAction | --> | S4 TrajSmooth | --> | S5 AtomicSegment| --> | S6 ExportLeRobot|
|    (CPU)      |     |    (GPU)       |     |    (CPU)      |     |    (CPU)      |     |    (CPU)       |     |    (CPU)       |
 +--------------+     +---------------+     +--------------+     +--------------+     +----------------+     +----------------+
```

6 个步骤严格串行执行，每步通过 dependencies 声明对前一步的依赖。仅 S2 需要 GPU 资源。

单步独立执行 每个步骤也提供独立的工作流模板，可以单独提交运行。适用于调试单个算子或从中间步骤恢复执行。

例如单独运行 S2 3D 手部重建：

```
  version: v1
kind: PipelineTemplate

inputs:
- name: image
  type: string
  defaultValue: ccr-registry.baidubce.com/aihc/aihc-daft-gpu-0.5.2.1-cu121-py3.11-ubuntu22.04

taskTemplates:
- name: p2-2-hand-recon
  type: CustomTask
  inputs:
  - name: queue_id
    type: string
  - name: pfs_id
    type: string
  - name: input_file
    type: string
  - name: output_file
    type: string
  - name: tmp_dir
    type: string
  - name: model_dir
    type: string
    defaultValue: /opt/aihc/model
  - name: concurrency
    type: int
    defaultValue: 1
  spec:
    queue: '{{inputs.parameters.queue_id}}'
    priority: high
    jobType: RayJob
    command: >-
      DAFT_RUNNER=ray python3 /opt/aihc/workflow/p2_2_hand_recon.py
      --input {{inputs.parameters.input_file}}
      --output {{inputs.parameters.output_file}}
      --tmp-dir {{inputs.parameters.tmp_dir}}
      --model-dir {{inputs.parameters.model_dir}}
      --concurrency {{inputs.parameters.concurrency}}
    # jobSpec / datasources 配置同完整模板（Worker 需配置 GPU）

tasks:
- name: p2-hand-recon-task
  taskTemplateName: p2-2-hand-recon
  inputs:
  - name: queue_id
    value: '<YOUR_QUEUE_ID>'
  - name: pfs_id
    value: '<YOUR_PFS_ID>'
  - name: input_file
    value: '/mnt/pfs/<YOUR_PATH>/tmp_data/p2/s1_clips.jsonl'
  - name: output_file
    value: '/mnt/pfs/<YOUR_PATH>/tmp_data/p2/s2_recon.jsonl'
  - name: tmp_dir
    value: '/mnt/pfs/<YOUR_PATH>/tmp_data/p2'
```

使用建议

- S2 手部重建是计算瓶颈，建议根据数据量调整 `worker_replicas` 和 `concurrency`
- 首次使用可设置 `no_split=true` 快速验证全链路
- `use_videos=true` 会在导出时生成视频文件（增加存储开销），默认仅导出 parquet 格式动作数据
- `vcodec` 推荐使用 `h264` 兼容性最好；`libsvtav1` 压缩率更高但编码较慢
- 如需自定义坐标系或旋转格式，可通过 S3 的 `coordinate_frame` 和 `rotation_format` 参数调整

多模态感知驱动的 Caption 生成、自动起止打标与意图识别

概述 用于对操作视频进行多模态感知标注，包括目标检测、相机标定、人体姿态估计、3D 框提升和自然语言 Caption 生成。为下游训练数据集提供丰富的空间理解和语义描述。

- **输入**：清洗后的高质量视频片段（manifest.json）
- **输出**：包含 YOLO 检测框、相机参数、人体姿态、3D BBox 和自然语言 Caption 的标注文件（JSONL）
- **业务价值**：自动化的多模态标注替代人工标注，为视觉-语言-动作模型提供对齐的训练数据

处理流程

```
视频片段 → S1 场景切分 → S2 首帧提取 → S3 YOLO检测(GPU) → S4 相机标定(GPU) → S5 姿态估计(GPU) → S6 3D框提升 → S7 Caption生成(GPU) → S8 输出
```

算子映射 |步骤|算子|GPU|说明| |-|-|-|-| |S1|`SplitScene`|否|按场景切分视频为独立 clip| |S2|`FirstFrame`|否|提取每个 clip 的首帧用于感知分析| |S3|`YOLO`|**是**|YOLOv11x 目标检测，输出 2D BBox| |S4|`CameraCalib`|**是**|MoGe 单目深度估计 + 相机内参标定| |S5|`Pose`|**是**|DWPose 全身姿态估计（关键点）| |S6|`LiftBBox`|否|结合深度和标定将 2D 框提升为 3D BBox| |S7|`Caption`|**是**|QwenVL 视频理解生成操作意图描述| |S8|`Writeout`|否|字段规整，合并输出最终标注文件|

数据流向

```
input_file (manifest.json)
  → S1 → tmp_dir/s1_split_scene.jsonl
  → S2 → tmp_dir/s2_first_frame.jsonl
  → S3 → tmp_dir/s3_yolo.jsonl
  → S4 → tmp_dir/s4_calib.jsonl
  → S5 → tmp_dir/s5_pose.jsonl
  → S6 → tmp_dir/s6_lift_bbox.jsonl
  → S7 → tmp_dir/s7_caption.jsonl
  → S8 → output_file (labeled.jsonl)
```

前置条件 |资源|说明| |-|-| |AIHC 队列|需要有可用的计算队列（queue_id）| |PFS 存储|用于存放中间数据和最终输出（pfs_id）| |BOS Bucket|用于挂载算子脚本、模型和输入数据，默认 aihc-rdw-bj/aihc-daft| |GPU 资源|S3/S4/S5/S7 需要 GPU（默认各 1 卡）| |镜像|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04| |模型|YOLO11x、MoGe-2-ViTL、DWPose、QwenVL（BOS 挂载 /opt/aihc/model 下）|

全局参数说明 |参数名|类型|必填|默认值|说明| |-|-|-|-|-| | image|string|**是**|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.1-cu121-py3.11-ubuntu22.04|镜像地址（含 tag），不填则使用默认值| |queue_id|string|**是**|—|AIHC 队列 ID| |priority|string|否|high|任务优先级| |pfs_id|string|**是**|—|PFS 存储 ID| |pfs_source_path|string|否|/|PFS sourcePath| |pfs_mount_path|string|否|/mnt/pfs|PFS 挂载点| |bos_source_path|string|否|aihc-rdw-bj/aihc-daft|BOS sourcePath| |bos_mount_path|string|否|/opt/aihc|BOS 挂载点| |model_dir|string|否|/opt/aihc/model|模型根目录| |input_file|string|**是**|/opt/aihc/data/embodied_pipelines/input_data/p3/manifest.json|输入文件绝对路径| |output_file|string|**是**|—|最终输出文件路径（PFS 上）| |tmp_dir|string|**是**|—|中间数据目录（8 个 step 共用）| |no_split|string|否|false|是否跳过场景切分（true/false）| |gpu_concurrency|int|否|1|GPU 算子并发数| |perception_gpu_frac|string|否|0.3|感知算子 GPU 显存占比| |head_cpu|int|否|4|Head 节点 CPU 数| |head_memory|int|否|8|Head 节点内存（GB）| |shared_memory|int|否|8|sharedMemory 大小（GB）| |worker_replicas|int|否|1|Worker 副本数| |worker_cpu|int|否|6|Worker 节点 CPU 数| |worker_memory|int|否|12|Worker 节点内存（GB）| |worker_gpu|int|否|1|Worker 节点 GPU 数（GPU 步骤使用）| |caption_prompt|string|否|请用一句话描述视频中人手正在执行的操作意图。|Caption 提示词| |qwenvl_model_name|string|否|Qwen2.5-VL-7B-Instruct|QwenVL 模型名称| |qwenvl_tensor_parallel|int|否|1|张量并行 GPU 数（7B=1, 32B=2, 72B=4）| |qwenvl_dtype|string|否|auto|推理精度（auto/bfloat16/float16/float32）| |qwenvl_quantization|string|否|空|权重量化方式（awq/awq_marlin/空=自动探测）| |qwenvl_gpu_mem|string|否|0.8|QwenVL GPU 显存占比| |qwenvl_max_model_len|int|否|8192|QwenVL 最大模型长度| |qwenvl_enforce_eager|string|否|false|是否强制 eager 模式| |qwenvl_max_pixels|int|否|602112|QwenVL 最大像素数| |qwenvl_fps|string|否|2.0|QwenVL 采样帧率|

步骤详解 S1 场景切分（Split Scene） 将输入视频按场景边界切分为独立 clip。支持跳过（no_split=true）。

参数	类型	默认值	说明
no_split	string	false	是否跳过切分

- **输入**：manifest.json
- **输出**：tmp_dir/s1_split_scene.jsonl

S2 首帧提取（First Frame） 提取每个 clip 的首帧图像，供后续感知算子使用（YOLO、相机标定等基于静态帧分析）。

参数	类型	默认值	说明
output_format	string	jpg	输出图片格式

- **输入**：tmp_dir/s1_split_scene.jsonl
- **输出**：tmp_dir/s2_first_frame.jsonl（含首帧图片路径）

S3 YOLO 目标检测（YOLO） GPU 使用 YOLOv11x 对首帧进行目标检测，输出 2D 检测框和类别信息。

参数	类型	默认值	说明
yolo_model	string	/opt/aihc/model/yolo11x.pt	YOLO 模型路径

- **输入**：tmp_dir/s2_first_frame.jsonl
- **输出**：tmp_dir/s3_yolo.jsonl（含 2D BBox + 类别）
- **资源需求**：GPU（baidu.com/rtx_rpbzzz6_96g_cgpu）

S5 全身姿态估计（Pose） GPU 使用 DWPose 进行全身姿态估计，输出人体/手部关键点坐标。

参数	类型	默认值	说明
dwpose_dir	string	/opt/aihc/model/dwpose	DWPose 模型目录

- **输入**：tmp_dir/s4_calib.jsonl
- **输出**：tmp_dir/s5_pose.jsonl（含人体关键点坐标）
- **资源需求**：GPU（baidu.com/rtx_rpbzzz6_96g_cgpu）

S6 2D→3D 框提升（Lift BBox） 结合 S4 的深度图和相机内参，将 S3 的 2D 检测框提升为 3D BBox。纯 CPU 计算。

参数	类型	默认值	说明
（无独有参数）	—	—	—

- **输入**：tmp_dir/s5_pose.jsonl
- **输出**：tmp_dir/s6_lift_bbox.jsonl（含 3D BBox）

S7 Caption 生成（QwenVL Caption） GPU 使用 QwenVL 视频理解模型对操作视频生成自然语言描述（操作意图）。支持多种模型规模和量化配置。

参数	类型	默认值	说明
caption_prompt	string	请用一句话描述视频中人手正在执行的操作意图。	提示词
qwenvl_model_name	string	Qwen2.5-VL-7B-Instruct	模型名称
qwenvl_tensor_parallel	int	1	张量并行数（按模型规模设定）
qwenvl_gpu_mem	string	0.8	显存占比

- **输入**：tmp_dir/s6_lift_bbox.jsonl
- **输出**：tmp_dir/s7_caption.jsonl（含自然语言 caption）
- **资源需求**：GPU（baidu.com/rtx_rpbzzz6_96g_cgpu）

S8 字段规整输出（Writeout） 将所有中间步骤产出的标注信息（检测框、标定、姿态、3D 框、Caption）合并规整，输出最终标注文件。

参数	类型	默认值	说明
（无独有参数）	—	—	—

- **输入**：tmp_dir/s7_caption.jsonl
- **输出**：output_file（最终标注产物）

任务拓扑

```
 +-----------+    +-----------+    +--------+    +-----------+    +--------+    +----------+    +----------+    +----------+
| S1 Split  | --> | S2 1stFrame| --> | S3 YOLO| --> | S4 Calib  | --> | S5 Pose| --> | S6 Lift  | --> |S7 Caption| --> | S8 Write |
|  (CPU)  |    |  (CPU)  |    | (GPU) |    |  (GPU)  |    | (GPU) |    |  (CPU)  |    |  (GPU)  |    |  (CPU)  |
+-----------+    +-----------+    +--------+    +-----------+    +--------+    +----------+    +----------+    +----------+
```

8 个步骤严格串行执行。S3、S4、S5、S7 需要 GPU 资源，其余为 CPU 计算。

使用建议

- 首次使用建议设置 `no_split=true` 跳过切分快速验证
- GPU 步骤是瓶颈，可通过增加 `worker_replicas` 和 `gpu_concurrency` 提升吞吐
- `perception_gpu_frac=0.3` 意味着感知算子（S3-S5）各占约 30% 显存，适合小模型共存场景
- QwenVL 模型选择建议：
  - 7B（1 卡）：速度快，适合大批量标注
  - 32B（2 卡）：质量与速度平衡
  - 72B（4 卡）：最高质量，适合小批量精标
- `qwenvl_quantization` 设为 awq 可减少显存占用但略降精度
- 如显存不足，可降低 `qwenvl_max_pixels` 或开启 `qwenvl_enforce_eager=true`

基于 Lance 宽表的视频全链路精选与多格式生成管线

概述 此管线基于 Lance 列式表为核心数据面，对海量原始视频做全链路质量精选：从场景切分、语义去重、美学评分、NSFW 过滤到结构化 Caption，最终输出训练可用的 WebDataset 分片和 LeRobot v3.0 数据集。

- **输入**：原始视频/图像清单（manifest.jsonl，含 sample_id / modality / url）
- **输出**：
  - WebDataset tar 分片（供视觉-语言训练）
  - LeRobot v3.0 数据集（供具身策略训练）
- **业务价值**：将散乱原始素材一站式精选为高质量训练数据，前置 CPU/GPU 联合清洗显著降低下游训练噪声

处理流程

```
 manifest.jsonl → S00 写入Lance → S01 校验入库 → S02 场景切分 → S03 视频嵌入(GPU)
→ S04 KMeans聚类 → S05 语义去重 → S06 美学评分(GPU) → S07 NSFW检测(GPU)
→ S08 质量过滤 → S09 结构化Caption(GPU) → S10 WebDataset分片 / S11 导出LeRobot v3.0
```

算子映射 |**步骤**|**算子**|**GPU**|**说明**| |-|-|-|-| |S00|`IngestToLance`|否|从 manifest.jsonl 写入 Lance 宽表| |S01|`Ingest`|否|字段校验 + 默认值填充| |S02|`VideoSplitByScene`|否|ContentDetector 场景切分，1→N 行 explode| |S03|`CosmosVideoEmbedding`|**是**|Cosmos-Embed1-448p 视频嵌入| |S04|`KMeansClustering`|否|嵌入向量 KMeans 聚类| |S05|`SemanticDedupByCosine`|否|余弦相似度语义去重| |S06|`DOVERAesthetic`|**是**|DOVER 视频美学评分| |S07|`VideoNsfwDetect`|**是**|Falconsai NSFW 检测| |S08|`Filter`|否|按去重/美学/NSFW 三维过滤| |S09|`StructuredCaptionPipeline`|**是**|Qwen2.5-VL 结构化 Caption| |S10|`ExportToWebDataset`|否|输出 WebDataset tar 分片| |S11|`ExportLeRobotV30`|否|输出 LeRobot v3.0 数据集|

数据流向

```
manifest.jsonl
→ S00 → lance://bos://<YOUR-BUCKET-NAME>/aihc/p6-run1/episodes.lance
→ S01 → (Lance 表内 keep=True 校验)
→ S02 → (Lance 表追加 clip 列，行 explode)
→ S03 → (Lance 表追加 video_embed 列)
→ S04 → (Lance 表追加 cluster_id 列)
→ S05 → (Lance 表追加 dedup_keep 列)
→ S06 → (Lance 表追加 dover_aesthetic 列)
→ S07 → (Lance 表追加 video_nsfw 列)
→ S08 → (Lance 表更新 keep + drop_reasons)
→ S09 → (Lance 表追加 caption 列)
→ S10 → output_dir/*.tar (WebDataset 分片)
→ S11 → output_dir/ (LeRobot v3.0 数据集)
```

前置条件 |**资源**|**说明**| |-|-| |AIHC 队列|需要 CPU 队列和 GPU 队列（分别用于 CPU/GPU 步骤）| |BOS Bucket|存放 Lance 表、输入数据和模型（默认 `storage-aihc`,需提前创建）| |GPU 资源|S03/S06/S07 需要 RTX 4090；S09 需要 2×RTX 4090| |镜像|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.12.2-cu129-py3.12-p6-sm120-sm89-ubuntu22.04| |模型|Cosmos-Embed1-448p、DOVER、Falconsai/nsfw_image_detection、Qwen2.5-VL-7B-Instruct（镜像内 /opt/aihc/models）|

全局参数说明 |**参数名**|**类型**|**必填**|**默认值**|**说明**| |-|-|-|-|-| |`image`|string|**是**|ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.12.2-cu129-py3.12-p6-sm120-sm89-ubuntu22.04|镜像地址| |`cpu_queue_id`|string|**是**|—|CPU 计算队列 ID| |`gpu_queue_id`|string|**是**|—|GPU 计算队列 ID| |`priority`|string|否|high|任务优先级| |`lance_uri`|string|**是**|lance://bos://storage-aihc/aihc/p6-run1/episodes.lance|Lance 表 URI，需要替换为自己的 BOS 地址| |`manifest`|string|**是**|—|输入 manifest.jsonl 路径（BOS 上）| |`output_dir`|string|**是**|—|最终输出目录（WebDataset 分片 + LeRobot 数据集）| |`bos_env_file`|string|**是**|-|BOS 凭证文件路径（需要包含 ACCESS_KEY/SECRET_KEY，要求可以读写`lance_uri`参数中的填写的BOS地址），lance 数据的读写基于 bos endpoint，填写示例见下方| |`source`|string|否|`manifest`|数据来源标签| |`source_dataset`|string|否|manifest 父目录名|数据集标签| |`head_cpu`|int|否|6|Head 节点 CPU 数| |`head_memory`|int|否|12|Head 节点内存（GB）| |`worker_replicas`|int|否|2|Worker 副本数| |`worker_cpu`|int|否|16|Worker 节点 CPU 数| |`worker_memory`|int|否|64|Worker 节点内存（GB）| |`worker_gpu`|int|否|1|Worker 节点 GPU 数（GPU 步骤使用）|

bos_env_file文件内容要求：

```
 export BOS_ENDPOINT=https://bj.bcebos.com
export BOS_S3_ENDPOINT=https://s3.bj.bcebos.com
export BOS_REGION=bj
export BOS_ACCESS_KEY=xxxxxxx
export BOS_SECRET_KEY=xxxxxxx
```

步骤详解 S00 写入 Lance（Ingest to Lance） 从 manifest.jsonl 读取记录，写入 Lance 宽表作为后续所有步骤的统一数据面。支持增量追加（按 sample_id 去重），`--force` 可全量覆盖。

参数	类型	默认值	说明
manifest	string	—	输入 manifest.jsonl 路径
output	string	—	Lance URI
source	string	manifest	来源标签
source_dataset	string	父目录名	数据集标签
force	bool	false	是否强制覆盖

- **输入**：manifest.jsonl（字段：sample_id, modality, url）
- **输出**：Lance 表（schema：sample_id, modality, url, source, source_dataset, ingest_time, keep, meta）
- **资源**：CPU-only，纯 IO 操作

S01 字段校验（Ingest） 校验 manifest.jsonl 字段合法性，跳过缺失 sample_id / 非法 modality / 无 url 的行，为合法行填充 keep=True 默认值。

参数	类型	默认值	说明
allow_empty	bool	false	允许空输入不报错

- **输入**：manifest.jsonl
- **输出**：校验后的 JSONL
- **资源**：CPU-only，纯 IO 操作

S02 场景切分（Split Scene） 使用 PySceneDetect ContentDetector 检测画面内容突变点，将长视频切分为独立场景片段。切分后行 explode，每行对应一个 clip。仅处理 modality=video 的行。

参数	类型	默认值	说明
detector	string	ContentDetector	场景检测算法
threshold	float	27.0	检测灵敏度阈值（越小越敏感）
min_scene_len	int	15	最小场景帧数

- **输入列**：url（视频路径）
- **输出列**：新增 clip（切分后的片段路径）
- **资源**：CPU-only（4 CPU / 0 GPU）

S03 视频嵌入（Embed Video）-GPU 使用 NVIDIA Cosmos-Embed1-448p 模型生成视频语义嵌入向量，供后续聚类和去重使用。

参数	类型	默认值	说明
model_name	string	nvidia/Cosmos-Embed1-448p	嵌入模型标识
model_path	string	/opt/aihc/models	模型根目录
model_batch_size	int	4	GPU forward batch size
num_frames	int	8	每段视频采样帧数

- **输入列**：视频路径（优先级：compressed > cropped > clip > url）
- **输出列**：新增 video_embed（嵌入向量）
- **资源**：1×RTX 4090，2 CPU，16 CPU / 64GB 内存

S04 KMeans 聚类（KMeans） 对嵌入向量做 KMeans 聚类，分配 cluster_id，为语义去重提供分组依据。

参数	类型	默认值	说明
n_clusters	int	1024	聚类数
backend	string	auto	聚类后端（auto/cuml/faiss/sklearn）
max_iter	int	20	最大迭代数

- **输入列**：video_embed 或 image_embed
- **输出列**：新增 cluster_id（-1 表示无嵌入）
- **资源**：CPU-only，全局单次操作（concurrency=1）

S05 语义去重（Dedup） 在每个 cluster 内按余弦相似度去重，标记重复样本。

参数	类型	默认值	说明
threshold	float	0.95	余弦相似度去重阈值

- **输入列**：embed, cluster_id
- **输出列**：新增 dedup_keep（bool，True=保留）
- **资源**：CPU-only，全局单次操作

S06 美学评分（Dover Aesthetic）- GPU 使用 DOVER 模型对视频进行美学质量评分。

参数	类型	默认值	说明
backend	string	dover	评分后端（dover/laion）
model_path	string	/opt/aihc/model/dover	DOVER 权重目录
num_frames	int	32	采样帧数
num_clips	int	1	采样 clip 数

- **输入列**：视频路径
- **输出列**：新增 dover_aesthetic（float 评分）

- **资源**：0.5×RTX 4090，2 CPU；Workers 3×(16 CPU / 64GB)

S07 NSFW 检测（NSFW Video）- GPU 使用 Falconsai/nsfw_image_detection 模型检测视频 NSFW 概率。

参数	类型	默认值	说明
model_name	string	Falconsai/nsfw_image_detection	模型标识
model_path	string	/opt/aihc/models	模型根目录
num_frames	int	8	均匀采样帧数

- **输入列**：视频路径
- **输出列**：新增 video_nsfw（概率 float）
- **资源**：0.25×RTX 4090，2 CPU；Workers 3×(16 CPU / 64GB)

S08 质量过滤（Filter） 根据去重、美学评分、NSFW 检测三个维度综合过滤，更新 keep 标记。

参数	类型	默认值	说明
dover_aesthetic_min	float	3.0	美学评分最低阈值
nsfw_threshold	float	0.5	NSFW 概率上限

- **输入列**：dedup_keep, dover_aesthetic, video_nsfw
- **输出列**：更新 keep（bool）+ 新增 drop_reasons（列表）
- **资源**：CPU-only，纯 pandas/pyarrow 操作
- **过滤规则**：
  - dedup_keep=False → drop_reason: "dedup"
  - dover_aesthetic < dover_aesthetic_min → drop_reason: "dover_aes_low"
  - video_nsfw > nsfw_threshold → drop_reason: "video_nsfw"

S09 结构化 Caption（Caption）- GPU 使用 Qwen2.5-VL-7B-Instruct 通过 vLLM 对保留视频生成结构化文本描述。支持视频和图像双分支。

参数	类型	默认值	说明
model_name	string	Qwen/Qwen2.5-VL-7B-Instruct	模型标识
model_path	string	/opt/aihc/models	模型根目录
backend	string	auto	推理后端（auto/vllm/transformers）
tensor_parallel_size	int	-1（自动）	张量并行 GPU 数
gpu_memory_utilization	float	0.85	vLLM 显存占比
max_model_len	int	-1（自动）	最大序列长度
enforce_eager	bool	false	禁用 CUDA graphs
lora_path	string	空	LoRA 适配器路径（可选）

- **输入列**：keep=True 的视频/图像路径
- **输出列**：新增 caption（结构化文本）
- **资源**：2×RTX 4090，16 CPU / 128GB 内存

S10 WebDataset 分片（Shard） 将 keep=True 的样本导出为 WebDataset tar 分片。

参数	类型	默认值	说明
output_dir	string	—	分片输出目录
shard_prefix	string	shard	tar 文件前缀
max_shard_size_mb	int	512	单个分片最大 MB
max_samples_per_shard	int	10000	单个分片最大样本数

- **输入**：Lance 表中 keep=True 的行
- **输出**：output_dir/shard-000000.tar 等
- **资源**：CPU-only，全局单次有序操作

S11 导出 LeRobot v3.0（Export LeRobot） 将 keep=True 的具身数据（含 action/state）导出为 LeRobot v3.0 格式数据集。

参数	类型	默认值	说明
out	string	—	LeRobot v3.0 输出目录
repo_id	string	p6_curated_v3	数据集 repo ID
fps	int	30	视频帧率
robot_type	string	generic_hand	机器人类型标签
action_dim	int	7	动作维度
use_videos	bool	false	是否包含视频列
vcodec	string	h264	视频编码器

- **输入**：Lance 表中 keep=True 且含 action/state 的行
- **输出**：LeRobot v3.0 数据集目录（data/*.parquet）
- **资源**：CPU-only，Workers 2×(16 CPU / 64GB)

任务拓扑 [流程图 7342279715ae42a68dfbc53857a9b751 mxGraph]

- S00 → S09：严格串行，每步依赖前一步

- S09 → S10 / S11：**并行执行**（两个输出分支互不依赖）
- GPU 步骤：S03、S06、S07、S09

完整工作流YAML

```yaml
version: v1
kind: PipelineTemplate

inputs:
- name: image
  type: string
  hint: 镜像地址
  defaultValue: ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.2.12.2-cu129-py3.12-p6-sm120-sm89-ubuntu22.04
- name: cpu_queue_id
  type: string
  hint: CPU 队列 ID
  defaultValue: <YOUR_QUEUE_ID>
- name: gpu_queue_id
  type: string
  hint: GPU 队列 ID
  defaultValue: <YOUR_QUEUE_ID>
- name: priority
  type: string
  defaultValue: high
- name: pfs_id
  type: string
  hint: PFS 实例 ID
  defaultValue: <YOUR_PFS_ID>
- name: pfs_source_path
  type: string
  defaultValue: /
- name: pfs_mount_path
  type: string
  defaultValue: /mnt/pfs
- name: bos_source_path
  type: string
  hint: BOS sourcePath（代码/模型）
  defaultValue: aihc-rdw-bj/aihc-daft
- name: bos_mount_path
  type: string
  hint: BOS 挂载点（代码/模型）
  defaultValue: /opt/aihc
- name: input
  type: string
  hint: 源视频 manifest.jsonl 路径
  defaultValue: /mnt/pfs/<YOUR_PATH>/input_data/p6/manifest.jsonl
- name: lance_uri
  type: string
  hint: Lance 宽表 URI（s01-s10 共用）
  defaultValue: bos://storage-aihc/aihc/<YOUR_PATH>/episodes.lance
- name: tmp_dir
  type: string
  hint: 中间数据目录
  defaultValue: /mnt/pfs/<YOUR_PATH>/tmp_data/p6
- name: output_dir
  type: string
  hint: 最终产物输出目录
  defaultValue: /mnt/pfs/<YOUR_PATH>/output_data/p6
###### BOS 凭证
- name: bos_env_file
  type: string
  hint: BOS 凭证文件路径（含 BOS_ACCESS_KEY/BOS_SECRET_KEY）
  defaultValue: /mnt/pfs/<YOUR_PATH>/env/bos_env
###### S00 IngestToLance
- name: s00_source
  type: string
  defaultValue: manifest
- name: s00_force
  type: string
  hint: 传 "--force" 则强制覆盖 Lance 表，留空则增量 append
  defaultValue: "--force"
###### S02 SplitScene
- name: threshold
  type: string
  defaultValue: "27.0"
- name: min_scene_len
  type: string
  defaultValue: "15"
- name: s02_num_cpus
  type: string
  defaultValue: "4.0"
- name: s02_worker_replicas
  type: int
  defaultValue: 4
- name: s02_worker_cpu
  type: int
  defaultValue: 16
- name: s02_worker_memory
  type: int
  defaultValue: 64
###### S03 EmbedVideo
- name: embed_model_name
  type: string
  defaultValue: /opt/aihc/model/nvidia/Cosmos-Embed1-448p
- name: s03_num_gpus
  type: string
  defaultValue: "1.0"
- name: s03_worker_replicas
```

```
    type: int
    defaultValue: 2
  - name: s03_worker_gpu
    type: int
    defaultValue: 1
  ###### S04 KMeans
  - name: n_clusters
    type: string
    defaultValue: "1024"
  ###### S06 DOVER
  - name: dover_model_name
    type: string
    defaultValue: /opt/aihc/model/dover
  - name: dover_backend
    type: string
    defaultValue: dover
  - name: dover_laion_mlp_path
    type: string
    defaultValue: /opt/aihc/model/laion_aesthetic_v2
  - name: dover_clip_model
    type: string
    defaultValue: /opt/aihc/model/openai/clip-vit-large-patch14
  - name: dover_num_frames
    type: string
    defaultValue: "32"
  - name: dover_num_clips
    type: string
    defaultValue: "1"
  ###### S07 NSFW
  - name: nsfw_model_name
    type: string
    defaultValue: /opt/aihc/model/Falconsai/nsfw_image_detection
  ###### S08 Filter
  - name: dover_aesthetic_min
    type: string
    defaultValue: "3.0"
  - name: nsfw_threshold
    type: string
    defaultValue: "0.5"
  ###### S09 Caption
  - name: caption_model_name
    type: string
    defaultValue: /opt/aihc/model/Qwen2.5-VL-7B-Instruct
  - name: s09_num_gpus
    type: string
    defaultValue: "2.0"
  - name: s09_backend
    type: string
    defaultValue: auto
  - name: s09_tensor_parallel_size
    type: string
    defaultValue: "-1"
  - name: s09_gpu_memory_utilization
    type: string
    defaultValue: "0.85"
  - name: s09_max_model_len
    type: string
    defaultValue: "-1"
  - name: s09_worker_replicas
    type: int
    defaultValue: 2
  - name: s09_worker_cpu
    type: int
    defaultValue: 16
  - name: s09_worker_memory
    type: int
    defaultValue: 128
  - name: s09_worker_gpu
    type: int
    defaultValue: 2
  ###### S10 Shard
  - name: max_shard_size_mb
    type: string
    defaultValue: "512.0"
  - name: shard_prefix
    type: string
    defaultValue: shard
  - name: max_samples_per_shard
    type: string
    defaultValue: "10000"
  ###### S11 Export LeRobot
  - name: repo_id
    type: string
    defaultValue: p6_curated_v3
  - name: s11_fps
    type: string
    defaultValue: "30"
  - name: s11_robot_type
    type: string
    defaultValue: generic_hand
  - name: s11_action_dim
    type: string
    defaultValue: "7"
  ###### 通用资源
  - name: head_cpu
    type: int
    defaultValue: 6
  - name: head_memory
    type: int
    defaultValue: 12
```

```
- name: shared_memory
  type: int
  defaultValue: 30

taskTemplates:
###### ========== S00: Ingest to Lance ==========
- name: p6-00-ingest-to-lance
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.cpu_queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      source {{pipeline.parameters.bos_env_file}} &&
      export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
      python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_00_ingest_to_lance.py
      --manifest {{pipeline.parameters.input}}
      --output {{pipeline.parameters.lance_uri}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --source {{pipeline.parameters.s00_source}}
      --num-cpus 1.0
      {{pipeline.parameters.s00_force}}
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: 2
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
```

```
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S01: Ingest ==========
- name: p6-01-ingest
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.cpu_queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      source {{pipeline.parameters.bos_env_file}} &&
      export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
      python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_01_ingest.py
      --input {{pipeline.parameters.lance_uri}}
      --output {{pipeline.parameters.lance_uri}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --num-cpus 1.0
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: 2
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
```

```
      name:
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S02: Split Scene (CPU) ==========
  - name: p6-02-split-scene
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.cpu_queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        source {{pipeline.parameters.bos_env_file}} &&
        export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_02_split_scene.py
        --input {{pipeline.parameters.lance_uri}}
        --output {{pipeline.parameters.lance_uri}}
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --threshold {{pipeline.parameters.threshold}}
        --min-scene-len {{pipeline.parameters.min_scene_len}}
        --num-cpus {{pipeline.parameters.s02_num_cpus}}
        --batch-size 1
      jobSpec:
        Head:
          replicas: 1
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: '{{pipeline.parameters.head_cpu}}'
          - name: memory
            quantity: '{{pipeline.parameters.head_memory}}'
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
          - name: HF_HUB_OFFLINE
            value: "1"
          - name: TRANSFORMERS_OFFLINE
            value: "1"
          command: ""
          args: ""
          postStart: ""
          preStop: ""
        Worker:
          replicas: '{{pipeline.parameters.s02_worker_replicas}}'
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: '{{pipeline.parameters.s02_worker_cpu}}'
          - name: memory
            quantity: '{{pipeline.parameters.s02_worker_memory}}'
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
          - name: HF_HUB_OFFLINE
            value: "1"
          - name: TRANSFORMERS_OFFLINE
            value: "1"
          command: ""
          args: ""
          postStart: ""
          preStop: ""
      datasources:
      - type: pfs
        sourcePath: '{{pipeline.parameters.pfs_source_path}}'
```

```
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S03: Embed Video (GPU) ==========
- name: p6-03-embed-video
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.gpu_queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      source {{pipeline.parameters.bos_env_file}} &&
      export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_03_embed_video.py
      --input {{pipeline.parameters.lance_uri}}
      --output {{pipeline.parameters.lance_uri}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --model-name {{pipeline.parameters.embed_model_name}}
      --num-cpus 2.0
      --num-gpus {{pipeline.parameters.s03_num_gpus}}
      --batch-size 1
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        - name: AIHC_MODEL_DIR
          value: "{{pipeline.parameters.embed_model_name}}"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: '{{pipeline.parameters.s03_worker_replicas}}'
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        - name: baidu.com/rtx_4090_cgpu
          quantity: '{{pipeline.parameters.s03_worker_gpu}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
```

```
        - name: AIHC_MODEL_DIR
          value: "{{pipeline.parameters.embed_model_name}}"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      datasources:
      - type: pfs
        sourcePath: '{{pipeline.parameters.pfs_source_path}}'
        mountPath: '{{pipeline.parameters.pfs_mount_path}}'
        name: '{{pipeline.parameters.pfs_id}}'
      - type: bos
        name: ""
        sourcePath: '{{pipeline.parameters.bos_source_path}}'
        mountPath: '{{pipeline.parameters.bos_mount_path}}'
      labels:
        aijob.cce.baidubce.com/create-from-aihcp: "true"
        aijob.cce.baidubce.com/log-collection: v3
      autoCreatePVC: true
      hostNetwork: false
  ###### ========== S04: KMeans (CPU) ==========
  - name: p6-04-kmeans
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.cpu_queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        source {{pipeline.parameters.bos_env_file}} &&
        export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_04_kmeans.py
        --input {{pipeline.parameters.lance_uri}}
        --output {{pipeline.parameters.lance_uri}}
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --n-clusters {{pipeline.parameters.n_clusters}}
        --num-cpus 4.0
      jobSpec:
        Head:
          replicas: 1
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: '{{pipeline.parameters.head_cpu}}'
          - name: memory
            quantity: '{{pipeline.parameters.head_memory}}'
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
          - name: HF_HUB_OFFLINE
            value: "1"
          - name: TRANSFORMERS_OFFLINE
            value: "1"
          command: ""
          args: ""
          postStart: ""
          preStop: ""
        Worker:
          replicas: 1
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: 16
          - name: memory
            quantity: 64
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
          - name: HF_HUB_OFFLINE
```

```
        value: "1"
      - name: TRANSFORMERS_OFFLINE
        value: "1"
      command: ""
      args: ""
      postStart: ""
      preStop: ""
  datasources:
  - type: pfs
    sourcePath: '{{pipeline.parameters.pfs_source_path}}'
    mountPath: '{{pipeline.parameters.pfs_mount_path}}'
    name: '{{pipeline.parameters.pfs_id}}'
  - type: bos
    name: ""
    sourcePath: '{{pipeline.parameters.bos_source_path}}'
    mountPath: '{{pipeline.parameters.bos_mount_path}}'
  labels:
    aijob.cce.baidubce.com/create-from-aihcp: "true"
    aijob.cce.baidubce.com/log-collection: v3
  autoCreatePVC: true
  hostNetwork: false
###### ========== S05: Dedup (CPU) ==========
- name: p6-05-dedup
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.cpu_queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      source {{pipeline.parameters.bos_env_file}} &&
      export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_05_dedup.py
      --input {{pipeline.parameters.lance_uri}}
      --output {{pipeline.parameters.lance_uri}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --threshold 0.95
      --num-cpus 4.0
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: 3
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
```

```
        value: "1"
      - name: HF_HUB_OFFLINE
        value: "1"
      - name: TRANSFORMERS_OFFLINE
        value: "1"
      command: ""
      args: ""
      postStart: ""
      preStop: ""
  datasources:
  - type: pfs
    sourcePath: '{{pipeline.parameters.pfs_source_path}}'
    mountPath: '{{pipeline.parameters.pfs_mount_path}}'
    name: '{{pipeline.parameters.pfs_id}}'
  - type: bos
    name: ""
    sourcePath: '{{pipeline.parameters.bos_source_path}}'
    mountPath: '{{pipeline.parameters.bos_mount_path}}'
  labels:
    aijob.cce.baidubce.com/create-from-aihcp: "true"
    aijob.cce.baidubce.com/log-collection: v3
  autoCreatePVC: true
  hostNetwork: false
###### ========== S06: DOVER Aesthetic (GPU) ==========
- name: p6-06-dover-aesthetic
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.gpu_queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      source {{pipeline.parameters.bos_env_file}} &&
      export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_06_dover_aesthetic.py
      --input {{pipeline.parameters.lance_uri}}
      --output {{pipeline.parameters.lance_uri}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --model-path {{pipeline.parameters.dover_model_name}}
      --backend {{pipeline.parameters.dover_backend}}
      --laion-mlp-path {{pipeline.parameters.dover_laion_mlp_path}}
      --clip-model {{pipeline.parameters.dover_clip_model}}
      --num-frames {{pipeline.parameters.dover_num_frames}}
      --num-clips {{pipeline.parameters.dover_num_clips}}
      --num-cpus 2.0
      --num-gpus 0.5
      --batch-size 1
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        - name: AIHC_MODEL_DIR
          value: "{{pipeline.parameters.dover_model_name}}"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: 3
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        - name: baidu.com/rtx_4090_cgpu
          quantity: 1
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
```

```
        value: "true"
      - name: DAFT_RUNNER
        value: "ray"
      - name: BOS_ENDPOINT
        value: "https://bj.bcebos.com"
      - name: BOS_S3_ENDPOINT
        value: "https://s3.bj.bcebos.com"
      - name: BOS_REGION
        value: "bj"
      - name: LANCE_DATA_STORAGE_VERSION
        value: "stable"
      - name: LANCE_ENABLE_V2_MANIFEST_PATHS
        value: "1"
      - name: HF_HUB_OFFLINE
        value: "1"
      - name: TRANSFORMERS_OFFLINE
        value: "1"
      - name: AIHC_MODEL_DIR
        value: "{{pipeline.parameters.dover_model_name}}"
      command: ""
      args: ""
      postStart: ""
      preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S07: NSFW Video (GPU) ==========
  - name: p6-07-nsfw-video
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.gpu_queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        source {{pipeline.parameters.bos_env_file}} &&
        export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_07_nsfw_video.py
        --input {{pipeline.parameters.lance_uri}}
        --output {{pipeline.parameters.lance_uri}}
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --model-name {{pipeline.parameters.nsfw_model_name}}
        --num-cpus 2.0
        --num-gpus 0.25
        --batch-size 1
      jobSpec:
        Head:
          replicas: 1
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: '{{pipeline.parameters.head_cpu}}'
          - name: memory
            quantity: '{{pipeline.parameters.head_memory}}'
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
          - name: HF_HUB_OFFLINE
            value: "1"
          - name: TRANSFORMERS_OFFLINE
            value: "1"
          - name: AIHC_MODEL_DIR
            value: "{{pipeline.parameters.nsfw_model_name}}"
          command: ""
          args: ""
          postStart: ""
          preStop: ""
        Worker:
          replicas: 3
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: 16
```

```
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        - name: baidu.com/rtx_4090_cgpu
          quantity: 1
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        - name: AIHC_MODEL_DIR
          value: "{{pipeline.parameters.nsfw_model_name}}"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S08: Filter (CPU) ==========
  - name: p6-08-filter
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.cpu_queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        source {{pipeline.parameters.bos_env_file}} &&
        export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
        python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_08_filter.py
        --input {{pipeline.parameters.lance_uri}}
        --output {{pipeline.parameters.lance_uri}}
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --dover-aesthetic-min {{pipeline.parameters.dover_aesthetic_min}}
        --nsfw-threshold {{pipeline.parameters.nsfw_threshold}}
        --num-cpus 1.0
      jobSpec:
        Head:
          replicas: 1
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: '{{pipeline.parameters.head_cpu}}'
          - name: memory
            quantity: '{{pipeline.parameters.head_memory}}'
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
          - name: HF_HUB_OFFLINE
            value: "1"
          - name: TRANSFORMERS_OFFLINE
            value: "1"
          command: ""
          args: ""
          postStart: ""
          preStop: ""
        Worker:
```

```
      worker:
        replicas: 2
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S09: Caption (GPU) ==========
  - name: p6-09-caption
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.gpu_queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        source {{pipeline.parameters.bos_env_file}} &&
        export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_09_caption.py
        --input {{pipeline.parameters.lance_uri}}
        --output {{pipeline.parameters.lance_uri}}
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --model-name {{pipeline.parameters.caption_model_name}}
        --backend {{pipeline.parameters.s09_backend}}
        --tensor-parallel-size {{pipeline.parameters.s09_tensor_parallel_size}}
        --gpu-memory-utilization {{pipeline.parameters.s09_gpu_memory_utilization}}
        --max-model-len {{pipeline.parameters.s09_max_model_len}}
        --num-cpus 2.0
        --num-gpus {{pipeline.parameters.s09_num_gpus}}
        --concurrency 1
        --batch-size 1
      jobSpec:
        Head:
          replicas: 1
          restartPolicy: Never
          image: '{{pipeline.parameters.image}}'
          resources:
          - name: cpu
            quantity: '{{pipeline.parameters.head_cpu}}'
          - name: memory
            quantity: '{{pipeline.parameters.head_memory}}'
          - name: sharedMemory
            quantity: '{{pipeline.parameters.shared_memory}}'
          cardLabel: ""
          envs:
          - name: LOG_COLLECTION
            value: "true"
          - name: DAFT_RUNNER
            value: "ray"
          - name: BOS_ENDPOINT
            value: "https://bj.bcebos.com"
          - name: BOS_S3_ENDPOINT
            value: "https://s3.bj.bcebos.com"
          - name: BOS_REGION
            value: "bj"
          - name: LANCE_DATA_STORAGE_VERSION
            value: "stable"
          - name: LANCE_ENABLE_V2_MANIFEST_PATHS
            value: "1"
```

```
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        - name: AIHC_MODEL_DIR
          value: "{{pipeline.parameters.caption_model_name}}"
        - name: CUDA_VISIBLE_DEVICES
          value: "0,1"
        - name: VLM_NUM_GPUS
          value: "2"
        - name: VLM_CONCURRENCY
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: '{{pipeline.parameters.s09_worker_replicas}}'
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.s09_worker_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.s09_worker_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        - name: baidu.com/rtx_4090_cgpu
          quantity: '{{pipeline.parameters.s09_worker_gpu}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        - name: CUDA_VISIBLE_DEVICES
          value: "0,1"
        - name: VLM_NUM_GPUS
          value: "2"
        - name: VLM_CONCURRENCY
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S10: Shard (CPU) ==========
- name: p6-10-shard
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.cpu_queue_id}}'
    priority: '{{pipeline.parameters.priority}}'
    jobType: RayJob
    command: >-
      source {{pipeline.parameters.bos_env_file}} &&
      export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
      DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_10_shard.py
      --input {{pipeline.parameters.lance_uri}}
      --output {{pipeline.parameters.lance_uri}}
      --tmp-dir {{pipeline.parameters.tmp_dir}}
      --output-dir {{pipeline.parameters.output_dir}}/shards
      --shard-prefix {{pipeline.parameters.shard_prefix}}
      --max-shard-size-mb {{pipeline.parameters.max_shard_size_mb}}
      --max-samples-per-shard {{pipeline.parameters.max_samples_per_shard}}
      --num-cpus 2.0
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
```

```
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: 0
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false
  ###### ========== S11: Export LeRobot v3.0 (CPU) ==========
  - name: p6-11-export-lerobot-v30
    type: CustomTask
    spec:
      queue: '{{pipeline.parameters.cpu_queue_id}}'
      priority: '{{pipeline.parameters.priority}}'
      jobType: RayJob
      command: >-
        source {{pipeline.parameters.bos_env_file}} &&
        export LD_LIBRARY_PATH=$(python3 -c "import os,sys;print(os.path.join(sys.prefix,'lib'))"):$LD_LIBRARY_PATH &&
        DAFT_RUNNER=ray python3 {{pipeline.parameters.bos_mount_path}}/workflow/p6_11_export_lerobot_v30.py
        --input {{pipeline.parameters.lance_uri}}
        --output {{pipeline.parameters.lance_uri}}
        --tmp-dir {{pipeline.parameters.tmp_dir}}
        --out {{pipeline.parameters.output_dir}}/lerobot_v3
        --repo-id {{pipeline.parameters.repo_id}}
        --fps {{pipeline.parameters.s11_fps}}
        --robot-type {{pipeline.parameters.s11_robot_type}}
        --action-dim {{pipeline.parameters.s11_action_dim}}
        --num-cpus 1.0
        --concurrency 1
        --batch-size 1
      jobSpec:
```

```
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: '{{pipeline.parameters.head_cpu}}'
        - name: memory
          quantity: '{{pipeline.parameters.head_memory}}'
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
      Worker:
        replicas: 2
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: sharedMemory
          quantity: '{{pipeline.parameters.shared_memory}}'
        cardLabel: ""
        envs:
        - name: LOG_COLLECTION
          value: "true"
        - name: DAFT_RUNNER
          value: "ray"
        - name: BOS_ENDPOINT
          value: "https://bj.bcebos.com"
        - name: BOS_S3_ENDPOINT
          value: "https://s3.bj.bcebos.com"
        - name: BOS_REGION
          value: "bj"
        - name: LANCE_DATA_STORAGE_VERSION
          value: "stable"
        - name: LANCE_ENABLE_V2_MANIFEST_PATHS
          value: "1"
        - name: HF_HUB_OFFLINE
          value: "1"
        - name: TRANSFORMERS_OFFLINE
          value: "1"
        command: ""
        args: ""
        postStart: ""
        preStop: ""
    datasources:
    - type: pfs
      sourcePath: '{{pipeline.parameters.pfs_source_path}}'
      mountPath: '{{pipeline.parameters.pfs_mount_path}}'
      name: '{{pipeline.parameters.pfs_id}}'
    - type: bos
      name: ""
      sourcePath: '{{pipeline.parameters.bos_source_path}}'
      mountPath: '{{pipeline.parameters.bos_mount_path}}'
    labels:
      aijob.cce.baidubce.com/create-from-aihcp: "true"
      aijob.cce.baidubce.com/log-collection: v3
    autoCreatePVC: true
    hostNetwork: false

######  ========== DAG: 线性 12 步 ==========
tasks:
- name: step00-ingest-to-lance
  taskTemplateName: p6-00-ingest-to-lance

- name: step01-ingest
  taskTemplateName: p6-01-ingest
  dependencies:
  - step00-ingest-to-lance

- name: step02-split-scene
  taskTemplateName: p6-02-split-scene
  dependencies:
  - step01-ingest
```

```
- name: step03-embed-video
  taskTemplateName: p6-03-embed-video
  dependencies:
  - step02-split-scene

- name: step04-kmeans
  taskTemplateName: p6-04-kmeans
  dependencies:
  - step03-embed-video

- name: step05-dedup
  taskTemplateName: p6-05-dedup
  dependencies:
  - step04-kmeans

- name: step06-dover-aesthetic
  taskTemplateName: p6-06-dover-aesthetic
  dependencies:
  - step05-dedup

- name: step07-nsfw-video
  taskTemplateName: p6-07-nsfw-video
  dependencies:
  - step06-dover-aesthetic

- name: step08-filter
  taskTemplateName: p6-08-filter
  dependencies:
  - step07-nsfw-video

- name: step09-caption
  taskTemplateName: p6-09-caption
  dependencies:
  - step08-filter

- name: step10-shard
  taskTemplateName: p6-10-shard
  dependencies:
  - step09-caption

- name: step11-export-lerobot
  taskTemplateName: p6-11-export-lerobot-v30
  dependencies:
  - step09-caption
```

单步独立执行 每个步骤也提供独立的工作流模板，可以单独提交运行，适用于调试或从中间步骤恢复执行。

例如单独运行 S03 视频嵌入：

```yaml
  version: v1
kind: PipelineTemplate

inputs:
- name: image
  type: string
  defaultValue: ccr-registry.baidubce.com/aihc/aihc-daft-gpu-0.5.2.12.2-cu129-py3.12-p6-sm120-sm89-ubuntu22.04
- name: queue_id
  type: string
  hint: GPU 队列 ID
- name: lance_uri
  type: string
  hint: Lance 表 URI
- name: model_name
  type: string
  defaultValue: nvidia/Cosmos-Embed1-448p
- name: num_frames
  type: int
  defaultValue: 8
- name: model_batch_size
  type: int
  defaultValue: 4

taskTemplates:
- name: p6-03-embed-video
  type: CustomTask
  spec:
    queue: '{{pipeline.parameters.queue_id}}'
    priority: high
    jobType: RayJob
    command: >-
      DAFT_RUNNER=ray python3 /opt/aihc/workflow/p6_03_embed_video.py
      --input {{pipeline.parameters.lance_uri}}
      --output {{pipeline.parameters.lance_uri}}
      --model-name {{pipeline.parameters.model_name}}
      --num-frames {{pipeline.parameters.num_frames}}
      --model-batch-size {{pipeline.parameters.model_batch_size}}
    jobSpec:
      Head:
        replicas: 1
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 6
        - name: memory
          quantity: 12
      Worker:
        replicas: 2
        restartPolicy: Never
        image: '{{pipeline.parameters.image}}'
        resources:
        - name: cpu
          quantity: 16
        - name: memory
          quantity: 64
        - name: baidu.com/rtx_4090_cgpu
          quantity: 1

tasks:
- name: embed-video-task
  taskTemplateName: p6-03-embed-video
```

使用建议

- **Lance 增量处理**：S00 支持增量追加（按 sample_id diff），无需每次全量重跑；`--force` 可强制覆盖重建
- **GPU 步骤是瓶颈**：S03/S06/S07/S09 需要 GPU，可通过增加 `worker_replicas` 提升吞吐
- **S09 Caption 资源较重**：默认需要 2×RTX 4090 + 128GB 内存，如显存不足可降低 `gpu_memory_utilization` 或使用更小模型
- **过滤阈值可调**：`dover_aesthetic_min` 和 `nsfw_threshold` 可根据数据质量灵活调整
- **S10/S11 并行输出**：WebDataset 分片供视觉-语言训练，LeRobot 供具身策略训练，按需启用
- **离线模式**：镜像内模型预装在 `/opt/aihc/models`，设置 `HF_HUB_OFFLINE=1` 避免网络依赖
- **全局单次操作**：S04（KMeans）和 S05（Dedup）为全局操作，concurrency 和 batch-size 参数无效

Human-to-Robot 数据合成管线

P9 Human-to-Robot 数据合成管线

概述 将人类第一视角（ego-centric）手部操作采集数据，自动转换为 Franka Panda 双臂机器人演示数据， 支持 episode 粒度分布式并行处理。

- **输入**：EgoVerse 格式的原始 zarr 数据集（第一视角 RGB 视频 + 手部位姿/关键点 + 头部位姿）
- **输出**：LeRobot v3.0 数据集（state/action parquet + H.264 编码视频 + 元信息）+ 3×3 分屏质检视频
- **业务价值**：真实机器人数据采集成本高，人手演示采集便宜可规模化。本 pipeline 把人手演示自动转成机器人演示，绕过真机采集，降低具身智能训练数据成本

数据处理分屏 Demo 示例

您浏览器不支持视频播放
您浏览器不支持视频播放

处理流程

```
原始 zarr（人手操作录像 + 位姿）
→ S1 数据加载清洗 → S2 动作对齐
→ S3 人手分割（GPU） → S4 背景补全（GPU） → S5 场景深度估计（GPU）
→ S6 Panda 重定向渲染（GPU，深度感知遮挡合成）
→ S7 LeRobot 数据集写出 → S8 分屏 Demo（GPU） → S9 数据集校验
```

算子映射 |步骤|算子|说明| |-|-|-| |S1|EgoVerseLoader|zarr v3 读取 + 哨兵帧过滤 + FPS 重采样 + head-relative 坐标变换| |S2|ActionAlignment|state/action 向量构造 + Savitzky-Golay 平滑| |S3|SAM3HandMask|SAM3 视频预测器时序分割人手/前臂/人体，输出二值 mask| |S4|ProPainterInpaint|ProPainter 视频 inpainting，擦除人体补全背景| |S5|DepthAnything3|DA3 单目深度估计，输出米制场景深度图| |S6|PandaRetarget|末端位姿 → 6-DOF IK → MuJoCo EGL 渲染 → 深度感知遮挡合成| |S7|LeRobotWriter|LeRobot v3.0 数据集写出 + H.264 编码 + 全局 stats 聚合| |S8|DemoGrid|3×3 分屏对比视频生成（含 MuJoCo 重渲染），供人工质检| |S9|DatasetValidate|LeRobot 数据集结构校验 + 加载器 smoke test|

数据流向

```
<raw_data_root>/（EgoVerse zarr，每个子目录一个 episode）
→ S1 → workspace/01_load/{ep}.npz
→ S2 → workspace/02_align/{ep}.npz
→ S3 → workspace/03_mask/{ep}/masks.npz + person_masks.npz + hand_arm_masks.npz + mask_overlay.mp4
→ S4 → workspace/04_inpaint/{ep}/bg.mp4 + inpaint_meta.json
→ S5 → workspace/05_depth/{ep}/scene_depth.npz + depth_vis.mp4
→ S6 → workspace/06_retarget/{ep}_ik.npz + {ep}_robot_on_bg.mp4 + summary.json
→ S7 → workspace/07_lerobot/（最终训练数据集）
→ S8 → workspace/08_demo/{ep}_demo.mp4（质检视频）
→ S9 → stdout 校验报告
```

前置条件 |资源|说明| |-|-| |GPU |S3/S4/S5/S6/S8 依赖 GPU，需有 Pro6k（96GB 显存）或同等 GPU| |CPU |S1/S2/S7/S9 纯 CPU，建议使用独立 CPU 队列避免浪费 GPU 资源| |PFS 存储|跨 step 共享中间产物，是 pipeline 正常运行的必要条件；按 77MB/episode 预估容量| |镜像|ccr-registry.baidubce.com/aihc/human2robot:daft-gpu-0.5.2-v4| |模型权重|镜像自带 `/opt/h2r_assets/`：SAM3（`sam3/sam3.pt`）、Depth-Anything-3-Base（`DA3-BASE/`）、ProPainter（`propainter/`）、mujoco_menagerie（`mujoco_menagerie/`）| |样例数据|镜像自带 `/data/egoverse/narrow_tabletop`，可用于验证流程|

全局参数说明 |参数名|类型|必填|默认值|说明| |-|-|-|-|-| | image|string|否|ccr-registry.baidubce.com/aihc/human2robot:daft-gpu-0.5.2-v4|| |priority|string|否|high|任务优先级| |cpu_queue_id|string|**是**|—|CPU 步骤队列 ID，在百舸「资源池→队列」查询，格式 aihcq-xxxxxxxx| |gpu_queue_id|string|**是**|—|GPU 步骤队列 ID，需确认队列中有 Pro6k 或同等 GPU| |gpu_card_resource|string|否|baidu.com/rtx_rpbzzz6_96g_cgpu|K8s GPU 资源名，固定为 Pro6k 资源名，一般不需要修改| |pfs_id|string|**是**|—|PFS 实例 ID，格式 pfs-xxxxxxxx，需与资源池同地域| |pfs_source_path|string|**是**|—|PFS 内源路径，即你在 PFS 上的根目录，如 /your-username| |pfs_mount_path|string|**是**|—|PFS 挂载到容器内的目标路径，格式 /mnt/pfs/xxxxxx，在百舸 PFS 挂载配置页面查看| |pfs_base_dir|string|**是**|—|PFS 挂载点下的工作根目录，建议 {pfs_mount_path}/{project}，所有产物写入此目录| |code_root|string|否|/workspace/LoongForge/loongforge/embodied/tools/ego2robot|镜像内代码根目录，一般不需要修改| |assets_root|string|否|/opt/h2r_assets|镜像内模型权重根目录，一般不需要修改| |raw_data_root|string|**是**|/data/egoverse/narrow_tabletop|输入数据根目录，包含所有 EgoVerse zarr episode 子目录（见「步骤详解 S1」中的格式说明）| |run_id|string|**是**|—|本次跑批唯一标识，建议带日期如 20260916-run1，同时决定工作区目录名；相同 run_id 会复用断点标记跳过已完成 episode| |target_fps|string|否|30.0|S1 重采样目标帧率，需与 LeRobot 数据集 FPS 一致| |action_skip|string|否|5|S2 action 前瞻步数，action[t] = state[t+skip]（绝对目标位姿），5 帧在 30fps 下约 166ms| |smooth_window|string|否|11|S2 Savitzky-Golay 平滑窗口，只作用于平移列与夹爪列| |body_prompt|string|否|person|S3 SAM3 文本 prompt| |mask_mode|string|否|both|S3 分割通道：both（person 文本 + 手臂几何框）/ person / arms| |mask_dilation|string|否|4|S4 送 ProPainter 前的 mask 膨胀像素，偏小会在 bg.mp4 留人体边缘残影| |inpaint_chunk_frames|string|否|120|S4 ProPainter 分块帧数，显存不足往下调| |depth_mode|string|否|depth-aware|S5/S6 合成模式；改为 alpha 会**整段跳过 S5**，step2 只跑 mask+inpaint| |depth_epsilon|string|否|0.02|S6 遮挡判定容差（米）| |depth_process_res|string|否|504|S5 DA3 推理分辨率| |retarget_height|string|否|360|S6 渲染高度，必须与 bg.mp4 一致| |retarget_fy|string|否|490.1961|S6 相机纵向焦距，用于换算 MuJoCo fovy；缺省值对应样例数据内参| |robot_preset|string|否|panda_dual_g2|S7 机器人布局预设，决定 state/action 维度为 18| |num_gpus_mask_inpaint_depth|string|否|0.5|S3+S4+S5 每 Actor GPU 占比，Pro6k 上建议 0.5（每卡 2 并发）| |num_gpus_retarget|string|否|0.5|S6 每 Actor GPU 占比| |num_gpus_demo|string|否|0.5|S8 每 Actor GPU 占比| |max_fail_ratio|string|否|0.05|单 step 允许的 episode 失败比例，超过则该 step 判定失败| |head_cpu|int|否|6|Head pod CPU 核数| |head_memory|int|否|24|Head pod 内存（GB）| |shared_memory|int|否|64|共享内存（GB），ProPainter 在 /dev/shm 落中间帧，step2 必须给足| |submitter_backoff_limit|int|否|0|RayJob Submitter 启动失败重试次数|

步骤详解 S1 数据加载清洗（EgoVerseLoader） 读取 EgoVerse zarr episode，清洗数据并标准化。

参数	类型	默认值	说明
target_fps	string	30.0	重采样目标帧率，统一时间刻度

- **输入**：`<raw_data_root>/{ep}/`，每个 episode 子目录须包含：
  - `zarr.json` — episode 元信息（帧数、采集帧率等），也是 episode 目录识别依据
  - `images.front_1/` — 第一视角 RGB 帧（逐帧 JPEG，zarr 格式）
  - `obs_head_pose/` — 头部位姿 (N, 7)
  - `left.obs_ee_pose/`、`right.obs_ee_pose/` — 双手末端位姿 (N, 7)
  - `left.obs_keypoints/`、`right.obs_keypoints/` — 手部关键点 (N, K, 3)
  - `annotations/` — 任务描述文本
- **输出**：`workspace/01_load/{ep}.npz`（清洗后的位姿、关键点、annotations，12 个字段）

S2 动作对齐（ActionAlignment） 将人手位姿序列转换为机器人训练可用的 state/action 向量。

参数	类型	默认值	说明
action_skip	string	5	action 前瞻步数，action[t] = state[t+skip]
smooth_window	string	11	Savitzky-Golay 平滑窗口

- **输入**：`workspace/01_load/{ep}.npz`
- **输出**：`workspace/02_align/{ep}.npz`（含 `state(N,16)`、`action(N,16)`、`annotations`，8 个字段）

16 维 = 左右手 `ee_pose` 各 7 维（位置 3 + 四元数 4）+ 夹爪宽度各 1 维。

S3 人手分割（SAM3HandMask） SAM3 视频预测器时序分割人手/前臂/人体，产出二值 mask。

参数	类型	默认值	说明
body_prompt	string	person	文本 prompt
mask_mode	string	both	分割通道：both / person / arms
num_gpus_mask_inpaint_depth	string	0.5	每 Actor GPU 占比（S3+S4+S5 共用）

- **输入**：原始 zarr 视频帧 + 手部 3D 关键点投影为 2D 几何框提示
- **输出**：`workspace/03_mask/{ep}/masks.npz`、`person_masks.npz`、`hand_arm_masks.npz`、`mask_overlay.mp4`

S4 背景补全（ProPainterInpaint） ProPainter 视频 inpainting，按 S3 mask 擦除人体并补全背景。

参数	类型	默认值	说明
inpaint_chunk_frames	string	120	分块帧数，显存不足往下调
mask_dilation	string	4	送 ProPainter 前的 mask 膨胀像素
num_gpus_mask_inpaint_depth	string	0.5	每 Actor GPU 占比（S3+S4+S5 共用）

- **输入**：原始 zarr 视频帧 + workspace/03_mask/{ep}/masks.npz
- **输出**：workspace/04_inpaint/{ep}/bg.mp4（去人体背景视频）、inpaint_meta.json

S5 场景深度估计（DepthAnything3） Depth Anything V3 单目深度估计，输出米制场景深度图，供 S6 判断机器人与场景的前后遮挡关系。

参数	类型	默认值	说明
depth_mode	string	depth-aware	改为 alpha 时整段跳过本步骤
depth_process_res	string	504	推理分辨率
num_gpus_mask_inpaint_depth	string	0.5	每 Actor GPU 占比（S3+S4+S5 共用）

- **输入**：workspace/04_inpaint/{ep}/bg.mp4（在去人背景上估深度，避免人手干扰）
- **输出**：workspace/05_depth/{ep}/scene_depth.npz（float16，N×360×640，单位米）、depth_vis.mp4

S6 Panda 双臂重定向（PandaRetarget） 将人手末端位姿重定向到 Panda 关节角（IK），MuJoCo EGL 渲染并按深度合成到背景。

参数	类型	默认值	说明
depth_mode	string	depth-aware	depth-aware 用深度判定遮挡，alpha 退化为纯 alpha 混合
depth_epsilon	string	0.02	遮挡判定容差（米）
retarget_height	string	360	渲染高度，必须与 bg.mp4 一致
retarget_fy	string	490.1961	相机纵向焦距，用于换算 MuJoCo fovy
num_gpus_retarget	string	0.5	每 Actor GPU 占比

- **输入**：原始 zarr（世界系位姿 + 相机参数）+ workspace/02_align/{ep}.npz（仅取帧数 N）
  - workspace/04_inpaint/{ep}/bg.mp4 + workspace/05_depth/{ep}/scene_depth.npz
- **输出**：workspace/06_retarget/{ep}_ik.npz（关节角序列 + IK 误差统计）、{ep}_robot_on_bg.mp4（合成视频，640×360）、summary.json

S7 LeRobot 数据集写出（LeRobotWriter） 将所有 episode 的产物打包为 LeRobot v3.0 数据集。

参数	类型	默认值	说明
robot_preset	string	panda_dual_g2	机器人布局，(7,2,7,2) → 18 维 state/action

- **输入**：workspace/06_retarget/（全部 episode 的 _ik.npz 与 _robot_on_bg.mp4）
  - workspace/02_align/（annotations 任务文本）
- **输出**：

```
workspace/07_lerobot/
meta/info.json          ← codebase_version "v3.0" / robot_type "panda_dual" / fps 30
meta/stats.json         ← 各字段统计（min/max/mean/std/quantile）
meta/tasks.parquet      ← 任务描述文本
meta/episodes/          ← 每个 episode 的起止帧索引
data/chunk-000/file-000.parquet          ← state/action 帧数据（18 维）
videos/observation.images.ego/.../file-000.mp4  ← H.264 编码视频
```

S8 分屏 Demo（DemoGrid） 生成 3×3 分屏对比视频，供人工质检处理效果。

参数	类型	默认值	说明
num_gpus_demo	string	0.5	每 Actor GPU 占比

- **输入**：原始 zarr + workspace/03_mask/ + workspace/04_inpaint/
  - workspace/05_depth/ + workspace/06_retarget/
- **输出**：workspace/08_demo/{ep}_demo.mp4（3×3 九宫格）

九格内容：原始画面 / 手部关键点 / 夹爪间距曲线 / SAM3 mask / inpaint 背景 / 场景深度可视化 / 纯机器人渲染 / 机器人合成 / IK 误差曲线。

S9 数据集校验（DatasetValidate） 对 LeRobot 数据集做结构与一致性校验。

- **输入**：workspace/07_lerobot/
- **输出**：stdout 校验报告，全部通过输出 All validation checks passed!

任务拓扑

```
 +------------------+   +-----------------------------+
| step1-preprocess |   | step2-mask-inpaint-depth    |
|   S1+S2 (CPU)    | \ /     S3+S4+S5 (GPU)          |
+------------------+  \/  +--------------------------+
                      |
                      ↓
              +-----------------+
              | step3-retarget  |
              |    S6 (GPU)     |
              +-----------------+
                /             \
               ↓               ↓
+--------------------+   +-----------+
| step4-lerobot-merge |   | step5-demo |
|    S7+S9 (CPU)      |   |  S8 (GPU)  |
+--------------------+   +-----------+
```

5 个工作流 step，step1/step2 并行启动（菱形结构），step3 等两者完成后启动，step4/step5 并行执行。

工作流 Step	包含算子	资源	依赖	Ray 形态
step1-preprocess	S1 + S2	CPU	无（与 step2 并行）	每 episode 一个 task
step2-mask-inpaint-depth	S3 + S4 + S5	GPU	无（与 step1 并行）	Actor 池
step3-retarget	S6	GPU	step1 + step2	Actor 池
step4-lerobot-merge	S7 + S9	CPU	step3（与 step5 并行）	单 task（barrier）
step5-demo	S8	GPU	step3（与 step4 并行）	Actor 池

断点续跑机制 每个 episode 每个步骤完成后写入 `.done` 标记文件，重新提交工作流时自动跳过已完成的 episode。

```
{pfs_base_dir}/workspace-{run_id}/.done/
{ep}_s1  {ep}_s2  {ep}_s3  {ep}_s4  {ep}_s5  {ep}_s6  {ep}_s8
```

标记内容是一行 JSON（时间戳 + 帧数或耗时），仅记录「跑完了」，不校验产物完整性。

- **同一个** `run_id` **重跑** → 只补跑失败/未完成的 episode，已完成的秒过
- **换** `run_id` **重跑** → 新工作区，全量重算
- **S7/S9（step4）无 episode 级标记** → 只要被调度就重新扫描 06_retarget ∩ 02_align 并重写 07_lerobot/

手动触发重跑：删除对应标记后重新提交工作流。

```
###### 重跑某个 episode 的某步（S3 重算需连带删 s4/s5，否则下游不会更新）
rm workspace-{run_id}/.done/{ep}_s3 workspace-{run_id}/.done/{ep}_s4 workspace-{run_id}/.done/{ep}_s5

###### 重跑某一步的全部 episode
rm workspace-{run_id}/.done/*_s6

###### 重跑全部
rm -rf workspace-{run_id}/.done/
```

UMI 人类演示数据处理

概述

将 UMI（Universal Manipulation Interface）手持夹爪采集的 GoPro 演示视频，重建为包含图像、末端（TCP）位姿和夹爪开合宽度的 Zarr 数据集，同时为每段演示（episode）生成任务描述（caption）和子任务分段（subtask）候选。

- **输入**：收件箱目录。每个子目录是一个采集会话（session），包含 1 段建图视频、至少 1 段夹爪标定视频和若干段演示视频。
- **输出**：每个 session 一份 Zarr 数据集 `dataset.zarr.zip`、数据集规划 `dataset_plan.pkl`、质量报告和语义标注结果；每个批次一份交付清单 `manifest.jsonl` 和完成标记 `_SUCCESS.json`。
- **业务价值**：UMI 采集不需要机器人本体，单人手持即可完成示教，成本低、易于多人多地并行。本管线把建图定位、标定、对齐、语义标注和质检串成一条工作流：
  - **放入即处理**：自动识别收件箱中的 session，无需逐个配置。
  - **并行处理**：视频和 episode 在 Worker 间流式并行；数据集生成与语义标注两个分支同时运行。以示例数据为例，3 个 session（每个 5 段演示）从提交到完成约 27 分钟。
  - **结果可追溯**：每个批次绑定镜像代码、配置和输入文件的 SHA256，每个产物记录大小和 SHA256。

> **说明**：本模板不包含目标机器人重定向、LeRobot 格式转换、模型训练和实机执行。

处理前后示例

输入为一段原始 GoPro 视频，处理后得到 224×224 的图像序列、整段描述和分段描述。下图为同一段演示处理前后的画面及对应描述。



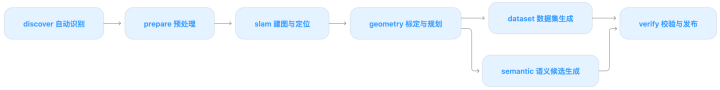
整段描述

> 机械臂从桌子上方接近一个蓝色杯子，逐渐闭合夹爪并抓取杯子，随后将杯子抬起并移动到桌子另一侧，最终将杯子放置在桌面上。整个过程中，夹爪的宽度变化和TCP位置变化与操作同步，未观察到与杯子的直接接触证据，但夹爪闭合后杯子位置发生移动，表明成功抓取和释放。

分段描述（第 1 段）

> 机械臂从初始位置向桌子上的蓝色杯子移动，夹爪保持张开状态，TCP位置逐渐靠近杯子。 以上描述为模型生成的原文。语义结果属于候选，使用前需人工审核。

处理流程




```
原始 GoPro 视频（收件箱）
→ S1 discover  自动识别 session，生成输入清单
→ S2 prepare   整理视频，提取 IMU
→ S3 slam      建图，逐视频定位
→ S4 geometry  标记检测、标定、数据集规划
→ S5 dataset   生成 Zarr 数据集        ┐
→ S6 semantic  生成语义候选（GPU）     ┘ 两者并行
→ S7 verify    汇总质量报告，发布交付物
```

S5 dataset 与 S6 semantic 都只依赖 S4 geometry，两者并行执行；S7 verify 在两者都完成后执行。

算子映射

步骤	节点	主要算子	说明
S1	discover	`discover.sh`	扫描收件箱，生成输入清单
S2	prepare	`umi_mapper_prepare_session`	整理视频，提取 GoPro IMU
S3	slam	`umi_mapper_create_map` `umi_mapper_slam_video`	ORB-SLAM3 建图与定位

S4	geometry	`umi_mapper_detect_aruco`
		`umi_mapper_calibrate_session`
		`umi_mapper_dataset_plan`
			ArUco 检测、标定、规划

S5	dataset	`umi_mapper_replay_buffer` `umi_mapper_verify_dataset`	生成并校验 Zarr

S6	semantic	`umi_mapper_semantic_prepare`
		`umi_mapper_vlm_annotate`
		`umi_mapper_semantic_summary`
			抽帧、模型标注、汇总

S7	verify	`umi_mapper_verify_report`	合并质量报告，发布

S2～S7 每一步结束时由 `umi_mapper_record_artifacts` 记录产物的路径、大小和 SHA256，下游步骤开始前会先校验这些记录。各步骤的完整算子链由镜像内的 `configs/<步骤>.yaml` 定义。

数据流向

所有产物写在同一个批次目录下，批次目录由 `batch_tag` 决定：

```
<inbox>/<session_id>/raw_videos/          原始视频
→ S1 discover   /mnt/data/umi/deployments/umi-<batch_tag>/{runtime.json, inputs.jsonl}
→ S2 prepare    runs/<batch_tag>/sessions/<session_id>/demos/*/{raw_video.mp4, imu_data.json}
→ S3 slam       demos/mapping/map_atlas.osa、demos/*/camera_trajectory.csv
→ S4 geometry   demos/*/tag_detection.pkl、gripper_range.json、dataset_plan.pkl
→ S5 dataset    dataset.zarr.zip、quality-report.json
→ S6 semantic   semantic/{captions.jsonl, subtasks.jsonl, index.html, summary.json}
→ S7 verify     verify/quality-report.json
→ 发布          deliveries/<batch_tag>/verify-attempts/<attempt_id>/
                deliveries/<batch_tag>/verify/_SUCCESS.json
```

S2～S7 都在同一个 session 目录 `runs/<batch_tag>/sessions/<session_id>/` 中追加产物，步骤之间不复制数据。

前置条件

资源	说明
资源池与队列	全托管资源池，已安装工作流组件
GPU	S6 semantic 依赖 GPU，单卡显存不低于 24 GB（已验证 RTX 4090）
CPU	S1～S5、S7 为纯 CPU 步骤
PFS 存储	收件箱、中间产物、交付物和模型缓存都在 PFS 上
镜像	〔待确认：公开镜像地址〕
模型权重	Qwen3-VL-32B-Instruct，需预先放入模型缓存目录
标定文件	镜像内置 `/opt/umi/example/calibration`
样例数据	〔待确认：公开样例数据入口〕

资源总量：S5 dataset 与 S6 semantic 并行运行，会同时申请资源。按默认参数，两者合计 60 核 CPU、368 GiB 内存、2 张 GPU。RayJob 按整组调度，队列空闲资源不足时整个任务排队，不会只启动一部分 Worker。提交前请确认队列空闲资源覆盖两者之和，并为 Submitter 预留少量资源。

模型权重：S6 semantic 离线加载 Qwen3-VL-32B-Instruct（4 bit 量化，单卡运行），不会在运行时下载。请提前将权重放到 PFS 上，使其在容器内的路径为 `/mnt/data/umi/cache/semantic-models`。〔待确认：权重目录的具体布局，例如是否为 Hugging Face 缓存格式。〕

标定文件：镜像内置的标定文件 `gopro_intrinsics_2_7k.json` 和 `aruco_config.yaml` 适用于下文"采集要求"中的 GoPro 与标记配置。相机型号、镜头模式或标记规格不同时，需要替换标定文件，示例标定不能直接用于其他采集设备。

输入说明

采集要求

项目	要求
相机	GoPro，Max Lens 鱼眼模式（已验证 HERO9）
视频规格	2.7K 4:3，2704×2028，59.94 fps
元数据	保留 GPMF 惯导轨道、timecode、相机序列号和拍摄时间
夹爪标记	ArUco DICT_4X4_50，ID 0/1，边长 0.016 m
桌面标记	ArUco ID 12～19，边长 0.16 m

> **注意**：请勿转码或剪辑原始 MP4，否则 IMU 和时间元数据会丢失，S2 无法提取惯导数据。

目录结构

收件箱的每个子目录表示一个 session，子目录名即 `session_id`：

```
<inbox>/
└── <session_id>/
    └── raw_videos/
        ├── mapping.mp4                  # 必需，建图视频
        ├── gripper_calibration/
        │   └── <标定视频>.MP4             # 必需，至少 1 个
        ├── <演示视频 1>.MP4
        └── <演示视频 2>.MP4
```

- **建图视频**：缓慢环绕桌面场景拍摄，用于建立 SLAM 地图。
- **夹爪标定视频**：完整开合夹爪数次，用于标定开口范围。
- **演示视频**：每段完成一次任务，生成一个或多个 episode。

S1 discover 会跳过不符合上述结构的子目录，并在日志中给出原因。运行后请在 discover 节点日志中核对识别出的 session 数量。

全局参数说明

参数名	类型	必填	默认值	说明
image	string	是	无	UMI 工作流镜像地址
queue_id	string	是	无	CPU 步骤队列 ID，格式 aihcq-xxxxxxxx
gpu_queue_id	string	是	无	S6 semantic 队列 ID，可与 queue_id 相同
priority	string	否	normal	任务优先级
pfs_id	string	是	无	PFS 实例 ID，格式 pfs-xxxxxx
pfs_source_path	string	否	/	PFS 内的源路径
pfs_mount_path	string	否	/mnt/data	容器内挂载路径，请勿修改
batch_tag	string	是	无	批次标识，每批数据使用新值
inbox	string	是	无	收件箱在容器内的绝对路径
head_cpu	int	否	4	CPU 步骤 Head 核数
head_memory	int	否	32	CPU 步骤 Head 内存（GiB）
shared_memory	int	否	32	共享内存（GiB），不低于 16
worker_replicas	int	否	3	CPU Worker 数量
worker_cpu	int	否	12	单个 CPU Worker 核数，不低于 4
worker_memory	int	否	64	单个 CPU Worker 内存（GiB）
gpu_worker_replicas	int	否	2	GPU Worker 数量
gpu_per_worker	int	否	1	单个 GPU Worker 的卡数
gpu_resource_name	string	是	无	GPU 资源名，如 baidu.com/rtx_4090_cgpu
gpu_head_cpu	int	否	4	S6 semantic Head 核数
gpu_head_memory	int	否	16	S6 semantic Head 内存（GiB）
gpu_worker_cpu	int	否	8	单个 GPU Worker 核数，不低于 6
gpu_worker_memory	int	否	64	单个 GPU Worker 内存（GiB）
thread_limit	string	否	"1"	计算库线程上限，建议 1～2

参数填写要点：

- **pfs_mount_path**：运行配置和清单中使用的是 /mnt/data 开头的绝对路径，修改挂载路径会导致各步骤找不到文件。inbox 也应填写 /mnt/data 下的路径，例如 /mnt/data/umi/inbox/<目录名>。
- **batch_tag**：决定批次目录 /mnt/data/umi/deployments/umi-<batch_tag>/。批次会绑定镜像代码、配置和输入文件，处理新数据或更换镜像时请使用新的 batch_tag，例如 20261113-run1。
- **CPU 资源**：Head 只运行调度程序，不运行算子，算子都在 Worker 上执行。单个算子任务最多申请 4 核，因此 worker_cpu 不能低于 4。数据量增大时，优先增加 worker_replicas。
- **GPU 资源**：总卡数 = gpu_worker_replicas × gpu_per_worker，每张卡运行一个模型实例。每个 GPU Worker 需同时容纳模型实例（2 核）和片段预处理（4 核），因此 gpu_worker_cpu 不能低于 6。模型加载到显存前会先读入内存，gpu_worker_memory 建议保持 64。
- **thread_limit**：OpenCV、FFmpeg、OpenMP、MKL 默认按物理核数开线程，容易超额占用 CPU。本参数通过环境变量传入各进程，以字符串形式填写。
- **调整资源**：资源申请量、算子实际占用和吞吐不成正比。修改 Worker 规格后，请同时核对队列空闲资源是否覆盖 S5 与 S6 两个任务之和。

以下处理配置由 S1 discover 写入 runtime.json，不作为全局参数开放：

配置	当前值
输出图像分辨率	224×224
语义抽帧数	每个 episode 20 帧
语义输出语言	中文
标定目录	/opt/umi/example/calibration
模型缓存目录	/mnt/data/umi/cache/semantic-models

步骤详解

S2～S7 启动后，Head 上的调度程序先校验批次绑定和上游产物（大小、SHA256），再按 configs/<步骤>.yaml 的算子链在 Worker 上流式执行。某一行（session、视频或 episode）出错时，该行被标记为失败并记录错误，其他行继续处理。

S1 discover：自动识别

扫描收件箱，识别符合目录结构的 session，计算每个源文件的 SHA256，创建批次目录并生成运行配置。本步骤只启动 Head，不启动 Worker。

- **输入**：inbox 下的各 session 子目录
- **输出**：runtime.json（运行配置）、inputs.jsonl（每个 session 一行，含文件路径、大小和 SHA256）

S2 prepare：视频整理与 IMU 提取

每个 session 一行。将原始视频复制到 session 目录并逐个校验 SHA256，按相机序列号和拍摄时间整理视频目录，再从 MP4 内嵌的 GPMF 轨道中提取加速度计和陀螺仪数据。

- **输入**：inputs.jsonl
- **输出**：demos/<视频目录>/raw_video.mp4、imu_data.json

S3 slam：建图与定位

先用建图视频为每个 session 建立 ORB-SLAM3 地图（单目 + 惯导），再展开为每个视频一行，基于同一张地图逐个视频定位。地图丢帧比例超过 20% 时自动重建，最多运行 2 次。夹爪标定视频不做定位。

- **输入**：S2 整理后的视频和 IMU 数据
- **输出**：demos/mapping/map_atlas.osa（地图）、demos/*/camera_trajectory.csv（每段视频的相机轨迹）

> **说明**：本步骤成功表示程序生成了轨迹。轨迹精度是否满足应用要求，需要结合跟踪状态、标定结果和独立参考另行评估。

S4 geometry：标记检测、标定与数据集规划

展开为每个视频一行，逐帧检测夹爪手指标记和桌面标记。随后按 session 汇总，计算 SLAM 坐标系到桌面标记坐标系的变换，用夹爪标定视频确定开口范围，最后生成数据集规划：按恒定帧率对齐图像、轨迹和标记，在丢跟踪处切分 episode。

- **输入**：S3 的相机轨迹、镜像内置标定文件
- **输出**：`demos/*/tag_detection.pkl`、`demos/mapping/tx_slam_tag.json`、`gripper_range.json`、`dataset_plan.pkl`

TCP 位姿以桌面标记坐标系为参考，平移单位为米，旋转为轴角（弧度）。接入目标机器人前，需另行完成坐标适配和动作验证。

S5 dataset：Zarr 数据集生成

每个 session 一行。按规划逐帧解码，修补标记区域、遮挡夹爪本体，缩放到 224×224 后写入 `dataset.zarr.zip`，再对照规划校验帧对齐和数值有效性。图像保留鱼眼视角，未做去畸变。

- **输入**：`dataset_plan.pkl`、原始视频
- **输出**：`dataset.zarr.zip`、`quality-report.json`（本步骤的阶段报告）

S6 semantic：语义候选生成

按规划展开为每个 episode 一行。CPU 算子剪出 episode 片段，从第一路相机抽取 20 个关键帧，附加夹爪局部放大图和宽度、位姿读数；GPU 算子调用 Qwen3-VL-32B 生成中文任务描述和子任务分段。每张卡常驻一个模型实例，模型只加载一次。结果未通过结构校验时，会把错误信息加入提示词重试 1 次。

- **输入**：`dataset_plan.pkl`、原始视频
- **输出**：`semantic/` 目录，含 `captions.jsonl`、`subtasks.jsonl`、审核页 `index.html` 和 `summary.json`

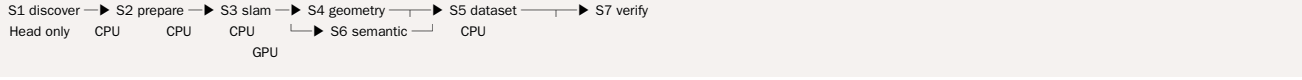
子任务分段采用整数帧的左闭右开区间 [start_frame, end_frame)。恰好落在 `end_frame` 上的证据帧会被去掉，修改记录写入 `conversion.json`。

S7 verify：质量汇总与发布

每个 session 一行。将 S6 的语义状态合并进 S5 的质量报告，生成最终质量报告 `verify/quality-report.json`（S5 的阶段报告保持不变）。随后发布每个 session 的数据集、规划、最终质量报告和语义目录，写入 `manifest.jsonl` 和 `summary.json`，最后写入完成标记 `_SUCCESS.json`。

- **输入**：S5 与 S6 的产物清单
- **输出**：`deliveries/<batch_tag>/` 下的版本化交付目录

任务拓扑



工作流节点	步骤	资源	依赖	Ray 行粒度
umi-discover	S1	仅 Head	无	不使用 Worker
umi-prepare	S2	CPU	S1	每 session 一行
umi-slam	S3	CPU	S2	每视频一行
umi-geometry	S4	CPU	S3	每视频一行，再汇总
umi-dataset	S5	CPU	S4	每 session 一行
umi-semantic	S6	GPU	S4	每 episode 一行
umi-verify	S7	CPU	S5、S6	每 session 一行

每个节点运行一个 RayJob，节点结束后集群即释放。S2～S5、S7 共用同一个 CPU 任务模板，通过 `stage` 参数区分步骤。

输出说明

交付目录

```
/mnt/data/umi/deployments/umi-<batch_tag>/deliveries/<batch_tag>/
├── verify/_SUCCESS.json                  # 完成标记，指向本次发布的版本
└── verify-attempts/<attempt_id>/
    ├── manifest.jsonl                    # 交付清单，每个文件的路径、大小、SHA256
    ├── summary.json                      # 批次汇总
    ├── <session_id>.zarr.zip             # 数据集
    ├── <session_id>.dataset_plan.pkl     # 数据集规划，含时间戳
    ├── <session_id>.quality-report.json  # 最终质量报告
    └── <session_id>.semantic/            # 语义候选与审核页
```

读取交付物时，请先打开 `verify/_SUCCESS.json`，按其中指向的 `attempt_id` 读取 `manifest.jsonl`，再按清单取文件，避免混用不同次运行的产物。

数据集 dataset.zarr.zip

Zarr v2 格式的 ZIP 存储，结构与 UMI 官方 Diffusion Policy 训练代码的 ReplayBuffer 一致，可用 `zarr.ZipStore` 直接打开。所有 episode 首尾相接存为一组数组，用 `meta/episode_ends` 切分。

字段	形状	类型	含义
meta/episode_ends	[E]	int64	每个 episode 的结束位置（不含）
data/camera0_rgb	[N,224,224,3]	uint8	处理后的 RGB 图像
data/robot0_eef_pos	[N,3]	float32	TCP 位置（米）
data/robot0_eef_rot_axis_angle	[N,3]	float32	TCP 姿态，轴角（弧度）
data/robot0_gripper_width	[N,1]	float32	夹爪开口宽度（米）
data/robot0_demo_start_pose	[N,6]	float64	所属 episode 起始位姿
data/robot0_demo_end_pose	[N,6]	float64	所属 episode 结束位姿

- N 为总帧数，E 为 episode 数。以单相机、单夹爪的示例为例，一个 session 含 5 个 episode、2317 帧，`episode_ends` 为 [468, 932, 1303, 1712, 2317]。多相机或多夹爪时，会出现 `camera1_rgb`、`robot1_*` 等字段。
- `camera0_rgb` 使用 JPEG XL 有损压缩（`imagecodecs_jpegxl`，`level=99`），读取环境需安装 `imagecodecs`。
- 字段名中的 `robot0` 沿用 UMI 命名，表示第 1 个夹爪，不代表目标机器人可直接执行的动作。

> **注意**：时间戳不存储在 Zarr 中，而是存储在同版本的 `dataset_plan.pkl` 的 `episode_timestamps` 中。使用数据集时请一并保留该文件。

语义候选 semantic/

文件	内容
captions.jsonl	每个 episode 一行，整段任务描述
subtasks.jsonl	每个 episode 一行，子任务分段列表
index.html	人工审核页，视频与候选结果对照
summary.json	生成状态与审核状态

`subtasks.jsonl` 中的子任务示例：

{"start_frame": 0, "end_frame": 104, "description": "机械臂从初始位置向桌子上的蓝色杯子移动，夹爪保持张开状态，TCP位置逐渐靠近杯子。", "evidence_frames": [0, 26, 52, 78]}

所有候选的初始状态为 `review_status=pending`、`semantic_accepted=false`、`training_label=false`。结构校验通过只表示 JSON 格式和字段正确，不代表描述与视频相符。模型可能出现描述偏差，例如把手持夹爪称为"机械臂"，或在缺少接触证据时仍判断抓取成功。请在 `index.html` 中对照视频核对操作对象、接触、动作结果和分段边界，审核通过后再作为训练标签使用。

质量报告 quality-report.json

字段	含义	示例结果
status	总体状态：PASS / WARN / FAIL	WARN
structural	数据集与规划一致、数值有效、边界正确	PASS
rgb_source_correspondence	逐帧图像与原视频对应帧比对	PASS
semantic	语义候选状态	WARN（待审核）
physical	物理指标判定	unknown
calibration	标定残差	unknown

`status=WARN` 是当前版本的正常结果：结构与对齐检查全部通过，但物理指标（最大速度、角速度、开口范围等）只计算、不设阈值，标定没有独立真值，语义候选也待审核。`summary.json` 中的 `delivery_pass=true` 表示交付物完整、哈希一致，不等同于 `quality_pass=true`。

> **说明**：请以 `verify/quality-report.json`（交付目录中的 `<session_id>.quality-report.json`）为准。S5 阶段报告中的语义状态在 S7 合并前尚未更新。

快速开始

1. 将采集好的 session 目录上传到 PFS，确保挂载后路径为 `/mnt/data/<收件箱路径>/<session_id>/raw_videos/`，并将模型权重放到 `/mnt/data/umi/cache/semantic-models`。
2. 在百舸控制台进入 **工作流 > 工作流模板**，在 UMI 人类演示数据处理管线卡片上单击 **创建工作流**，填写名称并选择资源池。
3. 在编辑器中填写必填参数：`image`、`queue_id`、`gpu_queue_id`、`pfs_id`、`gpu_resource_name`、`inbox` 和新的 `batch_tag`。可单击 **查看队列信息** 核对空闲资源。
4. 保存并运行工作流。在运行详情的任务页中查看各节点状态和日志，首先在 discover 节点日志中核对识别出的 session 数量。
5. 全部节点成功后，按"输出说明"读取交付目录，并在 `index.html` 中审核语义候选。

首次接入时，建议先在收件箱中只放 1 个 session 跑通全流程，确认节点日志和交付目录后再扩大数据量。示例中 3 个 session（每个含 1 段建图视频、1 段标定视频和 5 段演示视频）从创建运行到全部完成约 27 分钟，其中 S3 slam 约 7 分钟，S6 semantic 约 10 分钟。

失败处理

某个节点失败时，可在节点日志中查看出错的 session 或视频。每个步骤的逐行处理结果写在 `runs/<batch_tag>/<步骤>/<attempt_id>/rows.jsonl`，可据此定位失败行。

〔待确认：研发说明使用相同 `batch_tag` 重新运行时，会复用已完成且校验通过的产物，只重做失败部分；输入、镜像或配置发生变化时会拒绝运行。该机制尚未经过失败恢复测试，确认后补充操作步骤。〕

使用限制

- **采集设备**：仅验证了 GoPro HERO9、2.7K Max Lens 模式、单相机单夹爪的数据。多相机、双手和其他相机型号需要另行验证。
- **语义结果**：只从第一路相机抽帧，结果为候选，需人工审核。
- **物理精度**：轨迹与标定没有独立真值，质量报告中的物理指标仅供参考，不作为通过或失败的依据。
- **GPU 型号**：仅验证了 RTX 4090（24 GB）。更换卡型时，请确认单卡显存可容纳 4 bit 量化的 Qwen3-VL-32B。

修改 YAML 的注意事项

如需在 YAML 模式下修改模板，请保留以下写法：

- 节点的启动命令写在 `spec.command` 中，`Head` 和 `Worker` 下的 `command` 保持为空。
- 不要在 `envs` 中设置 `RAY_ADDRESS`，该变量由平台注入。
- `advancedSettings.SubmitterBackoffLimit` 填写不带引号的数字 0。
- 不要设置 `UMI_OVERLAY`，否则会用外部代码覆盖镜像内的代码。

AI数据处理

算子列表

文本

空白字符标准化器

简介

空白字符标准化器 - 将文本中不同种类的空白符号替换成标准空格

功能描述

- 空白字符识别：自动识别各种Unicode空白字符

- 标准化处理：将所有空白字符替换为标准空格

算子参数

输入

输入	含义
text	待处理的文本列，要求元素类型为字符串

输出

输出	含义
normalized_text	标准化后的文本列

调用示例

```
  from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.whitespace_normalizer import WhitespaceNormalizer

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "Hello\u00a0World",
            "中文\u3000全角空格\u3000测试",
            "Tab\t分隔\t文本",
            "多　个　普　通　空　格",
        ]
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "url_ratio",
        aihc_udf(
            UrlRatioCalculator,
            construct_args={},
        )(col("text")),
    )
    ds.show()
```

中文简繁体转换器

简介

基于OpenCC的中文简繁体转换器

功能描述

- **多方向转换**：支持简繁体、台湾正体、香港繁体等多种转换方向
- **混合文本处理**：正确处理中英文混杂内容，仅转换中文部分
- **高效批处理**：支持大批量文本的快速转换处理

算子参数

输入

输入	含义
texts	包含待转换文本的列，元素类型为字符串。

输出

输出	含义
converted_text	转换后的文本列，元素类型为字符串。

参数

参数名称	类型	默认值	描述
direction	str	t2s	转换方向 可选值：["t2s", "s2t", "t2tw", "s2tw", "t2hk", "s2hk", "tw2s", "hk2s"] 默认值："t2s" (繁体转简体)

调用示例

```
 from __future__ import annotations

import os
import daft
import pandas as pd
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.chinese_text_converter import ChineseTextConverter

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "這是一段純繁體中文測試句子，歡迎使用軟體開發工具。",
            "使用 Python 進行資料處理，pip install 安裝套件非常方便。",
            "这是简体，這是繁體，and this is English，三种文字混合测试。",
            None,
        ]
    }

    df = pd.DataFrame(samples)
    ds = daft.from_pandas(df)

    ds = ds.with_column(
        "converted_text",
        aihc_udf(
            ChineseTextConverter,
            construct_args={
                "direction": "t2s",
            },
            num_gpus=0,
            num_cpus=1,
            batch_size=1,
            concurrency=1,
        )(col("text")),
    )

    ds.show()
```

BOS 预签名URL生成器

HTML文本净化器

简介

HTML文本净化器: 去除文本中的 HTML 标签，只提取纯文本内容

功能描述

- 标题提取：自动识别`<h1>`-`<h6>`标签
- 正文抽取：智能识别文章主体内容
- 冗余过滤：移除`<script>`/`<style>`等非文本标签

算子参数

输入

输入	含义
texts	包含HTML内容的文本数组，元素类型为字符串

输出

输出	含义
cleaned_text	清理后的文本数组，元素类型为字符串

参数

参数	类型	默认值	含义
separator	str	\n	分隔符
strip	bool	True	是否去除空白

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.clean_html_tag import CleanHtmlTag

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            """<!-- 这是 HTML 注释 -->
    <!DOCTYPE html>
    <html>
    <head>
        <!-- CSS -->
        <link rel="stylesheet" href="styles.css">
    </head>
    <body>
        <!--
            多行注释#1
            多行注释#2
            多行注释#3
        -->
        <div class="content" id="main-content">
            <p class="text">Hello World!</p>
        </div>
    </body>
    </html>""",
            None,
        ]
    }

    separator = "\n"
    strip = True
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "cleaned_text",
        aihc_udf(
            CleanHtmlTag,
            construct_args={"separator": separator, "strip": strip},
        )(col("text")),
    )
    ds.show()
```

MD5哈希值计算

简介

MD5哈希值计算器

功能描述

- 针对每条文本数据生成对应的MD5哈希值
- 输出固定长度（32位小写十六进制）指纹
- 支持批量处理

算子参数

输入

输入列名	说明
text	待处理的文本列，要求元素类型为字符串

输出

输出	含义	示例
md5_hash	计算后的哈希值	

调用示例

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.md5_calculator import Md5Calculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            'Hello, World!',
            "Hello, World !",
            'Hello,World!',
        ]
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "md5_hash",
        aihc_udf(Md5Calculator)(col("text")),
    )

    ds.show()
```

URL占比计算器

简介

URL占比计算器 - 基于URL字符占比的文本特征提取

功能描述

- URL占比计算：精确统计URL字符在文本中的占比
- 多协议支持：支持HTTP、HTTPS等多种URL协议
- 智能识别：使用正则表达式精确识别URL格式

算子参数

输入

输入	含义
text	待处理的文本列，要求元素类型为字符串

输出

输出	含义
url_ratio	占比结果列，元素为浮点数，表示URL字符的占比

调用示例

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.url_ratio_calculator import UrlRatioCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "今天天气很好",
            "请访问 https://baidu.com 获取更多信息",
            "官网 http://example.com 和镜像站 https://mirror.example.org 均可访问",
            "这段文字中没有任何链接",
            "联系我们：https://contact.us 或发邮件至 support@test.com",
            "http://a.com https://b.com https://c.com 三个链接",
            "",
        ]
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "url_ratio",
        aihc_udf(
            UrlRatioCalculator,
            construct_args={},
        )(col("text")),
    )
    ds.show()
```

最大英文单词长度计算器

最大英文单词长度计算器

简介 最大英文单词长度计算器，统计文本中最长英文单词的字符数，用于识别包含异常长字符串（URL、乱码等）的低质量文本。

功能描述

- 提取文本中所有英文单词，返回最长单词的字符数
- 空文本、无英文单词或 None 返回 0 或 None
- 无外部依赖

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |max_word_length|最长英文单词的字符数（float64），空文本返回 None|

参数 无

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.maximum_word_length_calculator import MaximumWordLengthCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["hello world", "aVeryLongWordThatShouldBeFiltered text", "normal sentence here"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "max_word_length",
        aihc_udf(
            MaximumWordLengthCalculator,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    # 过滤最大单词长度异常的文本
    ds = ds.where(col("max_word_length") <= 30)
    ds.show()
```

特殊字符占比计算器

特殊字符占比计算器

简介 特殊字符占比计算器，统计文本中特殊字符（非字母、数字、空格的字符）占总字符数的比例，用于过滤符号密集型低质量文本。

功能描述

- 统计文本中特殊字符（标点、符号等）占总字符数的比例
- 空文本或 None 返回 None
- 无外部依赖

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |special_ratio|特殊字符占比（float64），范围 [0, 1]，空文本返回 None|

参数 无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.special_characters_ratio_calculator import SpecialCharactersRatioCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["Normal text here.", "!!!@@@###$$$", "Mix of text! And symbols@"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "special_ratio",
        aihc_udf(
            SpecialCharactersRatioCalculator,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds = ds.where(col("special_ratio") < 0.3)
    ds.show()
```

CommonCrawl_WARC文件内容提取

CommonCrawl WARC文件内容提取

简介 CommonCrawl WARC 文件内容提取算子，从 CommonCrawl 格式的 WARC 文件中批量提取正文文本，支持 trafilatura 和 justext 两种提取引擎。

功能描述

- 解析 WARC 格式文件，批量提取网页正文
- 支持 trafilatura（推荐）和 justext 两种正文提取引擎
- 支持 WARC 文件的本地路径和远程路径输入
- 支持二进制 WARC 数据输入
- 可通过 max_records 限制处理记录数
- 依赖 trafilatura（`pip install trafilatura`）

算子参数 输入 |输入|含义| |-|-| |warc_col|WARC 文件路径或二进制数据数组，类型由 warc_src_type 决定|

输出 |输出|含义| |-|-| |extracted_texts|从 WARC 文件中提取的正文文本列表（list[string]），每个元素对应一条 WARC 记录的正文|

参数 |参数名称|类型|默认值|描述| |-|-|-| |warc_src_type|str|（必填）|WARC 数据来源类型：warc_path（本地/远程文件路径）、warc_binary（二进制数据）| |extractor_type|str|"trafilatura"|正文提取引擎：trafilatura 或 justext| |max_records|int 或 None|None|每个 WARC 文件最多处理的记录数，None 表示处理全部|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.commoncrawl_content_extractor import CommonCrawlContentExtractor

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "warc_path": [
            "/tmp/CC-MAIN-2024-01.warc.gz",
            "/tmp/CC-MAIN-2024-02.warc.gz",
        ]
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "warc_src_type": "warc_path",
        "extractor_type": "trafilatura",
        "max_records": 1000,
    }
    ds = ds.with_column(
        "texts",
        aihc_udf(
            CommonCrawlContentExtractor,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=4,
        )(col("warc_path")),
    )
    ds.show()
```

字符占比计算器

字符占比计算器

简介 字符占比计算器，计算文本中字母字符数与总字符数（或 token 数）的比值，用于过滤非自然语言文本。

功能描述

- 默认模式：计算字母字符数 / 总字符数
- tokenization 模式：计算字母字符数 / token 数（需加载分词器）
- 空文本或 None 返回 None
- 可选择是否启用 tokenization 模式（需提供模型路径）

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |alphanumeric_ratio|字符占比（float64），范围通常为 [0, 1]（tokenization 模式下可能 > 1）|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |tokenization|bool|False|是否启用 tokenization 模式（字母数 / token 数），需配置 model_path/model_name| |model_path|str|"/opt/aihc/models"|tokenization 模式下使用的分词器模型根目录| |model_name|str|"pythia-6.9b-deduped"|分词器模型名称|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.alphanumeric_ratio_calculator import AlphanumericRatioCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["Hello World 123", "!!###@@@", "正常中文文本", "abc def ghi"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "alpha_ratio",
        aihc_udf(
            AlphanumericRatioCalculator,
            construct_args={"tokenization": False},
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds = ds.where(col("alpha_ratio") > 0.5)
    ds.show()
```

版权声明移除

版权声明移除

简介

版权声明移除算子，从文本中识别并移除版权声明（Copyright、©、All Rights Reserved 等），用于清洗训练数据中的版权信息。

功能描述

- 识别并移除常见的版权声明模式（英文 Copyright、中文版权声明等）
- 支持多行版权声明的识别与清除
- 空文本或 None 返回 None

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
cleaned_text	移除版权声明后的文本（large_string），空文本返回 None

参数

无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.copyright_cleaner import CopyrightCleaner

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "Some article content.\nCopyright © 2024 Example Corp. All Rights Reserved.",
            "Normal text without copyright.",
            "版权所有 © 2023 某公司\n文章内容在这里。",
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "cleaned_text",
        aihc_udf(
            CopyrightCleaner,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds.show()
```

特定字符替换

特定字符替换

简介 特定字符替换算子，基于正则表达式对文本进行批量字符串替换，支持多组 pattern/replacement 规则。

功能描述

- 支持多个正则表达式模式的顺序替换
- 支持 Python regex 语法（包括 Unicode 属性等扩展语法）
- patterns 和 replacements 一一对应；replacements 为单值时对所有 pattern 使用同一替换字符串
- 空文本或 None 返回 None

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |replaced_text|替换后的文本（large_string），失败时返回 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |patterns|list[str]|（必填）|正则表达式模式列表，支持 Python regex 语法| |replacements|list[str]|（必填）|替换字符串列表，与 patterns 一一对应；若只有 1 个则对所有 pattern 使用相同替换|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.regex_replacement import RegexReplacer

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["Hello   World!!!", "特殊字符：@#$%^&*()", "Normal text."]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        # 将多个空格替换为单个空格，将非字母数字中文字符替换为空
        "patterns": [r"\s+", r"[^\w\u4e00-\u9fff\s]"],
        "replacements": [" ", ""],
    }
    ds = ds.with_column(
        "replaced_text",
        aihc_udf(
            RegexReplacer,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds.show()
```

重复行计算器

重复行计算器

简介 重复行计算器，计算文本中重复行占总行数的比例，用于检测低质量重复内容。

功能描述

- 将文本按换行符分割为行，统计重复行（完全相同）的占比
- 空文本或 None 返回 None
- 无外部依赖

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |repeated_lines_ratio|重复行占比（float64），范围 [0, 1]，空文本返回 None|

参数 无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.repeated_lines_calculator import RepeatedLinesCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["line1\nline1\nline2\nline3", "abc\nabc\nabc", "unique line"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "repeated_ratio",
        aihc_udf(
            RepeatedLinesCalculator,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds.show()
```

项目符号行占比计算器

项目符号行占比计算器

简介 项目符号行占比计算器，统计文本中以项目符号（如 `-`、`*`、`·`、`•` 等）开头的行占总行数的比例，用于识别结构性列表文本。

功能描述

- 识别以常见项目符号开头的行（`-`、`*`、`·`、`•`、`▪` 等）
- 计算项目符号行在所有非空行中的占比

- 空文本或 None 返回 None

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |bullet_ratio|项目符号行占比（float64），范围 [0, 1]，空文本返回 None|

参数 无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.bullet_line_ratio_calculator import BulletLineRatioCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["- item1\n- item2\n- item3", "normal paragraph text", "* a\n* b\nsome text"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "bullet_ratio",
        aihc_udf(
            BulletLineRatioCalculator,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds.show()
```

文本链接移除

文本链接移除

简介 文本链接移除算子，从文本中识别并删除超链接（HTTP/HTTPS URL），用于清洗训练数据中的噪声链接。

功能描述

- 使用正则表达式识别文本中的 HTTP/HTTPS URL
- 将 URL 替换为空字符串或空格
- 空文本或 None 返回 None

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |cleaned_text|移除链接后的文本（large_string），空文本返回 None|

参数 无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.remove_links import RemoveLinks

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "Visit https://example.com for more info.",
            "Check http://foo.bar/path?q=1 and https://baz.com",
            "No links here.",
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "cleaned_text",
        aihc_udf(
            RemoveLinks,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds.show()
```

Email地址清理

Email 地址清理

简介 Email 地址清理算子，从文本中识别并移除电子邮件地址，用于数据脱敏和文本净化。

功能描述

- 使用正则表达式识别文本中的电子邮件地址
- 将识别到的邮件地址替换为空字符串或指定占位符
- 空文本或 None 返回 None

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |cleaned_text|移除邮件地址后的文本（large_string），空文本返回 None|

参数 无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.clean_email import CleanEmail

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["Contact us at admin@example.com for help.", "No email here.", "Send to user@test.org or admin@foo.bar"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "cleaned_text",
        aihc_udf(
            CleanEmail,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds.show()
```

文本chunk切分

文本 chunk 切分

简介 文本 chunk 切分算子，基于 llama-index SentenceSplitter 将长文本按语义边界切分为固定大小的文本块，适用于 RAG 检索增强生成的文档预处理场景。

功能描述

- 基于句子边界进行切分，保证语义完整性
- 支持 text、md（Markdown）、html 三种内容类型的预处理
- 可配置 chunk_size（目标块大小，token 数）和 chunk_overlap（块间重叠 token 数）
- 依赖 llama-index-core（`pip install llama-index-core`）

算子参数 输入 |输入|含义| |-|-| |texts|文本字符串数组|

输出 |输出|含义| |-|-| |chunks|切分后的文本块列表（list[string]），失败时返回空列表|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |content_type|str|"text"|输入内容类型：text（纯文本）、md（Markdown）、html（HTML）| |chunk_size|int|500|每个 chunk 的最大 token 数| |chunk_overlap|int|50|相邻 chunk 之间的重叠 token 数，保证上下文连续性|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.chunk_text_sentence_splitter import ChunkTextSentenceSplitter

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "This is a long article. " * 100,
            "# Title\n\nSection 1 content here.\n\n## Sub-section\n\nMore content." * 20,
        ]
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "content_type": "text",
        "chunk_size": 500,
        "chunk_overlap": 50,
    }
    ds = ds.with_column(
        "chunks",
        aihc_udf(
            ChunkTextSentenceSplitter,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=32,
        )(col("text")),
    )
    ds.show()
```

文本长度计算器

文本长度计算器

简介 文本长度计算器，计算文本字符串的字符数（字节/Unicode 字符数），用于数据质量过滤。

功能描述

- 计算文本的 Unicode 字符总数
- 空字符串或 None 输入返回 None
- 无外部依赖，处理速度极快

算子参数 输入 |输入|含义| |-|-| |text|文本字符串数组|

输出 |输出|含义| |-|-| |text_length|文本字符数（int64），空文本返回 None|

参数 无

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.text_length_calculator import TextLengthCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"text": ["Hello World", "你好世界", "", None, "a" * 1000]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "text_length",
        aihc_udf(
            TextLengthCalculator,
            num_cpus=1,
            concurrency=4,
            batch_size=1024,
        )(col("text")),
    )
    ds = ds.where(col("text_length") > 50)
    ds.show()
```

句相似度语义切分

简介

句相似度语义切分 - 基于句子相似度的智能文本切分解决方案

功能描述

- 语义分块策略
- 基于句子相似度的智能切分
- 结合语义与语法规则
- 重叠优化保持上下文连贯性
- 支持中英文混合文本
- 支持中英文字符和标点
- 智能识别句子边界
- 分块引擎
- `LlamaIndex`：使用语义分割节点解析器
- `HuggingFace`：使用预训练嵌入模型计算句子相似度
- 分块算法
- 基于句子相似度的断点检测
- 支持自定义断点阈值

算子参数 输入 |输入|含义| |-|-| |texts|待处理的文本数组，要求元素类型为字符串。|

输出 |输出|含义| |-|-| |result|pyarrow.Array: 切分后的文本块，元素为List[str]类型。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |chunk_size|int|500|chunk长度 描述：chunk的长度，单位为字符 默认值：500| |chunk_overlap|int|50|chunk重叠长度 描述：切分文本时chunk之间重叠的最大长度 默认值：50| |model_path|str|'/opt/aihc/models'|模型路径 描述：嵌入模型的根目录路径 默认值："/opt/aihc/models"| |embedding_model_name|str|'BAAI/bge-m3'|嵌入模型名称 描述：用于计算句子相似度的嵌入模型名称 默认值："BAAI/bge-m3"| |breakpoint_percentile_threshold|int|80|断点百分位阈值 描述：确定语义断点的百分位阈值，值越高，切分越保守 默认值：80|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.chunk_text_sentence_similarity import ChunkTextSentenceSimilarity

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "chunk_size": 500,
        "chunk_overlap": 50,
        "model_path": '/opt/aihc/models',
        "embedding_model_name": 'BAAI/bge-m3',
        "breakpoint_percentile_threshold": 80,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ChunkTextSentenceSimilarity,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

文本安全评分

简介 文本安全性评分器 - 基于ShieldLM-6B-chatglm3的安全性评估

功能描述

- 多语言支持**：支持中文和英文文本安全性评估
- 三分类评估**：输出safe、unsafe、controversial三类概率
- 批量处理**：支持批量文本安全性评估，提升处理效率
- 模型核心**：基于ShieldLM-6B-chatglm3
- 推理优化**：支持GPU加速和批量推理
- 内容安全审核
- 文本风险评估
- 多语言安全过滤

算子参数 输入 |输入|含义| |-|-| |texts|pyarrow.Array，元素类型为str|

输出 |输出|含义| |-|-| |safe|Float64，模型预测文本为“安全”的概率| |unsafe|Float64，模型预测文本为“不安全”的概率| |controversial|Float64，模型预测文本为“有争议”的概率|

参数 |参数名称|类型|默认值|描述| |-|-|-| |lang|str|'zh'|语种 描述：需要评估的文本的语种 可选值：["en", "zh"] 默认值："zh"| |model_path|str|'/opt/aihc/models'|模型文件所在的基础路径 默认值："/opt/aihc/models"| |model_name|str|'thu-coai/ShieldLM-6B-chatglm3'|模型名称 默认值："thu-coai/ShieldLM-6B-chatglm3"| |batch_size|int|1|批处理大小 描述：控制模型推理时的批量处理大小 影响：较大的batch_size可以提高GPU利用率和吞吐量，但会增加显存占用 建议：ShieldLM-6B模型显存占用较大，默认设置为1以确保稳定性和兼容性 调优：在显存充足的环境下可适当增加到2-4以提升处理效率 默认值：1| |rank|int|None|GPU编号 描述：指定使用的GPU编号，None表示自动选择 默认值：None|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.text_safety_scorer import TextSafetyScorer

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "lang": 'zh',
        "model_path": '/opt/aihc/models',
        "model_name": 'thu-coai/ShieldLM-6B-chatglm3',
        "batch_size": 1,
        "rank": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            TextSafetyScorer,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

词汇重复率计算

简介 词重复比例计算器 - 基于N-gram词组重复比例的文本特征提取

功能描述

- 词重复比例计算：精确统计文本中重复词组的比例
- 双语言支持：支持中文、英文的分词处理
- 灵活配置：支持不同长度的N-gram词组计算
- 文本质量评估
- 数据清洗和预处理
- 内容重复检测
- 基于空格分词：适用于英文等空格分隔的语言
- 基于模型分词：使用sentencepiece模型，中文须使用此模式
- 可配置N-gram长度：支持1-N个词的组合
- 智能过滤：自动过滤特殊字符，提高计算准确性

算子参数 输入 |输入|含义| |-|-| |texts|待处理的文本列，要求元素类型为字符串|

输出 |输出|含义| |-|-| |result|pa.Array: 词重复比例列，元素为浮点数，表示重复词组比例|

参数 |参数名称|类型|默认值|描述| |-|-|-| |repetition|int|5|词组长度 描述：用于拼接成词组的单词数量 默认值：5| |lang|str|'zh'|语种 描述：文本的语种，支持zh(中文)、en(英文) 默认值："zh"| |tokenization|bool|True|是否使用分词器 描述：是否使用sentencepiece模型进行分词，中文须开启 默认值：True| |model_path|str|'/opt/aihc/models'|模型文件所在的路径 默认值："/opt/aihc/models"| |model_name|str|'kenlm/wikipedia'|模型名称 默认值："kenlm/wikipedia"|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.word_repetition_calculator import WordRepetitionCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "repetition": 5,
        "lang": 'zh',
        "tokenization": True,
        "model_path": '/opt/aihc/models',
        "model_name": 'kenlm/wikipedia',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            WordRepetitionCalculator,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

英文文本质量评分

简介 英文文本质量评分算子 - 基于FastText的文本质量评估

功能描述

- 质量评分**：使用FastText模型对英文文本质量进行评分，偏好于科学知识，只支持CPU环境。
- 批量处理**：支持批量处理文本，提高处理效率
- 0: 低质量 (Low)
- 1: 中等质量 (Mid)
- 2: 高质量 (High)
- 最终得分为0-2之间的浮点数，分数越高表示质量越好
- 一般来讲，分数超过0.5，则表示文本质量较好。

算子参数 输入 |输入|含义| |-|-| |texts|包含待处理文本的列，元素类型为字符串。|

输出 |输出|含义| |-|-| |result|pyarrow.Array: 包含文本质量分数的列，元素类型为float64。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|模型文件所在的基础路径 默认值："/opt/aihc/models"| |model_name|str|'llm-data-textbook-quality-fasttext-classifier-v2/model_quantized.bin'|模型文件名 默认值："llm-data-textbook-quality-fasttext-classifier-v2/model_quantized.bin"|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.en_text_quality_scorer import EnTextQualityScorer

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'llm-data-textbook-quality-fasttext-classifier-v2/model_quantized.bin',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            EnTextQualityScorer,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

文本困惑度计算

简介 困惑度计算算子 - 基于语言模型的文本质量评估解决方案

功能描述

- 语言模型评估
- 基于 KenLM 语言模型计算文本困惑度
- 支持中英文文本质量评估
- 提供文本可读性指标
- 质量评估
- 困惑度越低，文本质量越高
- 适用于文本质量筛选和评估
- 模型核心
- KenLM: 高效的语言模型推理
- SentencePiece: 中英文分词处理
- 计算优化
- 批量处理提升效率
- 内存友好的模型加载

算子参数 输入 |输入|含义| |-|-| |texts|待处理的文本列，要求元素类型为字符串。|

输出 |输出|含义| |-|-| |result|pyarrow.Array: 困惑度值列，元素为浮点数类型。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |lang|str|'zh'|语种 描述：需要计算的文本的语种 可选值：["en", "zh"] 默认值："zh"| |model_path|str|'/opt/aihc/models'|模型文件所在的路径 默认值："/opt/aihc/models"| |model_name|str|'kenlm/wikipedia'|模型名称 默认值："kenlm/wikipedia"| |rank|int|0|GPU 编号 描述：GPU 编号，用于模型加载 默认值：0|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.perplexity_calculator import PerplexityCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "lang": 'zh',
        "model_path": '/opt/aihc/models',
        "model_name": 'kenlm/wikipedia',
        "rank": 0,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            PerplexityCalculator,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

多语言文本翻译

简介 Seed-X 多语言文本翻译模型 - 跨语言文本翻译 核心功能

功能描述

- 多语言智能翻译
- 支持多种语言间的文本转换，可通过source_language和target_language参数自定义源语言和目标语言
- 基于Seed-X-Instruct-7B/Seed-X-PPO-7B模型，提供高质量翻译结果
- 支持语种请参考: https://huggingface.co/ByteDance-Seed/Seed-X-Instruct-7B
- 一般场景中，推荐使用Seed-X-PPO-7B模型，其在翻译质量上效果更佳
- 灵活配置与优化
- 支持多种计算精度选择（bfloat16等），适配不同性能需求
- 集成张量并行处理和前缀缓存技术，显著提升推理效率
- 支持自动或手动设备分配，完美适配单卡/多卡环境
- 资源使用
- 请使用24G以上显存的GPU
- 推荐使用NVIDIA L20、A10、A800、A100等高性能GPU以获得最佳推理性能
- 广泛适用于跨语言内容转换、多语言文档处理、国际化应用开发等场景
- 内置批处理机制，高效支持大规模文本数据并行翻译
- 支持精确控制最大生成 tokens 数量，满足多样化业务需求

算子参数 输入 |输入|含义| |-|-| |contents|包含待翻译文本的数组，元素类型为字符串。|

输出 |输出|含义| |-|-| |result|处理后的数组，元素为每个文本的翻译结果。对于处理失败的文本，返回空字符串。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|本地模型文件存储的绝对路径，默认为容器内预置路径。当使用自定义模型时需修改此路径 默认值："/opt/aihc/models"| |model_name|str|'Seed-X-PPO-7B'|支持的多语言模型名称，当前支持Seed-X-Instruct-7B/Seed-X-PPO-7B系列模型 可选值：["Seed-X-Instruct-7B", "Seed-X-PPO-7B"] 默认值："Seed-X-PPO-7B"| |dtype|str|'bfloat16'|模型推理精度选择 默认值："bfloat16"| |max_model_len|int|32768|模型支持的最大序列长度 默认值：32768| |max_num_seqs|int|128|模型同时处理的最大序列数量 默认值：128| |tensor_parallel_size|int|1|张量并行计算的设备数量，用于多GPU并行推理 默认值：1| |enable_prefix_caching|bool|True|是否启用前缀缓存机制，可提升重复前缀的推理效率 默认值：True| |gpu_memory_utilization|float|0.9|GPU内存使用比例，范围0-1 默认值：0.9| |use_cot|bool|False|是否使用思维链(Chain-of-Thought)模式进行翻译 默认值：False| |source_language|str|'Chinese'|源语言名称，支持的语言请参考模型文档 默认值："Chinese"| |target_language|str|'English'|目标语言名称，支持的语言请参考模型文档 默认值："English"| |max_tokens|int|1024|模型生成翻译结果的最大token数 默认值：1024| |batch_size|int|4|单次推理处理的文本样本数量 默认值：4| |seed|int|42|随机数种子，用于结果复现 默认值：42|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.multilingual_text_translate import MultilingualTextTranslate

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"contents": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'Seed-X-PPO-7B',
        "dtype": 'bfloat16',
        "max_model_len": 32768,
        "max_num_seqs": 128,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            MultilingualTextTranslate,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("contents")),
    )
    ds.show()
```

文本 sparse & dense embedding

简介 基于 BGE-M3 的文本嵌入模型，支持稀疏/稠密/token 三级向量生成

功能描述

- 稀疏向量：词项权重表示，适合关键词检索
- 稠密向量：1024维语义表示，适合语义相似度计算
- Token向量：细粒度上下文表征
- 硬件加速**：支持 `FP16` 量化与 GPU 并行计算

算子参数 输入 |输入|含义| |-|-| |texts|包含待处理文本的数组，元素类型为str。|

输出 |输出|含义| |-|-| |dense_embedding|稠密嵌入向量| |sparse_embedding|稀疏嵌入向量| |token_embedding|可选的token级嵌向量|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |is_output_token_vec|bool|False|是否输出 token 向量 默认值：False| |dtype|str|'float32'|模型精度，支持 float32 和 float16 可选值：["float32", "float16"] 默认值："float32"| |batch_size|int|512|模型推理时的批处理大小 默认值：512| |model_path|str|'/opt/aihc/models'|模型文件所在的路径 默认值："/opt/aihc/models"| |model_name|str|'BAAI/bge-m3'|模型名称 可选值：["BAAI/bge-m3"] 默认值："BAAI/bge-m3"| |rank|int|None|GPU 编号 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.embedding.bge_sparse_dense_embedding import BgeSparseDenseEmbedding

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "is_output_token_vec": False,
        "dtype": 'float32',
        "batch_size": 512,
        "model_path": '/opt/aihc/models',
        "model_name": 'BAAI/bge-m3',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            BgeSparseDenseEmbedding,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

文本 embedding(BEG模型)

简介 基于 BGE 系列的文本嵌入模型，支持稠密向量生成

功能描述

- 支持 BGE-M3、BGE-Large-zh-v1.5、BGE-Large-en-v1.5、BGE-Multilingual-Gemma2 等模型
- 每个模型有不同的输入提示，用户可以根据需要选择不同的模型
- 硬件加速**：支持 `FP16` 量化与 GPU 并行计算

算子参数 输入 |输入|含义| |-|-| |texts|包含待处理文本的数组，元素类型为str。|

输出 |输出|含义| |-|-| |result|处理后的数组，包含每个文本对应的稠密嵌入向量|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |dtype|str|'float32'|模型精度，支持 float32 和 float16 可选值：["float32", "float16"] 默认值："float32"| |batch_size|int|512|模型推理时的批处理大小 默认值：512| |model_path|str|'/opt/aihc/models'|模型文件所在的路径 默认值："/opt/aihc/models"| |model_name|str|'BAAI/bge-m3'|模型名称 可选值：["BAAI/bge-m3", "BAAI/bge-large-zh-v1.5", "BAAI/bge-large-en-v1.5", "BAAI/bge-multilingual-gemma2"] 默认值："BAAI/bge-m3"| |rank|int|None|GPU 编号 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.embedding.bge_embedding import BgeEmbedding

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "dtype": 'float32',
        "batch_size": 512,
        "model_path": '/opt/aihc/models',
        "model_name": 'BAAI/bge-m3',
        "rank": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            BgeEmbedding,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

语言识别

简介 文本语种识别算子 - 基于FastText模型提供多语言识别能力

功能描述

- 支持识别176种语言（基于fasttext/lid.176系列模型）
- 批量推理优化，适合处理大规模文本数据
- 支持同时输出语种标签及置信度分数
- 支持本地路径和BOS路径的模型文件
- 多语言文本分类
- 语种过滤和筛选
- 多语言数据集构建
- 输入文本长度建议不少于10个字符以提高识别准确率
- 需要至少4GB内存以进行模型批式推理

算子参数 输入 |输入|含义| |-|-| |texts|原始字符串列，要求元素类型为字符串|

输出 |输出|含义| |-|-| |result|pyarrow.Array: 包含语种标签和置信度分数的结构体列 每个元素包含 language 和 confidence 字段|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|模型文件所在的路径 默认值："/opt/aihc/models"| |model_name|str|'fasttext/lid.176.bin'|模型文件名，支持 "fasttext/lid.176.bin" 或 "fasttext/lid.176.ftz"| |batch_size|int|1000|批量处理大小，较大的batch_size可提升吐吐但增加内存消耗|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.language_recognition import LanguageRecognitionOperator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'fasttext/lid.176.bin',
        "batch_size": 1000,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            LanguageRecognitionOperator,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("texts")),
    )
    ds.show()
```

LaTeX 图表上下文提取

简介 LatexFigureContextExtractor 从 LaTeX 源码中抽取图表的 caption、label、图片路径及其引用段落。

功能描述 • 解析 figure / wrapfigure 环境、子图（subfigure/subfloat）、minipage 等结构，抽取 caption、label 与 \includegraphics 图片路径。

• 按 citation_commands 匹配引用命令（如 \ref / \cref），回溯定位引用该图 label 的段落作为上下文。

• 一行 LaTeX 源码可 fan-out 出多张图，聚合为一行的图 struct 列表返回（列式 1 进 1 出），无图样本返回空列表。

算子参数 输入 |输入|含义| |-|-| |text|完整的 LaTeX 源码字符串|

输出 |输出|含义| |-|-| |result|类型 list<struct{images: list<large_string>, caption, label, citing_paragraphs: list<large_string>, parent_caption, parent_label}>；每个元素对应一张图及其引用上下文，无图返回空列表|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |citation_commands|list |None|None|引用命令列表；None 时默认 [r"\ref", r"\cref", r"\Cref", r"\autoref"]，用于匹配引用图 label 的段落| |paragraph_separator|str|"\n\n"|段落分隔符，用于将去环境后的正文切分为段落||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.latex_figure_context_extractor import LatexFigureContextExtractor

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    latex = r"""
    As shown in Figure~\ref{fig:demo}, the result is good.
    \begin{figure}
        \includegraphics{demo.png}
        \caption{A demo figure.}
        \label{fig:demo}
    \end{figure}
    """
    samples = {"text": [latex]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "figures",
        aihc_udf(
            LatexFigureContextExtractor,
            construct_args={},
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("text")),
    )
    ds.show()
```

文本分块

简介 TextChunk 按最大长度或分隔模式将文本切分为多个块，纯 regex（按字符长度）实现，可选 HuggingFace tokenizer（按 token 长度）。

功能描述 • 仅给 split_pattern：按 pattern 切分并过滤空段；仅给 max_len：按长度切分，块间可 overlap_len 重叠。

• 两者都给时递归切分：超过 max_len 时在 pattern 处强制切断。

• 可选传入 tokenizer 名称，则按 token 数而非字符数计长，切分后再解码回文本。

算子参数 输入 |输入|含义| |-|-| |text|待切分的文本字符串|

输出 |输出|含义| |-|-| |result|类型 list<large_string>，即该条文本切出的块列表；输入为 None 时返回 None，出错时返回 [原文]|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |max_len|int |None|None|每块最大长度（tokenizer 为空时按字符数，否则按 token 数）；与 split_pattern 不能同时为 None| |split_pattern|str |None|r"\n\n"|分隔正则模式；与 max_len 不能同时为 None| |overlap_len|int|0|块间重叠长度，必须小于 max_len|| |tokenizer|str |None|None|可选 HuggingFace tokenizer 名称/路径；提供后按 token 计长，惰性加载| |trust_remote_code|bool|False|加载 tokenizer 时是否信任远程代码||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.text_chunk import TextChunk

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "text": [
            "First paragraph.\n\nSecond paragraph.\n\nThird paragraph.",
            "Single block without separator.",
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "chunks",
        aihc_udf(
            TextChunk,
            construct_args={"split_pattern": r"\n\n", "max_len": None},
            num_cpus=1,
            concurrency=1,
            batch_size=2,
        )(col("text")),
    )
    ds.show()
```

IP 地址清理

简介

IP 地址清理算子，删除文本中的 IPv4 / IPv6 地址，用于语料脱敏与净化。默认正则同时覆盖 IPv4 与 IPv6 全写形式，默认替换串为空串（即直接删除）。

功能描述

- 默认 pattern 由两段组成：前半段匹配 IPv4 点分十进制（每段按 0-255 限定），后半段匹配 IPv6 的 8 组全写形式 ([da-fA-F]{1,4}:){7}[da-fA-F]{1,4}，命中即替换成 repl
- 支持自定义 pattern；正则在算子构造阶段编译，非法正则直接抛 ValueError，不会拖到运行时才失败
- 兼容配置里常见的 r'...' / r"..." 包裹写法，构造时自动剥掉外层的 r 前缀与引号
- 正则以 re.DOTALL 预编译复用；逐行先 search，命中才 sub，未命中的行原样返回
- 输入为 None 的行返回 None；单行处理抛异常时记日志并返回 None，不中断整批，结束时打印处理行数与失败行数

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
result	删除 IP 地址后的文本（string），输入 None 返回 None

参数

参数名称	类型	默认值	描述
pattern	str	""	自定义正则；空串表示使用内置的 IPv4+IPv6 默认 pattern。支持 r'...' 包裹写法
repl	str	""	替换串，默认空串（即删除）

注意事项

- 默认 IPv6 分支只认 8 组全写形式，2001:db8::1 这类 :: 压缩写法不会被匹配；fe80:0:0:0:0:0:0:1 才会命中。

- 正则不做词边界锚定，四段数字串与 IPv4 无法区分：`ver 1.2.3.4 ok` 会被清成 `ver  ok`；`300.1.1.1` 的后缀 `0.1.1.1` 命中，结果剩 `30`。版本号密集的语料建议自定义 pattern。
- `repl` 按 `re.sub` 的替换模板语义处理，串里的 `1`、`g<1>` 会被当成反向引用（默认 IPv6 分支带一个捕获组），需要输出字面反斜杠时要自行转义。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.clean_ip import CleanIp

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "texts": [
            "server at 192.168.1.1 is down",
            "ipv6 2001:0db8:85a3:0000:0000:8a2e:0370:7334 unreachable",
            "no ip here",
            None,
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(CleanIp, construct_args={}, num_cpus=1, concurrency=1, batch_size=4)(col("texts")),
    )
    ds.show()
```

参考文献移除

简介

参考文献移除算子：识别 LaTeX 文档里的参考文献 / 附录起始标记，把标记本身及其后到文末的内容整段删除。

功能描述

- 识别 `appendix`、`begin{references}`、`begin{REFERENCES}`、`begin{thebibliography}`、`bibliography{...}` 五个标记，从标记本身开始一路删到文末（标记也删掉）
- 正则以 `re.DOTALL` 编译，尾部 `.*$` 跨行贪婪匹配，一次替换即清空到文末
- 无构造参数，行为完全固定
- 命中不到任何标记的文本原样返回；输入 None 的行返回 None；单行异常记日志并返回 None，不中断整批

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
result	删除参考文献段后的文本（string），输入 None 返回 None

参数

无（`__init__` 只接受透传给基类的 `**kwargs`）

注意事项

- `appendix` 是 LaTeX 的附录起始标记，不专指参考文献。带正文价值的附录同样会被整段删除，这是本算子的既定口径。
- 删除范围是「标记到文末」，不是「到 `end{...}`」：`intro begin{references} r end{references} more text` 的结果是 `intro`，`end{references}` 之后的正文一并丢失。
- 标记大小写只认 `references` 与 `REFERENCES` 两种拼写，`begin{References}` 不命中。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.remove_bibliography import RemoveBibliography

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

body = "\section{Intro}
real content here
"
    samples = {
        "texts": [
            body + "\begin{thebibliography}{9}
\bibitem{a} Foo
\end{thebibliography}",
            body + "\appendix
appendix stuff",
            "plain text without bibliography",
            None,
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(RemoveBibliography, construct_args={}, num_cpus=1, concurrency=1, batch_size=4)(col("texts")),
    )
    ds.show()
```

LaTeX 注释移除

简介

文档注释移除算子，删除 LaTeX 源码里的 `%` 注释：行内注释与整行注释两条规则分别可开关，转义的 `%` 不会被误删。`doc_type` 当前只实现了 `tex`，不处理 Python / C 等编程语言的注释。

功能描述

- 行内注释规则 `[^\]%.+$`（`re.MULTILINE`）：转义的 `%` 不会被误当注释删掉
- 整行注释规则 `^%.*`

`?`（`re.MULTILINE`）：连同该行的换行符一起删除

- 两条规则分别由 `inline` / `multiline` 控制，默认都开；执行顺序是先删行内注释，再删整行注释
- `doc_type` 不在 `("tex",)` 中时，构造阶段直接抛 `ValueError`
- 输入 None 的行返回 None；单行异常记日志并返回 None，不中断整批

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
result	删除注释后的文本（string），输入 None 返回 None

参数

参数名称	类型	默认值	描述
doc_type	str	"tex"	文档类型；当前只支持 tex，其他值构造时报错
inline	bool	True	是否删除行内注释
multiline	bool	True	是否删除整行注释

注意事项

- 行内注释的正则会连 `%` 前面的那一个字符一起吃掉：`code = 1 % trailing comment` 的结果是 `code = 1`（`%` 前的空格被删），`a%b` 会被整体删成空串。
- `%` 后至少要有一个字符才算行内注释，行尾孤立的 `%`（如 `x %`）不会被删。
- 行首的 `%` 不满足 `[^\]` 的前导字符要求，只能靠 `multiline=True` 删除；把 `multiline` 关掉，整行注释会保留。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.remove_comments import RemoveComments

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "texts": [
            "% full line comment
real line
code = 1 % trailing comment",
            "escaped 50\% stays here",
            None,
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            RemoveComments,
            construct_args={"doc_type": "tex", "inline": True, "multiline": True},
            num_cpus=1,
            concurrency=1,
            batch_size=4,
        )(col("texts")),
    )
    ds.show()
```

文档头部移除

简介

文档头部移除算子，删除 LaTeX 正文之前的 header（`documentclass`、宏包、`itle` 等 preamble），从第一个章节命令开始保留。

功能描述

- pattern 形如 `^(.*?)(<章节命令>)`，替换成第 2 个捕获组，即把第一个章节命令之前的内容全部删掉、命令本身保留
- 识别的章节命令：`chapter`、`part`、`section`、`subsection`、`subsubsection`，均支持 `*` 变体与可选的 `[短标题]` 参数
- `^` 未开 `re.MULTILINE`，只在字符串起点匹配，因此一篇文档只裁剪一次，后面的章节命令不受影响
- 整篇找不到章节命令时：`drop_no_head=True` 返回空串 `""`（不是 None），`drop_no_head=False` 原样返回
- 输入 None 的行返回 None；单行异常记日志并返回 None，不中断整批

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
result	删除 header 后的文本（string）；无 header 且 drop_no_head=True 时为空串，输入 None 返回 None

参数

参数名称	类型	默认值	描述
drop_no_head	bool	True	整篇没有章节命令时是否把文本清空，默认 True

注意事项

- pattern 在 `paragraph` 与 `subparagraph` 两个分支之间没有 | 分隔，这一支实际要求 `paragraph{...}subparagraph{...}` 紧邻才命中，单独出现的 `paragraph` 不会被当作 header 边界。
- `drop_no_head` 默认 True，非 LaTeX 的普通文本会被整条清成空串。混排语料务必显式传 `False`，否则会静默丢数据。
- 章节命令必须带 `{}` 参数才命中，裸写的 `section` 不算 header 边界。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.remove_header import RemoveHeader

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "texts": [
            "\documentclass{article}
\title{X}
\section{Intro}
body text",
            "no latex header at all",
            None,
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            RemoveHeader,
            construct_args={"drop_no_head": True},
            num_cpus=1,
            concurrency=1,
            batch_size=4,
        )(col("texts")),
    )
    ds.show()
```

超长单词移除

简介

超长单词移除算子，按词长过滤：删除长度不落在 [min_len, max_len] 区间内的词，切分、判定、合并三步走。

功能描述

- 按 `

→ → 空格 三级切分文本，逐词判定，再按空格 → → ` 顺序合并回文本

- 判定规则：原始长度落在 [min_len, max_len] 直接保留；只有超过 max_len 的词才去掉首尾特殊字符后复判，短于 min_len 的词直接丢弃、不做复判
- 特殊字符集取 daft.aihc.common.utils.special_characters.SPECIAL_CHARACTERS：ASCII 标点 + 数字 + 空白 + 一批全角/异体符号 +emoji 包收录的全部 emoji
- 构造阶段校验 0 <= min_len <= max_len，不满足抛 ValueError
- 输入 None 的行返回 None；单行异常记日志并返回 None，不中断整批

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
result	按词长过滤后的文本（string），输入 None 返回 None

参数

参数名称	类型	默认值	描述
min_len	int	1	词长下限，短于该值的词被删除
max_len	int	sys.maxsize（64 位平台为 9223372036854775807）	词长上限，超过该值且去掉首尾特殊字符后仍超过的词被删除；默认等于不限制

注意事项

- 合并阶段会顺带做空白归一：连续空格压成一个、行首尾空格去掉、空行（含只有空白的行）整行丢掉。即使用默认参数（不删除任何词）文本也可能变化，例如 ``a b

c'→'a b c'。 与 ` 本身的层级结构会保留。

- 词长按 Python len()（Unicode 码点数）计算，中文按字数计。一段没有空格的中文长句会被当成一个超长词整体删掉，中文语料上设 max_len 要谨慎。
- 切分只按空白，标点不单独切开：word, 是长度 5 的词；只有被判为超长时才会去掉首尾标点再复判。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.remove_long_words import RemoveLongWords

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "texts": [
            "short words stay but supercalifragilisticexpialidocious goes away",
            "line one col two
line three",
            None,
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            RemoveLongWords,
            construct_args={"min_len": 1, "max_len": 10},
            num_cpus=1,
            concurrency=1,
            batch_size=4,
        )(col("texts")),
    )
    ds.show()
```

非中文字符移除

简介

非中文字符移除算子，默认只保留中文汉字（一-龥），可通过开关额外保留英文字母、数字以及一组固定的标点白名单，其余字符全部删除。

功能描述

- 构造时按三个开关拼出否定字符类：基底 [^一-龥，keep_alphabet 追加 A-Za-z，keep_number 追加 0-9，keep_punc 追加写死的标点白名单 .，,-。%《*》/•、&&(—)(+)：?!！“”·
- 三个开关默认全 True；keep_punc=False 时 pattern 收成 [^一-龥A-Za-z0-9]，空格也一并删除
- 拼好的 pattern 保存在 self.pattern，以 re.DOTALL 预编译；逐行先 search，命中才 sub，未命中的行原样返回
- 输入 None 的行返回 None；单行异常记日志并返回 None，不中断整批

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
result	只保留中文（以及按开关保留的字母/数字/标点）后的文本（string），输入 None 返回 None

参数

参数名称	类型	默认值	描述
keep_alphabet	bool	True	是否保留英文字母 A-Za-z
keep_number	bool	True	是否保留数字 0-9
keep_punc	bool	True	是否保留白名单里那组中英文标点（含半角空格）

注意事项

- 换行与 Tab 都不在白名单里，任何开关组合下都会被删除，多行文本会被压成一行：``第一行

第二行'→'第一行第二行'`。

- 汉字范围只有基本区 一-龥，扩展区生僻字、日文假名、韩文等都会被删。
- 标点白名单是写死的一组，；、【】、「」、[] 都不在其中：'分号；括号【】' → '分号括号'。
- keep_punc=True 时白名单包含半角空格，英文分词能保住；关掉它会把英文词粘成一串。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.remove_non_chinese_character import RemoveNonChineseCharacter

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"texts": ["百舸 AIHC 算子 v0.5.3 ⚡ ok", "纯中文文本", None]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "keep_all",
        aihc_udf(
            RemoveNonChineseCharacter,
            construct_args={},
            num_cpus=1,
            concurrency=1,
            batch_size=4,
        )(col("texts")),
    )
    ds = ds.with_column(
        "chinese_only",
        aihc_udf(
            RemoveNonChineseCharacter,
            construct_args={"keep_alphabet": False, "keep_number": False, "keep_punc": False},
            num_cpus=1,
            concurrency=1,
            batch_size=4,
        )(col("texts")),
    )
    ds.show()
```

Token 数量计算

简介

文本 token 数统计算子，用 HuggingFace tokenizer 统计每条文本的 token 数量，默认 tokenizer 为`EleutherAI/pythia-6.9b-deduped`。算子只产出计数，按上下界过滤交给 pipeline 的 `where`。

功能描述

- 只加载 `model_path/model_name` 目录下的分词器，不加载模型权重，纯 CPU 即可运行
- 分词时 `add_special_tokens=False`，计数不含 BOS/EOS
- 按 `batch_size` 微批分词；某个微批整批失败时降级为逐条重试，只把失败的那一条置 None，不带走整批
- 空字符串、纯空白、非字符串（含 null）不进入分词，直接返回 None
- tokenizer 目录不存在抛 `FileNotFoundError`、加载失败抛 `RuntimeError`，都发生在算子初始化阶段，不静默降级成全 None

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
num_token	token 数量（int64）；空文本或分词失败返回 None

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"EleutherAI/pythia-6.9b-deduped"	相对 model_path 的 tokenizer 目录名
trust_remote_code	bool	False	是否允许加载 tokenizer 仓库内的自定义代码
batch_size	int	64	算子内部分词的微批大小，与 aihc_udf 的 batch_size 相互独立

注意事项

- 与存量「文本长度计算器」（`TextLengthCalculator`，按字符数计）量纲不可比，不能复用同一套阈值。
- 换 tokenizer 后同一段文本的 token 数会变，阈值需随 `model_name` 重新校准。
- 默认目录名带 `6.9b`，但只用到目录里的分词器文件（词表与 tokenizer 配置），不需要下载 6.9B 模型权重。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.token_count_calculator import TokenCountCalculator

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "text": [
            "Hello world, this is a tokenizer test.",
            "百度智能云 AIHC 数据处理算子。",
            "",
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "num_token",
        aihc_udf(
            TokenCountCalculator,
            construct_args={
                "model_path": "/path/to/models",
                "model_name": "EleutherAI/pythia-6.9b-deduped",
            },
            num_cpus=1,
            concurrency=1,
            batch_size=64,
        )(col("text")),
    )
    ds.show()
```

指令跟随难度评分（IFD）

简介

指令跟随难度（Instruction Following Difficulty，论文 https://arxiv.org/abs/2308.12032）打分算子。对每条样本做两次因果 LM 前向，输出 `IFD = loss(response | query) / loss(response)`：比值越小说明 query 对生成 response 帮助越大。算子只产出分数，不按上下界过滤样本。

功能描述

- 双输入列（query 指令 + response 回答），逐条计算，每条两次前向：带 query 前缀的条件 loss 与不带前缀的无条件 loss
- loss 只在 response 段上统计（前缀位置的 label 置 -100）
- 拼接方式为 `f"{prefix} {response}".strip()`，前缀为空时直接用 `response.strip()`；两侧可分别套 `query_template` / `response_template`
- response 的 token 数按单独分词的结果扣掉 BOS 计算，必须 >0 且不超过拼接后长度，否则该行按异常处理
- 拼接后不足 2 个 token 时因果 LM 没有有效 label 位（loss 为 nan），该行返回 None；无 query 的 loss 为 0 也返回 None
- 任一列为空/纯空白/非字符串返回 None；单条异常只影响该行，其余行照算
- 输出保留 6 位小数
- tokenizer 的 `pad_token` 缺失时用 `eos_token` 顶上，`padding_side` 与 `truncation_side` 都取 left
- `aihc_udf` 的 `num_gpus` > 0 且 CUDA 可用时按 `cuda:{rank % 卡数}` 选卡，否则跑 CPU
- `dtype` 非法抛 `ValueError`，权重目录不存在抛 `FileNotFoundError`，都发生在初始化阶段

算子参数

输入

输入	含义
queries	指令/问题文本列
responses	回答文本列，IFD 的 loss 只在这一段上算

输出

输出	含义
ifd_score	float64，条件 loss 与无条件 loss 的比值；空值、response 过短或计算异常返回 None

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"Qwen/Qwen2.5-0.5B"	相对 model_path 的权重子目录
dtype	str	"float32"	权重精度，可选 float16 / float32 / bfloat16；float32 数值更稳，故为默认
max_length	int 或 None	None	tokenizer 截断长度，None 表示不截断
query_template	str	"{query}"	query 侧模板，占位符 {query}
response_template	str	"{response}"	response 侧模板，占位符 {response}
rank	int	0	多卡场景 worker 序号，实际设备取 cuda:{rank % 卡数}

注意事项

- 与存量「困惑度计算」（`PerplexityCalculator`，kenlm + sentencepiece 的语言模型困惑度）语义不同，两者分值不可互相比较，也不能复用阈值。
- 逐条两次前向，吞吐显著低于批量文本算子；长文本可用 `max_length` 截断控制耗时。
- 换 `model_name` 或 `dtype` 都会改变分值，阈值需重新校准。
- 算子不联网下载权重，权重缺失时在初始化阶段直接抛 `FileNotFoundError`。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.instruction_following_difficulty_scorer import (
    InstructionFollowingDifficultyScorer,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "query": [
            "What is the capital of France?",
            "Explain what a neural network is in one sentence.",
            "",
            "Translate to French: hello",
        ],
        "response": [
            "The capital of France is Paris.",
            "A neural network is a model made of layers of weighted connections.",
            "some response without query",
            "Bonjour",
        ],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "ifd_score",
        aihc_udf(
            InstructionFollowingDifficultyScorer,
            construct_args={
                "model_path": "/path/to/models",
                "model_name": "Qwen/Qwen2.5-0.5B",
                "dtype": "float32",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=4,
        )(col("query"), col("response")),
    )
    ds.show()
```

文本对相似度（CLIP）

简介

文本对相似度算子，两段文本各过 CLIP 文本塔后取余弦相似度。算子只产出分数，不按 min/max 过滤，过滤交给 pipeline 的 `where`。CLIP 封装复用 `daft/aihc/functions/multimodal/_clip_dual_tower.py`。

功能描述

- 复用 `ClipDualTower`（transformers `CLIPModel` / `CLIPProcessor`）：两侧文本各编码一次，向量做 L2 归一化后点积即余弦，取值 [-1, 1]，输出保留 6 位小数
- 两列按行配对，长度不一致直接抛 `ValueError`
- 只对两侧都非空的行推理；任一侧为空、纯空白或非字符串留 None
- 按 `batch_size` 微批推理；某批异常时该批整批留 None（记 exception 日志），其余批不受影响
- 文本编码按 CLIP 文本塔上限截断（truncation=True, max_length=77）
- `aihc_udf` 的 `num_gpus` > 0 且 CUDA 可用时按 `cuda:{rank % 卡数}` 选卡，否则跑 CPU；CPU 上 float16 由封装层自动回落 float32
- `dtype` 非法抛 `ValueError`，权重目录不存在抛 `FileNotFoundError`，都发生在初始化阶段

算子参数

输入

输入	含义
texts_a	文本 A 列
texts_b	文本 B 列，与 A 按行配对

输出

输出	含义
sim	float64，两段文本 CLIP 文本向量的余弦相似度；任一侧为空或该批推理失败返回 None

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"openai/clip-vit-base-patch32"	相对 model_path 的 CLIP 权重子目录
dtype	str	"float32"	权重精度，可选 float16 / float32 / bfloat16
batch_size	int	32	算子内部推理的微批大小，与 aihc_udf 的 batch_size 相互独立
rank	int	0	多卡场景 worker 序号，实际设备取 cuda:{rank % 卡数}

注意事项

- CLIP 文本塔有 77 token 上限，超长文本的尾部不参与计算。长文档比对建议先切分，或改用「文本 Embedding 相似度」算子。
- CLIP 是图文对比学习出来的模型，不是文本检索模型，分数区间与专用文本 embedding 模型（BGE、Qwen3-Embedding 等）不同，阈值不能互相套用。
- 权重需包含 `CLIPModel` 与 `CLIPProcessor` 的完整文件；`ClipDualTower` 加载整个 CLIP（含图像塔），即使本算子只用文本塔。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.text_pair_similarity import TextPairSimilarity

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict(
        {
            "a": ["a dog on grass", "a dog on grass", "a dog on grass"],
            "b": ["a dog on grass", "quantum physics lecture notes", ""],
        }
    )
    ds = ds.with_column(
        "sim",
        aihc_udf(
            TextPairSimilarity,
            construct_args={
                "model_path": "/path/to/models",
                "model_name": "openai/clip-vit-base-patch32",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=32,
        )(col("a"), col("b")),
    )
    ds.show()
```

文本 Embedding 相似度

简介

文本与参照集的平均 embedding 余弦相似度算子：把每条文本编码成向量，与一组参照文本的向量逐个算余弦后取均值，用来衡量样本与「目标风格 / 目标任务」参照集的接近程度。只产出分数，`min_score` / `max_score` 一类阈值过滤交给 pipeline 的 `where`。

功能描述

- 只走本地模型（`AutoTokenizer` + `AutoModel`），算子内不联网，不调用任何在线 embedding 接口
- 三种 pooling：`None` 取最后一个 token 的 hidden state（默认）、`"mean"` 均值、`"weighted_mean"` 按位置加权均值；left padding 时按 `attention_mask` 取真实末位
- 参照集两种形态，必须二选一：`valid_texts` 传原文（在 `__init__` 里现场编码），或 `valid_embeddings_path` 指向预先算好的 `.npy` / `.npz`（`.npz` 取 `embeddings` 键，路径支持本地 / BOS / S3 / HTTP）。都不给或都给抛 `ValueError`
- 预计算向量会校验形状为 `(n, dim)` 且 `n>0`、`dim` 等于模型 `hidden_size`，不符抛 `ValueError`；随后做 L2 归一化（零向量的模按 1 处理）
- `n_shot` 只取参照集前 n 条
- `input_template` 是单占位符模板（默认 `"{text}"`），对参照集与待测文本同样套用，可用来加 Qwen3-Embedding 推荐的 instruct 前缀

- max_length 是分词截断长度，默认 4096
- 批量编码，与逐条编码数值一致：mean / weighted_mean 只差一个正标量，pooling 后紧跟 L2 normalize 把标量约掉；last-token pooling 用 attention_mask 取真实末位下标，与 batch 无关
- 待测文本与参照集向量都已归一化，点积即余弦，对参照集取均值即输出分数
- 空文本 / 非字符串留 None；整批编码失败时自动退回逐条编码，避免一条坏数据带走整批；非有限值（nan / inf）写 None
- CPU 上非 float32 会告警并强制回落 float32；GPU 上每次 transform 结束调用 empty_cache
- 只接收单列文本，多字段拼接由上游完成

算子参数

输入

输入	含义
texts	文本字符串数组

输出

输出	含义
score	float64，与参照集各向量余弦的均值；文本为空或编码失败返回 None

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"Qwen/Qwen3-Embedding-0.6B"	相对 model_path 的 embedding 模型目录；也可换 BAAI/bge-m3 等
pooling	str 或 None	None	取向量的方式：None 取最后一个 token（默认）、"mean"、"weighted_mean"；其它值抛 ValueError
valid_texts	list[str] 或 None	None	参照集原文，适合几十条以内
valid_embeddings_path	str 或 None	None	预先算好的参照集向量文件，.npy（二维数组）或 .npz（取 embeddings 键），可为 BOS 路径
n_shot	int 或 None	None	只取参照集前 n 条；None 表示全用
input_template	str	"{text}"	单占位符模板，按 template.format(text=<行内容>) 套用
max_length	int	4096	分词截断长度
batch_size	int	16	编码批大小，与 aihc_udf 的 batch_size 相互独立
dtype	str	"float32"	模型精度：float32 / float16 / bfloat16
trust_remote_code	bool	True	透传给 transformers
rank	int	0	多卡场景 worker 序号，实际设备取 cuda:{rank % 卡数}

注意事项

- valid_texts 形态下参照集会在每个 UDF actor 冷启时各编码一遍，参照集上千条时改用 valid_embeddings_path。
- 预计算向量必须由同一个模型、同一种 pooling 算出。算子只校验维度，pooling 或模型换了但维度恰好相同时不会报错，只会算出错的分数。
- 分数是「与给定参照集的平均余弦」，参照集不同则分数不可比，阈值需跟参照集一起固定。
- 与「文本对相似度（CLIP）」的分数量纲不可比：一个是 CLIP 图文对齐空间的两两余弦，一个是文本 embedding 模型对参照集的均值。
- 算子不联网下载权重，权重缺失时在初始化阶段直接抛 FileNotFoundError。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.text.text_embd_similarity import TextEmbdSimilarity

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "text": [
            "There is a lovely cat.",
            "It is challenging to train a large language model.",
            None,
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "score",
        aihc_udf(
            TextEmbdSimilarity,
            construct_args={
                "model_path": "/path/to/models",
                "model_name": "Qwen/Qwen3-Embedding-0.6B",
                # 参照集形态 1：直接传原文
                "valid_texts": ["There is a cute cat.", "The cat there is lovely."],
                # 参照集形态 2（大参照集推荐，与形态 1 二选一）：
                # "valid_embeddings_path": "bos://your-bucket/ref/text_embd_valid.npy",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=16,
        )(col("text")),
    )
    ds.show()
```

图片

图片哈希计算处理器

简介

图片哈希计算处理器

功能描述

- 支持五种哈希方法：`ahash`、`dhash`、`phash`、`whash`、`md5`
- 输入类型可选：`image_url`、`image_base64`、`image_binary`
- 支持批量处理

算子参数

输入

输入	含义
image	包含输入图像的数组，支持URL/二进制格式

输出

输出	含义
hash_hex	图像的十六进制哈希值。对于 md5 方法，该值为 16 个字符（截取完整摘要的前 64 位）
hash_bin	图像的二进制哈希值。对于 md5 方法，该值为 64 位

参数

参数	类型	含义	默认值
image_src_type	str	输入图片的来源类型可选值：["image_url", "image_base64", "image_binary"]	"image_url"
method	str	哈希计算方法。md5 使用 hashlib.md5 计算，其它方法使用 imagededup 库。可选值：{"ahash", "dhash", "phash", "whash", "md5"}	"phash"
batch_size	int	批处理大小，用于在吞吐量和内存消耗之间进行平衡	64

调用示例

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_hash import ImageHash

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ]
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_hash",
        aihc_udf(
            ImageHash,
            construct_args={
                "image_src_type": "image_url",
                "method": "phash",
            },
            num_gpus=0,
            batch_size=1,
            concurrency=1,
        )(col("image")),
    )

    ds.show()
```

图像裁剪处理器

简介

图像裁剪处理器

功能描述

- 输入格式：
  - URL 地址（image_url）
  - BOS路径 ("bos://{bucket}/{path}")
  - Base64 编码（image_base64）
  - 二进制流（image_binary）
- 裁剪方式（支持多种算法/模式）：
  - 坐标裁剪：指定左上角/右下角坐标精准裁剪
  - 比例裁剪：按宽高比例裁剪（0-1）
  - 中心裁剪：按指定尺寸从图像中心裁剪
- 插值算法：nearest / bilinear / bicubic / lanczos（裁剪后尺寸适配时使用）
- 输出模式：
  - 返回裁剪后图片的 Base64 字符串
  - 可选将图片保存到本地目录
- 格式适配：
  - JPEG：RGBA 转 RGB + 质量压缩
  - PNG：保留透明通道 + 无损压缩
  - WebP：支持透明通道 + 质量/无损压缩

算子参数

输入

输入	含义
image	包含输入图像的数组，支持URL/base64/二进制格式
image_name	可选参数，包含图像标识名的数组，用于生成输出文件名

输出

输出	含义
base64	压缩后图像的base64编码
image_path	本地存储路径（当配置输出目录时有效）

参数

参数	类型	含义	默认值
image_suffix	str	保存到本地时使用的文件后缀，需为支持的格式。可选值: [".jpg", ".jpeg", ".png", ".webp"]	".jpg"
output_dir	str	将裁剪后的图像保存到的输出路径，为空则不保存到输出，	""
image_src_type	str	输入图像的格式类型。可选值: ["image_url", "image_base64", "image_binary"]	"image_url"
crop_type	str	裁剪方式。支持三种裁剪模式，需配合对应参数使用：coordinate: 需指定 crop_coords=[x1, y1, x2, y2]ratio: 需指定 crop_ratio=[width_ratio, height_ratio]（0-1）center: 需指定 crop_size=[width, height]（或使用默认比例）可选值: ["coordinate", "ratio", "center"]	"center"
crop_coords	list	坐标裁剪参数。格式：[x1, y1, x2, y2]，x1/y1=左上角坐标，x2/y2=右下角坐标。	None
crop_ratio	list	比例裁剪参数。格式：[width_ratio, height_ratio]，值范围 0-1（如 [0.8, 0.8] 裁剪80%区域）。	[0.8, 0.8]
crop_size	list	中心裁剪参数。格式: [width, height]，指定裁剪后的宽高，None 则按原尺寸80%裁剪。	None
quality	int	保存质量参数（适配不同格式）：JPEG/WebP：1-100（数值越高质量越好）PNG：映射为 compress_level（1→9，100→0）	85
method	str	插值算法（裁剪后尺寸适配时使用）。可选值: ["nearest", "bilinear", "bicubic", "lanczos"]	"lanczos"

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_crop import ImageCrop

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ],
        "image_name": ["sample_1.jpg", "sample_2.jpg", "sample_3.jpg"],
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_crop",
        aihc_udf(
            ImageCrop,
            construct_args={
                "image_suffix": ".jpg",
                "output_dir": "./test_images_output",
                "image_src_type": "image_url",
                "crop_type": "center",
                "crop_size": [500, 500],
                "crop_ratio": [0.8, 0.8],
                "crop_coords": [100, 100, 800, 800],
                "quality": 85,
                "method": "lanczos",
            },
            batch_size=1,
        )(col("image"), col("image_name")),
    )
    ds.show()
```

图像格式转换处理器

简介

图像格式转换处理器

功能描述

- 多种图像格式支持：
  - 输入：URL地址、BOS地址、Base64编码、二进制流
  - 输出：PNG、JPEG/JPG、WebP
- 质量调整：
  - 支持调整有损压缩格式（JPEG、WebP）的输出质量
  - 可设置质量范围为-1-100，-1表示使用默认值
- 智能格式处理：
  - 自动处理透明通道（JPEG不支持透明通道）
  - 自动处理色彩模式转换（如CMYK转RGB）
- 多种输出模式：
  - Base64编码直出

- 支持本地存储
- 支持直接存储到BOS

算子参数

输入

输入	含义
image	包含输入图像的数组，支持URL/base64/二进制格式
image_name	可选参数，包含图像标识名的数组，用于生成输出文件名

输出

输出	含义
base64	压缩后图像的base64编码
image_path	本地存储路径（当配置输出目录时有效）

参数

参数名称	类型	默认值	描述
output_format	str	"png"	输出图像的格式。描述: 输出图像的格式，支持多种常见图像格式。可选值: ["png", "jpg", "jpeg", "webp"]默认值: "png"
image_suffix	str	""	保存到本地图像后缀描述: 保存本地的图像后缀，如果不设置，将根据output_format自动生成默认值: ""
output_dir	str		保存图像的本地文件夹路径。描述: 转换后图像的本地存储目录，如果为空，则转换后的图像不保存到本地默认值: ""
bos_dir	str	""	保存图像到对象存储BOS中描述: 转换后图像的存储BOS地址, 例如"bos://{bucket}/{path}"默认值: ""
image_src_type	str	"image_url"	输入图像的格式类型，支持：url地址(image_url)base64编码(image_base64)二进制流(image_binary)描述: 输入图像的格式类型。可选值: ["image_url", "image_base64", "image_binary"]默认值: "image_url"
quality	int	-1	输出图像的质量（仅适用于有损压缩格式描述: 输出图像的质量，范围-1-100，仅适用于jpg、jpeg、webp等有损压缩格式当设置为-1时，将使用各格式的内置默认质量值（JPEG默认75，WebP默认80）默认值: -1

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_format_convert import ImageFormatConvert

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ]
    }

    image_src_type = "image_url"
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_format_convert",
        aihc_udf(
            ImageFormatConvert,
            construct_args={
                "output_format": "jpg",
                "image_src_type": image_src_type,
                "quality": -1,
                "output_dir": "./test_images_output",
            },
            batch_size=1,
        )(col("image"), col("image_name")),
    )

    ds.show()
```

图像黑边检测裁剪处理器

简介

图像黑边检测裁剪处理器

功能描述

- 专业级黑边检测算法：
  - 阈值检测（threshold） - 速度最快，适合纯黑边/纯色边
  - 边缘检测（edge） - 基于Canny边缘，适合模糊/渐变黑边
  - 直方图检测（histogram） - 基于像素分布，适合低对比度黑边
  - 自动检测（auto） - 智能选择最优算法（推荐）

- 多格式输入支持：
  - URL地址（image_url）
  - BOS路径 ("bos://{bucket}/{path}")
  - Base64编码（image_base64）
  - 二进制流（image_binary）
- 多格式输出支持：
  - JPEG/JPG：自动转换RGBA→RGB，保留DPI
  - PNG：保留透明通道，无损保存
  - WebP：支持透明通道，灵活保存
- 双输出模式：
  - Base64编码直出
  - 可选本地存储
- 鲁棒性保障：
  - 最小黑边尺寸过滤（避免过度裁剪）
  - 边界校验（防止裁剪超出图像范围）
  - 检测失败自动降级为原图返回

算子参数

输入

输入	含义
image	包含输入图像的数组，支持URL/base64/二进制格式
image_name	可选参数，包含图像标识名的数组，用于生成输出文件名。

输出

输出	含义
base64	压缩后图像的base64编码
image_path	本地存储路径（当配置输出目录时有效）

参数

参数	类型	含义	默认值
image_suffix	str	裁剪后图像的保存格式，支持多格式适配。可选值: [".jpg", ".jpeg", ".png", ".webp"]	".jpg"
output_dir	str	将裁剪后的图像保存到的输出路径，为空则不保存到输出，	""
image_src_type	str	输入图像的格式类型，支持URL/base64/二进制。可选值: ["image_url", "image_base64", "image_binary"]	"image_url"
detect_algorithm	str	黑边检测算法。支持4种检测算法，auto模式自动选择最优方案。可选值: ["threshold", "edge", "histogram", "auto"]	"auto"
black_threshold	int	灰度阈值（仅threshold/auto算法生效）：像素灰度值低于该值视为黑边（0-255），值越小检测越严格。	10
edge_sensitivity	float	边缘检测灵敏度（仅edge/auto算法生效）：灵敏度越高，越容易检测到弱边缘（0.1-2.0）。	1.0
min_border_size	int	最小黑边尺寸（像素）：小于该尺寸的黑边不裁剪，避免过度裁剪。	1
target_dpi	list	保存图像时的DPI参数，格式为 [width, height]。	[72, 72]
quality	int	保存质量参数（适配不同格式）：JPEG/WebP：1-100（数值越高质量越好）PNG：映射为compress_level（1→9，100→0）	85

调用示例

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_blackborder_crop import ImageBlackBorderCrop

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ],
        "image_name": ["sample_1.jpg", "sample_2.jpg", "sample_3.jpg"],
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_blackborder_crop",
        aihc_udf(
            ImageBlackBorderCrop,
            construct_args={
                "image_suffix": ".jpg",
                "output_dir": "./test_images_output",
                "image_src_type": "image_url",
                "detect_algorithm": "auto",
                "black_threshold": 10,
                "edge_sensitivity": 1.0,
                "min_border_size": 1,
                "target_dpi": [72, 72],
            },
            batch_size=1,
        )(col("image"), col("image_name")),
    )

    ds.show()
```

图像清晰度计算处理器

简介

图像清晰度计算处理器

功能描述

- 提供4种专业级清晰度评估方法：
  - 拉普拉斯方差法（laplacian） - 基于二阶导数的边缘检测，计算简单快速
  - Tenengrad法（tenengrad） - 基于Sobel梯度的边缘强度，对噪声敏感
  - Brenner法（brenner） - 基于相邻像素灰度差的简单计算，速度最快
  - FFT高频能量法（fft_highfreq） - 基于频域分析的清晰度评估，对模糊更敏感
- 多格式输入支持：
  - URL地址（image_ur), BOS地址
  - Base64编码（image_base64）
  - 二进制流（image_binary）
- 高精度计算：
  - 支持处理各种尺寸和格式的图像
  - 自动处理NaN值和异常情况

算子参数

输入

输入	含义
image	包含输入图像的数组，支持URL/base64/二进制格式

输出

输出	含义
image_sharpness	包含清晰度计算结果的浮点数组

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像的格式类型，支持：image_url：url地址image_base64：base64编码image_binary：二进制流默认值: "image_url"
method	str	"laplacian"	清晰度计算方法，支持：laplacian(拉普拉斯方差)tenengrad(Tenengrad梯度)brenner(Brenner梯度)fft_highfreq(FFT高频能量)默认为laplacian。

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_sharpness import ImageSharpness

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ]
    }

    image_src_type = "image_url"
    method = "laplacian"

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_sharpness",
        aihc_udf(
            ImageSharpness,
            construct_args={
                "image_src_type": image_src_type,
                "method": method,
            },
            batch_size=1,
        )(col("image")),
    )

    ds.show()
```

图像压缩处理器

图像重采样处理器

简介

图像重采样算子用于对输入图像进行尺寸重采样（仅支持降采样）

功能描述

- 支持 4 种插值算法（$nearest / bilinear / bicubic / lanczos$）与$ .jpg/.png $输出格式
- 输入支持本地/挂载目录/对象存储BOS 中原始数据输入
- 仅支持降采样：target_size 的宽高不得超过原图。
- 输入图像格式支持：JPEG/PNG/TIFF（JPG/TIF 作为别名支持）
- 输入图像大小限制：文件大小不超过 100MB；像素总数不超过 225,000,000

算子参数

输入

输入	含义
images	包含输入图像的数组，支持URL/BOS/base64/二进制格式
images_name	可选参数，包含图像标识名的数组，用于生成输出文件名。建议传入以确保文件名唯一性，且不要带后缀；如果不传入，会为无法解析出名称的图片随机生成名称。

输出

输出	含义
base64	压缩后图像的base64编码
image_path	本地存储路径（当配置输出目录时有效）

参数

参数	类型	含义	默认值
image_suffix	str	重采样的图像格式。可选值: [".jpg", ".png"]	'.jpg'
local_dir	str	重采样后图像的本地存储目录.如果为空，则调整后的图像不保存到本地。	""
image_src_type	str	输入图像的格式类型，支持格式："image_url" (url地址), "image_base64" (base64编码), "image_binary" (二进制流)	"image_url",
target_size	list	重采样后的图像尺寸。格式为 [width, height] ,	[200, 200],
target_dpi	list	图像 DPI。格式为 [width, height]	[72, 72]
method	str	重采样方法。支持方法："nearest" (最近邻插值)、"bilinear" (双线性插值)、"bicubic" (双三次插值)、"lanczos" (Lanczos插值)	"lanczos"

调用示例

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_resample import ImageResample

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ],
        "image_name": ["sample_1.jpg", "sample_2.jpg", "sample_3.jpg"],
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_resample",
        aihc_udf(
            ImageResample,
            construct_args={
                "image_suffix": ".jpg",
                "bos_dir": "",
                "local_dir": "./test_images_output",
                "image_src_type": "image_url",
                "target_size": (200, 200),
                "target_dpi": (72, 72),
                "method": "lanczos",
            },
            num_gpus=0,
            batch_size=1,
        )(col("image"), col("image_name")),
    )

    ds.show()
```

图像 Embedding（ViT 系列模型）

简介

ImageViTEmbedding基于 Daft + Ray 分布式框架的图像嵌入向量提取组件，利用 ViT 类模型（如 DINOv2）将图像编码为固定维度的向量表示，适用于图像检索、分类、聚类等下游任务。

功能

- 支持通过图像 URL 批量提取 ViT 嵌入向量
- 基于 Ray Actor 实现 GPU 加速推理
- 支持多种 ViT 模型（如 facebook/dinov2-large）
- 支持 FP16 精度推理，降低显存占用

> 该算子运行需要依赖对应的模型权重文件，平台已将所需权重文件预置在公共数据集aihc_daft_public_model中，需要在创建开发负载时挂载该数据集即可正常使用。

参数

参数名称	类型	默认值	描述
image_src_type	str	image_url	输入图像的格式类型，支持：bos/http 地址(image_url)base64 编码(image_base64)二进制流(image_binary)可选值：["image_url", "image_base64", "image_binary"]默认值："image_url"
dtype	str	float32	模型推理精度选择：bfloat16: 平衡精度与速度（TPU上更快）float16: 更快的推理速度float32: 最高精度但显存消耗最大可选值：["bfloat16", "float16", "float32"]默认值："float16"
batch_size	int	32	批处理大小默认值: 32
model_path	str	/opt/aihc/models	模型文件存储路径默认值: "/opt/aihc/models"
model_name	str	facebook/dinov2-large	使用的图像向量模型名称可选值: ["google/vit-base-patch16-224-in21k","google/vit-large-patch16-224-in21k","facebook/dinov2-base","facebook/dinov2-large"]默认值: "facebook/dinov2-large"
use_cls_token_embedding	bool	true	是否使用CLS Token特征默认值: True
rank	int	0	指定使用的GPU设备编号（多卡环境有效）。例如：0表示第一张GPU，1表示第二张GPU默认值：0

输入

输入列名	说明	说明
images	包含图像数据的数组，元素类型可为图像URL、Base64编码或二进制数据	图像 URL 地址

以 Daft DataFrame 形式传入，列名为 image。

输出

包含特征向量的数组，每个元素为float类型的嵌套数组， 数组维度由模型输出决定

使用示例

```
 from __future__ import annotations

import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.embedding.image_vit_embedding import ImageViTEmbedding

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    image_src_type = "image_url"
    batch_size = 64
    model_path = os.getenv("MODEL_PATH", "/opt/aihc/models")
    model_name = "facebook/dinov2-large"
    dtype = "float16"
    use_cls_token_embedding = True
    rank = 0
    num_gpus = 1

    samples = {
        "image": [
            "https://{bucket}.bj.bcebos.com/image.png",
        ]
    }

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "embedding",
        aihc_udf(
            ImageViTEmbedding,
            construct_args={
                "image_src_type": image_src_type,
                "batch_size": batch_size,
                "model_path": model_path,
                "model_name": model_name,
                "dtype": dtype,
                "use_cls_token_embedding": use_cls_token_embedding,
                "rank": rank,
            },
            num_gpus=num_gpus,
            batch_size=1,
        )(col("image")),
    )

    ds.show()
** ╭──────────────
```

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_aesthetic_score import ImageAestheticScore

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"image_inputs": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "batch_size": 32,
        "model_path": '/opt/aihc/models',
        "clip_model_name": 'openai/clip-vit-large-patch14',
        "mlp_model_name": 'laion_aesthetic_v2/sac+logos+ava1-l14-linearMSE.pth',
        "device": 'cpu',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ImageAestheticScore,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("image_inputs")),
    )
    ds.show()
```

图像OCR文字识别

简介 基于 EasyOCR 的多语言OCR识别组件，支持中英文混合场景下的文本检测与识别。

功能描述

- 支持 100+ 种语言识别（需配置对应语言模型）
- BOS URL
- Base64编码
- 二进制流
- Numpy数组
- GPU 加速推理
- 模型量化（默认开启）
- 批量处理优化
- 简体中文(ch_sim)
- 英文(en)
- 日文(ja)
- 韩文(ko)
- 法文(fr)
- 德文(de)

算子参数 输入 |输入|含义| |-|-| |images|包含图像数据的数组，支持URL/base64/二进制格式|

输出 |输出|含义| |-|-| |result|pyarrow.Array: 包含OCR识别结果的数组，元素类型为字符串|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|'image_url'|输入图像的格式类型，支持： - bos/http 地址 (image_url) - base64 编码 (image_base64) - 二进制流 (image_binary) 可选值: ["image_url", "image_base64", "image_binary"] 默认值: "image_url"| |model_path|str|'/opt/aihc/models'|模型存储路径。 默认值: "/opt/aihc/models"| |model_name|str|'EasyOCR'|使用的模型名称，当前仅支持 "EasyOCR"。 默认值: "EasyOCR"| |quantize|bool|True|是否启用模型量化加速推理。 默认值: True| |lang_list|list[str]|['en', 'ch_sim']|支持识别的语言列表，详见 https://www.jaided.ai/easyocr/ 默认值: ["en", "ch_sim"]| |batch_size|int|16|GPU 推理批次大小（可根据显存调整）。 默认值: 16|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_easyocr import ImageEasyOcr

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"images": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "image_src_type": 'image_url',
        "model_path": '/opt/aihc/models',
        "model_name": 'EasyOCR',
        "quantize": True,
        "lang_list": ['en', 'ch_sim'],
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ImageEasyOcr,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("images")),
    )
    ds.show()
```

图像人脸检测

简介 图片人脸检测算子

功能描述

- 自动检测图片中的人脸并返回矩形框列表 (x1, y1, x2, y2)
- 支持 URL、本地路径、Base64、二进制等多种输入方式
- 输出人脸矩形框列表
- cpu 和 gpu 环境均可运行
- 隐私合规：发布前对图片中的人脸进行脱敏打码
- 数据预处理：为训练/评测数据集生成带人脸矩形框的版本
- 内容审核/展示：对涉及敏感人脸的图片进行自动矩形框处理
- 单条样本处理失败时，该条结果返回 None，不影响同一批次内其他样本

算子参数 输入 |输入|含义| |-|-| |images|输入图片列，内容类型由 image_src_type 决定： - 当为 "image_url" 时：每个元素为字符串 URL/本地路径/BOS 路径； - 当为 "image_base64" 时：每个元素为 Base64 编码字符串； - 当为 "image_binary" 时：每个元素为图片二进制数据（bytes）。|

输出 |输出|含义| |-|-| |result|包含人脸边界框列表的列，每个元素为一个二维列表，表示图片中检测到的人脸边界框。 每个边界框为一个四元组 (x1, y1, x2, y2)，表示左上角和右下角坐标。 如果图片中未检测到人脸，则对应元素为 None。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|人脸检测模型的基础目录路径，用于 InsightFace 模型加载。 默认值："/opt/aihc/models"。| |model_name|str|'insightface'|InsightFace 模型子目录名称，用于拼接实际模型路径 model_path/model_name。 默认值："insightface"。| |image_src_type|str|'image_url'|输入图片的数据类型。 可选值："image_url"、"image_base64"、"image_binary"。 - "image_url": HTTP/BOS/本地路径等可通过路径访问的图片 - "image_base64": Base64 编码字符串 - "image_binary": 图片二进制数据（bytes） 默认值："image_url"。| |det_thresh|float|0.5|InsightFace 人脸检测置信度阈值。 建议：隐私打码场景可适当降低以减少漏检，精细修图可提高以减少误检。 默认值：0.5。| |det_size|tuple[int, int]|(640, 640)|InsightFace 人脸检测输入尺寸 (width, height)。 普通网络图片/头像等使用默认值 (640, 640) 即可。 默认值：(640, 640)。|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_face_detect import ImageFaceDetect

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"images": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'insightface',
        "image_src_type": 'image_url',
        "det_thresh": 0.5,
        "det_size": (640, 640),
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ImageFaceDetect,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("images")),
    )
    ds.show()
```

图像安全性检测

简介 图像安全性(NSFW)检测器，支持多源输入与批量推理。

功能描述

- 使用预训练的图像分类模型进行 NSFW 概率检测，输出每张图片的 NSFW 置信度分数。
- URL 地址（image_url）
- Base64 编码（image_base64）
- 二进制流（image_binary）
- 通过配置 batch_size 进行批量推理以提升吞吐性能。
- 内容安全审核：电商、社交平台的图片审核流程。
- 生产管线中的预过滤：在图像处理/生成任务之前进行安全性筛查。
- 仅输出数值型 NSFW 置信度（0~1），不保存图像；如需持久化请在上游/下游管线中实现。
- 请确保模型路径(model_path)与模型名称(model_name)在本地可用，或已正确配置权重下载策略。

算子参数 输入 |输入|含义| |-|-| |images|包含输入图像的数组，支持 URL/base64/数组 格式。|

输出 |输出|含义| |-|-| |result|包含检测结果的数组，每个元素为NSFW 置信度分数(float)，若检测失败则为 None。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|'image_url'|输入图像的格式类型，支持： - url地址(image_url) - base64编码(image_base64) - 二进制流(image_binary) 描述: 输入图像的格式类型。 可选值: ["image_url", "image_base64", "image_binary"] 默认值: "image_url"| |model_path|str|'/opt/aihc/models'|模型基础路径。 描述: 预训练模型在本地的根目录路径。 默认值: "/home/ray/workdir/models"| |model_name|str|'Falconsai/nsfw_image_detection'|模型名称/子目录。 描述: 在 model_path 下的具体模型目录名称。 可选值: ["Falconsai/nsfw_image_detection"] 默认值: "Falconsai/nsfw_image_detection"| |dtype|str|'float16'|模型推理精度选择： - float16: 更快的推理速度 - float32: 最高精度但显存消耗最大 可选值：["float16", "float32"] 默认值："float16"| |batch_size|int|16|批量推理的图片数量。 描述: 每次送入模型进行推理的图片数量，批量越大吞吐越高，但显存占用也更高。 默认值: 16| |rank|int|0|GPU 编号。 描述: 推理所使用的 GPU 编号，CPU 推理时可保持为 0。 默认值: 0|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_nsfw_detect import ImageNsfwDetect

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"images": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "image_src_type": 'image_url',
        "model_path": '/opt/aihc/models',
        "model_name": 'Falconsai/nsfw_image_detection',
        "dtype": 'float16',
        "batch_size": 16,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ImageNsfwDetect,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("images")),
    )
    ds.show()
```

图像人脸车牌模糊（Egoblur）

ImageEgoBlur 使用说明 简介 ImageEgoBlur 是基于 Daft 的图片隐私脱敏算子，使用 EgoBlur TorchScript 模型检测图片中的人脸和车牌，并对检测区域进行模糊处理。适用于图片发布前的隐私保护、数据集脱敏、自动化合规处理等场景。

功能

- 支持检测并模糊图片中的人脸区域
- 支持检测并模糊图片中的车牌区域
- 支持 EgoBlur gen1 和 gen2 模型
- 支持返回处理后图片的 Base64 编码
- 支持将处理后图片保存到 BOS 或本地目录

算子参数 输入 |字段|类型|说明| |-|-|-| |image_path|str|图片地址。测试示例中使用 HTTP URL，如 https://{bucket}.bj.bcebos.com/face.jpg|

输出 egoblur_result 列，类型为 struct。

字段	类型	说明
base64	str	null
image_path	str	null
face_bounding_boxes	list[list[int]]	检测到的人脸框，格式为 [x1, y1, x2, y2]
lp_bounding_boxes	list[list[int]]	检测到的车牌框，格式为 [x1, y1, x2, y2]

单条样本处理失败时，该条结果会返回空结果，例如 base64=None、image_path=None、检测框为空列表，不影响同批次其他样本。

参数 |参数|类型|默认值|说明| |-|-|-|-| |face_model_path|str|""|人脸检测模型路径，如 /opt/aihc/models/Egoblur/ego_blur_face_gen1.jit| |lp_model_path|str|""|车牌检测模型路径，如 /opt/aihc/models/Egoblur/ego_blur_lp_gen1.jit| |output_dir|str|""|处理后图片保存目录，支持 BOS 或本地路径。为空时不保存图片| |image_src_type|str|"image_url"|输入图片类型，可选 image_url、image_base64、image_binary| |model_generation|str|"auto"|模型代际，可选 auto、gen1、gen2。auto 会从模型文件名识别 gen2| |camera_name|str|"camera-rgb"|gen2 模型使用的相机名，用于选择默认阈值| |face_score_threshold|float | None|None|人脸检测置信度阈值。gen1 默认 0.9，gen2 默认按 camera_name 选择| |lp_score_threshold|float | None|None|车牌检测置信度阈值。gen1 默认 0.9，gen2 默认按 camera_name 选择| |nms_iou_threshold|float | None|None|NMS IoU 阈值。gen1 默认 0.3，gen2 默认 0.5| |scale_factor|float|1.0|模糊区域缩放系数，>1 会扩大模糊区域| |return_base64|bool|False|是否在输出中返回处理后图片的 Base64 编码| |return_image_format|str|"png"|输出图片格式，可选 png、jpg、jpeg|

至少需要提供 face_model_path 或 lp_model_path 之一。

调用示例

```
 from __future__ import annotations

import inspect
import os

import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_egoblur import ImageEgoBlur


def _get_optional_float(name: str) -> float | None:
    value = os.getenv(name)
    return float(value) if value not in {None, ""} else None


def _get_bool(name: str, default: bool = False) -> bool:
    value = os.getenv(name)
    if value is None:
        return default
    return value.lower() in {"1", "true", "yes", "y", "on"}


if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    model_base = os.getenv("MODEL_PATH", "/opt/aihc/models")
    model_generation = os.getenv("MODEL_GENERATION", "gen1").lower()
    if model_generation == "gen2" and "model_generation" not in inspect.signature(ImageEgoBlur.__init__).parameters:
        raise RuntimeError(
            "MODEL_GENERATION=gen2 requires the gen2-capable ImageEgoBlur implementation. "
            "The imported ImageEgoBlur is still the old gen1-only version."
        )

    face_model_path = os.getenv("FACE_MODEL_PATH", os.path.join(model_base, f"ego_blur_face_{model_generation}.jit"))
    lp_model_path = os.getenv("LP_MODEL_PATH", os.path.join(model_base, f"ego_blur_lp_{model_generation}.jit"))
    output_dir = os.getenv("OUTPUT_DIR", "")

    samples = {"image_path": ["https://{bucket}.bj.bcebos.com/face.jpg"]}
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "face_model_path": face_model_path,
        "lp_model_path": lp_model_path,
        "model_generation": model_generation,
        "camera_name": os.getenv("CAMERA_NAME", "camera-rgb"),
        "output_dir": output_dir,
        "return_base64": _get_bool("RETURN_BASE64"),
    }
    optional_float_kwargs = {
        "face_score_threshold": _get_optional_float("FACE_SCORE_THRESHOLD"),
        "lp_score_threshold": _get_optional_float("LP_SCORE_THRESHOLD"),
        "nms_iou_threshold": _get_optional_float("NMS_IOU_THRESHOLD"),
    }
    constructor_kwargs.update({key: value for key, value in optional_float_kwargs.items() if value is not None})

    ds = ds.with_column(
        "egoblur_result",
        aihc_udf(
            ImageEgoBlur,
            construct_args=constructor_kwargs,
            num_gpus=0,
            batch_size=1,
            concurrency=1,
        )(col("image_path")),
    )

    ds.show()
** ┌──────────────────────────────┬──────────────────────────────────────────────────────────────────────────────┐
** │ image_path                   ┆ egoblur_result                                                               │
** │ ---                          ┆ ---                                                                          │
** │ String                       ┆ Struct{base64: String, image_path: String, face_bounding_boxes: ⋯}          │
** ╞══════════════════════════════╪══════════════════════════════════════════════════════════════════════════════╡
** │ https://{bucket}.bj.bcebos.com/ ┆ {base64: None, image_path: None, face_bounding_boxes: [[...]], ⋯}          │
** └──────────────────────────────┴──────────────────────────────────────────────────────────────────────────────┘
*
```

3D模型渲染

简介 3D模型渲染处理器，支持多种3D格式渲染为2D图片。

功能描述

- 3D模型文件路径（支持本地路径）
- Open3D后端：.obj, .ply, .stl, .glb, .gltf
- 多视角渲染（默认8个方向）
- 自定义贴图支持
- 返回渲染图片的 Base64 字符串列表

- 可选将图片保存到 BOS
- 可选将图片保存到本地目录

算子参数 输入 |输入|含义| |-|-| |model_paths|输入3D模型文件路径数组| |model_names|可选，模型名称数组（用于生成输出文件名）| |texture_paths|可选，每个模型的自定义贴图路径数组| |views_configs|可选，每个模型的自定义渲染视角配置数组 (list[dict])|

输出 |输出|含义| |-|-| |result|pa.Array: 结果字典数组，包含每个模型对应的多视角渲染结果|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_suffix|str|'.png'|输出图片的后缀名，默认 ".png"。| |output_dir|str|''|保存渲染图片的文件夹路径（支持本地路径或 BOS 路径）。| |texture_path|str|dict[str, str]|自定义贴图绝对路径，支持单张图片路径(str)或PBR贴图字典(dict)。 字典键支持: "albedo", "metallic", "roughness", "normal", "ao"。 如果提供将覆盖所有模型的贴图。| |texture_relative_path|str|dict[str, str]|自定义贴图相对路径（相对于模型文件所在目录）。 支持单张图片路径(str)或PBR贴图字典(dict)。 例如 "textures_base/texture_shaded.png" 或 {"albedo": "pbr/texture_base.png", "metallic": "pbr/metallic.png"}。 如果提供了texture_path，此参数将被忽略。| |views|dict[str, dict[str, Any]]|None|自定义渲染视角配置，字典格式 {view_name: {dist, elev, azim, ...}}。 如果为None，则使用默认的8个视角。| |target_size|list[int]|None|渲染图片尺寸 [width, height]。|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_render import ImageRender

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"model_paths": [...], "model_names": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "image_suffix": '.png',
        "output_dir": '',
        "texture_path": None,
        "texture_relative_path": 'textures_base/texture_shaded.png',
        "views": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ImageRender,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("model_paths"), col("model_names"), col("texture_paths")),
    )
    ds.show()
```

图像人脸模糊

简介 图片人脸模糊处理算子

功能描述

- 自动检测图片中的人脸并进行模糊处理
- 支持多种模糊类型（均值、盒式、高斯）
- 支持 URL、本地路径、Base64、二进制等多种输入方式
- 可选输出模糊后图片的 Base64 编码以及保存到的路径
- cpu 和 gpu 环境均可运行
- 隐私合规：发布前对图片中的人脸进行脱敏打码
- 数据预处理：为训练/评测数据集生成带人脸模糊的版本
- 内容审核/展示：对涉及敏感人脸的图片进行自动模糊处理
- 单条样本处理失败时，该条结果返回 {"base64": None, "image_path": None}，不影响同一批次内其他样本
- 当配置 output_dir 时，模糊后的图片会上传到指定BOS/本地目录

算子参数 输入 |输入|含义| |-|-| |images|-| |images_name|-|

输出 |输出|含义| |-|-| |result|算子输出|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_dir|str|''|将模糊处理后的图片保存到的 BOS 目录或者本地路径。 格式："bos://bucket/path/" 或 "/local/path/" 默认值："""| |model_path|str|'/opt/aihc/models'|人脸检测模型的基础目录路径，用于 InsightFace 模型加载。 默认值："/opt/aihc/models"。| |model_name|str|'insightface'|InsightFace 模型子目录名称，用于拼接实际模型路径 model_path/model_name。 默认值："insightface"。| |image_src_type|str|'image_url'|输入图片的数据类型。 可选值："image_url"、"image_base64"、"image_binary"。 - "image_url": HTTP/BOS/本地路径等可通过路径访问的图片 - "image_base64": Base64 编码字符串 - "image_binary": 图片二进制数据（bytes） 默认值："image_url"。| |blur_type|str|'gaussian'|人脸区域的模糊方式。 可选值："mean"、"box"、"gaussian"。 - "mean": 使用均值模糊（ImageFilter.BLUR） - "box": 使用盒式模糊（ImageFilter.BoxBlur） - "gaussian": 使用高斯模糊（ImageFilter.GaussianBlur） 默认值："gaussian"。| |radius|float|10.0|模糊半径，适用于 "box" 和 "gaussian" 模式。 要求：radius >= 0，默认值：10.0。| |det_thresh|float|0.5|InsightFace 人脸检测置信度阈值。 建议：隐私打码场景可适当降低以减少漏检，精细修图可提高以减少误检。 默认值：0.5。| |det_size|tuple[int, int]|(640, 640)|InsightFace 人脸检测输入尺寸 (width, height)。 普通网络图片/头像等使用默认值 (640, 640) 即可。 默认值：(640, 640)。| |return_base64|bool|False|是否在输出结果中包含模糊后图片的 Base64 编码。 默认值：False。| |return_image_format|str|'png'|输出图片的格式 可选值："png"、"jpg"、"jpeg"， 默认值："png"。|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_face_blur import ImageFaceBlur

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"images": [...], "images_name": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_dir": '',
        "model_path": '/opt/aihc/models',
        "model_name": 'insightface',
        "image_src_type": 'image_url',
        "blur_type": 'gaussian',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            ImageFaceBlur,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("images"), col("images_name")),
    )
    ds.show()
```

层级分类器

简介 HierarchicalClassifier 是基于 CLIP zero-shot 的层级分类算子，按照 label_tree 指定的 prompt 树对图像逐层分类，输出层级路径字符串（如 general/live_action/human）及沿路径的最小置信度。

功能描述 • 使用本地 CLIP 权重（默认 openai/clip-vit-base-patch16）做 zero-shot 分类，无需重新训练，适合冷启动。

• 通过 label_tree 自定义层级 prompt 树；预先为每一层 label 缓存 text embedding，避免重复编码。

• 逐层 softmax 取 argmax 向下深入，直至叶节点；当某层置信度低于 min_confidence 时输出 unknown。

• 支持 URL / base64 / binary 三种图像输入，视频可先抽首帧再喂入；GPU/CPU 均可运行。

算子参数 输入 |输入|含义| |-|-| |images|待分类的图像列，取值由 image_src_type 决定（图像 URL / base64 字符串 / 二进制）；视频请先抽取首帧|

输出 |输出|含义| |-|-| |result| struct{path: string, confidence: float32} ，path 为层级路径（如 general/live_action/human ，低置信度为 unknown） ，confidence 为路径上各层的最小置信度|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-|-| |image_src_type|str|"image_url"|图像输入类型，可选 image_url / image_binary / image_base64|| |model_path|str|"/opt/aihc/models"|CLIP 权重根目录，与 model_name 拼接为实际模型目录|| |model_name|str|"openai/clip-vit-base-patch16"|CLIP 模型名（相对 model_path 的子目录）|| |label_tree|dict[str, Any] |None|None|层级 prompt 树；为 None 时使用内置默认树 _DEFAULT_LABEL_TREE（general / physical_ai 两大顶层）| |prompt_template|str|"a photo of {label}"|每个 label 的 prompt 模板，{label} 会替换为叶节点名|| |batch_size|int|16|图像推理批大小|| |dtype|str|"float32"| 模型精度，可选 float32 / float16 / bfloat16（zero-shot 下 float32 更稳）|| |rank|int|0|多卡 rank，用于选择 cuda:{rank % cuda_device_count}|| |min_confidence|float|0.0|置信度低于该阈值时输出 unknown；默认 0.0 表示不做阈值过滤|| |fail_on_error|bool|False|批处理出错时是否抛异常；False 时对应行返回 None||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.hierarchical_classifier import HierarchicalClassifier

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "label",
        aihc_udf(
            HierarchicalClassifier,
            construct_args={
                "image_src_type": "image_url",
                "model_path": MODEL_PATH,
                "model_name": "openai/clip-vit-base-patch16",
                "min_confidence": 0.0,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像白底检测

简介 ImageWhiteBackgroundDetect 是白背景检测算子，基于 PIL / numpy 灰度直方图统计判断图像是否为白底图，输出布尔值。

功能描述 • 将图像转灰度后，分别统计边界带与全图中亮度 >= brightness_threshold 的白像素占比。

• 当边界带白像素占比与全图白像素占比同时超过 white_pixel_ratio 时判为白背景。

• 边界带宽度由 border_ratio 与图像最短边共同决定。

• 纯 CPU 实现，仅依赖 PIL 与 numpy，无需模型权重。

算子参数 输入 |输入|含义| |-|-| |images|待检测的图像列，取值由 image_src_type 决定（图像 URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result|bool，True 表示图像为白背景|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|"image_url"|图像输入类型，可选 image_url / image_binary / image_base64| |brightness_threshold|int|240|单像素灰度亮度阈值，>= 视为 "白"| |border_ratio|float|0.05|边界带占图像最短边的比例| |white_pixel_ratio|float|0.75|白像素占比阈值；边界带与全图占比同时超过该值判为 True| |fail_on_error|bool|False|出错时是否抛异常；False 时对应行返回 False|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_white_background_detect import ImageWhiteBackgroundDetect

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "white_bg",
        aihc_udf(
            ImageWhiteBackgroundDetect,
            construct_args={
                "image_src_type": "image_url",
                "brightness_threshold": 240,
                "border_ratio": 0.05,
                "white_pixel_ratio": 0.75,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像目标检测（YOLO）

简介 `ImageDetectionYolo` 是基于 ultralytics YOLO(11) 的目标检测算子，对每张图返回检测框（xywh）、类别 id 与置信度，是多模态打标链路中的 2D 检测环节。

功能描述 • 使用 ultralytics YOLO 加载权重（默认 `yolo11n.pt`），支持 CPU / GPU 推理（GPU 更快）。

• 可配置推理分辨率 `imgsz`、置信度阈值 `conf` 与 NMS IoU 阈值 `iou`。

• 输出每张图的检测框（xywh 格式）、类别 id 列表和置信度列表；无图或无检测时均为空列表。

算子参数 输入 |输入|含义| |-|-| |image|待检测的图像列，取值由 `image_src_type` 决定（图像 URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result| struct{bbox: list<list<float64>> (M×4, xywh), cls: list<int64>, conf: list<float64>}，无检测时三者均为空列表|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|"yolo11n.pt"|YOLO 权重路径或名称（如 `yolo11n.pt` / `yolo11x.pt`）| |image_src_type|str|"image_url"|图像输入类型，可选 `image_url` / `image_base64` / `image_binary`| |imgsz|int|640|推理分辨率| |conf|float|0.05|置信度阈值| |iou|float|0.5|NMS IoU 阈值| |fail_on_error|bool|False|推理异常时是否立即抛出；False 时对应行返回空检测|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_detection_yolo import ImageDetectionYolo

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "det",
        aihc_udf(
            ImageDetectionYolo,
            construct_args={
                "model_path": "yolo11n.pt",
                "image_src_type": "image_url",
                "imgsz": 640,
                "conf": 0.05,
                "iou": 0.5,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像真实感评分

简介 `ImageRealismScore` 是图像真实感评分算子，采用 CLIP zero-shot 方式对比 "真实照片" 与 "AI 生成图像" 两个 prompt 的相似度，输出 0-1 的真实感分数（越大越真实）。

功能描述 • 使用本地 CLIP 权重（默认 `openai/clip-vit-base-patch16`）做 zero-shot 打分，作为 SILA RealismScore 的开源冷启动替代。

• 初始化时预计算正/负 prompt 的文本特征，推理时仅对图像特征做相似度比较。

• 对每个 batch 的图像特征与正负 prompt 做 softmax，取正向 prompt 概率作为真实感分数。

• 支持 URL / base64 / binary 图像输入，GPU/CPU 均可运行。

算子参数 输入 |输入|含义| |-|-| |images|待评分的图像列，取值由 `image_src_type` 决定（图像 URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result|`float32`，0-1 真实感分数，越大越真实；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|"image_url"|图像输入类型，可选 `image_url` / `image_binary` / `image_base64`| |model_path|str|"/opt/aihc/models"|CLIP 权重根目录，与 `model_name` 拼接为实际模型目录| |model_name|str|"openai/clip-vit-base-patch16"|CLIP 模型名（相对 `model_path` 的子目录）| |batch_size|int|32|图像推理批大小| |rank|int|0|多卡 rank，用于选择 `cuda:{rank % cuda_device_count}`| |pos_prompt|str|"a real natural photo taken by a camera"|正向 prompt（代表真实照片），其概率作为分数| |neg_prompt|str|"an AI generated synthetic image"|负向 prompt（代表 AI 生成图像）| |fail_on_error|bool|False|批处理出错时是否抛异常；False 时对应行返回 None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_realism_score import ImageRealismScore

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "realism",
        aihc_udf(
            ImageRealismScore,
            construct_args={
                "image_src_type": "image_url",
                "model_path": MODEL_PATH,
                "model_name": "openai/clip-vit-base-patch16",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像水印检测

简介 ImageWatermarkDetect 是图像水印检测算子，默认加载 HuggingFace 分类器 amrul-hzz/watermark_detector 判断图像中是否含有水印，输出布尔值。

功能描述 • 使用本地权重（默认 amrul-hzz/watermark_detector）的图像分类模型做水印二分类。

• 自动从模型 id2label 中兼容识别 "watermark" 正类索引，适配不同权重的标签命名。

• 对每个 batch 做 softmax，正类概率 >= threshold 时判为有水印。

• 支持 URL / base64 / binary 图像输入，GPU/CPU 均可运行。

算子参数 输入 |输入|含义| |-|-| |images|待检测的图像列，取值由 image_src_type 决定（图像 URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result|bool，True 表示检测到水印；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|"image_url"|图像输入类型，可选 image_url / image_binary / image_base64| |model_path|str|"/opt/aihc/models"|模型权重根目录，与 model_name 拼接为实际模型目录| |model_name|str|"amrul-hzz/watermark_detector"|水印检测分类模型名（相对 model_path 的子目录）| |batch_size|int|16|图像推理批大小| |threshold|float|0.5|判为有水印的正类概率阈值| |rank|int|0|多卡 rank，用于选择 cuda:{rank % cuda_device_count}| |fail_on_error|bool|False|批处理出错时是否抛异常；False 时对应行返回 None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_watermark_detect import ImageWatermarkDetect

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "has_watermark",
        aihc_udf(
            ImageWatermarkDetect,
            construct_args={
                "image_src_type": "image_url",
                "model_path": MODEL_PATH,
                "model_name": "amrul-hzz/watermark_detector",
                "threshold": 0.5,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像模糊

简介 ImageBlur 是按概率对图像做模糊处理的算子，仅依赖 PIL，可用于隐私保护与数据增广。

功能描述 • 支持 mean / box / gaussian 三种模糊方式，gaussian / box 可通过 radius 控制模糊半径。

• 按概率 p 决定单张图是否模糊，未命中概率时原样返回原图路径。

• 处理后的图像先落地到本地 output_dir，若配置 output_bosdir 则上传 BOS 并返回 BOS 路径。

• 纯 CPU 实现，不依赖 data_juicer 包，可与其它 AIHC 算子共存于统一环境。

算子参数 输入 |输入|含义| |-|-| |image|待模糊的图像列，取值由 image_src_type 决定（bos/http URL、base64 或二进制）|

输出 |输出|含义| |-|-| |result| large_string，模糊后图像路径（本地或 BOS）；未命中概率时返回原图路径；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|"image_url"|图像输入类型（ image_url / image_binary / image_base64 ）| |p|float|0.2|模糊概率，取值须在 [0, 1]；1.0 表示强制模糊| |blur_type|str|"gaussian"|模糊类型，可选 mean / box / gaussian| |radius|float|2.0|模糊半径，须 >= 0；对 box / gaussian 生效（mean 使用固定 ImageFilter.BLUR）| |output_dir|str|"/tmp/aihc_image_blur"|模糊后图像的本地输出目录| |output_bosdir|str|""|BOS 输出目录；非空时上传并返回 BOS 路径，为空则返回本地路径|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_blur import ImageBlur

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "blurred",
        aihc_udf(
            ImageBlur,
            construct_args={
                "image_src_type": "image_url",
                "p": 1.0,  # 强制模糊
                "blur_type": "gaussian",
                "radius": 2.0,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像拼图检测

简介 `ImageCollageDetect` 是拼贴 / 多子图检测算子，通过 OpenCV Canny 边缘 + HoughLinesP 直线检测判断图像是否为拼贴图。

算子参数 输入 |输入|含义| |-|-| |images|待检测的图像列，取值由 `image_src_type` 决定（图像 URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result|`bool`，True 表示图像为拼贴 / 多子图|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|"image_url"|图像输入类型，可选 `image_url` / `image_binary` / `image_base64`| |canny_low|int|60|Canny 边缘检测低阈值| |canny_high|int|180|Canny 边缘检测高阈值| |hough_threshold|int|120|HoughLinesP 累加器阈值| |line_len_ratio|float|0.5|直线最小长度占图像最短边的比例阈值| |n_lines_threshold|int|3|满足条件的横线 + 竖线总数阈值，达到该值判为拼贴| |fail_on_error|bool|False|出错时是否抛异常；False 时对应行返回 False|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_collage_detect import ImageCollageDetect

IMAGE = "bos://your-bucket/sample.jpg"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"image": [IMAGE]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "is_collage",
        aihc_udf(
            ImageCollageDetect,
            construct_args={
                "image_src_type": "image_url",
                "canny_low": 60,
                "canny_high": 180,
                "hough_threshold": 120,
                "line_len_ratio": 0.5,
                "n_lines_threshold": 3,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像字节大小

简介

图像字节大小算子：只看文件字节数，**不解码图像**，逐行产出字节数，过滤交给 pipeline。

功能描述

- `image_src_type="image_url"` 且路径是本地路径（`/` 或 `file://` 开头）时直接 `stat` 取 `st_size`，不读文件内容
- 远端路径（`bos://` / `s3://` / http(s)）会把对象整个下载到临时目录后取字节长度
- `image_base64` 取 **解码后** 的字节数（等于原文件大小），不是 base64 字符串长度；`image_binary` 取 bytes 长度
- 全程不做图像解码，坏图/非图像文件也能算出字节数
- 单行失败（文件不存在、下载失败、输入类型与 `image_src_type` 不匹配）打 exception 日志后该行填 None，不中断整批
- 一行一张图，逐行产出该图的字节数

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（路径/URL 字符串、base64 字符串、bytes）

输出

输出	含义
image_size	int64，图像字节数。该行处理失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像编码形式：image_url / image_base64 / image_binary，取值非法在初始化阶段抛 ValueError

注意事项

- 本算子只产出统计字段，不做过滤。输出统一是字节数，`"5KB"` / `"1TB"` 这类带单位阈值的换算自己在 pipeline 里做。
- 输出的是**文件**大小，不是解码后的像素体积。要按分辨率筛就用 图像宽高（ImageShape）。
- 远端路径逐行下载整个对象，全量跑的开销与下载带宽相关；数据已在本地/挂载盘时才有「只 stat 不读内容」的快路径。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_size import ImageSize

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_size",
        aihc_udf(
            ImageSize,
            construct_args={"image_src_type": "image_url"},
            num_cpus=1,
            concurrency=4,
            batch_size=64,
        )(col("image")),
    )
    ds = ds.where(col("image_size") > 1024)
    ds.show()
```

图像对相似度（CLIP）

简介

图图相似度算子：两张图各过一遍 CLIP 图像塔，取余弦相似度，逐行产出分数，过滤交给 pipeline。CLIP 封装复用 `daft/aihc/functions/multimodal/_clip_dual_tower.py`。

功能描述

- 双输入列：图像 A、图像 B 按行配对，两列必须用同一种 `image_src_type`；两列行数不一致直接抛 ValueError
- 分数 = 两侧 L2 归一化向量的点积，等价于余弦相似度，结果 `round(..., 6)`，取值 [-1, 1]
- 权重从本地目录加载（{model_path}/{model_name}），不联网下载；目录不存在时初始化阶段抛 FileNotFoundError
- 设备由 `aihc_udf(num_gpus=...)` 决定：`num_gpus > 0` 且 `torch.cuda.is_available()` 时用 `cuda:{rank % 卡数}`，否则 CPU
- `dtype="float16"` 落到 CPU 上会被强制回退 float32（CPU 的 fp16 矩阵乘法不完整），并打 warning
- 图像先 `convert("RGB")` 再交给 `CLIPProcessor` 做 resize/归一化
- 任一侧为 None 或空白字符串的行不进推理，直接输出 None
- 按 `batch_size` 微批推理；微批内任一张图解码或推理失败，**整个微批**记 exception 日志并留 None，其余微批照常
- 两路输入按列组织，不要求 A、B 是不同的图；A/B 传同一张图会正常返回约 1.0

算子参数

输入

输入	含义
images_a	第一路图像列，内容形式由 image_src_type 决定
images_b	第二路图像列，与 images_a 按行配对，形式相同

输出

输出	含义
sim	float64，两张图的 CLIP 余弦相似度，取值 [-1, 1]；任一侧为空或该微批失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	两列图像共用的编码形式：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"openai/clip-vit-base-patch32"	相对 model_path 的 CLIP 权重子目录
dtype	str	"float32"	权重精度：float16 / float32 / bfloat16，其他值抛 ValueError
batch_size	int	16	推理微批大小（算子内部分批，与 aihc_udf(batch_size=...) 的行批是两回事）
rank	int	0	多卡场景 worker 序号，用来选 cuda:{rank % 卡数}

注意事项

- 本算子只产出统计字段，不做过滤。例如「只保留相似度落在 [0.1, 1.0] 的样本」这类条件要自己在 pipeline 里写。
- 加载的是完整 `CLIPModel`（含文本塔），只用到图像塔；显存占用按整模型算。
- 微批粒度的异常兜底意味着一张坏图会连带同微批的其它行变成 None。数据脏的时候把 `batch_size` 调小，能缩小连带范围。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_pair_similarity import ImagePairSimilarity

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "a": ["bos://your-bucket/sample.jpg"],
        "b": ["bos://your-bucket/sample.jpg"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "sim",
        aihc_udf(
            ImagePairSimilarity,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "openai/clip-vit-base-patch32",
                "dtype": "float32",
                "batch_size": 16,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=16,
        )(col("a"), col("b")),
    )
    ds.show()
```

图像扩散生成

简介

Stable Diffusion img2img 图像增广算子：以原图为条件、caption 为 prompt 走 img2img 重绘，为每张图生成 `aug_num` 张增广图并落盘，`strength` 控制改动强度、`guidance_scale` 控制 CFG 强度。默认权重 `CompVis/stable-diffusion-v1-4`。

功能描述

- 推理前把图 BILINEAR resize 到 512×512，出图后再 BILINEAR 缩回原图尺寸，输出图与输入图同分辨率
- caption 是可选的第二个输入列：该行 caption 为空、非字符串或整列不传时用 `default_prompt`；算子内不会再套一个大模型现算 caption
- 命中 pipeline 自带的 NSFW safety checker 时重跑采样，最多 3 次；3 次仍命中该行按失败处理（safety checker 被权重裁掉时跳过该判断）
- 列式算子行数恒定，输出列是该行生成的 `aug_num` 个图片路径列表，不含原图
- `aug_num > 1` 时逐张独立采样，耗时随 `aug_num` 线性增长
- 输出文件名 `<输入图 stem>_diffusion_<序号>.jpg`；输入不是路径（base64 / binary）时 stem 取 `row<行内下标>`
- 权重只从本地 {model_path}/{model_name} 加载，目录不存在时在构造期抛 FileNotFoundError，不联网下载、不做运行时 pip 安装
- 构造期校验 `image_src_type` 合法、`strength ∈ [0, 1]`、`aug_num > 0`
- 输入为 null 直接返回空列表；单行异常（解码失败、NSFW 重试用尽等）记日志后返回空列表，不影响其它行

算子参数

输入

输入	含义
image	图像输入，内容类型由 image_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制数据）
caption	可选，img2img 使用的 prompt 文本列；该行为空或整列不传时使用 default_prompt

输出

输出	含义
aug_images	list<large_string>：本行生成的 aug_num 个增广图路径（output_bosdir 非空时为 BOS 路径）；输入为空或该行失败时为空列表

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"CompVis/stable-diffusion-v1-4"	相对 model_path 的 SD 权重子目录
dtype	str	"float16"	权重精度：float32 / float16 / bfloat16。GPU 上建议 float16
strength	float	0.8	img2img 改动强度，取值 [0, 1]
guidance_scale	float	7.5	CFG 强度
aug_num	int	1	每张图生成几张增广图
default_prompt	str	"A photo of a scene"	caption 缺省时使用的 prompt
num_inference_steps	int	50	采样步数
output_dir	str	"/tmp/aihc_image_diffusion"	本地输出目录
output_bosdir	str	""	非空时把生成图上传到该 BOS 目录，并返回 BOS 路径
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 权重要求 safetensors 格式（use_safetensors=True）。
- 只有 aihc_udf(num_gpus=...) 大于 0 时才用 GPU；否则整条 SD 采样跑在 CPU 上，速度不可用。
- 输出文件名只由输入图 stem 与序号决定：不同目录下同名的输入图、或使用 base64/binary 输入时不同微批的相同行内下标，都会写到同一个输出路径而互相覆盖。批量场景建议按批设置不同的 output_dir。
- num_inference_steps 默认 50，配合 aug_num 会直接决定单行耗时；回归测试里取 20 步。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_diffusion import ImageDiffusion

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "image": ["bos://your-bucket/sample.jpg"],
        "caption": ["a photo of a street scene"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "aug_images",
        aihc_udf(
            ImageDiffusion,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "CompVis/stable-diffusion-v1-4",
                "dtype": "float16",
                "aug_num": 2,
                "num_inference_steps": 20,
                "output_dir": "/tmp/aihc_image_diffusion",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image"), col("caption")),
    )
    ds.show()
```

SDXL Prompt2Prompt 相似图对生成

简介

SDXL 交叉注意力控制生成相似图对的算子：给定一对 caption，用仓库内置的 Prompt2PromptPipeline（daft/aihc/functions/image/_prompt2prompt_pipeline.py）以 edit_type="refine" 的注意力替换，一次生成两张仅局部语义不同的图，常用于造 image-diff / 细粒度对比训练数据。默认权重 stabilityai/stable-diffusion-xl-base-1.0。

功能描述

- 输入是两个 prompt 列，没有图像输入：两个 prompt 作为一个 batch 一次采样，共享同一条噪声轨迹，第二张图只在注意力 refine 覆盖的位置上与第一张不同
- cross_attention_kwargs 固定 edit_type="refine"，n_self_replace 与 n_cross_replace 由参数控制
- seed = -1 时每行现取 random.randint(0, 9999)；seed >= 0 时全列复用同一 seed，结果可复现
- 生成尺寸可用 height / width 指定；尺寸非法（非正数或非 8 的倍数）在构造期就报错，因为 transform 会把行级异常吞成空列表
- 构造期同时校验 num_inference_steps > 0、n_self_replace ∈ [0, 1]、n_cross_replace ∈ [0, 1]
- 推理在 torch.no_grad() 下执行；输出恒为两张图，不支持一行出多对
- 输出文件名 image_pair_row<行内下标>_<seed>_1.jpg / _2.jpg；output_bosdir 非空时上传并返回 BOS 路径

- prompt 对中任一为空串或非字符串时跳过该行、返回空列表并记 warning；行级异常记日志后返回空列表
- 权重只从本地 {model_path}/{model_name} 加载，目录不存在时构造期抛 FileNotFoundError，不联网下载、不做运行时 pip 安装

算子参数

输入

输入	含义
caption	第一张图的 prompt 文本
caption_second	第二张图的 prompt 文本，通常与第一个 prompt 只差一个局部词

输出

输出	含义
image_pair	list<large_string>：[第一张图路径, 第二张图路径]（output_bosdir 非空时为 BOS 路径）；prompt 对非法或该行失败时为空列表

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"stabilityai/stable-diffusion-xl-base-1.0"	相对 model_path 的 SDXL 权重子目录
dtype	str	"float16"	权重精度：float32 / float16 / bfloat16。GPU 上建议 float16
variant	str	""	权重文件 variant 后缀；只下了 *.fp16.safetensors 时需传 "fp16"，空串表示默认权重
num_inference_steps	int	50	采样步数
guidance_scale	float	7.5	CFG 强度
n_self_replace	float	0.4	self-attention 替换比例，取值 [0, 1]
n_cross_replace	float	1.0	cross-attention 替换比例，取值 [0, 1]
seed	int	-1	随机种子；-1 表示每行随机
height	int 或 None	None	生成高度，必须是 8 的正整数倍；None 走 SDXL 默认 1024
width	int 或 None	None	生成宽度，语义同 height
output_dir	str	"/tmp/aihc_sdxl_prompt2prompt"	本地输出目录
output_bosdir	str	""	非空时把生成图上传到该 BOS 目录，并返回 BOS 路径
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 显存：prompt2prompt 要显式拿到注意力矩阵，走不了 SDPA。1024×1024 峰值实测 29.6G，24G 卡放不下；降到 768×768 是 14.3G。小卡上务必显式设 height / width。
- 若权重只有 fp16 分片，必须同时传 variant="fp16"，否则 from_pretrained 找不到权重文件。
- 输出文件名里的行内下标是「本次微批内」的下标，多个微批会从 0 重新计数：固定 seed 且 batch_size < 总行数 时不同批之间会互相覆盖，批量场景建议按批设置不同的 output_dir。
- 只有 aihc_udf(num_gpus=...) 大于 0 时才用 GPU。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.sdxl_prompt2prompt import SdxlPrompt2Prompt

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "caption": ["a photo of a squirrel eating a burger"],
        "caption_second": ["a photo of a squirrel eating a confetti burger"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "image_pair",
        aihc_udf(
            SdxlPrompt2Prompt,
            construct_args={
                "model_path": "/path/to/models",
                "model_name": "stabilityai/stable-diffusion-xl-base-1.0",
                "dtype": "float16",
                "variant": "fp16",
                "num_inference_steps": 20,
                "seed": 42,
                "height": 768,
                "width": 768,
                "output_dir": "/tmp/aihc_sdxl_prompt2prompt",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("caption"), col("caption_second")),
    )
    ds.show()
```

图像标签生成

简介

RAM++ 图像打标算子：用 RAM++（recognize-anything plus，swin_l backbone）给图像打开放词表标签，每张图输出一组去重保序的标签。默认权重 ram_plus_swin_large_14m.pth。

功能描述

- 标签来自 RAM++ generate_tag 输出的 "|" 分隔串，按分隔符拆开、去空白、去重且保持原顺序
- 先把整列图片解码成 RGB，再按 batch_size 分微批推理，而不是逐样本推理，吞吐更高
- 权重只从本地 {model_path}/{model_name} 加载，缺失时构造期抛 FileNotFoundError；算子内不做网络请求与安装
- 为兼容 RAM++ 自带的旧版 BERT 代码，加载时把 transformers.pytorch_utils 的 apply_chunking_to_forward / prune_linear_layer / find_pruneable_heads_and_indices 按需回填到 transformers.modeling_utils（RAM++ 照 transformers 4.2x 写，镜像里是 4.57），避免为一个包降级全环境的 transformers
- 解码失败的行单独跳过（记日志、该行留空列表），不影响同批其它行；某个微批推理抛异常时该批所有行都留空列表并记 exception
- 输入为 null 时该行返回空列表
- backbone 固定 swin_l，模型输入分辨率由 input_size 决定（默认 384）
- 设备：num_gpus > 0 且 torch.cuda.is_available() 时用 cuda:(rank % 可见卡数)，否则 CPU

算子参数

输入

输入	含义
image	图像输入，内容类型由 image_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制数据）

输出

输出	含义
tags	list<large_string>：该图的开放词表标签（去重、保序）；输入为空或该行失败时为空列表

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"ram_plus_swin_large_14m.pth"	相对 model_path 的 RAM++ 权重文件名
input_size	int	384	模型输入分辨率
batch_size	int	8	模型推理微批大小（<=1 时按 1 张一批）
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 需要镜像内已安装 `ram`（recognize-anything）包，未安装时构造期抛 `RuntimeError`；算子不在运行时装包。
- `construct_args` 里的 `batch_size` 是模型推理微批，`aihc_udf(batch_size=...)` 是一次 UDF 调用处理多少行，两者是不同的东西：单次调用会先把该次的所有行解码成 PIL 图（内存占用按行数走），再按推理微批进显存。
- `input_size` 要与权重的训练分辨率一致，默认 384；改了会影响标签质量。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_tagging import ImageTagging

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "tags",
        aihc_udf(
            ImageTagging,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "ram_plus_swin_large_14m.pth",
                "input_size": 384,
                "batch_size": 4,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=8,
        )(col("image")),
    )
    ds.show()
```

图像人脸数量

简介

图像人脸数量算子：用 OpenCV Haar 正脸级联检测图像中的人脸并计数，逐行输出人脸数量，过滤交给 pipeline。

功能描述

- 检测器：`cv2.CascadeClassifier` + OpenCV 自带的 `haarcascade_frontalface_alt.xml`（`cv2.data.haarcascades` 目录下），随 opencv 包安装，不需要纳管权重
- 纯 CPU，不用 GPU、不用 torch 推理
- `scale_factor` / `min_neighbors` / `min_size` / `max_size` 直接透传 `detectMultiScale`；`min_size`、`max_size` 为 None 时不传该参数（cv2 不接受 None）
- 灰度化：`convert("RGB")` 后走 `COLOR_RGB2GRAY`
- 只计数，不做检测框的边界裁剪（`min(w, width - x)`）——裁剪不影响数量
- 级联 xml 加载失败（`CascadeClassifier.empty()`）在**初始化**阶段抛 `FileNotFoundError`，整个 UDF 起不来，而不是逐行报错
- 单行失败（读不到文件、非图像数据、输入类型与 `image_src_type` 不匹配）打 exception 日志后该行填 None，不中断整批
- 一行一张图，逐行产出该图的人脸数

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（路径/URL 字符串、base64 字符串、bytes）

输出

输出	含义
face_count	int32，检出人脸数（无脸为 0）。该行处理失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像编码形式：image_url / image_base64 / image_binary
cv_classifier	str	""	Haar 级联 xml 路径，空字符串用 OpenCV 自带的 haarcascade_frontalface_alt.xml
scale_factor	float	1.1	detectMultiScale 的 scaleFactor
min_neighbors	int	3	detectMultiScale 的 minNeighbors
min_size	tuple[int,int] 或 None	None	最小人脸尺寸，None 表示不限制（不传 minSize）
max_size	tuple[int,int] 或 None	None	最大人脸尺寸，None 表示不限制（不传 maxSize）

注意事项

- 本算子只产出统计字段，不做过滤。例如「只留恰好一张脸的样本」这类条件（face_count == 1）要自己在 pipeline 里写。
- Haar 正脸级联的精度就是本算子的上限：侧脸、遮挡、远处小脸会漏检，纹理密集区域（布面印花、格栅、卡通图案）会误检。想要更准的人脸框，用存量的 daft.aihc.functions.image.image_face_detect.ImageFaceDetect（insightface，可用 GPU），本算子的定位是轻量 CPU 统计。
- 调大 min_size 能压掉一部分小尺寸误检，但同时会丢掉远处的真脸，改之前先在自己的数据上比对。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_face_count import ImageFaceCount

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "face_count",
        aihc_udf(
            ImageFaceCount,
            construct_args={
                "image_src_type": "image_url",
                "scale_factor": 1.1,
                "min_neighbors": 3,
            },
            num_cpus=1,
            concurrency=4,
            batch_size=32,
        )(col("image")),
    )
    ds = ds.where(col("face_count") == 1)
    ds.show()
```

图像人体姿态估计（DWPose）

简介

图像人体全身关键点估计算子，走两段式 ONNX 推理链路：yolox_l 人体检测 + dw-ll_ucoco_384（RTMPose 架构的 COCO-WholeBody 模型），复用仓库已有的 embodied/perception/dwpose_func.py，与 VideoWholeBodyPoseEstimation 共用同一套权重，不引入 mmpose / mmcv / mmdet。

功能描述

- 两段式推理：先用 yolox_l 检测人体（输入 640×640，NMS IoU 0.45，只保留 class 0 且分数 > 0.3 的框，框为原图像素坐标的 xyxy），再对每个框跑 DWPose 关键点
- 关键点布局是 134 点：COCO-WholeBody 133 点在 index 17 处插入 neck（两肩中点），并把前若干点按 openpose 顺序重排；keypoint_layout 字段固定为 dwpose_coco_wholebody_134，下游按该布局取点
- 检测不到人体时按「无人」返回空结构：底层 Wholebody 在无框时会退化成对整帧做单框推理，直接采信会得到虚假关键点
- kpt_thr > 0 时把分数低于阈值的关键点坐标置 0（keypoint_scores 原样保留），默认 0 表示不过滤
- bbox_scores 恒为空列表：检测后处理只回传过滤后的框、不回传分数；不输出点位名，改用 keypoint_layout 标明布局
- 图像统一转 RGB 后按 BGR ndarray 送前处理（与 VideoWholeBodyPoseEstimation 口径一致）
- 构造期校验 image_src_type 合法、kpt_thr ∈ [0, 1]，并检查两个 ONNX 权重都存在
- GPU 需要 num_gpus > 0 且 onnxruntime 报告 CUDAExecutionProvider 可用，否则回落 CPU
- 输入为 null 或行级异常时返回空结构（前四个字段为空列表 + keypoint_layout 字符串），不中断整列

算子参数

输入

输入	含义
image	图像输入，内容类型由 image_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制数据）

输出

输出列类型为 struct，字段如下：

输出	含义
pose.keypoints	list<list<list>>：外层=人，中层=关键点（134 个），内层=[x, y] 原图像素坐标
pose.keypoint_scores	list<list>：与 keypoints 同形状（去掉最内层），每人每点的置信度
pose.bboxes	list<list>：人体检测框，每行一个框（xyxy，原图像素坐标）
pose.bbox_scores	list：恒为空列表（DWPose 检测后处理不回传分数）
pose.keypoint_layout	large_string：固定 "dwpose_coco_wholebody_134"

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model/dwpose"	存放两个 ONNX 权重的目录
det_onnx	str	"yolox_l.onnx"	人体检测 ONNX 文件名
pose_onnx	str	"dw-ll_ucoco_384.onnx"	姿态 ONNX 文件名（COCO-WholeBody）
kpt_thr	float	0.0	关键点置信度阈值，取值 [0, 1]，低于该值的点坐标置 0；0 表示不过滤
rank	int	0	多卡场景 worker 序号（onnxruntime CUDA provider 用）

注意事项

- 检测与姿态两个 ONNX 权重（`det_onnx`、`pose_onnx`）必须放在同一个 `model_path` 目录下，任一缺失都在构造期抛 `FileNotFoundError`，不会自动联网下载。
- 用 GPU 需要装 `onnxruntime-gpu`。CUDA provider 起不来时算子会静默回落 CPU（速度差一个量级）；常见原因是 `LD_LIBRARY_PATH` 没带 pip 安装的 nvidia 库路径（缺 libcufft 等），而不是没装库。
- `bbox_scores` 恒空，需要按检测置信度过滤的场景拿不到分数。
- 图上没有人时 `keypoints` / `bboxes` 都是空列表，属正常返回而非失败。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_mmpose import ImageMmpose

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "pose",
        aihc_udf(
            ImageMmpose,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models/dwpose",
                "det_onnx": "yolox_l.onnx",
                "pose_onnx": "dw-ll_ucoco_384.onnx",
                "kpt_thr": 0.0,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像背景移除

简介

图像抠背景算子：用 u2net 显著性模型把主体从背景中抠出，输出带 alpha 通道的 PNG。直接用 onnxruntime 跑 u2net ONNX，不引入 `rembg` 依赖，显著性推理与抠图合成逻辑都实现在算子内。

功能描述

- 前处理：图 LANCZOS resize 到 320×320，先整体除以像素最大值，再按 `mean=(0.485,0.456,0.406)` / `std=(0.229,0.224,0.225)` 归一化，NCHW float32 送模型
- 后处理：取模型第一个输出做 min-max 归一化得到显著性 mask（输出全常量时 mask 全 0），LANCZOS resize 回原图尺寸，作为 alpha 通道合成 RGBA
- `bgcolor` 非空时把抠图 `alpha_composite` 到该 RGBA 纯色底上，输出不再透明；`bgcolor` 必须是 4 元组，否则构造期报错
- `alpha_matting=True` 走 pymatting 的 `estimate_alpha_cf` + `estimate_foreground_ml`（trimap 由前景/背景两个阈值裁出）；镜像未装 pymatting 时构造期直接抛 `ValueError`，不静默降级
- 不用 rembg 的原因：rembg >= 2.0.55 都要求 `pillow>=12.1`，而镜像里是 Pillow 11.3，装它会顶掉几乎所有图像/视频算子共用的 PIL；u2net 本身就是一个 ONNX 模型，onnxruntime 已在镜像内
- 权重只从本地 `{model_path}/{model_name}.onnx` 读，缺失时构造期抛 `FileNotFoundError`，不联网下载
- 每行输入先落到 `/tmp` 下的临时目录再处理，行处理结束自动清理；`image_url` 且是本地路径时不复制，直接读原文件
- 输出统一是 PNG，文件名 `<本地文件 stem>_nobg.png`：本地路径输入取原图 stem，`bos://` / `s3://` 输入取 `image_<行内下标>_0`，base64 / binary 输入取 `<unix 时间戳>_<行内下标>_0`
- GPU 需要 `num_gpus > 0` 且 onnxruntime 报告 `CUDAExecutionProvider` 可用，否则用 CPU（u2net 在 CPU 上可用，回归测试即 CPU 跑）
- 行级失败（输入为 null、下载或解码失败等）返回 None 并计入内部 failed 计数，不中断整列

算子参数

输入

输入	含义
image	图像输入，内容类型由 image_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制数据）

输出

输出	含义
cutout	large_string：抠图后 PNG 的路径（output_bosdir 非空时为 BOS 路径），bgcolor 为 None 时是 RGBA 透明底；该行失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"rembg/u2net"	相对 model_path 的权重名，不含 .onnx 后缀
alpha_matting	bool	False	是否启用 alpha matting 细化边缘，需要 pymatting
alpha_matting_foreground_threshold	int	240	alpha matting 前景阈值，mask 高于该值置 255
alpha_matting_background_threshold	int	10	alpha matting 背景阈值，mask 低于该值置 0
alpha_matting_erode_size	int	10	alpha matting 腐蚀核大小（接口保留，当前抠图路径未使用）
bgcolor	list[int] 或 None	None	输出背景色 RGBA 四元组；None 表示保留透明
output_dir	str	"/tmp/aihc_image_remove_background"	本地输出目录
output_bosdir	str	""	非空时把结果上传到该 BOS 目录，并返回 BOS 路径

注意事项

- `model_name` **不要带** `.onnx` **后缀**：传成 `rembg/u2net.onnx` 会多拼一次后缀而找不到权重文件。
- `alpha_matting_erode_size` 只是保留的参数位，当前实现的 trimap 只用前景/背景两个阈值，没有做腐蚀，改这个值不影响结果。
- 输出文件名只由输入文件 stem 决定：不同目录下同名的本地图片会写到同一个输出路径而互相覆盖，批量场景建议按批设置不同的 `output_dir`。
- 需要不透明结果时传 `bgcolor`（例如 `[255, 255, 255, 255]`），否则下游若按 RGB 打开 PNG 会得到黑色背景。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_remove_background import ImageRemoveBackground

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "cutout",
        aihc_udf(
            ImageRemoveBackground,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "rembg/u2net",
                "output_dir": "/tmp/aihc_image_remove_background",
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像分割（FastSAM）

简介

FastSAM 全图分割算子，用 ultralytics 的 FastSAM 做 segment-anything，返回每张图分割出的目标框。只保留框、不保留掩码，避免长列掩码撑爆内存。

功能描述

- 框取 ultralytics 的 `boxes.xywh`，即中心点 + 宽高，单位是原图像素；输出为 struct，并额外带回 `conf`，便于下游按置信度过滤
- 图像统一转 RGB 后按 ndarray 调 FastSAM，推理参数 `imgsz` / `conf` / `iou` / `retina_masks` 由构造参数控制，固定 `verbose=False`
- 设备：`num_gpus > 0` 且 `torch.cuda.is_available()` 时取整数卡号 `rank % 可见卡数`，否则 `"cpu"`
- 未分割出任何目标、或输入为 null 时，`bbox` 与 `conf` 都是空列表
- 行级异常默认记日志并返回空结构；`fail_on_error=True` 时直接抛出，让整个任务失败
- 权重只从本地 `{model_path}/{model_name}` 加载，缺失时构造期抛 `FileNotFoundError`，不联网下载
- 只输出框，不输出掩码

算子参数

输入

输入	含义
image	图像输入，内容类型由 image_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制数据）

输出

输出列类型为 struct，字段如下：

输出	含义
segments.bbox	list<list>：M×4 目标框，每行是 [cx, cy, w, h]（xywh，原图像素坐标）
segments.conf	list：与 bbox 等长的置信度

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"FastSAM-x.pt"	相对 model_path 的 FastSAM 权重文件名
imgsz	int	1024	推理分辨率
conf	float	0.05	置信度阈值
iou	float	0.5	NMS IoU 阈值
retina_masks	bool	True	是否用高分辨率掩码（影响框精度）
fail_on_error	bool	False	单行异常时是否直接抛出而不是返回空结构
rank	int	0	多卡场景 worker 序号，GPU 卡号取 rank % 可见卡数

注意事项

- `conf` 默认 0.05，阈值很低、出框多，下游按 `conf` 自行过滤。
- 输出只有框、没有掩码，需要像素级掩码的场景本算子不适用。
- `num_gpus=0` 时整图 1024 分辨率的 FastSAM 推理跑在 CPU 上，吞吐会明显下降。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_segment import ImageSegment

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "segments",
        aihc_udf(
            ImageSegment,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "FastSAM-x.pt",
                "imgsz": 1024,
                "conf": 0.05,
                "iou": 0.5,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图像宽高比

简介

图像宽高比算子，输出 `width / height` 比值。算子只产出比值，不按 min/max 区间过滤，过滤交给 pipeline。

功能描述

- 只用 PIL `Image.open` 读图像头部取 `image.size`，不做全图解码；纯 CPU，无模型权重
- 支持三种输入形式：`image_url`（本地路径 / file:// / http(s) / bos:// / s3://，远端路径先下载到临时目录）、`image_base64`（允许带 data URI 前缀，按第一个逗号切掉）、`image_binary`
- 输出 float64，口径是宽 / 高；取的是文件记录的原始像素尺寸，不做 EXIF 旋转矫正
- 高为 0 时该行返回 None
- 单行失败（读不到文件、非图像数据、输入类型与 `image_src_type` 不匹配）打 exception 日志后该行填 None，不中断整批

- 一行一张图，只产标量比值，不按 any/all 策略决定样本去留

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（路径/URL 字符串、base64 字符串、bytes）

输出

输出	含义
aspect_ratio	float64，宽 / 高。高为 0 或该行处理失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像编码形式：image_url / image_base64 / image_binary，取值非法在初始化阶段抛 ValueError

注意事项

- 算子只产出统计字段，不做过滤。等价过滤要自己在 pipeline 里写，常用区间是 min_ratio=0.333、max_ratio=3.0（见调用示例里的 where）。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_aspect_ratio import ImageAspectRatio

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "aspect_ratio",
        aihc_udf(
            ImageAspectRatio,
            construct_args={"image_src_type": "image_url"},
            num_cpus=1,
            concurrency=4,
            batch_size=64,
        )(col("image")),
    )
    ds = ds.where((col("aspect_ratio") > 0.333) & (col("aspect_ratio") < 3.0))
    ds.show()
```

图像人脸占比

简介

图像人脸占比算子，用 Haar 级联检测器检出图像中的人脸，输出**最大**人脸面积 / 图像面积。算子只产出占比字段，过滤交给 pipeline。

功能描述

- 检测器与 图像人脸数量（ImageFaceCount）共用一套代码：cv2.CascadeClassifier + OpenCV 自带的 haarcascade_frontalface_alt.xml，随 opencv 包安装，不需要纳管权重
- 只取**最大**一张脸的面积，不是所有脸面积之和；多脸图上小脸不影响结果
- 无人脸时返回 0.0，而不是 None
- 图像面积由 PIL 的 image.width * image.height 得到，取值范围 [0, 1]
- 纯 CPU，scale_factor / min_neighbors / min_size / max_size 直接透传 detectMultiScale，None 的可选项不传
- 面积按 detectMultiScale 返回的原始框计算，不额外把框裁剪到图像边界内。Haar 的检测窗口来自图内滑窗，正常图上该裁剪本就是空操作
- 级联 xml 加载失败在**初始化**阶段抛 FileNotFoundError；单行失败打 exception 日志后该行填 None，不中断整批

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（路径/URL 字符串、base64 字符串、bytes）

输出

输出	含义
face_ratio	float64，最大人脸面积 / 图像面积，取值 [0, 1]；无人脸为 0.0，该行处理失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像编码形式：image_url / image_base64 / image_binary
cv_classifier	str	""	Haar 级联 xml 路径，空字符串用 OpenCV 自带的 haarcascade_frontalface_alt.xml
scale_factor	float	1.1	detectMultiScale 的 scaleFactor
min_neighbors	int	3	detectMultiScale 的 minNeighbors
min_size	tuple[int,int] 或 None	None	最小人脸尺寸，None 表示不限制（不传 minSize）
max_size	tuple[int,int] 或 None	None	最大人脸尺寸，None 表示不限制（不传 maxSize）

注意事项

- 算子只产出统计字段，不做过滤。常用的保留区间是 face_ratio 落在 0.0 ~ 0.4（滤掉大头特写），条件要自己在 pipeline 里写。
- 0.0 有两种含义：确实没检出脸，或者脸太小/太侧被 Haar 漏掉。要区分「无脸」与「漏检」得配合 图像人脸数量 或更强的检测器。
- Haar 正脸级联对纹理密集区域（布面印花、卡通图案）会误检，误检框往往不小，会把 face_ratio 抬高。要更准的人脸框用存量的 daft.aihc.functions.image.image_face_detect.ImageFaceDetect（insightface，可用 GPU）。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_face_ratio import ImageFaceRatio

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "face_ratio",
        aihc_udf(
            ImageFaceRatio,
            construct_args={
                "image_src_type": "image_url",
                "scale_factor": 1.1,
                "min_neighbors": 3,
            },
            num_cpus=1,
            concurrency=4,
            batch_size=32,
        )(col("image")),
    )
    ds = ds.where(col("face_ratio") <= 0.4)
    ds.show()
```

图像宽高

简介

图像宽高算子：读取图像的原始像素宽高，以一个 struct（width + height）输出，逐行产出，按尺寸过滤交给 pipeline。

功能描述

- 只用 PIL Image.open 读图像头部取 image.size，不做全图解码；纯 CPU，无模型权重
- 支持三种输入形式：image_url（本地路径 / file:// / http(s) / bos:// / s3://，远端路径先下载到临时目录）、image_base64（允许带 data URI 前缀）、image_binary
- 宽高写在同一个 struct 里，下游用 col("shape")["width"] / col("shape")["height"] 取字段
- 取文件记录的原始像素尺寸，不做 EXIF 旋转矫正
- 单行失败（读不到文件、非图像数据、输入类型与 image_src_type 不匹配）打 exception 日志后整条 struct 填 None，不中断整批；不会出现 width 有值而 height 为 None 的半残行
- 一行一张图，逐行产出该图的宽高

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（路径/URL 字符串、base64 字符串、bytes）

输出

输出	含义
shape	struct，处理失败时整条为 None
shape.width	int32，图像宽（像素）
shape.height	int32，图像高（像素）

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像编码形式：image_url / image_base64 / image_binary，取值非法在初始化阶段抛 ValueError

注意事项

- 本算子只产出统计字段，不做过滤。按宽高的区间筛选（如 `width >= 1`、`height >= 1`，上界不限）要自己在 pipeline 里写。
- 输出是静态 schema 的 struct，字段固定为 `width` + `height`，不要指望后续版本往里加字段。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_shape import ImageShape

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {"image": ["bos://your-bucket/sample.jpg"]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "shape",
        aihc_udf(
            ImageShape,
            construct_args={"image_src_type": "image_url"},
            num_cpus=1,
            concurrency=4,
            batch_size=64,
        )(col("image")),
    )
    ds = ds.with_column("width", col("shape")["width"])
    ds = ds.with_column("height", col("shape")["height"])
    ds.show()
```

音频

音频格式标准化处理器

简介

音频格式标准化处理器

功能描述

- 支持采样率重采样
- 支持声道统一（如转为单声道）
- 支持响度归一化（目标 dBFS，带限制增益范围）
- 默认输入输出音频为字节（bytes）格式
- 支持本地目录, BOS目录

算子参数

输入

输入	含义
audio_col	输入音频的字节数组

输出

输出	含义
standardized_audio	处理后的音频结果；失败则返回 None

参数

参数名称	类型	默认值	描述
target_sr	int or None	None	目标采样率（单位 Hz），如 16000，为 None 时保留原始采样率
target_channels	int or None	None	目标声道数，如 1 表示单声道，为 None 时保留原始声道数
target_dbfs	float or None	None	目标响度（单位 dBFS），为 None 时不进行响度归一化
target_gain_range	list	[-3, 3]	归一化时允许的增益范围，如 [-3, 3]

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_standardization import AudioStandardization

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_path": [
            "file:///local/sample_1.wav",
            "file:///mnt/pfs/sample_2.wav",
            "file:///mnt/bos/sample_3.wav",
        ]
    }

    df = daft.from_pydict(samples)
    df = df.with_column(
        "standardized_audio",
        aihc_udf(
            AudioStandardization,
            construct_args={
                "target_sr": 16000,
                "target_channels": 1,
                "target_dbfs": -20.0,
                "target_gain_range": [-3.0, 3.0],
            },
            num_cpus=1,
            batch_size=1,
            concurrency=2,
        )(col("audio_path")),
    )

    df.show()
```

音频格式转换处理器

简介

音频格式转换处理器

功能描述

- 支持音频格式转换（WAV、MP3、FLAC）
- 自动选择合适的编码器
- 通过extra_params支持自定义ffmpeg参数
- 支持本地文件, BOS地址存储

算子参数

输入

输入	含义
input_col	包含输入音频路径的数组（支持本地路径）
output_col	输出音频路径的数组（支持本地路径）

输出

输出	含义
convert_result	包含转换结果路径的数组，成功返回输出路径，失败返回None

参数

参数名称	类型	默认值	描述
output_format	str	None	输出音频格式，仅支持 "wav", "mp3", "flac"，默认为 "wav"
timeout	int or None	None	ffmpeg 执行超时时间（秒），默认为 None（无超时）
extra_params	list or None	None	额外的 ffmpeg 参数列表，直接拼接到命令中 例如： - 音轨选择: ["-map", "0:a"] # 选择所有音轨 - 采样率: ["-ar", "44100"] - 比特率: ["-b:a", "192k"] # 适用于 MP3 - 压缩级别: ["-compression_level", "8"] # 适用于 FLAC

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_convert import AudioConvert

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "input_path": [
            "file:///local/sample_1.wav",
            "file:///mnt/pfs/sample_2.wav",
            "file:///mnt/bos/sample_3.wav",
        ]
        "output_path": [
            "file:///local/output/sample_1.map3",
            "file:///mnt/pfs/output/sample_2.mp3",
            "file:///mnt/bos/output/sample_3.mp3",
        ],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_format": "mp3",
        "sample_rate": 44100,
        "audio_map": "auto",
        "extra_params": ["-c:a", "libmp3lame", "-b:a", "192k", "-q:a", "2"],
    }
    ds = ds.with_column(
        "convert_result",
        aihc_udf(
            AudioConvert,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("input_path"), col("output_path")),
    )
    ds.show()
```

音频时长分析处理器

音频转MP3

音频转MP3

简介 音频转 MP3 处理器，将各种格式的音频文件转换为 MP3 格式，基于 FFmpeg 实现。

功能描述

- 支持将 WAV、FLAC、AAC、OGG 等音频格式转换为 MP3
- 可配置码率（bitrate）、采样率（sample_rate）和质量系数（quality）
- 支持通过 extra_params 传递自定义 FFmpeg 参数
- 支持本地路径、BOS、S3 等存储路径（自动下载后转换再上传）
- 失败时输出列返回 None

算子参数 输入 |输入|含义| |-|-| |input_col|输入音频文件路径数组（支持本地路径、BOS、S3）| |output_col|输出 MP3 文件路径数组（支持本地路径、BOS、S3）|

输出 |输出|含义| |-|-| |convert_result|转换后的 MP3 文件路径，失败时返回 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |bitrate|str|"192k"|MP3 输出码率，如 "128k"、"192k"、"320k"| |sample_rate|int 或 None|None|输出采样率（Hz），None 表示保留原始采样率| |quality|int|2|libmp3lame 质量系数（0-9，值越小质量越高），对应 FFmpeg -q:a 参数| |extra_params|list[str] 或 None|None|额外的 FFmpeg 参数列表，拼接在编码参数之后| |timeout|int 或 None|None|FFmpeg 执行超时时间（秒），None 表示不限制|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_convert_to_mp3 import AudioConvertToMp3

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "input_path": [
            "/tmp/test_audio1.wav",
            "/tmp/test_audio2.flac",
        ],
        "output_path": [
            "/tmp/test_output1.mp3",
            "/tmp/test_output2.mp3",
        ],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "bitrate": "192k",
        "sample_rate": 44100,
        "quality": 2,
    }
    ds = ds.with_column(
        "convert_result",
        aihc_udf(
            AudioConvertToMp3,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("input_path"), col("output_path")),
    )
    ds.show()
```

音频静音检测

音频静音检测

简介 音频静音检测算子，通过 FFmpeg volumedetect 滤镜分析音频的最大音量，判断是否为静音文件。

功能描述

- 使用 FFmpeg volumedetect 滤镜分析音频的最大音量（max_volume，单位 dBFS）
- 当最大音量低于设定阈值时判定为静音
- 支持本地路径和远程路径（自动下载后分析）
- 支持直接传入二进制音频数据
- 分析失败时返回 None

算子参数 输入 |输入|含义| |-|-| |audio_inputs|音频输入数组，支持本地文件路径（str）或音频二进制数据（bytes）|

输出 |输出|含义| |-|-| |is_silence|布尔值，True 表示为静音，False 表示非静音，失败时返回 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |silence_threshold_db|float|-60.0|静音判断阈值（dBFS），最大音量 ≤ 此值时判定为静音。典型值：-60.0（极静）、-40.0（较静）| |timeout|int 或 None|None|FFmpeg 执行超时时间（秒），None 表示不限制|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_silence_detection import AudioSilenceDetection

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_path": [
            "/tmp/test_speech1.mp3",
            "/tmp/test_silence.wav",
            "/tmp/test_music.mp3",
        ]
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "silence_threshold_db": -50.0,
        "timeout": 30,
    }
    ds = ds.with_column(
        "is_silence",
        aihc_udf(
            AudioSilenceDetection,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=16,
        )(col("audio_path")),
    )
    # 过滤掉静音文件
    ds = ds.where(col("is_silence") == False)
    ds.show()
```

音频片段切分-时间戳

音频片段切分(时间戳)

简介 音频片段切分处理器，按给定的时间戳区间列表将音频文件分割为多个片段，基于 FFmpeg 实现。

功能描述

- 根据传入的时间戳区间 [[start, end], ...] 精确切分音频片段
- 每条数据可对应不同的时间戳区间列表，适用于语音识别后的精细切分场景
- 支持本地路径输入或二进制数据输入
- 可选择同时返回片段的二进制内容
- 可指定输出格式，默认保留原始格式（codec copy 模式）
- 可将片段上传至 BOS 存储目录

算子参数 输入 |输入|含义| |-|-| |timestamp_ranges|时间戳区间数组（必填），每行为 [[start1, end1], [start2, end2], ...] 格式（秒）| |audio_paths|音频文件路径数组（可选，与 audio_binaries 二选一）| |audio_binaries|音频二进制数据数组（可选，与 audio_paths 二选一）| |audio_formats|音频格式标识数组，如 "mp3"（binary 输入时需提供）| |output_basenames|输出子目录名数组（可选）|

输出 |输出字段|类型|含义| |-|-|-| |segments|list[string]|切分后各片段的文件路径列表| |binaries|list[binary]|切分后各片段的二进制内容（仅 output_segments_binary=True 时有值）|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_bos_dir|str|""|片段上传目标 BOS 目录，为空则保存到本地| |output_segments_binary|bool|False|是否同时返回片段的二进制内容| |output_format|str 或 None|None|输出格式，None 表示保留原始格式（codec copy）|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_split_by_timestamps import AudioSplitByTimestamps

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_path": [
            "/tmp/test_interview1.mp3",
            "/tmp/test_interview2.mp3",
        ],
        # 每条数据对应不同的时间戳区间
        "timestamp_ranges": [
            [[0.0, 5.2], [8.1, 15.6], [20.0, 30.0]],
            [[0.5, 10.0], [12.3, 25.8]],
        ],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_segments_binary": False,
    }
    ds = ds.with_column(
        "split_result",
        aihc_udf(
            AudioSplitByTimestamps,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
            batch_size=4,
        )(col("timestamp_ranges"), col("audio_path")),
    )
    ds = ds.with_column("segments", col("split_result")["segments"])
    ds.show()
```

音频片段切分-时长

音频片段切分(时长)

简介 音频片段切分处理器，按固定时长将音频文件分割为多个片段，基于 FFmpeg 实现。

功能描述

- 按指定时长（秒）均匀分割音频文件
- 支持设置最小片段时长，过滤过短的尾部片段
- 支持本地路径输入或二进制数据输入
- 可选择同时返回片段的二进制内容
- 可指定输出格式，默认保留原始格式（codec copy 模式，速度快）
- 可将片段上传至 BOS 存储目录

算子参数 输入 |输入|含义| |-|-| |audio_paths|音频文件路径数组（可选，与 audio_binaries 二选一）| |audio_binaries|音频二进制数据数组（可选，与 audio_paths 二选一）| |audio_formats|音频格式标识数组，如 "mp3"、"wav"（当使用 binary 输入时需提供）| |output_basenames|输出子目录名数组（可选，不填则自动从文件名推断）|

输出 |输出字段|类型|含义| |-|-|-| |segments|list[string]|切分后各片段的文件路径列表| |binaries|list[binary]|切分后各片段的二进制内容（仅 output_segments_binary=True 时有值）|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |segment_duration|float|5.0|每个片段的时长（秒）| |min_segment_duration|float|0.0|最小片段时长（秒），短于此值的片段将被丢弃| |output_bos_dir|str|""|片段上传目标 BOS 目录（如 "bos://bucket/segments"），为空则保存到本地| |output_segments_binary|bool|False|是否在输出中同时返回片段的二进制内容| |output_format|str 或 None|None|输出格式，如 "mp3"、"wav"，None 表示保留原始格式（codec copy）|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_split_by_duration import AudioSplitByDuration

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_path": [
            "/tmp/test_audio1.mp3",
            "/tmp/test_audio2.wav",
        ]
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "segment_duration": 10.0,
        "min_segment_duration": 1.0,
        "output_segments_binary": False,
    }
    ds = ds.with_column(
        "split_result",
        aihc_udf(
            AudioSplitByDuration,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
            batch_size=4,
        )(col("audio_path")),
    )
    ds = ds.with_column("segments", col("split_result")["segments"])
    ds.show()
```

音频元数据提取

音频元数据提取

简介

音频元数据提取算子，基于 ffprobe 从音频文件中提取时长、格式、码率、编码器、采样率、声道数等元信息。

功能描述

- 使用 ffprobe 解析音频文件的 Format 和 Stream 信息
- 提取音频时长、格式名称、总码率
- 提取音频编码器、采样率、声道数
- 支持本地路径和 BOS 对象存储路径（通过预签名 URL 直接 probe，无需下载）
- 失败时返回 None，不影响整批数据处理

算子参数 输入 |输入|含义| |-|-| |input_col|音频文件路径数组，支持本地路径、BOS（bos://）、S3（s3://）路径|

输出 |输出字段|类型|含义| |-|-|-| |duration|float64|音频时长（秒）| |format_name|string|音频格式名称（如 mp3、wav、flac）| |bit_rate|int64|总码率（bps）| |audio_codec|string|音频编码器（如 mp3、aac、pcm_s16le）| |audio_sample_rate|int32|采样率（Hz）| |audio_channels|int32|声道数|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |timeout|int 或 None|None|ffprobe 执行超时时间（秒），None 表示不限制|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_metadata_extract import AudioMetadataExtract

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_path": [
            "/tmp/test_audio1.mp3",
            "/tmp/test_audio2.wav",
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "metadata",
        aihc_udf(
            AudioMetadataExtract,
            construct_args={"timeout": 30},
            num_cpus=1,
            concurrency=4,
            batch_size=16,
        )(col("audio_path")),
    )
    ds.show()
    # 访问具体字段
    ds = ds.with_column("duration", col("metadata").struct.get("duration"))
    ds = ds.with_column("audio_codec", col("metadata").struct.get("audio_codec"))
    ds.show()
```

音频滤镜处理

音频滤镜处理

简介 音频滤镜处理算子，通过 FFmpeg 的 audio filter 机制对音频施加任意滤镜（音量调整、高通/低通滤波、降噪等），适用于音频预处理场景。

功能描述

- 封装 FFmpeg audio filter 接口，支持任意 FFmpeg 内置音频滤镜
- 支持 volume、highpass、lowpass、equalizer、aecho 等常用滤镜
- 支持本地路径输入或二进制数据输入
- 可选择同时返回处理后音频的二进制内容
- 可指定输出音频格式，默认保留原始格式
- 可将处理结果上传至 BOS 存储目录

算子参数 输入 |输入|含义| |-|-| |audio_paths|音频文件路径数组（可选，与 audio_binaries 二选一）| |audio_binaries|音频二进制数据数组（可选，与 audio_paths 二选一）| |audio_formats|音频格式标识数组，如 "mp3"（binary 输入时需提供）| |output_basenames|输出文件基名数组（可选）|

输出 |输出字段|类型|含义| |-|-|-| |processed_audio_path|string|处理后的音频文件路径| |processed_audio_binary|binary|处理后的音频二进制内容（仅 output_audio_binary=True 时有值）|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |filter_name|str|（必填）|FFmpeg 音频滤镜名称，如 "volume"、"highpass"、"lowpass"、"aecho"| |filter_kwargs|dict 或 None|None|传给滤镜的参数字典，如 {"volume": 1.5}、{"frequency": 200}| |global_args|list[str] 或 None|None|FFmpeg 全局参数列表（如 ["-loglevel", "quiet"]）| |output_bos_dir|str|""|输出上传目标 BOS 目录，为空则保存到本地| |output_audio_binary|bool|False|是否同时返回处理后音频的二进制内容| |output_audio_format|str 或 None|None|输出音频格式，None 表示保留原始格式|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_ffmpeg_wrapped import AudioFFMPEGWrapped

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_path": [
            "/tmp/test_speech1.mp3",
            "/tmp/test_speech2.mp3",
        ]
    }
    ds = daft.from_pydict(samples)

    # 示例：将音量放大 1.5 倍
    constructor_kwargs = {
        "filter_name": "volume",
        "filter_kwargs": {"volume": 1.5},
        "output_audio_binary": False,
    }
    ds = ds.with_column(
        "processed",
        aihc_udf(
            AudioFFMPEGWrapped,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
            batch_size=4,
        )(col("audio_path")),
    )
    ds = ds.with_column("processed_path", col("processed")["processed_audio_path"])
    ds.show()
```

音频拼接

音频拼接

简介 音频拼接处理器，将多个音频文件按顺序拼接为一段音频，基于 FFmpeg concat 滤镜实现。

功能描述

- 支持将多个不同格式的音频文件拼接为一段输出音频
- 使用 FFmpeg concat 滤镜，强制重新编码以保证格式统一
- 支持输出为 WAV、MP3、FLAC 格式
- 支持本地路径和远程路径（自动下载到本地后拼接）
- 支持通过 extra_params 传递自定义 FFmpeg 参数（如音量归一化）

算子参数 输入 |输入|含义| |-|-| |audio_paths_list|音频路径列表的数组，每行为一个 list[str]，表示该行要拼接的多个音频文件路径| |output_col|输出文件路径数组（支持本地路径、BOS、S3）|

输出 |输出|含义| |-|-| |concat_result|拼接后的音频文件路径，失败时返回 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_format|str|"mp3"|输出音频格式，支持 "wav"、"mp3"、"flac"| |sample_rate|int|16000|输出采样率（Hz）| |timeout|int 或 None|None|FFmpeg 执行超时时间（秒），None 表示不限制| |extra_params|list[str] 或 None|None|额外的 FFmpeg 参数列表|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_concat import AudioConcat

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "audio_paths": [
            ["/tmp/test_part1.mp3", "/tmp/test_part2.mp3", "/tmp/test_part3.mp3"],
            ["/tmp/test_segA.wav", "/tmp/test_segB.wav"],
        ],
        "output_path": [
            "/tmp/test_merged1.mp3",
            "/tmp/test_merged2.mp3",
        ],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_format": "mp3",
        "sample_rate": 16000,
    }
    ds = ds.with_column(
        "concat_result",
        aihc_udf(
            AudioConcat,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
            batch_size=4,
        )(col("audio_paths"), col("output_path")),
    )
    ds.show()
```

音频信噪比计算

简介 音频信噪比（SNR）计算器，基于非负矩阵分解（NMF）进行信号-噪声分离

功能描述

- 对音频进行 STFT 频谱分析；
- 使用 NMF 将频谱分解为信号与噪声分量；
- 基于重建的时域信号与噪声估计能量比值，计算 SNR（单位 dB）。
- 常见音频格式：mp3, wav, flac, ogg, aac, m4a
- 支持本地路径与对象存储路径（bos:// 或 s3://）

算子参数 输入 |输入|含义| |-|-| |audio_paths|音频路径数组（支持本地与对象存储路径） - 本地路径：绝对路径或相对路径 - 对象存储：以 "bos://" 或 "s3://" 开头的路径|

输出 |输出|含义| |-|-| |result|pa.Array: 浮点数组（float64），每行为对应音频的 SNR 值（单位 dB）。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |n_components|int|2|NMF 分解的组件数量 默认值：2（通常认为第一个分量为信号、其余分量为噪声）| |max_iter|int|200|NMF 最大迭代次数 默认值：200|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_snr import AudioSNR

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audio_paths": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "n_components": 2,
        "max_iter": 200,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioSNR,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audio_paths")),
    )
    ds.show()
```

音频字节大小

简介 音频文件元数据分析处理器，精确计算文件大小

功能描述

- 精确计算音频文件字节大小
- 支持本地文件与BOS存储
- 轻量高效，适合批量处理
- MP3 (.mp3)
- WAV (.wav)
- FLAC (.flac)
- OGG (.ogg)
- AAC (.aac)
- M4A (.m4a)

算子参数 输入 |输入|含义| |-|-| |audio_paths|存放音频路径的列|

输出 |输出|含义| |-|-| |result|存放音频大小的列|

参数 无

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_size import AudioSize

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "audio_paths": ["bos://bucket/test/aihc/test_data/test_audio.wav"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioSize,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audio_paths")),
    )
    ds.show()
```

语音活动检测

简介 语音端点检测模块 - 基于 Silero VAD 的高效音频分割解决方案

功能描述

- 语音端点检测：自动识别音频中的语音片段起止时间，实现语音与静音的精准分割
- 批量处理：支持大批量音频数据的高效端点检测
- 多格式输入：兼容原始二进制、Base64 编码、BOS/HTTP 链接等多种音频输入方式
- GPU 加速：支持 GPU 环境下的高性能推理
- 通用性强：在处理不同领域、存在各种背景噪声和质量水平的音频时表现优异
- 建议输入 8k/16k 采样率、单声道的 WAV 格式音频，提升检测准确率
- GPU提速不明显，在不要求极致性能的场景中，可以使用CPU推理onnx模型
- 适用于语音活动检测、语音切分等场景
- silero_vad.onnx (对应 onnx_model_revision=16)
- silero_vad_16k_op15.onnx (对应 onnx_model_revision=15)
- silero_vad.jit
- 每条音频输出二维浮点数列表，表示所有语音片段的起止时间戳（单位：秒），如：[[0.0, 4.34], [5.50, 7.12]]
- 处理失败时返回 None

算子参数 输入 |输入|含义| |-|-| |videos|包含音频数据的列，支持以下格式： - audio_base64: base64 编码的音频字符串 - audio_url: 音频文件的 URL 路径 - audio_binary: 原始音频字节数据|

输出 |输出|含义| |-|-| |result|包含语音端点时间戳的列，每个元素为二维浮点数列表，表示音频中各语音片段的起止时间戳（单位：秒），例如：[[0.0, 4.34], [5.50, 7.12]]。若处理失败则为 None。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |audio_src_type|str|必填|音频格式类型 支持的音频格式类型，包含： - bos/http 地址(audio_url) - base64 编码(audio_base64) - 二进制流(audio_binary) 可选值：["audio_binary", "audio_url", "audio_base64"]| |model_path|str|'/opt/aihc/models'|模型路径 本地模型存储路径。 默认值："/opt/aihc/models"| |model_name|str|'silero-vad'|模型名称 使用的模型名称，包含： - silero-vad 可选值：["silero-vad"] 默认值："silero-vad"| |use_onnx_model|bool|True|是否使用onnx模型 默认值：True| |onnx_model_revision|int|16|onnx模型版本 可选值：[16, 15] 默认值：16|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_vad_silero import AudioVadSilero

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"videos": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'silero-vad',
        "use_onnx_model": True,
        "onnx_model_revision": 16,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioVadSilero,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("videos")),
    )
    ds.show()
```

多语言 CTC 对齐

简介 音频 CTC 对齐算子，用于将音频与文本按时间戳实施对齐，目前支持中文和英文。

算子参数 输入 |输入|含义| |-|-| |audios|包含音频数据的数组，支持以下格式： *audio_url: 音频文件 URL 路径（支持 HTTP/BOS/S3 等协议 URL，以及本地文件路径）；* audio_binary: 原始音频字节数据| |texts|与音频内容对应的文本列表| |langs|音频文本对应的语言，目前仅支持 "en" (英文) 和 "zh" (中文)|

输出 |输出|含义| |-|-| |result|算子输出|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|模型path| |model_name|str|'MMS/ctc_alignment_mling_uroman_model.pt'|模型名称|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_ctc_aligner import AudioCTCAligner

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audios": [...], "texts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'MMS/ctc_alignment_mling_uroman_model.pt',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioCTCAligner,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audios"), col("texts"), col("langs")),
    )
    ds.show()
```

同源音频快速拼接

简介 音频快速拼接处理器（同源音频）

功能描述

- 使用 concat demuxer 快速拼接同源音频（无需重编码）
- 适用于格式、编码、采样率完全相同的音频文件
- 速度快，无质量损失

- 本地文件路径
- BOS/S3存储路径
- HTTP/HTTPS路径
- 支持输出到指定路径
- 拼接同一录音分段
- 拼接相同格式的音频片段
- 需要快速拼接且无需格式转换
- 支持所有音频格式（mp3, wav, flac, aac, m4a等）
- 要求：所有输入音频必须具有相同的格式、编码、采样率、声道数
- AudioConcat: 使用 concat filter，支持不同格式音频，需要重编码
- AudioConcatFast: 使用 concat demuxer，仅支持同源音频，无需重编码，速度更快

算子参数 输入 |输入|含义| |-|-| |audio_paths_list|包含音频文件路径列表的列，每个元素是一个字符串列表| |output_col|包含输出音频文件路径的数组|

输出 |输出|含义| |-|-| |result|包含拼接后音频文件路径的数组，成功返回输出路径，失败返回None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_format|str|'mp3'|输出音频格式（仅用于确定文件扩展名），默认为 "mp3" 注意：由于使用 concat demuxer，输出格式应与输入音频格式一致| |timeout|int|None|ffmpeg执行超时时间（秒），默认为None（无超时）|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_concat_fast import AudioConcatFast

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audio_paths_list": [...], "output_col": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_format": 'mp3',
        "timeout": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioConcatFast,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audio_paths_list"), col("output_col")),
    )
    ds.show()
```

语音端点识别（FSMN 模型）

简介 语音端点检测模块 - 基于 FSMN VAD 的高效音频分割解决方案

功能描述

- 语音端点检测：自动识别音频中的语音片段起止时间，实现语音与静音的精准分割
- 批量处理：支持大批量音频数据的高效端点检测
- 多格式输入：兼容原始二进制、Base64 编码、BOS/HTTP 链接等多种音频输入方式
- GPU 加速：支持 GPU 环境下的高性能推理
- 建议输入 16k 采样率、单声道的 WAV 格式音频，提升检测准确率
- 长音频建议分段处理，单次处理时长建议不超过 1 小时
- 适用于语音活动检测、语音切分等场景
- `iic/speech_fsmn_vad_zh-cn-16k-common-pytorch`（中文通用 FSMN VAD）
- 每条音频输出二维浮点数列表，表示所有语音片段的起止时间戳（单位：秒），如：[[0.0, 4.34], [5.50, 7.12]]
- 处理失败时返回 None

算子参数 输入 |输入|含义| |-|-| |videos|包含音频数据的列，支持以下格式： - audio_base64: base64 编码的音频字符串 - audio_url: 音频文件的 URL 路径 - audio_binary: 原始音频字节数据|

输出 |输出|含义| |-|-| |result|包含语音端点时间戳的列，每个元素为二维浮点数列表，表示音频中各语音片段的起止时间戳（单位：秒），例如：[[0.0, 4.34], [5.50, 7.12]]。若处理失败则为 None。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |audio_src_type|str|必填|音频格式类型 支持的音频格式类型，包含： - bos/http 地址(audio_url) - base64 编码(audio_base64) - 二进制流(audio_binary) 可选值：["audio_binary", "audio_url", "audio_base64"]| |batch_size_s|int|3600|批量计算的秒数 每批次处理的音频时长（秒），仅在使用GPU时生效。 默认值：3600| |model_path|str|'/opt/aihc/models'|模型路径 本地模型存储路径。 默认值："/opt/aihc/models"| |model_name|str|'iic/speech_fsmn_vad_zh-cn-16k-common-pytorch'|模型名称 使用的模型名称，包含： - iic/speech_fsmn_vad_zh-cn-16k-common-pytorch 可选值：

["iic/speech_fsmn_vad_zh-cn-16k-common-pytorch"] 默认值："iic/speech_fsmn_vad_zh-cn-16k-common-pytorch"| |model_revision|str|'v2.0.4'|模型版本 指定模型的版本号，包含： - v2.0.4 可选值：["v2.0.4"] 默认值："v2.0.4"| |rank|int|0|指定使用的GPU设备编号（多卡环境有效）。例如：0表示第一张GPU，1表示第二张GPU 默认值：None|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_vad_fsmn import AudioVadFsmn

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"videos": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "batch_size_s": 3600,
        "model_path": '/opt/aihc/models',
        "model_name": 'iic/speech_fsmn_vad_zh-cn-16k-common-pytorch',
        "model_revision": 'v2.0.4',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioVadFsmn,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("videos")),
    )
    ds.show()
```

音频质量评分

简介 音频质量评分模块 - 使用 DNSMOS 模型评估音频质量

功能描述

- 使用 DNSMOS (Deep Noise Suppression Mean Opinion Score) 模型进行音频质量评估
- 提供整体质量 (OVRL)、信号质量 (SIG)、背景噪声 (BAK) 三个维度的评分
- 评分范围在 1 到 5 之间，分数越高表示音频质量越好
- 支持自动采样率转换和音频预处理
- 适用于语音质量评估、音频筛选、质量控制等场景
- OVRL (Overall): 音频整体质量评分，综合评估音频的可听性
- SIG (Signal): 信号质量评分，评估语音信号的清晰度和自然度
- BAK (Background): 背景噪声评分，评估背景噪声的干扰程度

算子参数 输入 |输入|含义| |-|-| |audio_col|包含音频二进制数据的数组（pa.binary 类型），每个元素应为一段完整的音频内容。|

输出 |输出|含义| |-|-| |result|一个结构化结果数组，其中每个元素包含以下字段： - ovrl (float): 音频整体质量评分，综合评估音频的可听性，范围在 1.0 到 5.0 之间 - sig (float): 信号质量评分，评估语音信号的清晰度和自然度，范围在 1.0 到 5.0 之间 - bak (float): 背景噪声评分，评估背景噪声的干扰程度，范围在 1.0 到 5.0 之间 处理失败的音频返回包含 null 值的结构。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|-| |device|str|'cpu'|-| |is_personalized_mos|bool|False|-|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_quality_score import AudioQualityScore

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audio_col": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "device": 'cpu',
        "is_personalized_mos": False,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioQualityScore,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audio_col")),
    )
    ds.show()
```

语音转文字（FireRed）

简介 语音识别模块 - 基于 FireRed ASR 模型的多语言语音转文字解决方案

功能描述

- 多语言识别：支持中英文、以及中文方言
- 多模型选择：支持多种模型，包括AED模型和LLM模型
- 音频类型： 支持单声道、16K采样率的wav音频文件
- 为保证识别效果，对于AED模型，建议音频长度不超过60s；对于LLM模型，建议音频长度不超过30s
- 使用LLM进行批量推理时，建议确保音频长度差异不大，如差异较大，建议设置batch_size=1
- 使用之前建议对音频进行标准化处理，转化成该算子支持的音频文件
- FireRedASR-AED-L
- FireRedASR-LLM-L

算子参数 输入 |输入|含义| |-|-| |contents|包含音频数据的数组，元素类型为字符串（文件路径或URL）。 支持本地文件路径以及bos://、s3://、http://、https://开头的远程文件链接| |contents_ids|包含音频数据的数组，元素类型为字符串，用于唯一标识音频 默认值：None|

输出 |输出|含义| |-|-| |result|处理后的数组，元素为每个音频的理解结果文本。对于处理失败的音频，返回空字符串。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |audio_src_type|str|必填|音频格式类型 支持的音频格式类型，包含： - bos/http 地址(audio_url) - base64 编码(audio_base64) - 二进制流(audio_binary) 可选值：["audio_binary", "audio_url", "audio_base64"]| |model_path|str|'/opt/aihc/models'|模型存储路径 默认值："/opt/aihc/models"| |model_name|str|'FireRedAsr/FireRedASR-AED-L'|模型名称 支持的FireRed系列模型： - FireRedAsr/FireRedASR-AED-L: 小模型 - FireRedAsr/FireRedASR-LLM-L: 大模型 可选值：["FireRedAsr/FireRedASR-AED-L", "FireRedAsr/FireRedASR-LLM-L"] 默认值："FireRedAsr/FireRedASR-AED-L"| |batch_size|int|1|单次处理的音频样本数量 默认值：1| |beam_size|int|3|解码时的beam宽度 控制解码时的搜索空间大小，数值越大，识别准确率可能提升但速度变慢 默认值：3| |decode_min_len|int|0|最小解码长度限制输出文本的最小长度，0表示不限制 默认值：0| |decode_max_len|int|0|最大解码长度 限制输出文本的最大长度，0表示不限制 默认值：0| |nbest|int|1|AED模型输出候选数 控制输出的候选文本数量，仅AED模型有效 默认值：1| |softmax_smoothing|float|1.25|AED模型softmax平滑系数 调整softmax分布的平滑程度，仅AED模型有效 默认值：1.25| |aed_length_penalty|float|0.6|AED模型长度惩罚 控制输出文本长度的惩罚系数，仅AED模型有效 默认值：0.6| |eos_penalty|float|1.0|AED模型终止符惩罚 控制终止符的惩罚系数，仅AED模型有效 默认值：1.0| |repetition_penalty|float|3.0|LLM模型重复惩罚 控制生成文本时重复内容的惩罚系数，仅LLM模型有效 默认值：3.0| |llm_length_penalty|float|1.0|LLM模型长度惩罚 控制生成文本长度的惩罚系数，仅LLM模型有效 默认值：1.0| |temperature|float|1.0|温度系数 控制生成文本的随机性（0.0-1.0） 较高值适合创造性场景，较低值适合确定性场景 默认值：1.0| |use_fp16|bool|False|是否使用FP16推理 是否启用半精度浮点数加速推理，节省显存 默认值：False|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_asr_firered import AudioAsrFireRed

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"contents": [...], "contents_ids": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'FireRedAsr/FireRedASR-AED-L',
        "batch_size": 1,
        "beam_size": 3,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioAsrFireRed,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("contents"), col("contents_ids")),
    )
    ds.show()
```

音频人声分离

简介 音频人声分离模块 - 使用 Demucs 模型提取人声分量

功能描述

- 使用 Demucs 模型进行人声分离（去除背景音乐、噪声）
- 自动进行采样率对齐、声道调整、响度恢复
- 适合用于语音增强、字幕生成、音视频前处理等场景

算子参数 输入 |输入|含义| |-|-| |audio_col|包含音频二进制数据的数组（pa.binary 类型），每个元素应为一段完整的音频内容。|

输出 |输出|含义| |-|-| |result|一个二进制数组（pa.binary 类型），每个元素为对应音频中提取出的人声部分，格式为编码后的音频 bytes。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|-| |chunk_batch_size|int|8|-| |device|str|'cuda' if torch.cuda.is_available() else 'cpu'|-| |num_coroutines|int|1|-|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_source_separation import AudioSourceSeparation

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audio_col": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "chunk_batch_size": 8,
        "device": 'cuda' if torch.cuda.is_available() else 'cpu',
        "num_coroutines": 1,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioSourceSeparation,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audio_col")),
    )
    ds.show()
```

说话人分离

简介 基于pyannote-audio的说话人分离处理器

功能描述

- 多说话人语音分离与时间戳标注
- 输出带说话人标签的语音分段元数据
- 输入：音频
- 输出：包含以下字段的结构化数据
- speaker: 说话人唯一标识
- start: 语音段开始时间（秒）
- end: 语音段结束时间（秒）
- 使用speaker-diarization-3.1说话人分离模型
- 支持GPU加速推理（需配置CUDA环境）

算子参数 输入 |输入|含义| |-|-| |audios|包含多个音频数据的数组|

输出 |输出|含义| |-|-| |result|pa.Array: 包含说话人分离结果的数组|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|模型文件所在的路径| |rank|int|0|用于指定使用的 GPU 设备编号|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_speaker_diarization import AudioSpeakerDiarization

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audios": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "rank": 0,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioSpeakerDiarization,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audios")),
    )
    ds.show()
```

语音识别(Whisper)

简介 语音识别模块 - 基于Whisper模型的多语言语音转文字解决方案

功能描述

- 多语言识别：支持中英文等主流语言
- 语音翻译：可将识别结果翻译为英文
- 30秒内长度的音频片段可支持 batch_size > 1，提高处理效率；否则建议 batch_size 设置为1
- 英文场景识别准确率最高
- `openai/whisper-large-v3-turbo`
- `openai/whisper-large-v3`
- `openai/whisper-medium`（中文支持一般）
- `openai/whisper-small`（中文输出繁体字）

算子参数 输入 |输入|含义| |-|-| |audios|包含音频数据的数组，支持以下格式： - audio_base64: base64编码的音频字符串； - audio_url: 音频文件URL路径； - audio_binary: 原始音频字节数据| |languages|-|

输出 |输出|含义| |-|-| |asr_result|语音识别文本结果；| |timestamps|时间戳对列表(开始/结束时间)；| |segments|分段文本结果列表|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |audio_src_type|str|必填|音频格式类型 支持的音频格式类型，包含： - bos/http 地址(audio_url) - base64 编码(audio_base64) - 二进制流(audio_binary) 可选值：["audio_binary", "audio_url", "audio_base64"]| |model_path|str|'/opt/aihc/models'|模型存储路径 默认值："/opt/aihc/models"| |model_name|str|'openai/whisper-large-v3'|模型名称 支持的Whisper系列模型： - whisper-small: 小模型 - whisper-medium: 中等模型 - whisper-large-v3: 最新大模型 - whisper-large-v3-turbo: 优化版大模型 可选值：["openai/whisper-small", "openai/whisper-medium", "openai/whisper-large-v3-turbo", "openai/whisper-large-v3"] 默认值："openai/whisper-large-v3"| |batch_size|int|10|单次处理的音频样本数量 默认值：10| |source_language|str|None|音频源语言 支持：chinese/english/japanese/korean等，可以设置为None以启用自动检测 默认值：None| |translate_to_english|bool|False|英文翻译模式 是否将识别结果翻译为英文 启用后输出文本将为英文翻译结果 默认值：False| |condition_on_prev_tokens|bool|True|历史依赖模式 是否基于历史token进行预测 关闭后会降低结果连贯性但提升处理速度 默认值：True| |compression_ratio_threshold|float|1.35|文本压缩阈值 控制生成文本的压缩程度（建议范围1.2-2.0） 值越大保留的重复内容越多 默认值：1.35| |temperature|float|0.5|温度系数 控制生成文本的随机性（0.0-1.0） 较高值适合创造性场景，较低值适合确定性场景 默认值：0.5| |logprob_threshold|float|-1.0|对数概率阈值 对数概率阈

值，过滤置信度过低的词。若词的对数概率低于此值，可能被拒绝。 默认为-1.0，不启用过滤，保留所有词。| |dtype|str|'bfloat16'|计算精度类型 模型推理使用的数值精度： - bfloat16: 平衡精度与速度（默认） - float16: 更快的推理速度 - float32: 最高精度 可选值：["bfloat16", "float16", "float32"] 默认值："bfloat16"| |rank|int|0|GPU设备编号 指定使用的GPU设备ID（多卡环境生效） 默认使用首张显卡（ID=0）|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_asr_whisper import AudioAsrWhisper

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"audios": [...], "languages": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'openai/whisper-large-v3',
        "batch_size": 10,
        "source_language": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            AudioAsrWhisper,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("audios"), col("languages")),
    )
    ds.show()
```

音频高斯噪声增广

简介 音频高斯噪声增广算子（AudioAddGaussianNoise），按给定概率为输入音频叠加高斯白噪声，用于语音/音频数据集的数据增广。

功能描述 • 对输入音频按概率 `p` 叠加振幅介于 [min_amplitude, max_amplitude] 的高斯噪声。

• 支持多种音频来源类型（bos/http url、base64、binary），由 `audio_src_type` 指定。

• 加噪后音频写为 `.wav` 文件，落地到本地目录 `output_dir`；若配置了 `output_bosdir`，则再上传到 BOS 并返回 BOS 路径。

• 逐行处理并做异常隔离：单行失败返回 `None`，不影响整批其他行；内部维护提交/成功/失败计数。

算子参数 输入 |输入|含义| |-|-| |audio|输入音频，取值为 bos/http url、base64 或 binary（具体解析方式由 `audio_src_type` 决定）|

输出 |输出|含义| |-|-| |result|`large_string` 类型；加噪后音频的路径（本地 `output_dir` 下路径，或配置 `output_bosdir` 后的 BOS 路径）；单行失败为 `None`|

参数 |参数名称|类型|默认值|描述| |-|-|-| |audio_src_type|str| "audio_url"|音频来源类型，用于决定如何将输入解析并保存到本地（如 `audio_url` 等）| |min_amplitude|float|0.001|高斯噪声最小振幅，必须小于 `max_amplitude`| |max_amplitude|float|0.015|高斯噪声最大振幅，必须大于 `min_amplitude`| |p|float|0.5|加噪概率，取值范围 [0, 1]；设为 1.0 表示强制加噪| |output_dir|str|"/tmp/aihc_audio_noise"|加噪后音频的本地输出目录，不存在时自动创建| |output_bosdir|str|""|BOS 输出目录（会去除首尾 `/`）；非空时将结果上传到该目录并返回 BOS 路径，为空则仅返回本地路径|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.audio.audio_add_gaussian_noise import AudioAddGaussianNoise

AUDIO = "bos://your-bucket/sample.wav"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    ds = daft.from_pydict({"audio": [AUDIO]})
    ds = ds.with_column(
        "noised",
        aihc_udf(
            AudioAddGaussianNoise,
            construct_args={
                "audio_src_type": "audio_url",
                "min_amplitude": 0.001,
                "max_amplitude": 0.015,
                "p": 1.0,  # 强制加噪，便于验证
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("audio")),
    )
    ds.show()
```

具身

LerobotV2.1格式转换V3.0

简介

将LeRobotV2.1格式数据集转换为LeRobotV3.0

功能描述

支持多个数据集粒度并行处理

输入

输入	含义
input_repoid	原始数据集repoid
input_path	原始v2.1数据集path
output_path	转换后v3.0数据集path

输出

输入	含义
output_path	单个数据集转换后输出path
convert_result	数据集转换状态:SUCCESS: 成功转换SKIP: 跳过，当前数据集已经是v3.0格式或已经有转换好的3.0格式数据集在output_path下FAILED: 转换失败

示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_dataset_v21_to_v30_udf import ConvertDatasetV21ToV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    tasks = {
        "input_repoid": [
            "lerobot/aloha_sim_insertion_human/",
            "lerobot/pusht/",
            "lerobot/pusht2/"
        ],
        "input_path": ["/mnt/pfs/xxx/dataset/lerobot_v2/"] * 3,
        "output_path": ["/mnt/pfs/xxx/dataset/lerobot_v3_output/"] * 3
    }
    num_datasets = len(tasks)

    ds = daft.from_pydict(tasks)
    ds = ds.into_partitions(num_datasets)

    ds = ds.with_column(
        "convert_result",
        aihc_udf(
            ConvertDatasetV21ToV30,
            construct_args={
            },
            num_cpus=1,
            batch_size=1,
            concurrency=num_datasets,
            use_process=True
        )(col("input_repoid"), col("input_path"), col("output_path")),
    )

    ds.show()
```

调用示例

```
 **启动命令**
**v2tov3_udf.py 为以上的示例代码**

**!/bin/bash**
echo "Start Execution."
python /mnt/pfs/cxy/test/aihc/embodied_ai/v2_to_v3/v2tov3_opreator.py
echo "Execution completed."
```

Rlds格式转换处理器

简介

将常用的 Open X-Embodiment (OXE) 标准的 RLDS 数据集转换为 LeRobot v3.0 格式

功能描述

- 将 Open X-Embodiment (OXE) 标准的 RLDS 数据集转换为 LeRobot v3.0 格式
- 支持按 split 数量自动划分 episode 范围，分布式并行转换
- 自动从 OXE_DATASET_CONFIGS 推导 fps、robot_type、version 等参数
- 支持视频编码（SVT-AV1）和图像模式
- 每次转换前清理旧目录，保证数据一致性
- 提供独立的 `merge_splits` 函数，将多个 split 结果合并为完整数据集

已知限制

- **仅支持 OXE 标准的 RLDS 数据集**：数据集需符合 Open X-Embodiment 的 episode/steps 结构，包含 `observation`、`action`、`language_instruction` 字段
- **仅读取 train split**：TFDS 数据源固定使用 `train` split（`builder.as_dataset(split="train[start:end]")`），不支持 val/test
- **深度图像不转换**：图像 key 过滤规则为包含 `image` 或 `rgb` 且不包含 `depth`，深度通道会被跳过
- **状态维度固定为 8**：proprio 向量拼接为 8 维，缺失的 key 用零填充
- **单 episode 全量加载**：每个 episode 的所有 steps 通过 `.batch(cardinality())` 一次性加载到内存，单个 episode 步数过多或图像分辨率过高时可能导致内存压力
- **视频编码为软件编码**：使用 SVT-AV1 软件编码器（libsvtav1），不支持 GPU 硬件加速
- **split 合并需单独调用**：算子只负责分片转换，合并步骤需在 `ds.show()` 之后显式调用 `merge_splits`
- **重跑会清除旧数据**：每次转换前自动删除对应 split 输出目录，不支持断点续传
- **不支持 ArrayRecord 格式**：仅支持 TFRecord 格式的 RLDS 数据集。TFDS 新版引入的 ArrayRecord 存储格式无法读取，需先转换为 TFRecord 或使用 TFRecord 版本的数据

输入

输入	含义	样例
split_idx	当前分片索引（从 0 开始）	[0, 1, 2, ..]

输出

输入	含义	
convert_result	转换结果字符串	SUCCESS:/path/to/output_split_0:120.5s
		FAILED:/path/to/output_split_1:error message

构造参数（construct_args）

参数	类型	含义	默认值
dataset_name	str	RLDS 数据集名称	**必填**
data_dir	str	RLDS 数据集所在目录（tfds builder 的 data_dir）	**必填**
output_path	str	转换输出目录路径	**必填**
num_splits	int	并行分片数量	**必填**
version	str	数据集版本号，空字符串则自动探测	""
repo_id	str	HuggingFace repo ID	默认同 dataset_name
fps	int	帧率，None 则从 OXE_DATASET_CONFIGS 获取	None（自动）
robot_type	str	机器人类型，None 则从 OXE_DATASET_CONFIGS 获取	None（自动）
use_videos	bool	是否将图像编码为视频	True
image_writer_processes	int	图像/视频写入进程数	5
image_writer_threads	int	图像/视频写入线程数	10

merge_splits 函数参数

参数	类型	含义	默认值
output_path	str	合并后的输出目录（与转换时一致）	**必填**
repo_id	str	HuggingFace repo ID	**必填**
num_splits	int	分片数量（与转换时一致）	**必填**
cleanup	bool	合并后是否删除 split 子目录	True

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_rlds_to_lerobot_udf import ConvertRldsToLerobot

os.environ["DAFT_RUNNER"] = "ray"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import logging
        import ray

        try:
            ray.init(address="auto", ignore_reinit_error=True)
        except ConnectionError:
            ray.init(ignore_reinit_error=True)

    daft.set_execution_config(
        actor_udf_ready_timeout=6000,
        min_cpu_per_task=0,
    )

    # === 配置 ===
    DATASET_NAME = "cmu_play_fusion"
    DATA_DIR = "/mnt/pfs/data/oxe/"
    OUTPUT_PATH = "/mnt/pfs/data/v3_output/cmu_play_fusion_lerobot"
    NUM_SPLITS = 5

    # 输入: split_idx 列表
    tasks = {
        "split_idx": list(range(NUM_SPLITS)),
    }

    ds = daft.from_pydict(tasks)
    ds = ds.into_partitions(NUM_SPLITS)

    ds = ds.with_column(
        "convert_result",
        aihc_udf(
            ConvertRldsToLerobot,
            construct_args={
                "dataset_name": DATASET_NAME,
                "data_dir": DATA_DIR,
                "output_path": OUTPUT_PATH,
                "num_splits": NUM_SPLITS,
                "use_videos": True,
                "image_writer_processes": 5,
                "image_writer_threads": 10,
            },
            num_cpus=1,
            batch_size=1,
            concurrency=NUM_SPLITS,
            use_process=True,
        )(col("split_idx")),
    )

    ds.show()

    # === 合并所有 split 子集 ===
    if NUM_SPLITS > 1:
        from daft.aihc.functions.embodied.convert_rlds_to_lerobot_udf import merge_splits
        merge_splits(
            output_path=OUTPUT_PATH,
            repo_id=DATASET_NAME,
            num_splits=NUM_SPLITS,
            cleanup=True,
        )
```

导出为WebDataset

简介 把训练样本按 WebDataset 规范打成 .tar shards（基于 webdataset.ShardWriter），可选把媒体文件字节一并写入；返回每条样本所在的 shard tar 路径，用于物化训练分片。

功能描述 • 标准 shards：使用 webdataset.ShardWriter，按 max_samples_per_shard / max_shard_size_mb 自动切分

• 媒体可选：include_media=True 时把 media_path 指向的文件下载后写入 tar；False 时仅写 meta.json

• 顺序写入：跨行状态保存在算子实例内，需配合 concurrency=1、batch_size=None 保证顺序

• 键兜底：sample_key 为空时用 md5 兜底生成键

算子参数 **输入** |输入|含义| |-|-| |sample_keys|每条样本的唯一键 str（为空时用 md5 兜底）。| |media_paths|本地或 BOS 媒体路径（会自动 download 到本地再打入 tar）。| |meta_jsons|结构化 caption/label/score 的 JSON 字符串，可为空。|

输出 |输出|含义| |-|-| |result|string，该样本被写入的 tar shard 路径；写入失败时为空字符串。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_dir|str|（必填）|shard 输出目录（PFS/BOS 挂载路径，不存在会自动创建）| |shard_prefix|str|'shard'|shard 文件前缀 默认值："shard"| |max_shard_size_mb|float|512.0|单 shard 目标大小上限（MB，近似值）默认值：512.0| |max_samples_per_shard|int|10000|单 shard 样本数上限 默认值：10000| |media_extension|str/None|None|媒体扩展名，为空则从 media_path 后缀自动推断 默认值：None| |include_media|bool|True|是否把 media 字节写入 tar（False 时仅写 meta.json）默认值：True| |fail_on_error|bool|False|出错是否抛异常 默认值：False|

调用示例

```
from __future__ import annotations

import json
import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.export.export_to_web_dataset import ExportToWebDataset

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # output_dir 为 mock 路径，请替换为实际 PFS/BOS 挂载目录
    OUTPUT_DIR = "/mnt/pfs/webdataset_shards"  # MOCK

    keys = [f"sample_{i}" for i in range(5)]
    samples = {
        "sample_key": keys,
        "media_path": ["" for _ in keys],   # include_media=False 时无需真实媒体
        "meta_json": [json.dumps({"idx": i, "label": "demo"}) for i in range(5)],
    }
    ds = daft.from_pydict(samples)
    # 顺序写入：concurrency=1, batch_size=None
    ds = ds.with_column(
        "shard",
        aihc_udf(
            ExportToWebDataset,
            construct_args={
                "output_dir": OUTPUT_DIR,
                "shard_prefix": "shard",
                "include_media": False,
                "max_samples_per_shard": 10000,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=None,
        )(col("sample_key"), col("media_path"), col("meta_json")),
    )
    ds.show()
```

轨迹平滑

简介 对手部/末端执行器的运动（动作/位姿）序列做平滑与离群点去除，纯 numpy、离线 CPU、无模型依赖。 **功能描述** • 平滑：支持 savgol（Savitzky-Golay）、moving_average、gaussian 三种方法

• 去离群：velocity / zscore / none，异常帧用相邻好帧线性插值替换

• 形状保持：输出序列与输入形状一致 [N, D]

• 依赖友好：savgol 需 scipy，缺失时自动回退 moving_average

算子参数 输入 |输入|含义| |-|-| |sequence|运动/位姿序列，每行形状 [N, D]，元素类型 list<list>。|

输出 |输出|含义| |-|-| |result|平滑并去离群后的序列，形状与输入一致（list<list>）。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |method|str|savgol|平滑方法，可选 savgol/moving_average/gaussian| |window|int|5|平滑窗口长度（建议奇数；savgol 要求奇数且 > polyorder）| |polyorder|int|2|savgol 多项式阶数（仅 method=savgol 时使用）| |outlier_method|str|velocity|离群检测，可选 velocity/zscore/none| |outlier_threshold|float|3.0|离群阈值（标准差倍数）|

调用示例

```
from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.trajectory_smooth import TrajectorySmooth

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    samples = {"sequence": [[[0.0,0.0,0.0,0.0],[0.1,0.0,0.0,0.0],[0.2,0.0,0.0,0.0],[0.3,0.0,0.0,0.0],[0.4,0.0,0.0,0.0]]]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "smoothed",
        aihc_udf(
            TrajectorySmooth,
            construct_args={"method":"savgol","window":5,"polyorder":2,"outlier_method":"velocity","outlier_threshold":3.0},
            num_cpus=1, concurrency=1, batch_size=8,
        )(col("sequence")),
    )
    ds.show()
```

手部动作计算

简介 从手部逐帧 6D 位姿（位置 + 旋转）+ 可选相机位姿计算对齐 LeRobot / Open X-Embodiment 约定的 7-DoF 相对动作序列[dx,dy,dz,drx,dry,drz,gripper] 及绝对状态向量，AIHC 自研，纯几何计算、离线 CPU、无模型依赖。

功能描述 • 7-DoF 动作：逐帧计算相对上一帧的平移增量 + 旋转增量 + gripper 绝对值（delta_mode=relative），或直接输出绝对量（absolute）

• 旋转编码：支持 axis_angle（默认，3 维无万向锁）/ euler_xyz / quaternion（取增量四元数虚部 3 维）

• 坐标系：默认相机系 camera；提供相机外参时可切世界系 world

• gripper 归一化：默认归一化到 [0,1]（0=闭,1=开），可用 gripper_open_dist 指定开合尺度或自动 min-max

算子参数 输入 |输入|含义| |-|-| |hand_pose_seq|每帧手部 6D 位姿，list，每个 struct 含 position([x,y,z]) + rotation（旋转矩阵 9 元素或四元数 [w,x,y,z]）+ 可选 gripper（原始开合度标量）。| |camera_pose（可选）| 相机外参（4x4 展平为 16 元素），coordinate_frame=world 时使用；不用可省略。|

输出 |输出|含义| |-|-| |result|struct{action: list<list>(N×7), state: list<list>(N×7)}；action 每帧为 [dx,dy,dz,drx,dry,drz,gripper]，state 每帧为绝对位姿 [x,y,z,rot(3),gripper]。|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |coordinate_frame|str|camera|坐标系，可选 camera/world（world 需提供 camera_pose）|| |rotation_format|str|axis_angle|旋转编码，可选 axis_angle/euler_xyz/quaternion|| |gripper_range|tuple[float,float]|(0.0, 1.0)|gripper 归一化目标区间 [g_lo, g_hi]|| |gripper_open_dist|float |None|None|gripper 完全张开对应的原始开合度；None 时按序列 min-max 归一化| |delta_mode|str|relative|动作模式，relative（相对上一帧增量）/ absolute（绝对量）||

调用示例

```
 from __future__ import annotations
import os
import math
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.hand_action_compute import HandActionCompute

def _rot_z(theta):
    c, s = math.cos(theta), math.sin(theta)
    return [c, -s, 0, s, c, 0, 0, 0, 1]

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # 一条 5 帧序列：沿 x 平移 0.01/帧，无旋转，无 gripper
    seq = [{"position": [0.01 * i, 0.0, 0.0], "rotation": _rot_z(0.0), "gripper": None} for i in range(5)]
    samples = {"hand_pose_seq": [seq]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            HandActionCompute,
            construct_args={
                "coordinate_frame": "camera",
                "rotation_format": "axis_angle",
                "delta_mode": "relative",
                "gripper_range": [0.0, 1.0],
            },
            num_cpus=1, concurrency=1, batch_size=1,
        )(col("hand_pose_seq")),
    )
    ds.show()
```

成功标签评估

简介 任务成败判定 + 2D→3D 目标框提升，AIHC 自研，纯 numpy 几何计算、离线 CPU、无模型/GPU 依赖，是 P3 多模态打标链路的关键环节；成败判定为可审计的几何首版，生产可将 eval 模式接入 VLM/奖励模型后端。

功能描述 • eval 模式：由末端 7-DoF 动作序列累积位移（前 3 维平移增量之和的模）+ 可选物体倾斜角，判定任务是否成功（bool）

• lift_bbox_3d 模式：由 2D 检测框 [x1,y1,x2,y2] + 相机内参/外参 + 深度先验，几何提升为一个 3D 目标框（8 个角点）

• 统一输出：两种模式返回同一 struct 结构，每种模式只填对应字段，保证返回类型静态可推断

• 阈值可配：min_displacement / max_tilt_deg / depth_prior / box_half_size / fx / fy 全部参数可配

算子参数 输入 |输入|含义| |-|-| |col_a|eval 模式：action_seq（list<list> [N,>=7]，末端 7-DoF 动作序列）；lift_bbox_3d 模式：bbox_2d（list [x1,y1,x2,y2]）。| |col_b（可选）|eval 模式：tilt_deg（float，物体倾斜角，度）；lift_bbox_3d 模式：camera（list，相机内参 3x3 展平 9 元素或外参 4x4 展平 16 元素）；可省略。|

输出 |输出|含义| |-|-| |result|struct{success: bool, bbox_3d: list<list>}；eval 模式填 success，lift_bbox_3d 模式填 bbox_3d（8×3 角点），另一字段留空/False。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |mode|str|eval|工作模式，可选 eval / lift_bbox_3d| |depth_prior|float|0.6|缺深度时的深度先验（米），用于 2D→3D 提升| |box_half_size|float|0.05|3D 框半边长（米）| |min_displacement|float|0.05|eval 判成功的最小末端位移（米）| |max_tilt_deg|float|15.0|eval 判成功允许的最大物体倾斜角（度）| |fx|float|1.0|归一化像素→相机系的焦距近似（camera 未给内参时使用）| |fy|float|1.0|归一化像素→相机系的焦距近似（camera 未给内参时使用）|

调用示例

```
 from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.success_label_eval import SuccessLabelEval

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # eval 模式：一条累计位移足够的序列（成功）与一条几乎不动的序列（失败）
    moving = [[0.02, 0.0, 0.0]] * 5   # 累计 dx=0.10 >= 0.05
    still = [[0.001, 0.0, 0.0]] * 5   # 累计 0.005 < 0.05
    samples = {"a": [moving, still]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "res",
        aihc_udf(
            SuccessLabelEval,
            construct_args={"mode": "eval", "min_displacement": 0.05},
            num_cpus=1, concurrency=1, batch_size=2,
        )(col("a")),
    )
    ds.show()
```

导出为LeRobotV3.0

简介 把视频 + 7-DoF 动作 + 状态组装导出为标准 LeRobot v3.0 数据集，AIHC 自研，是 7-DoF 动作链路的终点，依赖 lerobot（复用LeRobotDataset.create/add_frame/save_episode writer）+ 可选 OpenCV 读帧，离线 CPU，用于把处理好的 episode 落盘为可训练的 v3.0 数据集。

功能描述 • shard 写出：每个 transform batch 在独立的 <output_path>_shard_<...> 目录里完整走 create → add_frame → save_episode → finalize，返回 shard 路径

• 主进程合并：主脚本 ds.collect() 后对返回列去重，再调 merge_shards_to_final（或 lerobot 的 merge_datasets）合并为最终 output_path

• schema 可配：action_dim/state_dim/feature_schema 兼容不同本体（如 14-DoF 双臂），不绑定固定 7-DoF

• 容错开关：fail_on_error=True（默认）遇写入/finalize 失败即抛出，避免悄悄产坏 parquet

算子参数 输入 |输入|含义| |-|-| |video|该 episode 的视频本地路径或多相机 dict；use_videos=False 时可为 None。| |action|动作序列 [N, action_dim]，类型 list<list>。| |state|状态序列 [N, state_dim]，类型 list<list>。| |task|该 episode 的语言指令（string）。|

输出 |输出|含义| |-|-| |result|字符串（large_string）：该 episode 所在的 shard 目录路径；写入失败的行为 None。主脚本对该列去重后 merge。|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |output*path|str|（必填）|最终 v3.0 数据集输出目录；actor 在 `<output_path>_shard*写分片，主脚本负责 merge|| |repo_id|str|（必填）|数据集标识，如aihc/hand_demo`|| |fps|int|（必填）|帧率（须为正）|| |robot_type|str|generic_hand|本体类型标识，写入 v3 元数据|| |action_dim|int|7|动作维度（须为正；双臂等可配）|| |state_dim|int |None|None（等于 action_dim）|状态维度，None 时与 action_dim 相同| |image_key|str|observation.images.cam|视频 feature 的 key|| |image_shape|tuple[int,int,int]|(480, 640, 3)|视频帧 [H, W, C]|| |feature_schema|dict |None|None|完全自定义的 lerobot features dict；提供时覆盖 action/state/image 默认| |use_videos|bool|True|是否作为 video feature 写入|| |vcodec|str|h264|视频编码器|| |encoder_threads|int |None|None|每个编码器实例的线程数| |fail_on_error|bool|True|遇 episode 写入/finalize 失败是否抛出（False 保留吞异常成 None 的旧行为）||

调用示例

```
from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.export_to_lerobot_v30 import ExportToLeRobotV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    OUTPUT_DIR = "/mnt/pfs/lerobot_v30_out/ds"  # MOCK

    # 一个 5 帧无视频 episode：dx 递增、gripper 常开
    action = [[0.0 + 0.1 * i, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0] for i in range(5)]
    state = [row[:] for row in action]
    samples = {
        "video": [None],
        "action": [action],
        "state": [state],
        "task": ["pick up the block"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "shard",
        aihc_udf(
            ExportToLeRobotV30,
            construct_args={
                "output_path": OUTPUT_DIR,
                "repo_id": "aihc/hand_demo",
                "fps": 20,
                "robot_type": "generic_hand",
                "action_dim": 7,
                "state_dim": 7,
                "use_videos": False,
            },
            num_cpus=1, concurrency=1, batch_size=8,
        )(col("video"), col("action"), col("state"), col("task")),
    )
    ds.show()
    # 生产中：shards = set(ds.to_pydict()["shard"]); 再调 merge_shards_to_final(shards, "aihc/hand_demo", OUTPUT_DIR)
```

原子动作切分

简介 基于运动学启发式（停顿点 + gripper 翻转点）把一条连续的 7-DoF 动作序列切分为若干原子动作片段，AIHC 自研，纯 numpy、离线 CPU、无模型依赖，是 7-DoF 动作链路的中间处理环节。

功能描述 • 停顿切分：前 3 维平移速度低于阈值视为停顿，在"运动↔停顿"状态翻转处切分（segment_by 含 pause）

• 抓取翻转切分：第 7 维 gripper 开合状态翻转处切分（segment_by 含 gripper_flip）

• 短片段归并：长度小于 min_segment_len 的片段并入相邻片段，保证片段连续覆盖 [0, N)

• 片段输出：每个片段为半开区间 [start, end) 并带切分触发类型（pause/gripper_flip/start/merged）

算子参数 输入 |输入|含义| |-|-| |action_seq|一条 7-DoF 动作序列，形状 [N, 7]（[dx,dy,dz,drx,dry,drz,gripper]），类型 list<list>，与 HandActionCompute 输出对齐。|

输出 |输出|含义| |-|-| |result|片段列表，类型 list<struct{start:int32, end:int32, type:string}>，start/end 为半开区间，type 为切分触发类型。|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |segment_by|list[str] |None|None（等价于 ["pause","gripper_flip"]）|切分依据，可选项 pause / gripper_flip 的子集| |pause_velocity_thresh|float|0.01|停顿判定的平移速度阈值（前 3 维增量模长）|| |min_segment_len|int|5|最小片段长度，短于此长度的片段并入相邻段（须 >= 1）|| |gripper_flip_thresh|float|0.5|gripper 开合翻转阈值（小于阈值视为闭合）||

调用示例

```
from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.atomic_action_segment import AtomicActionSegment

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # 一条 12 帧 7-DoF 序列：前段运动、中段停顿、末段运动（会在停顿边界切分）
    seq = (
        [[0.1, 0.0, 0.0, 0, 0, 0, 1.0]] * 4     # 运动
        + [[0.0, 0.0, 0.0, 0, 0, 0, 1.0]] * 4   # 停顿
        + [[0.1, 0.0, 0.0, 0, 0, 0, 1.0]] * 4   # 运动
    )
    samples = {"action_seq": [seq]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "segments",
        aihc_udf(
            AtomicActionSegment,
            construct_args={"segment_by": ["pause"], "pause_velocity_thresh": 0.05, "min_segment_len": 3},
            num_cpus=1, concurrency=1, batch_size=1,
        )(col("action_seq")),
    )
    ds.show()
```

AgiBot转LeRobotV3.0

简介 把 AgiBot World 数据集目录转换为 LeRobot v3.0 数据集，AIHC UDF 封装，依赖 lerobot + ffmpeg（内部走convert_agibot_to_lerobot_v30.convert_dataset 的流式 H.264 编码、多线程解码 8 路相机、episode 级多进程 + aggregate_datasets 合并），用于把 AgiBot 采集数据接入标准 LeRobot v3.0 流水线。

功能描述 • 逐行转换：每行处理一个源数据集，源目录为 <input_path>/<input_repoid>，产物写到 <output_path>/<input_repoid>

• 视频编码：vcodec 默认 h264（libx264 CPU 编码，兼容性好），auto 映射为 h264 以避免误选 nvenc

• 并行解码：decode_threads 并发解码 8 路源 mp4，num_workers 控制 episode 级进程数

• 幂等跳过：目标已是 v3.0 时默认跳过（返回 SKIPPED），force_conversion=True 可强制重转

算子参数 输入 |输入|含义| |-|-| |input_repoid_col|数据集逻辑名（string），如 agibot/digital_world，写入 v3 元数据并作为子目录名。| |input_paths|源数据集父目录（string）；真实源在 <input_paths>/<input_repoid>（需含 meta_info/ 与 observations/）。| |output_paths|输出父目录（string）；产物写到 <output_paths>/<input_repoid>。|

输出 |输出|含义| |-|-| |result|字符串（large_string）：`"SUCCESS"` / `"SKIPPED: has already been v3.0"` / `"Failed: [repo] <ErrorType>: <msg>"`。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |num_workers|int|4|每个数据集的 episode 级进程数，<=1 时内联执行| |vcodec|str|h264|视频编码器，默认 h264（libx264）；h264_nvenc 需匹配 NVENC SDK 的驱动；auto 映射为 h264| |decode_threads|int|8|并发解码 8 路源 mp4 的线程数| |force_conversion|bool|False|即使输出已是 v3.0 也强制重新转换|

调用示例

```
 from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_agibot_to_lerobot_v30_udf import ConvertAgibotToLeRobotV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    INPUT_PATH = "/mnt/pfs/agibot_datasets"   # MOCK
    OUTPUT_PATH = "/mnt/pfs/agibot_v30_out"   # MOCK
    samples = {
        "input_repoid_col": ["sample_dataset"],
        "input_paths": [INPUT_PATH],
        "output_paths": [OUTPUT_PATH],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ConvertAgibotToLeRobotV30,
            construct_args={
                "num_workers": 4,
                "vcodec": "h264",
                "decode_threads": 8,
                "force_conversion": True,
            },
            num_cpus=1, concurrency=1, batch_size=1,
        )(col("input_repoid_col"), col("input_paths"), col("output_paths")),
    )
    ds.show()
```

LeRobotV3.0数据集对齐

简介 把一个已完成的 LeRobot v3.0 数据集对齐到共享的跨本体规格（统一 fps / robot_type / features、动作与状态维度投影、可选逐维归一化），依赖 lerobot（内部走 align_lerobot_v30.align_dataset + ffmpeg 视频重编码），用于多数据集合并前的预处理，使各输入通过 merge_datasets 的兼容性校验。

功能描述 • 统一规格：按同一个 spec（plan_unification 产出）重采样 fps、投影 action / observation.state 到规范宽度、统一 robot_type 与 features schema

• 归一化：支持 none / meanstd / minmax 三种逐维动作与状态缩放

• 视频重编码：vcodec 默认 h264（比 lerobot 的 libsvtav1 快 5-10 倍），auto 可选硬件编码器

• 逐行任务：每行 src（源 v3.0 数据集根目录）对齐输出到 out，返回成功/失败字符串

算子参数 输入 |输入|含义| |-|-| |src|源 v3.0 数据集根目录（string，本地或 BOS 路径）。| |out|对齐后数据集的输出根目录（string，本地或 BOS 路径）。|

输出 |输出|含义| |-|-| |result|字符串（large_string），成功为 `"SUCCESS: <out_path>"`，失败为 `"Failed: <src>: <ErrorType>: <msg>"`。|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |spec|dict|（必填）|统一目标规格， align_lerobot_v30.plan_unification 的输出；所有行共用同一 spec 以产出 byte-identical 的 fps/robot_type/features|| |normalize|str|none|逐维动作/状态缩放，可选 none/meanstd/minmax|| |vcodec|str|h264|视频编码器，可选 h264/auto/libsvtav1 等；auto 在可用时选硬件编码器|| |encoder_threads|int |None|None|每个编码器实例的线程数；大量数据集并发对齐时用于限流|

调用示例

```
 from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.align_lerobot_v30_udf import AlignLeRobotV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # 生产中 spec 应由 align_lerobot_v30.plan_unification([...]) 生成，此处给一个最小示例
    spec = {
        "fps": 20,
        "robot_type": "generic",
        "state_dim": 7,
        "action_dim": 7,
        "image_keys": ["cam_high"],
        "image_hw": (480, 640),
        "features": {
            "observation.state": {"dtype": "float32", "shape": (7,)},
            "action": {"dtype": "float32", "shape": (7,)},
        },
    }

    samples = {
        "src": ["bos://your-bucket/v30/dataset_a"],   # MOCK
        "out": ["bos://your-bucket/aligned/dataset_a"],  # MOCK
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            AlignLeRobotV30,
            construct_args={"spec": spec, "normalize": "none", "vcodec": "h264"},
            num_cpus=1, concurrency=1, batch_size=1,
        )(col("src"), col("out")),
    )
    ds.show()
```

LeRobotV3.0数据集合并

简介 把多个已完成的 LeRobot v3.0 数据集合并为一个 v3.0 数据集，AIHC UDF 封装，依赖 lerobot（内部走 `merge_lerobot_v30.merge_datasets` → `aggregate_datasets`/`aggregate_videos`，重索引 episode/frame/task/video）+ ffmpeg，用于把多来源数据集聚合为统一训练集。

功能描述 • 多入一出：每行描述一个 merge job，`input_paths`（显式列表）与 `base_path`（父目录递归扫描）二选一，输出到 `output_path`

• 输入校验：合并前校验每个输入是已完成的 v3.0 数据集（meta/info.json 且 codebase_version==v3.0），占位路径会被拒绝

• dry-run：`dry_run=True` 仅校验输入并报告计划而不写出（返回 `SUCCESS: dry-run`）

• 幂等跳过：输出已是 v3.0 且非 force/非 dry-run 时跳过（返回 SKIPPED）

算子参数 输入 |输入|含义| |-|-| |input_paths|待合并的 v3.0 数据集根目录列表（list）；与 base_path 互斥，用 base_path 时传空列表/None。| |base_path|递归扫描其下 v3.0 数据集的父目录（string）；与 input_paths 互斥。| |output_paths|合并后数据集的输出目录（string）。|

输出 |输出|含义| |-|-| |result|字符串（large_string）：`"SUCCESS"` / `"SKIPPED: has already been v3.0"` / `"SUCCESS: dry-run"` / `"Failed: [output] <ErrorType>: <msg>"`。|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-|-| |aggr_repo_id|str|merged|写入合并后元数据的逻辑 repo id|| |force_merge|bool|False|即使 output_path 已是 v3.0 也强制重新合并|| |dry_run|bool|False|仅校验输入并报告合并计划，不写出任何内容|| |data_files_size_in_mb|float |None|None|转发给底层聚合器的数据分片大小（None 用 lerobot 默认）| |video_files_size_in_mb|float |None|None|转发给底层聚合器的视频分片大小（None 用 lerobot 默认）| |chunk_size|int |None|None|转发给底层聚合器的 chunk 大小（None 用 lerobot 默认）|

调用示例

```
from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.merge_lerobot_v30_udf import MergeLeRobotV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    BASE_PATH = "/mnt/pfs/merge_src"     # MOCK，其下含 >=2 个 v3.0 数据集
    OUTPUT_PATH = "/mnt/pfs/merge_out"   # MOCK
    samples = {
        "input_paths": [None],           # 用 base_path 模式，input_paths 置空
        "base_path": [BASE_PATH],
        "output_paths": [OUTPUT_PATH],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            MergeLeRobotV30,
            construct_args={"aggr_repo_id": "merged", "force_merge": True},
            num_cpus=1, concurrency=1, batch_size=1,
        )(col("input_paths"), col("base_path"), col("output_paths")),
    )
    ds.show()
```

HDF5转LeRobot_v3.0

简介 把通用 HDF5 具身智能数据集（ALOHA / Mobile ALOHA / RoboMimic 布局）转换为 LeRobot v3.0 数据集，AIHC UDF 封装，依赖 lerobot + h5py + ffmpeg，源读取器可插拔并自动探测 schema，用于把常见 HDF5 采集数据无需逐集写代码地接入标准 LeRobot v3.0 流水线。

功能描述 • 逐行转换：每行处理一个源数据集，源目录为 `<input_path>/<input_repoid>`（需含至少一个 .hdf5/.h5），产物写到 `<output_path>/<input_repoid>`

• schema 自动探测：从首个 HDF5 文件自动识别布局；`schema_override` 可强制指定 aloha/mobile_aloha/robomimic；不匹配抛 `UnsupportedHdf5SchemaError`

• 视频编码：`vcodec` 默认 h264（libx264），auto 映射为 h264 避免误选 nvenc

• 元数据可配：`default_task` 填充缺失任务名，`robot_type` 写入 v3 元数据；目标已是 v3.0 时默认跳过

算子参数 输入 |输入|含义| |-|-| |input_repoid_col|数据集逻辑名（string），如 `aloha/transfer_cube`，写入 v3 元数据并作为子目录名。| |input_paths|源数据集父目录（string）；真实源在 `<input_paths>/<input_repoid>`（递归查找 .hdf5/.h5）。| |output_paths|输出父目录（string）；产物写到 `<output_paths>/<input_repoid>`。|

输出 |输出|含义| |-|-| |result|字符串（large_string）：`"SUCCESS"` / `"SKIPPED: has already been v3.0"` / `"Failed: [repo] <ErrorType>: <msg>"`。|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-|-| |num_workers|int|4|每个数据集的 episode 级进程数，<=1 时内联执行|| |vcodec|str|h264|视频编码器，默认 h264（libx264）；h264_nvenc 需匹配驱动；auto 映射为 h264|| |force_conversion|bool|False|即使输出已是 v3.0 也强制重新转换|| |default_task|str|default_task|源 schema 未提供任务名时，写入每个 episode 的默认任务名|| |robot_type|str|unknown|写入 v3 元数据的自由格式机器人本体标识|| |schema_override|str |None|None|强制指定 schema（aloha/mobile_aloha/robomimic），None 为自动探测|

调用示例

```
from __future__ import annotations
import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_hdf5_to_lerobot_v30_udf import ConvertHdf5ToLeRobotV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    INPUT_PATH = "/mnt/pfs/hdf5/rm_real_in"      # MOCK
    OUTPUT_PATH = "/mnt/pfs/hdf5/hdf5_v30_out"   # MOCK
    repoid = "robomimic/lift_ph"
    samples = {
        "input_repoid_col": [repoid],
        "input_paths": [INPUT_PATH],
        "output_paths": [OUTPUT_PATH],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ConvertHdf5ToLeRobotV30,
            construct_args={
                "num_workers": 4,
                "vcodec": "h264",
                "force_conversion": True,
                "default_task": repoid.rsplit("/", 1)[-1],
                "robot_type": "unknown",
                "schema_override": None,
            },
            num_cpus=1, concurrency=1, batch_size=1,
        )(col("input_repoid_col"), col("input_paths"), col("output_paths")),
    )
    ds.show()
```

VGGT 三维重建

简介

VGGT 三维重建算子：对输入视频抽帧后一次性送入 VGGT-1B 前向，输出相机内外参、深度图与点云图。VGGT 权重与 vggt 源码必须离线预置，算子内不下载权重也不安装依赖。

功能描述

- 抽帧走 VideoFrameSampler 在内存里完成、不落盘：5 种抽帧模式与时间窗口参数原样透传，默认 by_count_uniform + count_k=3，另加 max_frames=32 兜底，VGGT 显存随帧数增长
- 预处理等价于 vggt load_and_preprocess_images 的 "crop" 模式：BGR 转 RGB，宽固定缩放到 518，高按比例缩到 14 的倍数（下限 14），高超过 518 时中心裁剪，再归一化到 [0,1]；同一视频内各帧尺寸一致，直接 stack 成 (S,3,H,W)
- 相机内外参由 pose_encoding_to_extri_intri 从模型输出的 pose_enc 解出；point_maps_from_unprojection 由 unproject_depth_map_to_point_map 用深度 + 内外参反投影得到，只有同时拿到 pose_enc 与 depth 时才会填
- 每个张量统一存成 {shape, data}（形状 + 一维 float32 展平数据），而不是多层嵌套 list，arrow 侧内存可控、还原方式明确
- 四个 if_output_* 开关只控制字段是否填值，关掉即为空张量 {"shape": [], "data": []}，不改变输出 schema
- point_tracks 本期不实现：点跟踪要求样本自带 query_points，AIHC 没有上游算子产出该字段
- 精度自适应：GPU 上 bfloat16 需要 compute capability >= 8，不满足自动回落 float16；无 GPU 时强制 float32 并打 warning（CPU 上极慢）。非 float32 时前向包在 torch.autocast("cuda", dtype=...) 里
- 抽不到帧或单个视频前向抛异常时，该行所有张量为空、status 记录原因（error: no frames / error: <异常信息>），不中断整列；每个视频处理完调 torch.cuda.empty_cache()
- 输入支持 video_url（含 BOS 路径）、video_base64、video_binary 三种形态，由 video_src_type 指定；后两者按 video_format 解析容器格式，整列抽帧失败时整列返回空帧

算子参数

输入

输入	含义
videos	视频列。按 video_src_type 取值分别是视频路径/URL（含 bos://）、base64 字符串或二进制内容

输出

输出为一个 struct 列，字段如下（tensor 均指 struct<shape: list<int32>, data: list<float32>>，data 是按行优先展平的 float32）：

输出	含义
frame_count	int32，实际进入前向的帧数（即预处理后张量的第 0 维）
image_size	list，预处理后输入尺寸 [H, W]，失败时为空
camera_parameters.extrinsic	tensor，相机外参，形状 (1,S,3,4)；if_output_camera_parameters=False 时为空
camera_parameters.intrinsic	tensor，相机内参，形状 (1,S,3,3)；同上开关控制
depth_maps.depth_map	tensor，深度图，形状 (1,S,H,W,1)；if_output_depth_maps=False 时为空
depth_maps.depth_conf	tensor，深度置信度；模型未输出 depth_conf 时为空
point_maps_from_projection.point_map	tensor，点云头直接预测的点云图，形状 (1,S,H,W,3)；if_output_point_maps_from_projection=False 时为空
point_maps_from_projection.point_conf	tensor，上述点云的置信度；模型未输出时为空
point_maps_from_unprojection.point_map	tensor，由深度 + 内外参反投影得到的点云图，形状 (S,H,W,3)；默认关闭
status	string，success / error: no frames / error: <异常信息>

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	输入视频编码形式：video_url / video_base64 / video_binary，其他取值抛 ValueError
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"facebook/VGGT-1B"	相对 model_path 的 VGGT 权重子目录；model_path/model_name 不存在时抛 FileNotFoundError
vggt_source_path	str 或 None	None	vggt 源码目录，给出时 append 到 sys.path（镜像内预置源码的场景）；为 None 时要求 vggt 已 pip 安装
dtype	str	"bfloat16"	前向精度：bfloat16 / float16 / float32，其他取值抛 ValueError
sample_mode	str	"by_count_uniform"	抽帧模式，透传 VideoFrameSampler：by_count_uniform / by_interval_time / by_interval_frames / by_fps / by_timestamps
start_time_sec	float	0.0	抽帧起始秒
end_time_sec	float 或 None	None	抽帧结束秒，None 表示到视频末尾
count_k	int 或 None	3	by_count_uniform 模式抽帧数
interval_sec	float 或 None	None	by_interval_time 模式间隔秒
interval_frames	int 或 None	None	by_interval_frames 模式间隔帧
target_fps	float 或 None	None	by_fps 模式目标帧率
timestamps_sec	list[float] 或 None	None	by_timestamps 模式时间戳列表
max_frames	int 或 None	32	抽帧上限，VGGT 显存随帧数增长，默认 32 帧兜底
if_output_camera_parameters	bool	True	是否输出相机内外参
if_output_depth_maps	bool	True	是否输出深度图与置信度
if_output_point_maps_from_projection	bool	True	是否输出点云头直接预测的点云图
if_output_point_maps_from_unprojection	bool	False	是否输出由深度反投影得到的点云图，默认关（体积大且可由深度推出）
video_format	str	"mp4"	二进制 / base64 输入的容器格式
rank	int	0	多卡场景 worker 序号，设备取 cuda:rank % cuda_device_count

注意事项

- **权重缺失即失败**：算子在 __init__ 里检查权重是否存在，缺失时直接 FileNotFoundError，运行期不会联网下载。
- **vggt 源码不做运行时获取**：算子不会下载或安装 vggt，已有源码目录可用 vggt_source_path 指过去；vggt 不可 import 时 from vggt.models.vggt import VGGT 会直接 ImportError。
- **按单卡独占起**：显存与输出体积都随帧数、分辨率增长（深度 (1,S,H,W,1)、点云 (1,S,H,W,3)，518 宽下单帧就是 20 万像素量级）。测试与建议用法都是 num_gpus=1, concurrency=1, batch_size=1，靠 max_frames 控上限；point_maps_from_unprojection 除非下游确实要用，保持默认关闭。

- 该算子只用 PyTorch 前向，不需要自编 CUDA 扩展；Blackwell（sm_120）上已实测跑通，bf16 生效（cc=12 满足 >=8）。
- count_k 是上限而非保证值：短视频可能只抽到更少的帧，frame_count 以实际入参为准（实测 3 帧配置在短视频上抽到 2 帧）。
- CPU 上可以跑但极慢，算子会强制 float32 并打 warning，生产不要这么用。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.perception.vggt_reconstruct import VggtReconstruct

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "vggt",
        aihc_udf(
            VggtReconstruct,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "facebook/VGGT-1B",
                "count_k": 3,
                "if_output_point_maps_from_unprojection": True,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds = ds.with_column("frame_count", col("vggt")["frame_count"])
    ds = ds.with_column("status", col("vggt")["status"])
    ds.show()
```

图像三维人体重建（SAM 3D Body）

简介

SAM 3D Body 人体三维重建算子。对单张图像做人体检测 → （可选）分割 → FOV 估计 → SAM 3D Body 前向，输出每个人的三维网格、二维/三维关键点、MHR 姿态参数与相机平移。三方源码与权重需提前就位，算子内零网络零安装。

功能描述

- 四级流水线：vitdet 人体检测 → sam2 人体分割（可选）→ moge2 FOV 估计 → SAM 3D Body 重建，三个子模型通过 tools/build_detector.py / tools/build_sam.py / tools/build_fov_estimator.py 按文件路径加载（spec_from_file_location）
- sam-3d-body 源码目录由 sam_3d_body_repo_path 指定，sys.path 只在构建 estimator 与单张推理期间临时插入、用完即撤；源码目录不存在直接 FileNotFoundError，运行期不会 clone
- 主模型 ckpt 与 MHR 资源可显式给路径，为空时按 <model_path>/<model_name>/model.ckpt 和 <model_path>/<model_name>/assets/mhr_model.pt 推导，两者缺一即 FileNotFoundError
- 只有传了 segmentor_path 才会构建分割器，因此 use_mask=True 必须同时给 segmentor_path；detector_name 置空则跳过检测
- 上游 process_one_image 返回的是**每个人一个 dict 的 list**，逐人取 bbox / focal_length / pred_vertices / pred_keypoints_3d / pred_keypoints_2d / mhr_model_params / pred_cam_t；上游没有检测分数字段，输出里也不编造
- 张量统一存成 {shape, data}：形状 + 一维 float32 展平数据，不用多层嵌套 list
- 两个输出体积开关：max_instances 限制每张图保留的实例数，if_output_vertices=False 时不输出网格顶点（实测每人约 1.8 万顶点，人多时网格是输出体积主体）
- 未检出人体时 instances 为空列表、status 为 success: no person detected；单张图异常时该行 instances 为空、status 为 error: <异常信息>，不中断整列
- 输入支持 image_url（含 BOS 路径，先落到本地临时路径再推理）、image_base64、image_binary；后两者写成 /tmp 下的临时 jpg 文件后推理，用完即删
- 可视化（pyrender + EGL）本期不做

算子参数

输入

输入	含义
images	图像列。按 image_src_type 取值分别是图像路径/URL（含 bos://）、base64 字符串或二进制内容

输出

输出为一个 struct 列，字段如下（tensor 均指 struct<shape: list<int32>, data: list<float32>>，data 是按行优先展平的 float32；缺失时为 {"shape": [], "data": []}）：

输出	含义
instances	list，该图检出的人体实例，顺序与上游返回一致，受 max_instances 截断
instances[].bbox	list，人体检测框，取上游 bbox 的前 4 个值
instances[].focal_length	float32，该实例使用的焦距（上游 focal_length），取不到时为 0.0
instances[].vertices	tensor，人体网格顶点 (V,3)，实测 [18439,3]；if_output_vertices=False 时为空
instances[].keypoints_3d	tensor，三维关键点 (70,3)
instances[].keypoints_2d	tensor，二维关键点 (70,2)
instances[].pose_params	tensor，MHR 模型参数 mhr_model_params，实测 (204,)
instances[].translation	tensor，相机坐标系下的平移 pred_cam_t，(3,)
status	string，success / success: no person detected / error: <异常信息>

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	输入图像编码形式：image_url / image_base64 / image_binary，其他取值抛 ValueError
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"facebook/sam-3d-body-dinov3"	相对 model_path 的 SAM 3D Body 权重子目录
sam_3d_body_repo_path	str	"/opt/aihc/thirdparty/sam-3d-body"	预置的 sam-3d-body 源码目录，不存在抛 FileNotFoundError
checkpoint_path	str	""	主模型 ckpt 路径，空则取 <model_path>/<model_name>/model.ckpt
mhr_path	str	""	MHR 资源路径，空则取 <model_path>/<model_name>/assets/mhr_model.pt
detector_name	str	"vitdet"	人体检测器名；置空则不做检测
segmentor_name	str	"sam2"	人体分割器名
fov_name	str	"moge2"	FOV 估计器名；置空则不建 FOV 估计器
detector_path	str	""	检测器权重目录，空则用上游默认
segmentor_path	str	""	分割器权重目录；为空时不构建分割器
fov_path	str	""	FOV 模型权重路径，须是 MoGe 的 model.pt 文件
bbox_thresh	float	0.8	人体检测框置信度阈值，透传上游 process_one_image(bbox_thr=...)
use_mask	bool	False	是否把分割掩码喂给重建，需要同时配 segmentor_path
max_instances	int 或 None	None	每张图最多保留的人体实例数，None 表示不限
if_output_vertices	bool	True	是否输出人体网格顶点，关掉可大幅缩小输出体积
rank	int	0	多卡场景 worker 序号，设备取 cuda:rank % cuda_device_count

注意事项

- **算子内零网络零安装**：运行期不下载权重、不安装依赖、不拉源码；`sam_3d_body_repo_path` 指向的源码目录或权重缺失时，构造阶段直接抛 `FileNotFoundError`。
- `detector_path`**传目录，**`fov_path`**必须传**`model.pt`**文件**：vitdet 的 `detector_path` 指向包含 `model_final_f05665.pkl` 的目录；MoGe 的 `from_pretrained` 接受 `.pt` 文件或 HF repo id，`fov_path` 传目录会 `IsADirectoryError`。
- **sm_120（Blackwell）上 detectron2 需要自编 CUDA 扩展**：现成 wheel 不含 sm_120 kernel，需要按目标 GPU 架构自行编好 detectron2，算子内不做编译。
- **按**`concurrency=1`**起**：这条链路的并发安全性未经验证；本算子测试与建议用法均为 `num_cpus=1, num_gpus=1, concurrency=1, batch_size=1`。
- 依赖面很大：detectron2 + pytorch-lightning + hydra + pyrender 等 30 余个包，建议给它单独的运行环境。
- 输出体积：单人网格 [18439,3] 就是 5.5 万个 float32，人多的图建议配 `max_instances`，只要姿态参数与关键点时把 `if_output_vertices` 关掉。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.perception.image_sam_3d_body import ImageSam3dBody

MODEL_PATH = "/path/to/models"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"image": ["bos://your-bucket/sample.jpg"]})
    ds = ds.with_column(
        "body",
        aihc_udf(
            ImageSam3dBody,
            construct_args={
                "image_src_type": "image_url",
                "model_path": MODEL_PATH,
                "model_name": "facebook/sam-3d-body-dinov3",
                "sam_3d_body_repo_path": "/path/to/thirdparty/sam-3d-body",
                # 检测器传目录（内含 model_final_f05665.pkl）
                "detector_path": f"{MODEL_PATH}/sam3d_vitdet",
                # FOV 传 MoGe 的权重文件，不能传目录
                "fov_path": f"{MODEL_PATH}/Ruicheng/moge-2-vitl/model.pt",
                "bbox_thresh": 0.5,
                "max_instances": 4,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds = ds.with_column("status", col("body")["status"])
    ds.show()
```

视频相机位姿估计（MegaSaM）

简介

视频相机位姿估计算子。抽帧后先用 MoGe-2 逐帧估计单目深度与内参，再把 (image, depth, intrinsics) 喂给 MegaSaM 的 DROID-SLAM 做跟踪与全局 bundle adjustment，输出相机内参与每帧 camera-to-world 位姿。仅支持 GPU，三个 CUDA 扩展与全部权重必须离线预置，算子内零网络零编译。

功能描述

- 两级流水线：MoGe-2 出的是**归一化内参**，乘回 width / height 变成像素单位；第二级把每帧等比缩放到 384×512 面积、裁到 8 的倍数，内参同步按缩放比修正，深度用 `nearest-exact` 插值到同尺寸
- DROID 吃的是原始通道序（BGR），本实现不做转换；落 npz 时才转回 (T,H,W,3) RGB，与 mega-sam `cvd_opt` 的约定一致
- 逐帧 `track` → `track_final` → `terminate(_opt_intr=opt_intrinsics, full_ba=full_ba)` 做全局 BA；内参取 `droid.video.intrinsics[0] * 8.0`（SLAM 在 1/8 特征分辨率上算），位姿用 `lietorch.SE3(traj).inv().matrix()` 从 w2c 转成 c2w
- DROID 全部超参（`buffer` / `filter_thresh` / `warmup` / `keyframe_thresh` / frontend 四项 / backend 三项 / `beta`）逐个开放为算子参数，以普通对象而非 dict 传入（DROID 内部直接读属性）
- `count_k`**默认 64**：DROID 的 `warmup=8` 意味着 3 帧根本起不了轨迹；另加前置检查，抽到的帧数少于 `warmup` 时直接给出「调大 `count_k` / `max_frames`」的错误信息，而不是让 SLAM 崩在内部
- **大张量不进列**：`intrinsic` / `cam_c2w` 以 {shape, data}（形状 + 一维 float32）存出，`depths` 默认关闭，images 完全不进列（一段 300 帧视频的 images 与 depths 是百 MB 量级）。需要完整产物时设 `output_npz_dir`，落成键名与 mega-sam `cvd_opt` 一致（`images` / `depths` / `intrinsic` / `cam_c2w`）的 npz，文件名为 <视频名>_droid.npz（视频名取路径/URL 去参数后的 stem，取不到用 `video_<行号>`）
- `torch.load` 打补丁：mega-sam 的 `Droid.load_weights` 是裸 `torch.load`，torch>=2.6 默认 `weights_only=True` 会失败，构建 Droid 期间临时包一层 `weights_only=False`，构建完立即还原
- 两个源自 DROID 本身的既有行为：① `full_ba=True` 时 DROID 返回的是**关键帧**轨迹 `video.poses[:counter]`，逐帧的 `traj_filler` 结果在这条分支里被算出来又丢弃，所以 `frame_count` 通常小于抽帧数（实测抽 32 帧出 31 个关键帧）；② 输出的 `depths` 是 MoGe 深度按 SLAM 分辨率缩放后的结果，**不是** BA 优化后的 `depth_est`（BA 返回的优化深度未被使用）
- 抽帧走 `VideoFrameSampler` 在内存里完成，不把帧写盘；输入支持 `video_url`（含 BOS 路径）、`video_base64`、`video_binary`
- 抽不到帧或单个视频估计失败时，该行张量为空、`npz_path` 为空串、`status` 记录原因，不中断整列；每个视频处理完调 `torch.cuda.empty_cache()`

算子参数

输入

输入	含义
videos	视频列。按 video_src_type 取值分别是视频路径/URL（含 bos://）、base64 字符串或二进制内容

输出

输出为一个 struct 列，字段如下（`tensor` 均指 `struct<shape: list<int32>, data: list<float32>>`，`data` 是按行优先展平的 float32；缺失时为 {"shape": [], "data": []}）：

输出	含义
frame_count	int32，输出的位姿帧数，等于 min(output_max_frames, 关键帧数)；失败时 0
image_size	list，SLAM 分辨率下的 [H, W]，实测 [384, 512]
intrinsic	tensor，BA 后的像素单位内参矩阵，形状 (3,3)，fx=data[0]、fy=data[4]
cam_c2w	tensor，每帧 camera-to-world 位姿，形状 (frame_count,4,4)，每个 4×4 的最后一行是 [0,0,0,1]
depths	tensor，深度图 (frame_count,H,W)；if_output_depths=False（默认）时为空
npz_path	string，落盘的 npz 绝对路径（键 images / depths / intrinsic / cam_c2w）；未设 output_npz_dir 时为空串
status	string，success / error: no frames / error: <异常信息>

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	输入视频编码形式：video_url / video_base64 / video_binary，其他取值抛 ValueError
model_path	str	"/opt/aihc/model"	权重根目录
moge_model_name	str	"Ruicheng/moge-2-vitl/model.pt"	相对 model_path 的 MoGe-2 权重，须是 .pt 文件；默认使用 Ruicheng/moge-2-vitl
megasam_repo_path	str	"/opt/aihc/thirdparty/mega-sam"	mega-sam 源码根目录，须含 base/droid_slam
megasam_ckpt_name	str	"megasam/megasam_final.pth"	相对 model_path 的 DROID 权重
sample_mode	str	"by_count_uniform"	抽帧模式，透传 VideoFrameSampler：by_count_uniform / by_interval_time / by_interval_frames / by_fps / by_timestamps
start_time_sec	float	0.0	抽帧起始秒
end_time_sec	float 或 None	None	抽帧结束秒，None 表示到视频末尾
count_k	int 或 None	64	by_count_uniform 模式抽帧数。默认取 64 是因为帧数过少（如 3 帧）起不了 DROID 轨迹
interval_sec	float 或 None	None	by_interval_time 模式间隔秒
interval_frames	int 或 None	None	by_interval_frames 模式间隔帧
target_fps	float 或 None	None	by_fps 模式目标帧率
timestamps_sec	list[float] 或 None	None	by_timestamps 模式时间戳列表
max_frames	int 或 None	300	抽帧上限，显存与耗时随帧数线性增长
buffer	int	1024	DROID 关键帧缓冲区大小，短片可下调省显存
filter_thresh	float	2.0	MotionFilter 运动阈值，低于此值的帧不作为关键帧
warmup	int	8	frontend 启动所需帧数；抽帧数小于该值时算子直接报错
keyframe_thresh	float	2.0	关键帧剔除阈值
frontend_window	int	25	frontend 优化窗口
frontend_thresh	float	12.0	frontend 边构建距离阈值
frontend_radius	int	2	frontend 邻接半径
frontend_nms	int	1	frontend 非极大抑制
backend_thresh	float	16.0	backend 边构建距离阈值
backend_radius	int	2	backend 邻接半径
backend_nms	int	3	backend 非极大抑制
beta	float	0.3	几何 / 运动项权重
opt_intrinsics	bool	True	是否在 BA 里一并优化内参（对应 DROID terminate 的 _opt_intr）；DROID 内部若判定焦距不可观测会自动关掉
full_ba	bool	True	是否做全帧 BA（对应 DROID terminate 的 full_ba）
output_max_frames	int	1000	输出帧数上限
if_output_depths	bool	False	是否把深度图写进输出列，默认关（单帧 384×512 就是 20 万个 float32）
output_npz_dir	str 或 None	None	非空时把 images / depths / intrinsic / cam_c2w 落成 npz，目录会自动创建
video_format	str	"mp4"	二进制 / base64 输入的容器格式
rank	int	0	多卡场景 worker 序号，只影响 MoGe 的落卡

注意事项

- **三个 CUDA 扩展需要提前按目标 GPU 架构编好**：droid_backends、lietorch、torch_scatter。算子只做 import 检查，缺失时抛 ImportError，运行期不做编译。
- **编译产物与 python 版本、torch 版本、GPU sm 架构三重绑定**：换 python 小版本、换 torch 或换卡都得重编；要同时覆盖多种架构的卡，需要编出含对应架构的 fat binary。
- **必须有 GPU**：无 CUDA 时 __init__ 直接 RuntimeError，DROID-SLAM 的 CUDA 扩展没有 CPU 实现。
- moge_model_name**必须指到** .pt**文件**：MoGe 的 from_pretrained 传目录会被当成 HF repo id 处理而报错。megasam_ckpt_name 同样指向具体的 .pth 文件，两个路径任一不存在即 FileNotFoundError；megasam_repo_path 下缺 base/droid_slam 也会直接报错。
- **多卡靠 worker 级隔离**：Droid.load_weights 内部把网络固定放到 cuda:0，算子的 rank 只影响 MoGe 落卡，多卡场景需要 worker 级 CUDA_VISIBLE_DEVICES。
- **按**concurrency=1**起**：测试用法是 num_cpus=4, num_gpus=1, concurrency=1, batch_size=1；buffer 默认 1024 个关键帧槽位占显存，短片建议下调（测试用 256）。实测 test_video.mp4 抽 32 帧全程 85 秒，npz 42.7M。
- **首帧位姿只是接近单位矩阵**：DROID 以第一个关键帧为世界原点，但 full BA 会把所有位姿（含首帧）一起再优化、不做 gauge 固定，实测首帧 c2w 与单位矩阵偏差约 2e-3。做校验时要留容差。
- mega-sam README 里要求的 DepthAnything / RAFT 权重是它自己 mono-depth 与 cvd_opt 流程用的，本算子这条路径**不需要**。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.perception.video_camera_pose import VideoCameraPose

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "pose",
        aihc_udf(
            VideoCameraPose,
            construct_args={
                "model_path": "/path/to/models",
                "megasam_repo_path": "/path/to/thirdparty/mega-sam",
                "count_k": 32,
                "buffer": 256,
                "output_npz_dir": "/tmp/camera_pose_npz",
            },
            num_cpus=4,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds = ds.with_column("frame_count", col("pose")["frame_count"])
    ds = ds.with_column("status", col("pose")["status"])
    ds.show()
```

旋转表示转换

简介

旋转表示互转算子，支持 axis_angle / euler_angles / quaternion / rotation_6d / matrix 五种表示两两互转，全部经旋转矩阵中转。移植 GR00T-N1.5 gr00t/data/transform/state_action.py 的 RotationTransform（原版来自 diffusion_policy 的 rotation_transformer），不依赖 pytorch3d。

功能描述

- 五种表示的维度固定：axis_angle 3、euler_angles 3、quaternion 4、rotation_6d 6、matrix 9（3×3 行优先展平）
- 一律经 matrix 中转，forward/inverse 对称：库函数 convert_rotation(values, from_rep, to_rep) 把两个表示名对调就是逆变换。单射域内数值严格可逆（轴角范数 < π、规范 6d、matrix）；euler_angles 同一旋转有多组解、quaternion 有 ± 双覆盖，往返只保证旋转等价，不保证逐位相同
- 不引 pytorch3d（运行机未装）：欧拉角 / 轴角 / 四元数 ↔ matrix 走 scipy.spatial.transform.Rotation（pytorch3d 的同串大写 intrinsic 欧拉约定与 scipy 一致），rotation_6d ↔ matrix 自实现——6d 即旋转矩阵**前两行**展平，反向按 b1=normalize(a1)、b2=Gram-Schmidt(a2, b1)、b3=b1×b2
- 四元数沿用 GR00T / pytorch3d 的 **wxyz**（scalar-first）布局，进出 scipy 时重排为 xyzw；输入四元数先归一化
- 欧拉约定解析：euler_angles（默认 XYZ）、euler_angles_XYZ（大写轴串原样）、euler_angles_zyx（小写轴名转大写）、GR00T 缩写 euler_angles_rpy（r→X、p→Y、y→Z）；其它写法报错
- joint_dims 按给定顺序取出选中列组成旋转段，段长须是源表示维度的整数倍，因此一段里可以有多组旋转（如 36 维 = 6 组 rotation_6d），输出恒为展平的 组数 × 目标维度
- keep_other_dims=True 时未选中列按原相对顺序保留，转换段整体插在**最左选中列**的位置
- rotation_6d → matrix 的归一化用 eps 兜住零向量，不会除零
- from_rep 与 to_rep 相同（含欧拉约定相同）在构造时报错
- 行级容错：空行 / null / 段长不是源表示维度整数倍 / 下标越界，该行返回空列表，不打断整批

算子参数

输入

输入	含义
state	每行一个 1D 向量（list[float]），joint_dims 选中的列是旋转数据

输出

输出	含义
result	list。默认只输出转换后的段（长度 = 组数 × 目标表示维度）；keep_other_dims=True 时输出「其余维 + 转换段」拼接后的整行；失败行为空列表

参数

参数名称	类型	默认值	描述
from_rep	str	"axis_angle"	源表示，五种表示之一，欧拉可带约定后缀
to_rep	str	"rotation_6d"	目标表示，不能与 from_rep 相同
joint_dims	list[int] 或 None	None	参与转换的列下标，按给定顺序取值；None 表示整行都是旋转数据。须非空且不重复
keep_other_dims	bool	False	True 时未选中列一并输出
eps	float	1e-8	rotation_6d → matrix 归一化时的除零下限

注意事项

- 输入段长必须是源表示维度的整数倍，否则该行返回空列表（例如 from_rep="axis_angle" 时 2 维的行会被判为坏行）。
- rotation_6d 的转换隐含正交化：任意 6 维向量都会被 Gram-Schmidt 投影成合法旋转，第二次转换起是幂等的，但第一次会改值。上游若不是真旋转数据（例如把关节角当 6d 用），结果没有物理含义。
- matrix 是 3×3 按行优先展平的 9 维，不要按列优先解释。
- keep_other_dims=True 的输出不是「按原下标替换」：转换段整体落在最左选中列的位置，未选中列只保持彼此的相对顺序。
- 输出列长度随 from_rep / to_rep 变化，下游 schema 要同步（如 3 维轴角 → 6 维 6d）。

调用示例

```
 from __future__ import annotations

import os

import daft
import pyarrow.parquet as pq
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.rotation_rep_convert import RotationRepConvert

ROOT = "/path/to/lerobot_dataset"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

table = pq.read_table(
        f"{ROOT}/data/chunk-000/episode_000000.parquet", columns=["observation.state"]
    )
    ds = daft.from_pydict({"state": table.column("observation.state").to_pylist()})
    ds = ds.with_column(
        "result",
        aihc_udf(
            RotationRepConvert,
            construct_args={
                "from_rep": "rotation_6d",
                "to_rep": "matrix",
                "joint_dims": list(range(6)),
                "keep_other_dims": True,
                "eps": 1e-8,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=256,
        )(col("state")),
    )
    ds.show()
```

关节角 SinCos 编码

简介

关节角 sin/cos 双通道编码算子，把角度维映射成 [sin(θ), cos(θ)]，消除角度的 2π 周期性歧义（VLA 训练的标准输入形式）。移植 GR00T-N1.5 gr00t/data/transform/state_action.py 的 StateActionSinCosTransform.apply。

功能描述

- 编码结果沿最后一维拼接成 [sin(sel), cos(sel)]（与 GR00T 原版的拼接顺序一致），选中 n 维输出 2n 维；不是逐维交错的 [sin, cos, sin, cos...]
- joint_dims=None 时整行参与编码，此时行为与 GR00T 原版完全一致；给定 joint_dims 时只编码选中列，且按 joint_dims 给定的顺序取值
- keep_original=True 时输出 [原始整行, sin(sel), cos(sel)]，长度为 原维数 + 2 × 选中维数，便于回溯原值；默认只输出编码段
- 输入按弧度处理，不做任何角度归一化（sin/cos 本身对 ±2π 不敏感）
- joint_dims 为空或有重复在构造时报错；下标越界在处理该行时报错
- 行级容错：空行 / null / 非 1D / 下标越界，该行返回空列表，不打断整批
- 纯 numpy，CPU，无模型

算子参数

输入

输入	含义
state	每行一个 1D 向量（list[float]，单位弧度）

输出

输出	含义
result	list。默认为 [sin(sel), cos(sel)]（长度 = 2 × 选中维数）；keep_original=True 时为 [原始整行, sin(sel), cos(sel)]；失败行为空列表

参数

参数名称	类型	默认值	描述
joint_dims	list[int] 或 None	None	参与编码的列下标，按给定顺序取值；None 表示整行全部编码。须非空且不重复
keep_original	bool	False	True 时在编码段前面拼上原始整行

注意事项

- 输出维度会翻倍（keep_original=True 时是 原维数 + 2 × 选中维数），下游 schema 与训练配置要同步改。
- 只对角度维有物理意义。夹爪开合度、末端位置这类非角度维一起编码没有意义，用 joint_dims 圈出角度维。
- 编码不可逆推回具体的角度多值分支：atan2(sin, cos) 只能还原到 (-π, π]，原值超出这个区间时拿不回来。

调用示例

```
 from __future__ import annotations

import os

import daft
import pyarrow.parquet as pq
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.sincos_joint_encode import SinCosJointEncode

ROOT = "/path/to/lerobot_dataset"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

table = pq.read_table(
        f"{ROOT}/data/chunk-000/episode_000000.parquet", columns=["observation.state"]
    )
    ds = daft.from_pydict({"state": table.column("observation.state").to_pylist()})
    ds = ds.with_column(
        "result",
        aihc_udf(
            SinCosJointEncode,
            construct_args={
                "joint_dims": None,
                "keep_original": False,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=256,
        )(col("state")),
    )
    ds.show()
```

状态动作归一化

简介

state / action 归一化算子，按数据集统计量做 mean_std / min_max / q99 / scale / binary 五种归一化，并支持反归一化。移植 GR00T-N1.5 gr00t/data/transform/state_action.py 的 Normalizer，统计量可以直接吃 lerobot 数据集的 meta/stats.json。

功能描述

- 五种模式的 forward 与 GR00T 逐分支一致：
- ◦mean_std：(x - mean) / std，std == 0 的维透传原值
- ◦min_max：2*(x - min)/(max - min) - 1，min == max 的维置 0，不做截断
- ◦q99：2*(x - q01)/(q99 - q01) - 1 再截断到 [-1, 1]，q01 == q99 的维透传后同样受截断
- ◦scale：x / max(|min|, |max|)，abs_max == 0 的维置 0
- ◦binary：(x > 0.5) → {0, 1}，不需要统计量
- inverse=True 走反变换；scale 的逆在原版里缺失，此处按 x * abs_max 补齐，其余分支与原版语义一致
- 统计量三种来源：构造参数 statistics（dict）优先，其次读 statistics_path 指向的 stats.json，binary 模式无需统计量。dict 既可以是内层单特征形式 {"mean": [...], "std": [...]}，也可以是外层映射 {"observation.state": {...}, "action": {...}}——外层映射有多个特征时必须给 feature_key，只有一个特征时自动选中
- 构造时校验本模式需要的统计键（mean_std→mean/std，min_max/scale→min/max，q99→q01/q99），缺键立即报错；处理每行时校验向量维度与统计量维度一致
- 输出 struct 回填本模式 forward/inverse 实际用到的统计量，未用到的字段为 null，下游不必重读 stats.json 就能反演
- 行级容错：空行 / null / 维度不符时 normalized 为空列表且统计字段全 null（mode 仍回填），不打断整批
- 纯 numpy，CPU，无模型

算子参数

输入

输入	含义
state	每行一个 1D 向量（list[float]），维度须与统计量一致（binary 模式无此约束）

输出

输出	含义
result.normalized	list，归一化结果；inverse=True 时是还原值；失败行为空列表
result.mode	string，本次使用的模式，取值 mean_std / min_max / q99 / scale / binary
result.mean	list，mean_std 模式回填，其它模式为 null
result.std	list，mean_std 模式回填，其它模式为 null
result.min	list，min_max 与 scale 模式回填，其它模式为 null
result.max	list，min_max 与 scale 模式回填，其它模式为 null
result.q01	list，q99 模式回填，其它模式为 null
result.q99	list，q99 模式回填，其它模式为 null

参数

参数名称	类型	默认值	描述
mode	str	"mean_std"	归一化模式，取值 mean_std / min_max / q99 / scale / binary
statistics	dict 或 None	None	直接给统计量，内层单特征 dict 或外层 feature 映射；优先于 statistics_path
statistics_path	str 或 None	None	stats.json 路径（lerobot / 数据集统计算子写出的格式），文件不存在报错
feature_key	str 或 None	None	外层映射里取哪个特征，如 observation.state；多特征时必填
inverse	bool	False	True 走反归一化

注意事项

- 退化维不可逆：`mean_std` 的 `std == 0` 维 forward 透传原值，inverse 算的是 `x * 0 + mean`，还原不回原值；`min_max` 的 `min == max` 维 forward 置 0，inverse 得到 `min`。做往返校验时要把退化维排除。
- `q99` 的截断也不可逆：落在 [q01, q99] 之外的值被夹到 ±1，反归一化只能回到分位边界。
- `min_max` 与 `scale` 不做截断（GR00T 原语义），源值超出统计区间时输出会超出 [-1, 1]。
- `binary` 模式的 inverse 与 forward 相同（都是 `x > 0.5`），只是幂等，不能还原原值。
- 输入维度必须与统计量维度严格相等，否则该行返回空列表；不会按前缀对齐。
- 一个算子实例只带一个特征的统计量。`observation.state` 和 `action` 要各起一个实例（`feature_key` 不同），反归一化再起一个 `inverse=True` 的实例。

调用示例

```
 from __future__ import annotations

import os

import daft
import pyarrow.parquet as pq
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.normalize_state_action import NormalizeStateAction

ROOT = "/path/to/lerobot_dataset"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

table = pq.read_table(
        f"{ROOT}/data/chunk-000/episode_000000.parquet", columns=["observation.state"]
    )
    ds = daft.from_pydict({"state": table.column("observation.state").to_pylist()})
    ds = ds.with_column(
        "result",
        aihc_udf(
            NormalizeStateAction,
            construct_args={
                "mode": "mean_std",
                "statistics_path": f"{ROOT}/meta/stats.json",
                "feature_key": "observation.state",
                "inverse": False,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=256,
        )(col("state")),
    )
    ds.show()
```

LeRobot 数据集 ROI 裁剪

简介

LeRobot v3.0 数据集图像观测的 ROI 裁剪算子，按给定或自动推导的 ROI 裁剪每帧图像并 resize，产出一个新的 v3.0 数据集（源数据集只读）。移植 lerobot `rl/crop_dataset_roi.py` 的 `convert_lerobot_dataset_to_cropped_lerobot_dataset`，把上游的 cv2 窗口交互选框换成非交互配置。

功能描述

- 两种 ROI 模式：
- ◦`manual`：ROI 来自 `crop_params`，形如 {image_key: (top, left, height, width)}

- ◦center_square：对首帧里键名含 image 且形状为 3 维（C, H, W）的每个观测键，自动推导居中正方形 ROI（边长 min(H, W)，偏移 (H-side)//2 / (W-side)//2），是上游 select_square_roi_for_images 的非交互替代
- 裁剪后统一 resize 到 resize_size（(height, width)），并把 meta/info.json 里对应特征的 shape 同步改成 [3, height, width]
- 逐帧走 lerobot writer API（LeRobotDataset.create + add_frame + save_episode），episode 按 episode_index 变化切分；输出 repo_id 为 <源 repo_id>_cropped，fps / robot_type / features 沿用源数据集，源有 video 特征时输出也写视频
- task 保留：task=None（默认）时按 task_index 逐帧回填原 task 文本（上游强制所有帧写同一个字符串）；显式给 task 则整体覆盖
- dtype 归一：dataset[idx] 把特征统一成 float32 张量，写回时按目标 features 声明的 dtype（float32 / float64）转回；裁剪 resize 后的图像 clamp 到 [0, 1]
- 非图像特征（state / action 等）原样透传；task_index / timestamp / episode_index / frame_index / index / task 这些由 writer 自己生成的列不复制
- 输入校验：源目录必须是完成态 v3.0（meta/info.json 的 codebase_version == v3.0）；manual 模式缺 crop_params、ROI 的 top/left 为负或 height/width 非正都报错；center_square 一个图像键都没命中时报错
- 幂等与崩溃安全：输出已是完成态 v3.0 且 force=False 时跳过并返回 SKIPPED；转换中途失败会删掉半成品输出目录；写完再核一次输出是不是 v3.0，不是就报错
- 行级容错：dataset_path / output_path 缺失或转换抛异常都返回 Failed: ...，不打断整批

算子参数

输入

输入	含义
dataset_path	源 v3.0 数据集根目录，只读
output_path	输出数据集根目录，裁剪后的 v3.0 数据集直接写到这里

输出

输出	含义
result	string：SUCCESS / SKIPPED: output already exists / Failed: dataset_path and output_path are required / Failed: [<dataset_path>] <异常类型>: <信息>

参数

参数名称	类型	默认值	描述
roi_mode	str	"manual"	manual 用 crop_params；center_square 每个图像键自动取居中正方形
crop_params	dict 或 None	None	{image_key: (top, left, height, width)}，manual 模式必填；center_square 模式下会被自动推导的 ROI 覆盖
resize_size	tuple[int,int] 或 list[int]	(128, 128)	裁剪后的目标 (height, width)
task	str 或 None	None	覆盖所有帧的 task 文本；None 保留每帧原有 task
force	bool	False	输出已是完成态 v3.0 时也重建

注意事项

- crop_params 的键必须与 meta/info.json 的特征名完全一致（如 observation.images.cam）。写错的键不会报错，也不会裁剪，只是静默不生效。
- ROI 只校验 top/left ≥ 0、height/width > 0，不校验是否落在源图像内；超界部分由 torchvision F.crop 处理，不会报错。
- 输出目录会被清空重建：只要没走 SKIPPED 分支，已存在的 output_path 会被整个 rmtree 掉，不要把输出指向还有别的数据的目录。
- 图像 shape 元数据只对 crop_params / 自动推导命中的键更新。手工改动 ROI 或 resize 尺寸后必须让算子重跑（force=True），否则元数据与实际帧尺寸会不一致。
- 全量逐帧解码 + 逐帧重编码，耗时随总帧数线性增长；一行任务处理一个数据集，并发靠行数（batch_size=1），不要指望单行内并行。
- state / action 不随裁剪变化，下游训练配置需要自己对齐新的图像尺寸。

调用示例

```python
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.crop_dataset_roi_udf import CropDatasetROI

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        # 源目录必须是完成态 v3.0 数据集（meta/info.json 的 codebase_version == v3.0）
        "dataset_path": ["/path/to/lerobot_dataset"],
        "output_path": ["/tmp/lerobot_crop_out"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            CropDatasetROI,
            construct_args={
                # 手动 ROI：{"roi_mode": "manual",
                #            "crop_params": {"observation.images.cam": (0, 0, 24, 32)},
                #            "resize_size": (16, 16)}
                "roi_mode": "center_square",
                "resize_size": (32, 32),
                "task": None,
                "force": False,
            },
            num_cpus=2,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path"), col("output_path")),
    )
    ds.show()
```

LeRobot 数据集统计量计算

简介

对已落盘的 LeRobot v3.0 数据集重算 meta/stats.json。lerobot 0.4.4 只在写数据集的过程中算统计（LeRobotDataset.save_episode → compute_episode_stats → aggregate_stats → write_stats），没有给已成型数据集重算的入口，本算子补这个缺口：表列特征直接从 parquet 归约，视频特征抽帧统计，聚合与落盘复用 lerobot 自己的 aggregate_stats / write_stats。

源码锚点（lerobot 0.4.4）：

锚点	作用
datasets/compute_stats.py:477 compute_episode_stats	单 episode 逐特征统计
datasets/compute_stats.py:605 aggregate_stats	跨 episode 聚合
datasets/utils.py:295 write_stats	落盘 meta/stats.json
datasets/video_utils.py:127 decode_video_frames	按时间戳取视频帧

功能描述

- 表列特征（state / action 等）直接读 data/**/*.parquet：pyarrow 列投影 + 按 episode_index 行过滤，成本随请求的特征数走，不随数据集整体规模走
- 视频 / 图像特征按 video_frames_per_episode 个等距时间戳抽帧（decode_video_frames），逐通道在 (0, 2, 3) 上归约后除以 255，形状 (3, 1, 1)，与写入端 compute_episode_stats 的图像分支口径一致；从不全量解码
- 统计集合与写入端一致：跳过 index / episode_index / task_index 三个元数据列和所有 string 特征；features 白名单只能在这个集合内再收窄，传了未知键或元数据键直接报错并列出可用键
- 逐 episode 统计用 lerobot aggregate_stats 聚合、write_stats 落盘，产出布局与新写数据集的 meta/stats.json 一致
- v3.0 里多个 episode 顺序共享一个 mp4，采样区间按该 episode 元数据的 videos/<key>/from_timestamp 偏移，不是从文件 0 秒起算
- 抽样型数据集的 mp4 常出现「帧率元数据与真实 pts 栅格不一致 / 物理视频比元数据声称的短」，为此有三层防御：按容器 duration 与帧数钳制采样区间、采样点向下（floor）吸附到帧栅格且容差抬到 1/4 帧、按时间戳解码抛异常时回落到 pyav 按帧索引抽样（最多解 1024 帧后均匀取样）
- num_workers > 1 时 episode 级线程池并发（视频解码释放 GIL）
- skip-if-done：meta/stats.json 已存在且 force=False 时不重算，返回 SKIPPED，并把已有 stats.json 的内容原样回填到 features 里
- 每个特征的统计字典以 JSON 字符串返回，输出 schema 不随数据集特征集变化（__return_column_type__ 是静态的）
- write=False 时只算不落盘，stats_path 返回空串

算子参数

输入

输入	含义
dataset_path	已完成的 v3.0 数据集根目录（meta/info.json 存在且 codebase_version == "v3.0"）

输出

输出	含义
result.status	SUCCESS / SKIPPED: stats.json already exists / Failed: dataset_path is required / Failed: [<dataset_path>] <异常类型>: <信息>
result.stats_path	落盘的 meta/stats.json 路径；write=False 或失败时为空串
result.features	list，每个参与统计的特征一条：name = 特征名，stats_json = 该特征统计字典的 JSON 字符串（min / max / mean / std / count / q01 / q10 / q50 / q90 / q99）

参数

参数名称	类型	默认值	描述
features	list[str] 或 None	None	特征白名单。None 表示所有非元数据、非 string 特征
num_workers	int	1	episode 级并发线程数，取 max(1, num_workers)，1 为串行
video_frames_per_episode	int	32	每个 episode 每路视频抽多少帧做统计，取 max(1, ...)；实际抽样数为 min(该值, 该 episode 帧数)
tolerance_s	float	1e-4	视频取帧的时间戳容差；实际生效值会被抬到至少 1/4 帧（1/(4*fps)）
write	bool	True	是否把聚合结果写回 meta/stats.json
force	bool	False	True 表示 meta/stats.json 已存在也重算

注意事项

- 分位数 q01..q99 的聚合语义是「各 episode 分位数按帧数加权平均」（lerobot aggregate_feature_stats），不是全量数据的真分位数；单 episode 内部也是 5000 bin 直方图近似（RunningQuantileStats），样本不足 2 行时 q* 直接取 mean。
- 图像统计前会走 lerobot auto_downsample_height_width（长边 ≥300 时按整数因子下采样到约 150 像素），这是写入端的既有行为，统计值与全分辨率精确统计会有细微差别。
- 视频 / 图像特征统计是逐通道 3 个值、值域 [0,1]；表列特征逐维。
- 输入不是完成态 v3.0 数据集时直接失败，不会去猜 v2.x 布局。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.compute_dataset_stats_udf import (
    ComputeDatasetStats,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "dataset_path": ["/path/to/lerobot_dataset"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ComputeDatasetStats,
            construct_args={
                "features": None,
                "num_workers": 1,
                "video_frames_per_episode": 32,
                "write": True,
                "force": False,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path")),
    )
    ds.show()
```

LeRobot 分位数统计补齐

简介

给 meta/stats.json 里没有分位数（q01 / q10 / q50 / q90 / q99）的旧 v3.0 数据集补算统计。lerobot 只把这件事做成了 CLI 脚本，本算子把它包成库调用：判重用 has_quantile_stats，计算复用 compute_quantile_stats_for_dataset（逐 episode 遍历 dataset[idx] → get_feature_stats → aggregate_stats），结果用 lerobot write_stats 落盘。

源码锚点（lerobot 0.4.4）：datasets/v30/augment_dataset_quantile_stats.py:53 has_quantile_stats、:129 compute_quantile_stats_for_dataset、datasets/utils.py:295 write_stats。

功能描述

- 判重口径来自上游 has_quantile_stats：meta/stats.json 里任意一个特征带任意一个 q** 键就算「已有分位数」，overwrite=False 时直接返回 SKIPPED，并把现有 stats 原样回填到 features
- 计算走 lerobot 自己的 episode 循环：按 dataset_from_index / dataset_to_index 逐帧取 dataset[idx]，所以视频特征走的是数据集标准随机访问解码路径（真解码，不抽帧）
- 并发策略沿用上游：数据集不含 video key 时用最多 16 个线程并发处理 episode；含 video key 时强制顺序处理（视频解码非线程安全）
- 统计口径：跳过 string 特征；图像 / 视频的 uint8 数据先除 255 再在 (0, 2, 3) 上逐通道归约；分位数集合是 lerobot DEFAULT_QUANTILES = [0.01, 0.10, 0.50, 0.90, 0.99]
- 整份统计重算后覆盖写 meta/stats.json（不是只往里补 q* 键），mean / std / min / max 也一起刷新
- 与上游脚本的差异：不做 push_to_hub 与 hub 重打 tag（AIHC 数据集在 BOS 上，不进 HF hub）；不接受 repo_id，逻辑数据集名固定取数据集目录名

- 每个特征的统计字典以 JSON 字符串返回，输出 schema 不随数据集特征集变化

算子参数

输入

输入	含义
dataset_path	已完成的 v3.0 数据集根目录（meta/info.json 存在且 codebase_version == "v3.0"）

输出

输出	含义
result.status	SUCCESS / SKIPPED: quantile stats already exist / Failed: dataset_path is required / Failed: [<dataset_path>] <异常类型>: <信息>
result.stats_path	meta/stats.json 路径；失败时为空串
result.features	list，每个特征一条：name = 特征名，stats_json = 该特征统计字典的 JSON 字符串（min / max / mean / std / count / q01 / q10 / q50 / q90 / q99）

参数

参数名称	类型	默认值	描述
overwrite	bool	False	True 表示已有分位数也重算并覆盖 meta/stats.json

注意事项

- 分位数的聚合语义是「各 episode 分位数按帧数加权平均」，不是全量数据的真分位数；episode 内部又是 5000 bin 直方图近似。小 episode（几十帧量级）上与 `np.quantile` 有 1% 量级偏差，这是上游实现的口径，不是误差 bug。
- 内存：上游 `process_single_episode` 会把一个 episode 的所有帧 `torch.stack` 成一个张量，图像 / 视频数据集按「单 episode 全解开」估内存；本算子未改这个行为，长 episode 需要预留内存。
- 与「LeRobot 数据集统计量计算」的分工：本算子照抄 lerobot 的全量遍历口径（视频真解码、逐帧精确），代价是慢；只想快速刷统计用抽帧版算子。
- `overwrite=False` 的判重只看有没有 `q*` 键，不校验统计是否与当前数据一致；数据改过但 `q*` 键还在时不会重算，必须显式 `overwrite=True`。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.augment_quantile_stats_udf import (
    AugmentQuantileStats,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

**夹具可用 lerobot writer API 自建（见 test/cases/0.5.3/test_augment_quantile_stats.py）**
    samples = {
        "dataset_path": ["/tmp/lerobot_quantile_out"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            AugmentQuantileStats,
            construct_args={"overwrite": False},
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path")),
    )
    ds.show()
```

LeRobot Episode 删除

简介

从 LeRobot v3.0 数据集里删掉指定 episode，产出一份不含这些 episode 的新数据集，源数据集不动。包 lerobot `datasets/dataset_tools.py:81 delete_episodes`，外面加 crash-safe 与 skip-if-done 两层运维行为。

功能描述

- 非破坏：源数据集只读，结果写到 `output_path`；`output_path` 为空时缺省写到与源同级的 `<源目录名>_modified`
- 保留下来的 episode 按原顺序重索引成 0..N-1：`data/**/*.parquet` 的 `episode_index` / `index` / `task_index` 全部重编，`meta/episodes/**` 与 `meta/info.json`（`total_episodes` / `total_frames` / `total_tasks` / `splits`）同步重写
- 任务表按存活 episode 重建：删完之后没有 episode 引用的任务会从 `meta/tasks.parquet` 消失，`task_index` 相应重映射
- 视频按文件粒度处理：一个 mp4 里的 episode 全部保留时整文件拷贝（不重编码）；既有保留又有删除时用 PyAV 按帧区间重编码，并重排该文件内 episode 的 `from_timestamp` / `to_timestamp`
- `meta/stats.json` 由存活 episode 的逐 episode 统计重新聚合（lerobot `aggregate_stats`），不是照抄源 stats
- crash-safe：开跑前把已存在的输出目录整体删掉重建，中途失败再删一次半成品，避免下一次运行把半成品误判成完成态
- 完成后校验输出确实是合法 v3.0（`meta/info.json` + `codebase_version`），不是就报错
- skip-if-done：输出已是 v3.0 且 `force=False` 时不动手，返回 `SKIPPED`

- 参数校验沿用上游：episode_indices 为空、含越界索引、或把所有 episode 删光都直接失败（返回 Failed 状态行，不抛异常打断整批）

算子参数

输入

输入	含义
dataset_path	源 v3.0 数据集根目录（只读）
episode_indices	要删除的 episode 索引列表（list[int]，按源数据集的索引）
output_path	输出数据集根目录，可为空；为空时用 <源目录名>_modified

输出

输出	含义
result	SUCCESS / SKIPPED: output already exists / Failed: dataset_path and episode_indices are required / Failed: [<dataset_path>] <异常类型>: <信息>。输出数据集路径就是该行的 output_path

参数

参数名称	类型	默认值	描述
repo_id	str 或 None	None	写入新数据集元数据的逻辑数据集名。None 时由 lerobot 取 <源 repo_id>_modified（源 repo_id = 源目录名）
force	bool	False	True 表示输出已是 v3.0 也重新生成

注意事项

- output_path 指向的目录会被整体删除后重建（force=True 时连已完成的 v3.0 输出一起删），不要指向源数据集或还有其他内容的目录。
- 输出的 episode 索引是重排过的：删掉 ep1 之后原 ep2 变成新 ep1。下游若要引用原 episode 号，需要自己记映射。
- 视频文件沿用源的 chunk / file 索引命名，删除后输出里可能出现不连续的 file_index，属正常。
- 混合文件的重编码固定用 lerobot _copy_and_reindex_videos 的默认 libsvtav1 / yuv420p（本算子未暴露），这部分视频会多一次代际压缩损失；整文件保留的视频是字节拷贝，无损失。
- 单行只返回状态串（__return_column_type__ 是 large_string），需要输出路径请直接用输入列。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.delete_episodes_udf import DeleteEpisodes

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

**夹具可用 test/_helpers/lerobot_edit_fixture.py:build_tabular_dataset 生成**
    samples = {
        "dataset_path": ["/path/to/lerobot_dataset"],
        "episode_indices": [[1]],
        "output_path": ["/tmp/lerobot_deleted_out"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            DeleteEpisodes,
            construct_args={"repo_id": None, "force": False},
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path"), col("episode_indices"), col("output_path")),
    )
    ds.show()
```

LeRobot 数据集划分

简介

按 episode 把一份 LeRobot v3.0 数据集切成两份互不重叠的独立数据集（默认 train + val），源数据集不动。包 lerobot datasets/dataset_tools.py:144 split_dataset，外面加 crash-safe 与 skip-if-done。

功能描述

- 两种切法：行里给了 train_episode_indices 就按显式索引切（第二份拿剩下的全部 episode）；给 null 则按构造参数 train_fraction 切
- 比例模式的分配口径来自 lerobot _fractions_to_episode_indices：第一份取 int(总 episode 数 × fraction) 个，最后一份吃掉余数；episode 按索引顺序连续分配，不做 shuffle
- 每份 split 落在 <output_path>/<split_name>，是独立完整的 v3.0 数据集，episode 重索引成 0..N-1
- 重索引与元数据重写和 Episode 删除同一套逻辑：index / task_index 重编、任务表按该 split 里存活的任务重建、meta/stats.json 按该 split 的 episode 重新聚合、视频整文件保留时字节拷贝、混合文件用 PyAV 按帧区间重编码
- 两份 split 的逻辑数据集名由 lerobot 写成 <源 repo_id>_<split_name>
- crash-safe：开跑前把 output_path 整体删掉重建，中途失败再清一次半成品

- 完成后逐 split 校验都是合法 v3.0，缺任何一份直接报错
- skip-if-done：两份 split 都已是 v3.0 且 `force=False` 时返回 `SKIPPED`
- 校验：`train_fraction` 与 `train_episode_indices` 必须且只能生效一个（UDF 层保证行里给了索引就不传 fraction）；`train_fraction` 必须落在开区间 (0, 1)；索引越界、两份 split 有重叠、某份 split 为空都直接失败

算子参数

输入

输入	含义
dataset_path	源 v3.0 数据集根目录（只读）
output_path	两份 split 的父目录，split 落在 <output_path>/<split_name>
train_episode_indices	第一份 split 的显式 episode 索引列表（list[int]）；为 null 时改用构造参数 train_fraction

输出

输出	含义
result.status	SUCCESS / SKIPPED: output splits already exist / Failed: dataset_path and output_path are required / Failed: [<dataset_path>] <异常类型>: <信息>
result.splits	list，每份 split 一条：name = split 名，path = 该 split 的数据集根目录

参数

参数名称	类型	默认值	描述
train_fraction	float	0.8	第一份 split 占的 episode 比例（按索引顺序）；行里给了 train_episode_indices 时忽略
split_names	tuple[str, str] 或 list[str]	("train", "val")	两份 split 的名字，必须正好两个，否则构造算子时抛 ValueError
force	bool	False	True 表示两份 split 都已是 v3.0 也重新生成

注意事项

- `output_path` 是两份 split 的**父目录**，开跑前会被整体删除重建，不要指向源数据集或还有其他内容的目录。
- 比例是向下取整：4 个 episode + `train_fraction=0.8` → train 3 / val 1。episode 太少时第一份可能被算成 0 个，lerobot 会跳过这份 split，随后本算子的完整性校验报错（体现为`Failed`）。
- 划分是 episode 级、按索引顺序的连续切分，没有随机打乱；要随机划分请自己在行里给 `train_episode_indices`。
- 索引模式下第二份 split 拿剩余全部 episode；把所有 episode 都给了第一份会让第二份为空，上游直接报错。
- 每份 split 的 `meta/info.json` 里 `splits` 字段被 lerobot 统一写成 `{"train": "0:N"}`（N 是这份 split 自己的 episode 数），与 train/val 划分无关，不要拿它判断 split 身份。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.split_dataset_episodes_udf import (
    SplitDatasetEpisodes,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

**夹具可用 test/_helpers/lerobot_edit_fixture.py:build_tabular_dataset 生成**
    # train_episode_indices 给 [0, 2] 即切到索引模式，train_fraction 失效
    samples = {
        "dataset_path": ["/path/to/lerobot_dataset"],
        "output_path": ["/tmp/lerobot_split_out"],
        "train_episode_indices": [None],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            SplitDatasetEpisodes,
            construct_args={
                "train_fraction": 0.5,
                "split_names": ("train", "val"),
                "force": False,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path"), col("output_path"), col("train_episode_indices")),
    )
    ds.show()
```

LeRobot 任务描述修改

简介

改 LeRobot v3.0 数据集的任务文本（整体设一个默认任务，或按 episode 逐个覆盖），并把任务相关的四处元数据一起同步。包 lerobot `datasets/dataset_tools.py:1404 modify_tasks`（上游是原地改），默认先把数据集整目录拷到 `output_path` 再改拷贝，源保持不变。

功能描述

- 默认「拷贝后改」：`output_path` 非空时先 `shutil.copytree` 整份拷到目标再改目标，源数据集在 pipeline 里保持不可变；`output_path` 为空是文档化的原地改（超大数据集省一份拷贝），会打 warning
- 任务解析顺序沿用上游：某 episode 在 `episode_tasks` 里就用它，否则用 `new_task`，两者都没覆盖到就保留该 episode 原任务的第一条（原任务也为空则报错）
- 同步范围：`meta/tasks.parquet`、`data/**/*.parquet` 的 `task_index`、`meta/episodes/**/*.parquet` 的 `tasks`、`meta/info.json` 的 `total_tasks`
- 任务去重：所有 episode 的新任务取 `sorted(set(...))` 重新编号，文本相同的任务共用一个 `task_index`
- 任务内容来自构造参数而不是数据列（它是配置，不是每行数据）；`new_task` 与 `episode_tasks` 至少给一个，都不给直接失败
- `episode_tasks` 里的 episode 索引越界由上游校验拦下，返回 `Failed` 状态行
- skip-if-done：拷贝模式下输出已是 v3.0 且 `force=False` 时不动手，返回 `SKIPPED`；原地模式没有这个判断
- 不重算 `meta/stats.json`（任务改动不影响数值统计）

算子参数

输入

输入	含义
dataset_path	源 v3.0 数据集根目录
output_path	接收改动的拷贝目标根目录；为 null 表示原地改源数据集（慎用）

输出

输出	含义
result	SUCCESS / SKIPPED: output already exists / Failed: dataset_path is required / Failed: [<dataset_path>] <异常类型>: <信息>

参数

参数名称	类型	默认值	描述
new_task	str 或 None	None	所有 episode 的默认任务文本
episode_tasks	dict[int, str] 或 None	None	按 episode 覆盖，{episode_index: task}，优先级高于 new_task
force	bool	False	True 表示输出已是 v3.0 也重新拷贝并改写

注意事项

- `task_index` 会按新任务集合的字典序整体重编，与原编号没有对应关系；只改一个 episode 的任务也会让其他 episode 的 `task_index` 变。
- 每个 episode 的 `tasks` 被写成单元素列表，原本挂多个任务的 episode 会被压成一条。
- 拷贝模式会先删掉已存在的 `output_path` 再整目录拷贝，磁盘开销等于一份完整数据集（含视频），大数据集要预留空间。
- 改写阶段失败不会清理输出：拷出来的目录本身已经是合法 v3.0，下一次 `force=False` 会被 skip-if-done 当成已完成；重跑请显式 `force=True`。
- 原地模式（`output_path` 为 null）没有回滚，data parquet 是逐文件改写的，中途失败会留下部分改写的数据集。
- `episode_tasks` 的键是 episode 索引（int）。经 JSON 序列化传参时键会变成字符串，算子内部会 `int(k)` 还原。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.modify_tasks_udf import ModifyTasks

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

**夹具可用 test/_helpers/lerobot_edit_fixture.py:build_tabular_dataset 生成**
    samples = {
        "dataset_path": ["/path/to/lerobot_dataset"],
        "output_path": ["/tmp/lerobot_tasks_out"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ModifyTasks,
            construct_args={
                "new_task": "统一任务",
                "episode_tasks": {0: "特例"},
                "force": False,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path"), col("output_path")),
    )
    ds.show()
```

LeRobot 图片集转视频数据集

简介

把用 PNG 帧存图像观测的 LeRobot v3.0 数据集重新编码成分块 MP4 的视频数据集，产出一份新数据集，源不动。包 lerobot datasets/dataset_tools.py:1530 convert_image_to_video_dataset，编码走 lerobot datasets/video_utils.py encode_video_frames（PyAV / FFmpeg）。默认 libsvtav1 + crf 30，相比 PNG 帧有量级上的存储节省。

功能描述

- 只处理图像数据集：源数据集已经含任何 video key 时直接失败；被转换的特征是名字以 observation.image 开头的那些（observation.images.* 也命中），一个都没有则失败
- 特征表里图像特征的 dtype 从 image 改成 video，info 由编出来的首个 mp4 探测填入（lerobot get_video_info）；data/**/*.parquet 里的图像列被去掉，图像不再进 parquet
- episode 按估算体积成批：先用一小段样本试编估算每帧字节，再按数据集的 video_files_size_in_mb 上限把多个 episode 顺序编进同一个 mp4，逐 episode 记 videos/<key>/from_timestamp / to_timestamp
- 编码前把帧逐个落成临时 PNG 到 <output_path>/temp_images（num_workers 个线程写盘），每批编完即删，finally 里清掉整个临时目录
- 编码参数 vcodec / pix_fmt / g / crf / fast_decode 原样转给 lerobot encode_video_frames
- episode_indices 可只转指定 episode；meta/tasks.parquet 原样拷贝
- crash-safe：开跑前把已存在的输出目录整体删掉重建，中途失败再清一次半成品
- 完成后校验输出是合法 v3.0（meta/info.json + codebase_version）
- skip-if-done：输出已是 v3.0 且 force=False 时不动手，返回 SKIPPED

算子参数

输入

输入	含义
dataset_path	源 v3.0 图像数据集根目录（只读，不能含 video 特征）
output_path	输出视频数据集根目录

输出

输出	含义
result	SUCCESS / SKIPPED: output already exists / Failed: dataset_path and output_path are required / Failed: [<dataset_path>] <异常类型>: <信息>

参数

参数名称	类型	默认值	描述
vcodec	str	"libsvtav1"	编码器，取值受 lerobot resolve_vcodec 限制（见注意事项）
pix_fmt	str	"yuv420p"	像素格式
g	int	2	GOP 大小（关键帧间隔）
crf	int	30	码率控制参数，越大画质越低体积越小
fast_decode	int	0	解码优化档位，>0 时 libsvtav1 加 svtav1-params=fast-decode=N，其他编码器加 tune=fastdecode
episode_indices	list[int] 或 None	None	只转这些 episode。None 表示全部
num_workers	int	4	落临时 PNG 的线程数，取 max(1, num_workers)
force	bool	False	True 表示输出已是 v3.0 也重新转换

注意事项

- vcodec 合法取值由 lerobot resolve_vcodec 限定：h264 / hevc / libsvtav1 / auto，以及硬件编码器 h264_nvenc / hevc_nvenc / h264_vaapi / h264_qsv / h264_videotoolbox / hevc_videotoolbox。libx264**不在这个集合里，会直接抛**ValueError；没有 SVT-AV1 的机器请用 h264。auto 会去探测硬件编码器（有 NVIDIA 卡时会选 nvenc）。
- 输出的 meta/stats.json 是把源 stats 里的图像特征**剔除**后写入的，转换完视频特征没有统计。需要的话再跑一次「LeRobot 数据集统计量计算」补上。
- 一个 mp4 里顺序放了多个 episode，单独解某个 episode 必须按它的 videos/<key>/from_timestamp 偏移，从 0 秒读会拿到别的 episode 的帧。
- 只转部分 episode_indices 时输出保留原 episode_index（不重索引），total_episodes 只等于所选数量，索引可能不连续——这一点与 Episode 删除 / 数据集划分的重索引行为不同。
- 临时 PNG 落在输出目录下的 temp_images，转换过程中需要额外临时磁盘空间（按一批 episode 的原始帧算）。
- crf 在不同编码器上映射到不同参数（软编码 crf、nvenc qp、videotoolbox q:v）；libsvtav1 的 preset 被 lerobot 固定成 12，本算子未暴露。pix_fmt=yuv444p 与 libsvtav1 / hevc 不兼容，会被上游自动改回 yuv420p。
- 输出数据集的逻辑数据集名由 lerobot 固定成 <源目录名>_video，本算子未暴露 repo_id。
- output_path 目录会被整体删除重建，不要指向源数据集或还有其他内容的目录。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.convert_images_to_video_udf import (
    ConvertImagesToVideoDataset,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

**夹具可用 test/_helpers/lerobot_edit_fixture.py:build_image_dataset 生成**
    samples = {
        "dataset_path": ["/path/to/lerobot_image_dataset"],
        "output_path": ["/tmp/lerobot_video_out"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ConvertImagesToVideoDataset,
            construct_args={
                "vcodec": "libsvtav1",
                "pix_fmt": "yuv420p",
                "g": 2,
                "crf": 30,
                "fast_decode": 0,
                "episode_indices": None,
                "num_workers": 4,
                "force": False,
            },
            num_cpus=2,
            concurrency=1,
            batch_size=1,
        )(col("dataset_path"), col("output_path")),
    )
    ds.show()
```

Mcap格式转换处理器

简介

MCAP 格式转换算子，把 protobuf 编码的 MCAP episode（ABC-130k / YAM 双臂工作站布局，一个 episode 一个 episode.mcap）转换成 LeRobot v3.0 数据集。不需要安装 ROS，也不涉及 CDR 编码。

源数据话题：

话题	schema	说明
/instruction	Instructions	任务文本，单条
/{side}-arm-{state,action}	RobotState	每侧 6 个关节角（弧度），-state 另带 velocity/torque/pose
/{side}-ee-{state,action}	GripperState	夹爪归一化开合度（0 关 1 开）
/top-camera	foxglove.CompressedVideo	单目站（RealSense）顶部相机
/top-{left,right}-camera	foxglove.CompressedVideo	立体站（ZED-X）顶部双目
/{side}-wrist-camera	foxglove.CompressedVideo	腕部相机
/<视频话题>-info	foxglove.CameraCalibration	分辨率与内参

功能描述

- 自动探测站型与编码：单目 / 立体、每路相机逐个读 CompressedVideo.format 选 H.264 或 H.265，分辨率取自 CameraCalibration（探测不到才解码首帧）
- observation.state / action 按 left, right 顺序拼接每侧 [关节角, 夹爪]，只取 position 字段（与上游训练口径一致）
- 严格校验每侧 arm 为 6 维、ee 为 1 维，并在写出前校验所有 episode 的 topic/schema/维度/编码/分辨率一致；不补零掩盖缺失流
- **固定时钟因果重采样**：源流采样率各不相同（关节/夹爪 200~290Hz，相机约 30Hz，条数不等），行网格取 target_fps 固定周期，每条流取「tick 时刻或之前最近的一条采样」（causal ZOH）

- `CompressedVideo` 消息数与流里实际帧数（demux 出的 access unit 数）不一致时，按首尾消息时间戳线性重铺真实帧时间，避免把一条消息误当成一帧
- 任务名优先取 `/instruction`，缺失时回落到任务目录名，再回落到 `default_task`
- 视频逐帧流式解码，任意时刻内存里只有一帧：立体 episode 单路 3300 帧 1920×1200 全解开是 23GB RGB
- episode 级多进程分片，分片结果用 lerobot `aggregate_datasets` 合并
- 写出后逐路核对输出 mp4 的帧数与 parquet 行数，不一致直接失败并删掉半成品目录：lerobot 的流式编码器在自己队列打满 100ms 时会静默丢帧（实测 1920×1200 立体 episode 三路里两路各丢 5~6 帧），行还照样写进 parquet，之后每一行都会对上错的图。本算子已把入队改成阻塞等待，核对是第二道闸
- 只认 `episode.mcap`；已标注 episode 目录里的 `annotation.mcap`（子任务标签）会被跳过，不会被当成 episode
- 输出已是 v3.0 且 `force_conversion=False` 时跳过，返回 `SKIPPED`

算子参数

输入

输入	含义
input_path	源数据集目录，在这个目录下递归查找 episode.mcap
output_path	输出数据集目录，v3.0 数据集直接写到这里

输出

输出	含义
result	SUCCESS / SKIPPED: has already been v3.0 / Failed: [repo] <异常类型>: <信息>

参数

参数名称	类型	默认值	描述
num_workers	int	4	episode 级并发进程数，<=1 时串行
vcodec	str	"h264"	输出视频编码。auto 会被映射成 h264，避开 lerobot resolve_vcodec 误选 nvenc
force_conversion	bool	False	True 表示输出已是 v3.0 也重新转换
default_task	str	"default_task"	/instruction 和任务目录名都拿不到时的兜底任务名
robot_type	str	"unknown"	写入 v3 元数据的机器人标识
target_fps	int	30	输出数据集行频率，所有源流按此固定时钟重采样
top_camera_policy	str	"left"	立体站两路顶部相机的取法：left/right 取一只眼作为 observation.images.top；both 输出 top_left + top_right 两个特征；official 按 sha1(episode 目录名) 每个 episode 挑一只眼，复现上游训练集。单目站忽略此参数
resize	tuple[int,int] 或 None	None	所有相机统一缩放到 (height, width)。None 保持源分辨率——立体站是 1920×1200，全量转换建议开缩放
decode_threads	int	8	每路相机解码器的 FFmpeg 线程数。PyAV 默认单线程，在 1920×1200 H.265 上慢约 5 倍；不要设 0（会按核数开线程，与 episode 级并发叠加后线程过订阅）
repo_id	str 或 None	None	写入 v3 元数据的逻辑数据集名。None 取 input_path 的目录名

注意事项

- 立体站（ZED-X）四路相机都是 H.265、1920×1200；单目站（RealSense）是 H.264。编码按路判断，不要按站型假设。
- 立体 episode 如果同时存在 `/top-camera`，那是把左右眼缩略图烧进同一画面的低分辨率预览图，本算子在两只眼都存在时直接丢弃它。
- 源流不是等长的，不能按消息下标配对：实测某立体 episode 的 `/left-arm-state` 有 31954 条、`/top-left-camera` 只有 3332 条。

调用示例

```python
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_mcap_to_lerobot_v30_udf import (
    ConvertMcapToLeRobotV30,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "input_paths": ["/path/to/mcap_dataset/<task_name>"],
        "output_paths": ["/tmp/mcap_lerobot_out/<task_name>"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ConvertMcapToLeRobotV30,
            construct_args={
                "num_workers": 4,
                "vcodec": "h264",
                "force_conversion": True,
                "robot_type": "yam_bimanual",
                "target_fps": 30,
                "top_camera_policy": "left",
                "resize": None,
                "decode_threads": 8,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("input_paths"), col("output_paths")),
    )
    ds.show()
```

Rrd格式转换处理器

简介

Rerun .rrd 格式转换算子，把 VLA 数采导出的 rrd recording（一个 episode 一个 .rrd）转换成 LeRobot v3.0 数据集。通过 rerun.dataframe 直接读文件，不起 viewer、不走 gRPC。

源数据实体：

实体路径	archetype	说明
/metadatas/metadata.json	TextLog（静态）	task_prompt / fps / total_frames / state_space / action_space / videos[].key
/metadatas/stats.json	TextLog（静态）	采集质量统计（同步成功/跳过计数等），不参与转换
/task_info	TextLog（静态）	采集任务描述，不参与转换
/vla/timeseries/observation.state/<维度名>	Scalars	每个状态维度一个实体
/vla/timeseries/action/<维度名>	Scalars	每个动作维度一个实体
/vla/timeseries/{frame_index,collection_time}	Scalars	不转换（v3.0 自己会生成 frame_index）
/vla/camera/<相机名>	VideoStream	静态 codec + 每帧一条 sample（Annex-B）

功能描述

- 特征集自动探测：相机来自 VideoStream 实体，编码读静态 VideoStream:codec（fourcc avc1→H.264、hev1→H.265）逐路判断，分辨率解首帧拿到
- 相机特征名取 metadata.json 的 videos[].key（按实体叶子名匹配），即 observation.images.head_color / left_arm_color / right_arm_color；metadata.json 缺失或对不上时回落到 observation.images.<实体叶子名>
- observation.state / action 按 metadata.json 的 state_space / action_space **声明顺序**拼接（实测 29 维：torso 3 + 双臂各 6 + 双手各 6 + head 2）。实体扫描是按字母序的，与采集端的关节顺序不同，所以只有 metadata 缺失时才回落到字母序
- 写出前逐 episode 校验 state/action 维度顺序、相机实体、编码、分辨率和 fps；schema 漂移直接失败，不把错误值写进同一数据集
- **行与帧按下标配对**：采集端的同步器每个 tick 产出一行标量 + 每路相机一条 sample，健康 episode 各流条数相等，这个配对就是权威口径。两条时间线自己会相对漂移（实测 ±16.6ms，接近半个帧周期），按时间戳重新配对会把约 60% 的行挪到相邻帧上。只有条数不一致的残缺 episode 才回落到因果 ZOH（取「tick 时刻或之前最近的一条」），并打 warning
- VideoStream sample 条数与流里实际帧数（demux 出的 access unit 数）不一致时，按首尾 sample 时间戳线性重铺真实帧时间；残缺流在首个 tick 前没有历史样本时直接失败，禁止使用未来首帧
- 任务名优先取 metadata.json 的 task_prompt，再回落 task_description、上级目录名、default_task
- 视频样本先落盘成 Annex-B 流，再逐帧拉取，任意时刻内存里只有一帧（1240 帧 1280×720 全解开是 3.4GB RGB）
- 标量列中途有空洞时按前值保持；开头就缺样本的行（还没有任何因果值）整行裁掉，不补 0 伪造观测。裁行只改输出起点，帧配对仍按原始时间线的下标算，不会把相机重置回第 0 帧
- episode 级多进程分片，分片结果用 lerobot aggregate_datasets 合并
- 写出后逐路核对输出 mp4 的帧数与 parquet 行数，不一致直接失败并删掉半成品目录：lerobot 的流式编码器在自己队列打满 100ms 时会静默丢帧（实测 1920×1200 立体 episode 三路里两路各丢 5~6 帧），行还照样写进 parquet，之后每一行都会对上错的图。本算子已把入队改成阻塞等待，核对是第二道闸
- 输出已是 v3.0 且 force_conversion=False 时跳过，返回 SKIPPED

算子参数

输入

输入	含义
input_path	源数据集目录，在这个目录下递归查找 *.rrd
output_path	输出数据集目录，v3.0 数据集直接写到这里

输出

输出	含义
result	SUCCESS / SKIPPED: has already been v3.0 / Failed: [repo] <异常类型>: <信息>

参数

参数名称	类型	默认值	描述
num_workers	int	4	episode 级并发进程数，<=1 时串行
vcodec	str	"h264"	输出视频编码。auto 会被映射成 h264，避开 lerobot resolve_vcodec 误选 nvenc
force_conversion	bool	False	True 表示输出已是 v3.0 也重新转换
default_task	str	"default_task"	task_prompt 和上级目录名都拿不到时的兜底任务名
robot_type	str	"unknown"	写入 v3 元数据的机器人标识
fps	int 或 None	None	写入 v3 元数据的行频率。None 取 metadata.json 的 fps（采集端目标帧率）
resize	tuple[int,int] 或 None	None	所有相机统一缩放到 (height, width)。None 保持源分辨率（头部相机 1280×720，腕部 640×480）
decode_threads	int	8	每路相机解码器的 FFmpeg 线程数。PyAV 默认单线程，720p H.264 上慢数倍；不要设 0（会按核数开线程，与 episode 级并发叠加后线程过订阅）
repo_id	str 或 None	None	写入 v3 元数据的逻辑数据集名。None 取 input_path 的目录名

注意事项

- 只支持 VideoStream 里的 H.264 / H.265，其他编码直接抛 UnsupportedRrdSchemaError，不会交给 demuxer 静默产出坏图。
- 输入 rrd 由哪个 rerun 版本写的不重要，但读的版本不能太旧：rerun 0.26.2 读 0.27.2 写的文件会打前向兼容 warning（实测能读，结果与 0.27.3 一致）。
- metadata.json 的 total_frames 与实际标量行数不一致时只打 warning，按实际行数转换。
- 相机条数与标量行数不一致时会退到因果 ZOH，这时输出会重复或丢帧，日志里有明确 warning，值得回头查采集侧。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_rrd_to_lerobot_v30_udf import (
    ConvertRrdToLeRobotV30,
)

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "input_paths": ["/path/to/rrd_dataset/<task_name>"],
        "output_paths": ["/tmp/rrd_lerobot_out/<task_name>"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            ConvertRrdToLeRobotV30,
            construct_args={
                "num_workers": 4,
                "vcodec": "h264",
                "force_conversion": True,
                "robot_type": "EHR02",
                "fps": None,
                "resize": None,
                "decode_threads": 8,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("input_paths"), col("output_paths")),
    )
    ds.show()
```

SARM 子任务标注

简介

闭集子任务标注算子，用本地 Qwen3-VL 对一段机器人操作视频做时序阶段划分，输出每个子任务的起止秒。移植 lerobot data_processing/sarm_annotations/subtask_annotation.py 的 VideoAnnotator（提示词来自同文件的 create_sarm_prompt），面向 SARM reward model 的训练数据标注。

与 AtomicActionSegment 的分工：本算子做语义 subtask 阶段划分（VLM 驱动），后者做运动学动作链切分（无模型启发式）。

功能描述

- 闭集词表由构造参数 subtask_list 给出并写进提示词，模型只能使用这些标签；提示词约束为：从 00:00 起无缝覆盖全片、相邻段 end 等于下一段 start、每个标签按时序恰出现一次、先写文字时间线再输出 JSON
- 推理前用 ffmpeg 把 [start_time_sec, end_time_sec] 区间重编码成 **1 fps** 的临时 mp4（libx264 / ultrafast / crf 23 / 去音轨），推理后删除临时文件；end_time_sec 为空时用 cv2 读帧数与帧率算出全片时长
- 生成参数沿用上游：max_new_tokens=1024、do_sample=True、temperature=0.7；回复先剥掉 markdown 代码块围栏，再 json.loads + pydantic 校验（解析不出来时用正则兜一次最外层花括号），失败按 max_retries 重试，仍失败抛 RuntimeError
- 模型类按 checkpoint 的 config 解析（AutoModelForImageTextToText，老版本 transformers 回落 AutoModelForVision2Seq）后注入 VideoAnnotator：上游写死了 Qwen3VLMoeForConditionalGeneration，换非 MoE 权重会在 from_pretrained 里对不上 config，所以这里自己加载再注入，model_name 可以指向同族任意尺寸的权重
- 权重两段式解析 model_path + model_name，构造算子时先检查 <model_path>/<model_name>/config.json 存在，缺权重直接 FileNotFoundError，不自动下载、不占 GPU
- 模型在首次标注时才加载（每个 worker 一份）；设备由 UDF 的 num_gpus 决定：>0 走 cuda，=0 走 cpu
- 模型输出的 MM:SS 时间戳按整数秒解析，duration_sec = end_sec - start_sec
- 行级容错：video_path 为空、视频不存在或标注失败都返回 Failed: ...，不打断整批
- 数据集级子任务进度先验 compute_temporal_proportions（SARM 论文公式 (1)）留在库层 daft.aihc.functions.embodied.lerobot.sarm_subtask_annotate，它要聚合多个 episode，不是逐行算子

算子参数

输入

输入	含义
video_path	episode 视频路径（mp4）
start_time_sec	标注起始秒，null 按 0 处理
end_time_sec	标注结束秒，null 表示标到视频结尾

输出

输出	含义
result.status	SUCCESS / Failed: video_path is required / Failed: [<video_path>] <异常类型>: <信息>
result.subtasks	list，按时序排列的子任务段，字段 name（string，闭集标签）、start_sec、end_sec、duration_sec（均为 float64，秒）
result.raw_json	归一化后 subtasks 列表的 JSON 文本（json.dumps(..., ensure_ascii=False)），失败行为空串

参数

参数名称	类型	默认值	描述
subtask_list	list[str]	无（必填）	闭集子任务名列表，模型只能使用这些标签；为空报错
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"Qwen/Qwen3-VL-30B-A3B-Instruct"	权重名，与 model_path 拼成 <model_path>/<model_name>；可换同族其它尺寸
dtype	str	"bfloat16"	模型精度，取值 bfloat16 / float16 / float32
max_retries	int	3	JSON 解析或推理失败的重试次数，下限 1

注意事项

- 默认权重 Qwen3-VL-30B-A3B-Instruct 的 bf16 权重就有 58 GiB，24G 卡装不下。测试里按显存挑：≥64 GiB 用 30B MoE，≥20 GiB 用Qwen/Qwen2.5-VL-7B-Instruct，更小用 Qwen/Qwen2.5-VL-3B-Instruct；两条路径走同一份算子代码。
- 权重不会自动下载，缺失时算子构造阶段直接抛 FileNotFoundError。
- 上游是采样解码（do_sample=True、temperature=0.7），同一段视频两次标注结果可能不同；边界是整数秒精度，亚秒级切分拿不到。
- 需要 ffmpeg 在 PATH 里，否则抽段阶段报 ffmpeg not found；抽段会往系统临时目录写一份 1 fps 的 mp4。
- result.raw_json 是解析并归一化之后的 subtasks，不是模型的原始回复文本。
- 库层 compute_temporal_proportions 沿用 lerobot 语义：单个 episode 内同名子任务的时长是**覆盖**而不是累加（同名只保留最后一段），最后按各名次均值归一化到和为 1。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.lerobot.sarm_subtask_annotate_udf import SarmSubtaskAnnotate

SUBTASK_LIST = [
    "approach_object",
    "grasp_object",
    "lift_object",
    "move_object",
    "place_object",
    "release_object",
    "retreat",
]

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=3600, min_cpu_per_task=0)

samples = {
        "video_path": ["bos://your-bucket/sample.mp4"],
        "start_time_sec": [0.0],
        "end_time_sec": [40.0],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            SarmSubtaskAnnotate,
            construct_args={
                "subtask_list": SUBTASK_LIST,
                "model_path": "/path/to/models",
                "model_name": "Qwen/Qwen2.5-VL-7B-Instruct",
                "dtype": "bfloat16",
                "max_retries": 3,
            },
            num_cpus=4,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video_path"), col("start_time_sec"), col("end_time_sec")),
    )
    ds.show()
```

文档

Excel 表格解析处理器

简介

Excel 表格解析处理器

功能描述

- 支持 xlsx/xls 格式解析
- 输出 markdown 或 html 格式
- 保留表格结构与数据关系
- 支持多工作表处理

算子参数

输入

输入	含义
xlsx_col	包含 xlsx/xls 文件路径的列

输出

输出	含义
data_item_uri	原始文件路径
text	合并后的 markdown/html 文本
text_by_table	每个 sheet 的 markdown/html 文本列表

参数

参数名称	类型	默认值	描述
if_save_md_content	bool	True	是否保存为 markdown 格式，默认 True
if_save_html_content	bool	False	是否保存为 html 格式，默认 False 当两者都为 True 时，仅保存 markdown
output_path	str		处理后的文档输出位置

调用示例

```
 from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.doc.xlsx_parse import XlsxParse

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "xlsx_path": [
            "file:///local/test_doc_01.xlsx",
            "file:///mnt/pfs/test_doc_02.xlsx",
            "file:///mnt/bos/test_doc_02.xlsx",
        ],
    }
    output_path = "file:///local/test_doc_output/",
    df = daft.from_pydict(samples)

    constructor_kwargs = {
        "if_save_md_content": True,
        "if_save_html_content": True,
        "output_path": output_path,
    }

    df = df.with_column(
        "result",
        aihc_udf(XlsxParse, construct_args=constructor_kwargs, concurrency=1)(col("xlsx_path")),
    )
    df = df.with_column("data_item_uri", col("result")["data_item_uri"])
    df = df.with_column("text", col("result")["text"])
    df = df.with_column("text_by_table", col("result")["text_by_table"])
    df.show()
```

Markdown转DOCX

简介 Markdown 转 DOCX 转换器 - 将 Markdown 文档转换为 Word 文档

功能描述

- Markdown 转 DOCX：使用 Pandoc 将 Markdown 转换为 DOCX 格式
- 支持图片处理：处理 Markdown 中的图片引用
- 支持网络图片：自动下载网络图片并嵌入文档
- 本地和云端路径：支持本地和 BOS/S3 存储
- 标准 Markdown 语法：标题、列表、代码块、引用等
- 复杂表格：通过 HTML 中间格式支持复杂的 HTML 表格
- LaTeX 公式：支持将 LaTeX 公式转换为 MathML 格式
- 图片支持：本地图片和网络图片
- 行内格式：粗体、斜体、代码、删除线等
- 链接和引用：超链接和引用块

算子参数 输入 |输入|含义| |-|-| |markdown_files|输入的 Markdown 文件路径，支持 bos 与 https 路径|

输出 |输出|含义| |-|-| |result|输出的 DOCX 文件路径，仅支持 bos 路径|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_dir|str|必填|输出目录路径，可以是本地路径或 BOS/S3 路径|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.doc.markdown2docx import Markdown2Docx

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"markdown_files": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            Markdown2Docx,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("markdown_files")),
    )
    ds.show()
```

PPT转PDF或图片

简介 PPT 转换处理器，支持 PPT/PPTX 转 PDF 及每页图片

功能描述

- 支持将 PPT/PPTX 文档转换为 PDF 文件
- 支持将 PPT/PPTX 转换为按页拆分的 PNG 图片
- 自动处理本地路径与远程路径（BOS/HTTP 等）的下载与上传
- 提供转换超时控制与基础失败兜底
- 批量将演示文稿转换为用于预览或归档的 PDF
- 将演示文稿转换为逐页图片用于在线展示或内容审核
- 依赖本机已安装 LibreOffice（soffice 命令）
- 若未安装 pdf2image，将自动回退到系统的 pdftoppm 工具进行图片转换
- 当前支持生成 PNG/JPG 图片格式

算子参数 输入 |输入|含义| |-|-| |files|输入文件路径数组，支持 bos、http(s) 以及本地路径。|

输出 |输出|含义| |-|-| |result|输出处理后的路径字符串数组，与输入一一对应；单条失败时对应位置返回空字符串。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |target_format|str|必填|目标格式，"pdf" 表示仅生成 PDF，"images" 表示生成每页图片并返回图片文件夹路径。| |output_dir|str|必填|输出路径，必须提供，可以是本地路径或 BOS 路径。| |convert_time_out|int|600|转换超时时间（秒），用于控制 soffice / pdftoppm 执行时间。| |image_dpi|int|200|图片转换时的 DPI 分辨率，仅在 target_format="images" 时生效。| |image_format|str|'png'|图片格式，当前支持 "png" 、"jpg"、"jpeg"，仅在 target_format="images" 时生效。|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.doc.ppt_convert import PPTConvert

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"files": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "convert_time_out": 600,
        "image_dpi": 200,
        "image_format": 'png',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            PPTConvert,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("files")),
    )
    ds.show()
```

文档格式转换

简介 文档格式转换处理器，支持多种办公文档格式互转

功能描述

- 支持 doc/docx 到多种格式的转换
- 使用 LibreOffice 进行高质量转换
- 支持本地和云端存储路径
- 提供转换超时控制
- 自动处理文件上传下载
- 输入格式：Microsoft Word (.doc, .docx)
- 输出格式：PDF (.pdf), ODT (.odt), HTML (.html), 纯文本 (.txt), DOCX (.docx)
- 建议使用 docx 格式以获得最佳转换效果

算子参数 输入 |输入|含义| |-|-| |files|-|

输出 |输出|含义| |-|-| |result|输出文件路径，仅支持 bos 路径|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |target_format|str|必填|目标格式，支持 pdf/odt/html/txt/docx| |output_dir|str|必填|输出路径，必须提供，可以是本地路径或 BOS/S3 路径| |convert_time_out|int|60|转换超时时间，单位为秒|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.doc.doc_convert import DocConvert

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"files": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "convert_time_out": 60,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            DocConvert,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("files")),
    )
    ds.show()
```

其他

Tar文件解压

简介

解压tar文件算子

功能描述

tar文件解压, 支持批量处理

输入

输入	含义	默认值
base_path	支持传入base_path目录, 该目录下有多个子目录, 每个子目录有若干tar文件, 配合discover_datasets和create_tasks_from_datasets方法可便捷的将多个子目录数据集创建为多个task, 并发执行	无

输出

输入	含义	
output_path	解压后目录输出	
result	数据集转换状态:SUCCESS: 解压成功FAILED: 解压失败	

调用示例

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.process.tar_extractor_udf import TarUncompress
from daft.aihc.functions.process.tar_extractor_udf import discover_datasets
from daft.aihc.functions.process.tar_extractor_udf import create_tasks_from_datasets

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    base_path = "/path/dataset/lerobot/"
    datasets = discover_datasets(base_path, None, None)
    tasks = create_tasks_from_datasets(datasets)
    num_datasets = len(datasets)

    ds = daft.from_pydict(tasks)
    ds = ds.into_partitions(num_datasets)

    ds = ds.with_column(
        "result",
        aihc_udf(
            TarUncompress,
            construct_args={
            },
            num_cpus=1,
            num_gpus=0,
            batch_size=1,
            concurrency=num_datasets,
            use_process=True,
        )(col("input_path"), col("output_path")),
    )
    ds.show()
```

时间戳片段合并

时间戳片段合并

简介

时间戳片段合并算子，对秒级时间区间列表进行规范化、小间隙合并和最大时长约束切分，适用于语音识别（VAD/ASR）后的片段整理场景。

功能描述

- 对输入的时间区间列表进行格式规范化（支持多种嵌套格式）
- 可选：先合并相邻区间之间的小间隙（pre_merge_gap_seconds）
- 使用"最大间隙优先"算法，将超长连续区间按最大间隙切分，保证每段不超过 max_span_seconds
- 可选：对仍超过 max_span_seconds 的区间强制等分（enforce_chunking=True）
- 支持设置全局起始时间偏移（start_time）
- 输入为 None 或空时返回空列表，不抛出异常

算子参数

输入

输入	含义
timestamps	时间区间列表数组，每行为 [[start1, end1], [start2, end2], ...] 格式（秒），支持多种嵌套格式

输出

输出	含义
merged_timestamps	合并后的时间区间列表（list[list[float32]]），每个元素为 [start, end]

参数

参数名称	类型	默认值	描述
start_time	float	0.0	全局时间偏移量（秒），所有区间起点不小于此值
pre_merge_gap_seconds	float	0.0	预合并间隙阈值（秒），相邻区间间距小于此值时合并为一段
max_span_seconds	float	20.0	单个输出区间的最大时长（秒），超过此值时按最大间隙优先算法切分
enforce_chunking	bool	False	True 表示对仍超过 max_span_seconds 的区间强制等分切块

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.other.timestamps_merge import TimestampsMerge

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        # 每行为一个音频文件对应的时间戳区间列表（如 VAD 结果）
        "vad_timestamps": [
            [[0.5, 3.2], [3.5, 8.1], [9.0, 15.6], [16.0, 18.0], [18.5, 35.0]],
            [[0.0, 5.0], [5.5, 25.0], [26.0, 30.0]],
        ]
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "pre_merge_gap_seconds": 0.5,   # 间距 < 0.5s 的区间合并
        "max_span_seconds": 20.0,       # 单段最长 20s
        "enforce_chunking": False,
    }
    ds = ds.with_column(
        "merged_timestamps",
        aihc_udf(
            TimestampsMerge,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=64,
        )(col("vad_timestamps")),
    )
    ds.show()
```

文件下载(二进制)

简介 LoadFileBytes

算子参数 输入 |输入|含义| |-|-| |file_urls|文件url|

输出 |输出|含义| |-|-| |result|算子输出|

参数 无

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.file.download import LoadFileBytes

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"file_urls": [...]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            LoadFileBytes,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("file_urls")),
    )
    ds.show()
```

文件下载(Base64)

简介 LoadFileBase64

算子参数 输入 |输入|含义| |-|-| |file_urls|文件urls|

输出 |输出|含义| |-|-| |result|算子输出|

参数 无

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.file.download import LoadFileBase64

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"file_urls": [...]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            LoadFileBase64,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("file_urls")),
    )
    ds.show()
```

指定字段筛选

简介 指定字段筛选算子（SpecifiedFieldSelector），按指定字段做 Top-K / 区间 / 频次的数据集级选择。它是数据集级算子：transform 接收目标字段整列，返回布尔保留掩码，由上层 Daft .where() 完成过滤。纯 CPU。

功能描述 • 支持三种选择模式（由 mode 指定）：topk 取前 K、range 值区间筛选、frequency 频次筛选。

• topk 模式支持按条数 topk 或按比例 top_ratio 取值（两者同给取较小者），并可通过 reverse 控制升/降序；None 值按方向填充 ±inf。

• range 模式按闭区间 [lower, upper] 保留字段值落在区间内的行；None 值不入选。

• frequency 模式统计各取值出现次数，按 [freq_lower, freq_upper] 频次区间保留；输出为布尔列，交由 .where() 完成最终过滤。

算子参数 输入 |输入|含义| |-|-| |field|用于排序/筛选的字段整列（数值或可哈希值），算子据此计算保留掩码|

输出 |输出|含义| |-|-| |result|bool 类型的保留掩码列，True 表示该行入选；配合上层 Daft .where(col(...)) 过滤数据集|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |mode|str| "topk"|选择模式，可选值 topk / range / frequency，非法值抛 ValueError|| |topk|int |None|None|mode=topk 时保留的条数；None 表示不按条数限制| |top_ratio|float |None|None|mode=topk 时保留比例（0~1）；与 topk 同时给出时取较小的保留数| |reverse|bool|True|mode=topk 是否降序，True 取字段值最大的 K 个，False 取最小的 K 个|| |lower|float |None|None|mode=range 时字段值的下界（闭区间）；None 表示不设下界| |upper|float |None|None|mode=range 时字段值的上界（闭区间）；None 表示不设上界| |freq_lower|float |None|None|mode=frequency 时取值出现次数的下界；None 表示不设下界| |freq_upper|float |None|None|mode=frequency 时取值出现次数的上界；None 表示不设上界|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.other.specified_field_selector import SpecifiedFieldSelector

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # SpecifiedFieldSelector 无路径入参，用小 python 数据即可
    scores = [0.9, 0.2, 0.7, 0.5, 0.95]
    ds = daft.from_pydict({"score": scores})
    ds = ds.with_column(
        "keep",
        aihc_udf(
            SpecifiedFieldSelector,
            construct_args={"mode": "topk", "topk": 2, "reverse": True},
            num_cpus=1,
            concurrency=1,
            batch_size=len(scores),
        )(col("score")),
    ).where(col("keep"))  # 数据集级选择：仅保留 Top-2
    ds.show()
```

Tar文件递归解压

简介 Tar 包递归解压算子（RecursiveTarUncompress），基于 stdlib tarfile 递归解压可能嵌套的 tar 归档，是异构数据入湖链路的第一步。带 tar-slip（路径穿越）防护，无第三方依赖，离线 CPU 运行。

功能描述 • 将顶层 tar 归档解压到指定输出目录 output_path；若归档内还含 tar，则在原地继续解压，最多递归 max_depth 层。

• 嵌套 tar 解压完成后原地删除并去重，保证内容都落在输出目录根下，避免多余子目录。

• 内置路径穿越防护：解压前校验每个成员路径必须位于目标目录内，越界成员直接拦截报错。

• 逐行处理并做异常隔离：成功返回 "SUCCESS:<output_path>"，失败返回 "Failed: <错误信息>"；内部维护提交/成功/失败计数。

算子参数 输入 |输入|含义| |-|-| |input_col|待解压的 tar 归档文件路径（.tar / .tar.gz / .tgz）| |output_col|解压目标目录路径（不存在时自动创建）|

输出 |输出|含义| |-|-| |result| large_string 类型；成功为 "SUCCESS:<output_path>"，失败为 "Failed: <错误信息>"|

参数 |参数名称|类型|默认值|描述| |-|-|-| |max_depth|int| 3|递归解压的最大层数，用于限制嵌套 tar 的展开深度（会被转为 int）|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.process.recursive_tar_extractor_udf import RecursiveTarUncompress

INPUT_PATH = "/mnt/pfs/xxx/outer.tar"  # MOCK
OUTPUT_PATH = "/mnt/pfs/xxx/out"       # MOCK

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    ds = daft.from_pydict({"src": [INPUT_PATH], "dst": [OUTPUT_PATH]})
    ds = ds.with_column(
        "result",
        aihc_udf(
            RecursiveTarUncompress,
            construct_args={"max_depth": 3},
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("src"), col("dst")),
    )
    ds.show()
```

多模态

图文embedding-CLIP模型

简介 ClipEmbedding 是基于 Daft + Ray 分布式框架的多模态嵌入向量提取组件，利用 CLIP 模型同时支持文本和图像的 embedding 提取，适用于跨模态检索、图文匹配、内容相似度计算等场景。

功能

- 支持文本/图像嵌入向量提取，多模态统一编码
- 支持中文 CLIP 模型
- 支持跨模态相似度计算

> 该算子运行需要依赖对应的模型权重文件，平台已将所需权重文件预置在公共数据集aihc_daft_public_model中，需要在创建开发负载时挂载该数据集即可正常使用。

输入 |输入列名|说明| |-|-| |content|包含输入数据的数组，支持以下元素类型：文本模式: UTF-8字符串图像模式: Base64字符串/二进制数据/图像URL|

参数 |参数名称|类型|默认值|描述| |-|-|-| |content_type|str|image_url|输入图像的格式类型，支持：文本(text)bos/http 地址(image_url)base64 编码(image_base64)二进制流(image_binary)可选值：["text", "image_url", "image_base64", "image_binary"]默认值："image_url"| |batch_size|int|16|批量计算数据量，默认：16| |model_path|str|/opt/aihc/models|模型存储路径，默认：'/opt/aihc/models'（内部参数）| |model_name|str|iic/multi-modal_clip-vit-base-patch16_zh|模型名称，可选：'iic/multi-modal_clip-vit-base-patch16_zh''iic/multi-modal_clip-vit-huge-patch14_zh''iic/multi-modal_clip-vit-large-patch14_zh''iic/multi-modal_clip-vit-large-patch14_336_zh'默认：'iic/multi-modal_clip-vit-base-patch16_zh'| |model_version|str|v1.0.1|模型版本，当前仅支持'v1.0.1'| |rank|int|0|指定GPU设备编号（多卡环境有效），默认：0（内部参数）|

输出 包含浮点数嵌入向量的数组，每个元素为List[float]。

使用示例

```
 from __future__ import annotations

import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.embedding.clip_embedding import ClipEmbedding

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    model_path = os.getenv("MODEL_PATH", "/opt/aihc/models")
    model_name = "iic/multi-modal_clip-vit-base-patch16_zh"
    model_version = "v1.0.1"
    batch_size = 2
    rank = 0
    num_gpus = 1

    samples = {"text": ["树苗", "蜻蜓", "鸽子", "雪人", None]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "embedding",
        aihc_udf(
            ClipEmbedding,
            construct_args={
                "content_type": "text",
                "model_path": model_path,
                "model_name": model_name,
                "model_version": model_version,
                "batch_size": batch_size,
                "rank": rank,
            },
            num_gpus=num_gpus,
            batch_size=1,
            concurrency=1,
        )(col("text")),
    )

    ds.show()
** ╭──────┬──────────────────────────────────────────────────────────────────╮ **
** │ text ┆ embedding                                                        │ **
** │ ---  ┆ ---                                                              │ **
** │ String ┆ List[Float32]                                                  │ **
** ╞══════╪══════════════════════════════════════════════════════════════════╡ **
** │ 树苗 ┆ [0.089660645, -0.022903442, 0…                                   │ **
** ├╌╌╌╌╌╌┼╌╌╌╌╌╌
```

- 基于 Qwen2.5-VL 多模态大模型生成视频描述
- 使用 vLLM 推理引擎，支持高吞吐并行推理
- 支持自定义 prompt，灵活指定理解任务

> 该算子运行需要依赖对应的模型权重文件，平台已将所需权重文件预置在公共数据集`aihc_daft_public_model`中，需要在创建开发负载时挂载该数据集即可正常使用。

输入 |输入列名|说明| |-|-| |videos|包含视频数据的数组，元素类型为 字符串 或者 二进制。| |user_prompts|视频理解对应的prompt，默认为None。如果不传入，则默认使用参数中的prompt。|

参数 |参数名称|类型|默认值|描述| |-|-|-| |video_src_type|str|video_url|输入视频的格式类型。支持："video_url": bos/http地址"video_base64": base64编码"video_binary": 二进制流可选值：["video_url", "video_base64", "video_binary"]默认值："video_url"| |model_path|str|/opt/aihc/models|本地模型文件存储的绝对路径。默认为容器内预置路径，当使用自定义模型时需修改此路径。默认值："/opt/aihc/models"| |model_name|str|Qwen/Qwen2.5-VL-7B-Instruct|支持的视觉语言模型版本。当前仅支持Qwen2.5-VL系列模型。可选值：["Qwen/Qwen2.5-VL-7B-Instruct-AWQ", "Qwen/Qwen2.5-VL-7B-Instruct", "Qwen/Qwen2.5-VL-32B-Instruct-AWQ", "Qwen/Qwen2.5-VL-32B-Instruct"]默认值："Qwen/Qwen2.5-VL-7B-Instruct"| |prompt|str|请给出这段视频的详细描述。|用户理解视频内容的提示词，模型会根据提示词来生成视频的描述。设置为空时，建议针对每条数据传入特定的prompt。默认值："请给出这段视频的详细描述。"| |batch_size|int|4|单次推理处理的样本数量。较大的batch_size可提升吞吐但增加显存消耗，建议根据GPU显存调整。默认值：4| |dtype|str|bfloat16|模型推理精度选择。"bfloat16": 平衡精度与速度"float16": 更快的推理速度"float32": 最高精度但显存消耗最大可选值：["bfloat16", "float16", "float32"]默认值："bfloat16"| |max_model_len|int|128000|支持的最大模型输入长度（token数），影响可处理视频描述的长度，不能超过128000。默认值：128000| |max_num_seqs|int|128|单批次最大序列数，影响并发推理能力。默认值：128| |enable_prefix_caching|bool|True|是否启用前缀缓存以加速多轮推理。默认值：True| |gpu_memory_utilization|float|0.9|单卡GPU显存利用率上限，范围0~1。默认值：0.9| |enforce_eager|bool|False|是否强制使用eager模式推理，调试或特殊场景可用。默认值：False| |min_pixels|int or None|None|视频最小像素。不设置时，默认使用视频的原像素。视频像素越大，GPU显存占用越高。默认值：None| |max_pixels|int or None|None|视频最大像素。不设置时，默认使用视频的原像素。视频像素越大，GPU显存占用越高。默认值：None| |fps|float or None|None|视频帧率。不设置时，默认使用视频的原帧率。视频帧率越高，GPU显存占用越高。建议设置特定的帧率值。默认值：None| |temperature|float|1.0|采样温度，控制生成内容的多样性。值越高生成越随机。默认值：1.0| |top_p|float|0.2|nucleus采样的概率阈值，控制生成内容的多样性。值越小生成越保守。默认值：0.2| |repetition_penalty|float|1.05|重复惩罚系数，防止生成重复内容。值越大重复内容越少。默认值：1.05| |max_tokens|int|8192|单次生成的最大token数，影响描述长度。默认值：8192| |stop_token_ids|list[]|生成时遇到这些token id则停止。用于自定义生成终止条件。默认值：[]| |seed|int|42|随机种子，保证推理结果可复现。默认值：42|

输出 处理后的数组，元素为每个视频的视觉理解结果。

使用示例

```
 from __future__ import annotations

import os
import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.qwen_vl_video_understanding_vllm import QwenVLVideoUnderstandingVLLM

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video_path": [
            f"https://{bucket}.bj.bcebos.com/sample.mp4"
        ],
        "prompt": ["请给出视频的详细描述。"]
    }

    video_src_type = "video_url"
    model_path = os.getenv("MODEL_PATH", "/opt/aihc/models")
    model_name = os.getenv("MODEL_NAME", "Qwen/Qwen2.5-VL-7B-Instruct")
    dtype = "bfloat16"
    default_prompt = None
    max_caption_length = 256
    min_pixels = 320 * 160
    max_pixels = 320 * 160
    fps = 1
    batch_size = 2
    seed = 42
    max_model_len = 12800
    max_num_seqs = 128
    enable_prefix_caching = True
    gpu_memory_utilization = 0.95
    enforce_eager = True

    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "caption",
        aihc_udf(
            QwenVLVideoUnderstandingVLLM,
            construct_args={
                "video_src_type": video_src_type,
                "model_path": model_path,
                "model_name": model_name,
                "dtype": dtype,
                "prompt": default_prompt,
                "max_caption_length": max_caption_length,
                "min_pixels": min_pixels,
                "max_pixels": max_pixels,
                "fps": fps,
                "batch_size": batch_size,
                "seed": seed,
                "max_model_len": max_model_len,
                "max_num_seqs": max_num_seqs,
                "enable_prefix_caching": enable_prefix_caching,
                "gpu_memory_utilization": gpu_memory_utilization,
                "enforce_eager": enforce_eager,
            },
            num_gpus=1,
            batch_size=batch_size,
            concurrency=1,
        )(col("video_path"), col("prompt")),
    )

    ds.show()
** ┌──────────────────────┬──────────────────┬──────────────────────────────────────────────────┐**
*
** │ video_path           ┆ prompt           ┆ caption                                          │**
** │ ---                  ┆ ---              ┆ ---                                              │**
** │ String               ┆ String           ┆ String                                           │**
** ╞══════════════════════╪══════════════════╪══════════════════════════════════════════════════╡**
*
** │ https://{bucket}.bj.bcebos.com/ ┆ 请给出视频的详细描述。 ┆ 这段视频展示了一对男女在室内对话的场景。背景中可以看到一些… │**
** └──────────────────────┴──────────────────┴──────────────────────────────────────────────────┘**
*
```

图像内容理解

简介 Qwen2.5-VL 多模态图像理解模型，支持视觉语义解析与自然语言描述生成，适用于多种图像分析场景。

功能描述

- 多模态时序建模：支持三种图像输入格式（URL、Base64编码、二进制流），灵活适配不同数据源。
- 对话式提示支持：通过 `prompt` 参数自定义生成方向，满足多样化业务需求。
- 高效推理：集成VLLM推理引擎，支持 `bfloat16`、`float16`、`float32` 三种精度，充分利用GPU算力。
- 推荐使用48G及以上显存的GPU
- 图像内容理解与摘要
- 智能监控与事件分析

- 多模态对话与交互
- 其他需要图像语义解析的AI应用
- 仅支持GPU环境，建议根据显存和业务需求合理设置参数。
- 推荐在推理前统一图像，以获得最佳效果。

算子参数 输入 |输入|含义| |-|-| |images|包含图像数据的数组，元素类型为 字符串 或者 二进制。| |user_prompts|-|

输出 |输出|含义| |-|-| |result|pa.Array: 处理后的数组，元素为每个图像的理解结果。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |image_src_type|str|'image_url'|输入图像的格式类型。支持： - "image_url": bos/http地址 - "image_base64": base64编码 - "image_binary": 二进制流 可选值：["image_url", "image_base64", "image_binary"] 默认值："image_url"| |model_path|str|'/opt/aihc/models'|本地模型文件存储的绝对路径。默认为容器内预置路径，当使用自定义模型时需修改此路径。 默认值："/opt/aihc/models"| |model_name|str|'Qwen/Qwen2.5-VL-7B-Instruct'|支持的视觉语言模型版本。当前仅支持Qwen2.5-VL系列模型。 可选值：["Qwen/Qwen2.5-VL-7B-Instruct-AWQ", "Qwen/Qwen2.5-VL-7B-Instruct", "Qwen/Qwen2.5-VL-32B-Instruct-AWQ", "Qwen/Qwen2.5-VL-32B-Instruct", "Qwen/Qwen2.5-VL-72B-Instruct"] 默认值："Qwen/Qwen2.5-VL-7B-Instruct"| |prompt|str|'请给出这张图片的详细描述。'|用户理解视频内容的提示词，模型会根据提示词来生成视频的描述。设置为空时，建议针对每条数据传入特定的prompt。 默认值："请给出这段视频的详细描述。"| |batch_size|int|4|单次推理处理的样本数量。较大的batch_size可提升吞吐但增加显存消耗，建议根据GPU显存调整。 默认值：4| |dtype|str|'bfloat16'|模型推理精度选择。 - "bfloat16": 平衡精度与速度 - "float16": 更快的推理速度 - "float32": 最高精度但显存消耗最大 可选值：["bfloat16", "float16", "float32"] 默认值："bfloat16"| |max_model_len|int|128000|支持的最大模型输入长度（token数），影响可处理视频描述的长度，不能超过128000。 默认值：128000| |max_num_seqs|int|128|单批次最大序列数，影响并发推理能力。 默认值：128| |tensor_parallel_size|int|1|张量并行的GPU数量，提升推理速度。 默认值：1| |enable_prefix_caching|bool|True|是否启用前缀缓存以加速多轮推理。 默认值：True| |gpu_memory_utilization|float|0.9|单卡GPU显存利用率上限，范围0~1。 默认值：0.9| |enforce_eager|bool|False|是否强制使用eager模式推理，调试或特殊场景可用。 默认值：False| |resized_height|int|None|图像高度。不设置时，默认使用图像的原高度。图像高度越大，GPU显存占用越高。 默认值：None| |resized_width|int|None|图像宽度。不设置时，默认使用图像的原宽度。图像宽度越大，GPU显存占用越高。 默认值：None| |temperature|float|1.0|采样温度，控制生成内容的多样性。值越高生成越随机。 默认值：1.0| |top_p|float|0.2|nucleus采样的概率阈值，控制生成内容的多样性。值越小生成越保守。 默认值：0.2| |repetition_penalty|float|1.05|重复惩罚系数，防止生成重复内容。值越大重复内容越少。 默认值：1.05| |max_tokens|int|8192|单次生成的最大token数，影响描述长度。 默认值：8192| |stop_token_ids|list[int]|[]|生成时遇到这些token id则停止。用于自定义生成终止条件。 默认值：[]| |seed|int|42|随机种子，保证推理结果可复现。 默认值：42|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.qwen_vl_image_understanding_vllm import QwenVLImageUnderstandingVLLM

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"images": [...], "user_prompts": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "image_src_type": 'image_url',
        "model_path": '/opt/aihc/models',
        "model_name": 'Qwen/Qwen2.5-VL-7B-Instruct',
        "prompt": '请给出这张图片的详细描述。',
        "batch_size": 4,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            QwenVLImageUnderstandingVLLM,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("images"), col("user_prompts")),
    )
    ds.show()
```

视频Embedding（Cosmos）

简介 CosmosVideoEmbedding 基于 NVIDIA Cosmos-Embed1-448p 模型，对输入视频进行语义 embedding，输出 L2 归一化的浮点向量，用于视频检索、相似度计算与去重。

功能描述 • 使用 transformers 原生推理加载 Cosmos-Embed1 模型，从视频中均匀抽帧（默认 8 帧）后计算视频语义向量。

• 支持三种视频输入源：`video_url` / `video_base64` / `video_binary`。

• 输出投影后的视觉语义向量（`visual_proj`），与文本 embedding 处于同一空间，默认做 L2 归一化。

• 必须运行在 CUDA GPU 上，权重需预置于 `{model_path}/{model_name}`（不自动下载），出错时可选返回 None 或抛异常。

算子参数 输入 |输入|含义| |-|-| |videos|视频输入列，取值形式由 `video_src_type` 决定（视频 URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result|视频 embedding，类型 `list<float32>`；默认 L2 归一化。批次失败且 `fail_on_error=False` 时该行为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 `"video_url"` / `"video_base64"` / `"video_binary"`| |model_path|str|"/opt/aihc/models"|权重根目录，实际模型目录为 `{model_path}/{model_name}`| |model_name|str|"nvidia/Cosmos-Embed1-448p"|模型名称/子目录，默认使用 Cosmos-Embed1-448p| |batch_size|int|4|单批处理的视频数量| |num_frames|int|8|每段视频均匀抽取的帧数| |dtype|str|"auto"|计算精度，可选 `"auto"` / `"bfloat16"` / `"float16"` / `"float32"`； `auto` 时按 GPU 是否支持 bf16 自动选择| |rank|int|0|多卡 rank，设备为 `cuda:{rank % cuda_device_count}`| |normalize|bool|True|是否对输出向量做 L2 归一化| |fail_on_error|bool|False|批次出错时是否抛异常；`False` 时对应行返回 None（向后兼容旧调用方）|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.embedding.cosmos_video_embedding import CosmosVideoEmbedding

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

MODEL_PATH = "/opt/aihc/models"  # MOCK

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video": ["bos://your-bucket/sample.mp4"],  # MOCK
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "embedding",
        aihc_udf(
            CosmosVideoEmbedding,
            construct_args={
                "video_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "nvidia/Cosmos-Embed1-448p",
                "num_frames": 8,
                "normalize": True,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

语义去重（余弦相似度）

简介 在同一聚类簇内基于余弦相似度做近重复剔除。与「GPU KMeans 聚类」串联：先聚 K 簇，同簇内两两 cosine 相似度 > threshold 视为近重复，贪心保留先出现者、剔除后来者。

功能描述 • 簇内比较：仅在相同 cluster_id 内两两比较，不跨簇，复杂度按簇规模控制

• 贪心保留：按下标顺序保留最先出现者，后来者与任一已保留向量超过阈值则剔除

• 未聚类保留：cluster_id < 0 或 embedding 为空的行不参与比较，直接保留

• 阈值可调：threshold 越接近 1 越严格（判重越少）

算子参数 输入 |输入|含义| |-|-| |embeddings|L2 归一化的语义向量，元素类型为 list。| |cluster_ids|int32 簇号，通常由「GPU KMeans 聚类」产出。|

输出 |输出|含义| |-|-| |result|bool，True=保留，False=剔除。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |threshold|float|0.95|cosine 相似度阈值，超过判为近重复 默认值：0.95| |fail_on_error|bool|False|出错是否抛异常 默认值：False|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.cluster.semantic_dedup_by_cosine import SemanticDedupByCosine

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # emb：L2 归一化向量；cid：由 GPU KMeans 聚类产出的簇号
    samples = {
        "emb": [[1.0, 0.0, 0.0], [1.0, 0.01, 0.0], [0.0, 1.0, 0.0]],
        "cid": [0, 0, 0],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "keep",
        aihc_udf(
            SemanticDedupByCosine,
            construct_args={"threshold": 0.95},
            num_cpus=1,
            concurrency=1,
            batch_size=3,
        )(col("emb"), col("cid")),
    )
    ds.show()  # 期望 [True, False, True]：同簇近重复的后来者被剔除
```

结构化字幕生成

简介 StructuredCaptionPipeline 基于 Qwen2.5-VL-7B-Instruct，对图像或视频编排「5 象限扫描 + 时序动态 pass」并合并为结构化 JSON caption，输出序列化的 JSON 字符串。

功能描述 • 支持图像与视频共 6 种输入源：video_url / video_base64 / video_binary / image_url / image_base64 / image_binary。

• 通过多轮 prompt engineering 生成主体、光照、5 象限扫描、时序动态与音频推测等字段，合并成结构化 JSON。

• 双后端推理：vLLM（默认）与 transformers 兜底，backend="auto" 时在 sm_120 新卡自动回退 transformers；lora_path 非空时加载结构化 caption LoRA。

• 必须运行在 CUDA GPU 上，权重需预置于 {model_path}/{model_name}；do_temporal_pass 仅对视频生效，图像自动跳过。

算子参数 输入 |输入|含义| |-|-| |contents|图像或视频输入列，取值形式由 content_src_type 决定（URL / base64 / 二进制）|

输出 |输出|含义| |-|-| |result|结构化 caption，类型 string（序列化 JSON 字符串，含 subjects / lighting / quadrant_scan / temporal / audio）；失败且 fail_on_error=False 时为空字符串|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-|-| |content_src_type|str|"video_url"|输入类型，视频/图像各三种： video_url / video_base64 / video_binary / image_url / image_base64 / image_binary|| |model_path|str|"/opt/aihc/models"|权重根目录，实际模型目录为 {model_path}/{model_name}|| |model_name|str|"Qwen/Qwen2.5-VL-7B-Instruct"|QwenVL 模型名称/子目录|| |lora_path|str |None|None|结构化 caption LoRA 权重路径；None 表示无 LoRA，走原生 Qwen2.5-VL 冷启动| |batch_size|int|2|单批推理量（视频显存开销大）|| |dtype|str|"auto"|计算精度，可选 "auto" / "bfloat16" / "float16" / "float32"|| |max_model_len|int|32000|最大上下文长度|| |max_pixels|int |None|1280000|视觉输入像素上限| |min_pixels|int |None|None|视觉输入像素下限| |fps|float |None|2.0|视频抽帧帧率| |tensor_parallel_size|int|1|张量并行度，仅 vLLM 后端生效|| |gpu_memory_utilization|float|0.85|GPU 显存利用率，仅 vLLM 后端生效|| |enforce_eager|bool|False|是否强制 eager 模式，仅 vLLM 后端生效|| |temperature|float|0.1|采样温度|| |top_p|float|0.9|nucleus 采样 top-p|| |max_tokens|int|2048|单次生成最大 token 数|| |seed|int|42|采样随机种子|| |do_quadrant_scan|bool|True|是否做 5 象限扫描|| |do_temporal_pass|bool|True|是否做时序动态单独 pass（视频生效，图像自动跳过）|| |backend|str|"auto"|推理后端，可选 "auto" / "vllm" / "transformers"；sm_120 自动走 transformers|| |fail_on_error|bool|False|出错时是否抛异常；False 时对应行返回空字符串||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.structured_caption_pipeline import StructuredCaptionPipeline

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

MODEL_PATH = "/opt/aihc/models"  # MOCK

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video": ["bos://your-bucket/sample.mp4"],  # MOCK
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "caption",
        aihc_udf(
            StructuredCaptionPipeline,
            construct_args={
                "content_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "Qwen/Qwen2.5-VL-7B-Instruct",
                "fps": 2.0,
                "do_quadrant_scan": True,
                "do_temporal_pass": True,
                "backend": "auto",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

图文视频Embedding（Qwen-VL）

简介 QwenVLEmbedding 基于 Qwen3-VL-Embedding-8B，对文本或图像进行图文联合 embedding，输出 4096 维 L2 归一化向量，便于下游 cosine 相似度、KMeans 聚类与去重。

功能描述 • 支持四种输入内容类型：`text` / `image_url` / `image_binary` / `image_base64`，由 `content_type` 指定。

• 双后端推理：`vLLM`（默认高吞吐）与 `transformers` 兜底；`backend="auto"` 时在 sm_120 新卡上自动回退 transformers。

• 输出 4096 维 `list<float32>`，默认做 L2 归一化；支持 `last` / `mean` / `cls` 三种池化方式。

• 必须运行在 CUDA GPU 上，权重需预置于 `{model_path}/{model_name}`（不自动下载）。

算子参数 输入 |输入|含义| |-|-| |contents|待编码的内容列，为文本字符串或图像（URL / base64 / 二进制），具体形式由 `content_type` 决定|

输出 |输出|含义| |-|-| |result|图文 embedding，类型 `list<float32>`（4096 维）；默认 L2 归一化。批次失败且 `fail_on_error=False` 时该行为 None|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |content_type|str|"image_url"|输入格式，可选 `"text"` / `"image_url"` / `"image_binary"` / `"image_base64"`|| |model_path|str|"/opt/aihc/models"|权重根目录，实际模型目录为 `{model_path}/{model_name}`|| |model_name|str|"Qwen/Qwen3-VL-Embedding-8B"|模型名称/子目录|| |batch_size|int|8|单批推理量（8B 模型显存占用较大）|| |dtype|str|"auto"|计算精度，可选 `"auto"` / `"bfloat16"` / `"float16"` / `"float32"`；`auto` 时优先 BF16|| |max_pixels|int |None|None|视觉输入像素上限，控制显存激活峰值| |min_pixels|int |None|None|视觉输入像素下限| |pooling|str|"last"|取序列表征方式，`"last"`=最后一个 token，`"mean"`=均值池化，`"cls"`=第一个 token|| |normalize|bool|True|是否 L2 归一化（cosine 去重语义正确所必需）|| |tensor_parallel_size|int|1|张量并行度，仅 vLLM 后端生效|| |gpu_memory_utilization|float|0.85|GPU 显存利用率，仅 vLLM 后端生效|| |enforce_eager|bool|False|是否强制 eager 模式，仅 vLLM 后端生效|| |rank|int|0|多卡 rank，模型部署位置为 `cuda:{rank % cuda_device_count}`|| |backend|str|"auto"|推理后端，可选 `"auto"` / `"vllm"` / `"transformers"`；sm_120 或缺 vLLM 时自动走 transformers|| |fail_on_error|bool|False|出错时是否抛异常；False 时对应行返回 None||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.embedding.qwen_vl_embedding import QwenVLEmbedding

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

MODEL_PATH = "/opt/aihc/models"  # MOCK

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "content": ["bos://your-bucket/sample.jpg"],  # MOCK
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "embedding",
        aihc_udf(
            QwenVLEmbedding,
            construct_args={
                "content_type": "image_url",
                "model_path": MODEL_PATH,
                "model_name": "Qwen/Qwen3-VL-Embedding-8B",
                "pooling": "last",
                "normalize": True,
                "backend": "auto",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("content")),
    )
    ds.show()
```

VLM评判打分

简介 VLMJudgeScorer 以 VLM-as-Judge 方式（默认 Qwen2.5-VL-7B-Instruct 顶替 Gemma），对「媒体 + 候选描述」在 Faithfulness / Completeness / Correctness 三个维度各打 1-5 分。

功能描述 • 接收两个输入列：媒体（视频或图像）与 caption（StructuredCaptionPipeline 输出的 JSON 字符串或任意文本描述）。

• 通过评审 prompt 引导模型只输出一行 JSON（字段 faith / complete / correct），并用正则解析裁剪到 0-5 整数。

• 双后端推理：vLLM（默认）与 transformers 兜底，backend="auto" 时在 sm_120 新卡自动回退 transformers。

• 必须运行在 CUDA GPU 上，权重需预置于 {model_path}/{model_name}；解析失败或出错时三项分数全为 0。

算子参数 输入 |输入|含义| |-|-| |contents|媒体输入列，视频或图像（URL / base64 / 二进制），形式由 content_src_type 决定| |captions|候选描述列，StructuredCaptionPipeline 输出的 JSON 字符串或任意文本 caption|

输出 |输出|含义| |-|-| |result|评分结果，类型 struct{faith: int8, complete: int8, correct: int8}，各维度取值 1-5；解析失败或出错时全为 0|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |content_src_type|str|"video_url"|媒体输入类型，视频/图像各三种： video_url / video_base64 / video_binary / image_url / image_base64 / image_binary|| |model_path|str|"/opt/aihc/models"|权重根目录，实际模型目录为 {model_path}/{model_name}|| |model_name|str|"Qwen/Qwen2.5-VL-7B-Instruct"|评审模型名称/子目录|| |batch_size|int|2|单批推理量|| |dtype|str|"auto"|计算精度，可选 "auto" / "bfloat16" / "float16" / "float32"|| |max_model_len|int|16000|最大上下文长度|| |max_pixels|int |None|1280000|视觉输入像素上限| |fps|float |None|1.0|视频抽帧帧率| |tensor_parallel_size|int|1|张量并行度，仅 vLLM 后端生效|| |gpu_memory_utilization|float|0.85|GPU 显存利用率，仅 vLLM 后端生效|| |enforce_eager|bool|False|是否强制 eager 模式，仅 vLLM 后端生效|| |temperature|float|0.0|采样温度（评审默认贪心）|| |top_p|float|0.95|nucleus 采样 top-p|| |max_tokens|int|128|单次生成最大 token 数|| |seed|int|42|采样随机种子|| |backend|str|"auto"|推理后端，可选 "auto" / "vllm" / "transformers"；sm_120 自动走 transformers|| |fail_on_error|bool|False|出错时是否抛异常；False 时对应行返回全 0||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.vlm_judge_scorer import VLMJudgeScorer

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

MODEL_PATH = "/opt/aihc/models"  # MOCK

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video": ["bos://your-bucket/sample.mp4"],  # MOCK
        "caption": ["A person is demonstrating an action in the video."],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "judge",
        aihc_udf(
            VLMJudgeScorer,
            construct_args={
                "content_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "Qwen/Qwen2.5-VL-7B-Instruct",
                "fps": 1.0,
                "backend": "auto",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video"), col("caption")),
    )
    ds.show()
```

LanceDB_向量索引

简介 把 (sample_id, embedding, meta) 批量写入 LanceDB 表，并可选创建 IVF_PQ 向量索引；返回每行的顺序累计写入行号，作为向量检索链路的落库步骤。

功能描述 • 批量写入：按批写入 (sample_id, embedding, meta_json)，返回顺序累计的 int64 行号

• 可选建索引：写入完成且累计行数 ≥ min_rows_for_index 时自动创建 IVF_PQ 索引

• 维度校验：以首行向量维度为准，维度不一致的行跳过

• 空值稳健：embedding 或 sample_id 为空的行返回 -1

算子参数 输入 |输入|含义| |-|-| |sample_ids|每条样本的唯一键 str。| |embeddings|L2 归一化向量，元素类型为 list。| |meta_jsons|结构化 meta 的 JSON 字符串，可为空。|

输出 |输出|含义| |-|-| |result|int64 写入行号（顺序累计）；无效行返回 -1。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |uri|str|（必填）|LanceDB 目录 URI，本地路径或 s3://、bos:// 均可| |table_name|str|'p6_samples'|表名 默认值："p6_samples"| |vector_column|str|'vector'|向量列名 默认值："vector"| |create_ivf_pq|bool|True|写入完成后是否创建 IVF_PQ 索引 默认值：True| |num_partitions|int|64000|IVF 分区数 默认值：64000| |num_sub_vectors|int|64|PQ 子向量数 默认值：64| |metric|str|'cosine'|距离度量 可选值：["cosine", "l2", "dot"] 默认值："cosine"| |min_rows_for_index|int|20000|建索引最小行数，不足则仅写入不建索引 默认值：20000| |overwrite|bool|False|是否重建表 默认值：False| |fail_on_error|bool|False|出错是否抛异常 默认值：False|

调用示例

```
from __future__ import annotations

import json
import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.index.lance_db_vector_index import LanceDBVectorIndex

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # uri 为 mock 路径，请替换为实际 LanceDB 目录（本地路径 / s3:// / bos://）
    LANCEDB_URI = "bos://your-bucket/index/p6_lancedb"  # MOCK

    n, dim = 8, 16
    samples = {
        "sample_id": [f"id_{i}" for i in range(n)],
        "embedding": [[round(((i + j) % 5) * 0.1, 3) for j in range(dim)] for i in range(n)],
        "meta_json": [json.dumps({"i": i}) for i in range(n)],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "row",
        aihc_udf(
            LanceDBVectorIndex,
            construct_args={
                "uri": LANCEDB_URI,
                "table_name": "p6_samples",
                "metric": "cosine",
                "min_rows_for_index": 20000,  # 示例数据量小，不足则仅写入不建索引
                "overwrite": True,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=None,
        )(col("sample_id"), col("embedding"), col("meta_json")),
    )
    ds.show()
```

GPU*KMeans*聚类

简介 cuML GPU K-Means 优先、faiss/sklearn CPU 兜底的向量聚类算子。对一整列 embedding 一次性 fit+predict，输出等长的 cluster_id 列；常与「语义去重（余弦相似度）」串联，实现「先聚 K 簇、再簇内近重复剔除」。

功能描述 • 后端自适应：backend=auto 时按可用性选择 cuml > faiss > sklearn，无 GPU 自动回退 CPU

• 全列聚类：对整列 embedding 一次性 fit+predict，需配合 concurrency=1、batch_size=None 保证同批处理

• 小数据保护：当 n_samples < min_samples_ratio × n_clusters 时自动下调簇数，避免小数据集报错

• 空值稳健：embedding 为 None 的行分配 cluster_id = -1，其余正常聚类

算子参数 输入 |输入|含义| |-|-| |embeddings|embedding 向量列（1 列），元素类型为 list。|

输出 |输出|含义| |-|-| |result|int32 cluster_id；embedding 为空的行返回 -1。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |n_clusters|int|20000|目标簇数 默认值：20000| |backend|str|'auto'|后端选择 可选值：["auto", "cuml", "faiss", "sklearn"]，auto 时按 cuml>faiss>sklearn 默认值："auto"| |max_iter|int|20|最大迭代次数 默认值：20| |init|str|'scalable-k-means++'|初始化策略 可选值：["scalable-k-means++", "random"] 默认值："scalable-k-means++"| |seed|int|42|随机种子 默认值：42| |min_samples_ratio|float|2.0|样本数/簇数下限，不足时自动下调簇数 默认值：2.0| |fail_on_error|bool|False|出错是否抛异常 默认值：False|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.cluster.gpu_kmeans_cluster import GpuKMeansCluster

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(min_cpu_per_task=0)

    # embeddings：一整列 L2 归一化向量（示例造 3 组明显分离的簇）
    samples = {
        "emb": [
            [0.99, 0.10], [0.98, 0.12], [0.97, 0.14],
            [0.10, 0.99], [0.12, 0.98], [0.14, 0.97],
            [-0.70, -0.71], [-0.72, -0.69], [-0.68, -0.73],
        ]
    }
    ds = daft.from_pydict(samples)
    # 聚类为全列一次性 fit+predict：concurrency=1, batch_size=None
    ds = ds.with_column(
        "cluster_id",
        aihc_udf(
            GpuKMeansCluster,
            construct_args={
                "n_clusters": 3,
                "backend": "auto",
                "max_iter": 20,
                "seed": 42,
                "min_samples_ratio": 1.0,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=None,
        )(col("emb")),
    )
    ds.show()
```

短语定位召回率

简介

从 caption 里抽名词短语，用 Owl-ViT 做开集检测，统计有多少名词短语能在图上定位到，得出召回率。算子只产出召回率分数，不做阈值过滤，过滤交给 pipeline。

功能描述

- 名词短语抽取：word_tokenize 分词（先小写）→ nltk pos_tag → RegexpParser 按语法 NP: {<DT>?<JJ.*>*<NN.*>+} 取 NP 子树 → 去标点 → 去重
- 分词兜底：word_tokenize 在 nltk>=3.8.2 需要 punkt_tab，只预置了 punkt 时会 LookupError，此时自动回退到不需要数据文件的 TreebankWordTokenizer
- 检测：所有名词短语作为一组 text query 一次送 OwlViTProcessor + OwlViTForObjectDetection，post_process_object_detection 的阈值取 conf_thr
- 召回率统计：预测按置信度降序，只看前 短语数 个预测；label 越界的跳过；同一短语只记一次命中；框面积 / 图面积超过 large_area_ratio_thr 的（几乎覆盖整图）跳过；与已命中框 IoU 超过 iou_thr 的（重复框）跳过。召回率 = 命中短语数 / 短语总数，保留 6 位小数
- caption 抽不到名词短语时该行记 1.0（不是 0）
- 图像为 None、文本非字符串或去空白后为空、以及该行推理抛异常，都输出 None 并记 exception 日志，不中断整列
- 算子内不做网络请求与安装：nltk 数据要求离线预置在 nltk_data_path（缺失即构造期抛 FileNotFoundError），并把该目录插到 nltk.data.path 首位
- 逐行推理，不做批合并；图像统一 convert("RGB")，target_sizes 用原图尺寸，框坐标是原图像素的 [xmin, ymin, xmax, ymax]
- Owl-ViT 权重按默认精度（float32）加载，算子不提供 dtype 参数；设备为 cuda:(rank % 可见卡数)（use_gpu 且 CUDA 可用）否则 cpu
- 构造期校验 image_src_type 取值；两列长度不一致时抛 ValueError

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（URL/本地或 BOS 路径 / Base64 字符串 / 二进制）
texts	caption 文本列，名词短语从这里抽；空串或空白串该行输出 None

输出

输出	含义
grounding_recall	float64 —— 名词短语定位召回率，取值 [0, 1]；抽不到名词短语记 1.0；图像/文本为空或该行失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url（本地/BOS 路径）、image_base64、image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"google/owlvit-base-patch32"	相对 model_path 的 Owl-ViT 权重子目录
nltk_data_path	str	"/opt/aihc/model/nltk_data"	离线预置的 nltk 数据目录，需含 taggers/averaged_perceptron_tagger*
iou_thr	float	0.5	命中框之间的 IoU 抑制阈值，超过视为重复框
large_area_ratio_thr	float	0.95	框面积 / 图面积超过该比例视为覆盖整图，丢弃
conf_thr	float	0.0	检测置信度阈值，透传 post_process_object_detection；默认 0 表示不按置信度过滤
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 抽不到名词短语的样本返回 `1.0`（表示「无短语可定位，不惩罚」），下游按「低于阈值即丢弃」过滤时不会误杀这类样本，但也意味着 `1.0` 不能一律当作「定位很好」。
- nltk 数据缺失时构造期直接抛 `FileNotFoundError`，不会联网下载；POS 标注用 `averaged_perceptron_tagger`，若 `punkt_tab` 不可用，分词会退化到 `TreebankWordTokenizer` 兜底并打 warning 日志。
- NP 语法基于英文 POS，中文 caption 抽不到名词短语，结果恒为 `1.0`。
- 短语数越多，一次前向的 text query 越多，显存与耗时随之上升；同时「只看前 短语数 个预测」的口径意味着短语很多时低分预测会被裁掉。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.phrase_grounding_recall import PhraseGroundingRecall

MODEL_PATH = "/path/to/models"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

image = "bos://your-bucket/sample.jpg"
    ds = daft.from_pydict({"image": [image, image], "text": ["a photo of a dog and a red car", "quickly"]})
    ds = ds.with_column(
        "grounding_recall",
        aihc_udf(
            PhraseGroundingRecall,
            construct_args={
                "image_src_type": "image_url",
                "model_path": MODEL_PATH,
                "model_name": "google/owlvit-base-patch32",
                "nltk_data_path": f"{MODEL_PATH}/nltk_data",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=2,
        )(col("image"), col("text")),
    )
    ds.show()
```

图像描述生成

简介

BLIP2 图像描述（caption）生成算子，对每张图生成 `caption_num` 条候选描述，再按 `keep_candidate_mode` 归约成输出。权重只从本地目录加载。

功能描述

- 模型走 transformers `AutoProcessor` + `Blip2ForConditionalGeneration`，权重目录为 `{model_path}/{model_name}`，目录不存在直接抛 `FileNotFoundError`，不联网下载、不做运行时 pip 安装
- 候选生成方式：同一批图重复调用 `generate` `caption_num` 次，每次得到一条候选；`do_sample=False` 时多次调用退化成同一条贪心结果
- `keep_candidate_mode` 三种归约：`all` 返回全部候选；`random_any` 随机取一条；`similar_one_simhash` 取与参考文本 simhash 汉明距离最小的一条，参考文本缺失时退化为 `random_any`
- `similar_one_simhash` 的 simhash 是算子自带的 64 位实现（window_size=2 的 shingle + 汉明距离），不依赖额外 C 扩展；候选文本极端相近时汉明距离可能打平，选中哪一条不保证稳定
- `prompt` 非空时对整批图使用同一条提示词，为 `None` 时不带提示
- 本算子是列式算子，**行数恒定**，输出列直接给归约后的 caption 列表，是否保留原样本由 pipeline 决定
- 逐行解码图像，单行解码失败只记日志并留空列表；整批 `generate` 失败时该批涉及的行全部为空列表，不中断整列
- 输入图像为 `None` 时该行输出空列表
- 算子内部按 `batch_size` 再切微批做推理，与 `aihc_udf` 的 `batch_size` 是两个独立参数
- 构造期校验 `image_src_type`、`keep_candidate_mode` 取值以及 `caption_num > 0`

算子参数

输入

输入	含义
image	图像输入列，内容形式由 image_src_type 决定（URL/本地或 BOS 路径 / Base64 字符串 / 二进制）
reference_text	可选第二列，keep_candidate_mode=similar_one_simhash 时作为参考文本；不传或非字符串时该行退化为随机取一条

输出

输出	含义
captions	list<large_string> —— 归约后的 caption 列表。all 模式给全部 caption_num 条，其余模式给 1 条；无图或失败给空列表

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url（本地/BOS 路径）、image_base64、image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"Salesforce/blip2-opt-2.7b"	相对 model_path 的 BLIP2 权重子目录
dtype	str	"float16"	权重精度：float32 / float16 / bfloat16。GPU 上建议 float16
caption_num	int	1	每张图生成的候选数，必须为正
keep_candidate_mode	str	"random_any"	候选归约方式：random_any / similar_one_simhash / all
prompt	str 或 None	None	全局提示词，None 表示不带提示
max_new_tokens	int	128	生成长度上限
do_sample	bool	True	是否采样，caption_num > 1 时需为 True 才有多样性
batch_size	int	8	算子内部推理微批大小
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)，无 GPU 时为 cpu

注意事项

- 权重缺失时构造期直接抛 FileNotFoundError，不会自动联网下载。
- caption_num > 1 时必须保持 do_sample=True，否则重复的贪心解码会给出完全相同的候选。
- all 模式下输出列长度等于 caption_num，其余模式恒为 1；下游按固定长度取值时要区分。
- 算子本身不做样本增删，输出行数恒等于输入行数；需要「保留原样本」「生成样本翻倍」这类语义要在 pipeline 层自己做。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.image_captioning import ImageCaptioning

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"image": ["bos://your-bucket/sample.jpg"]})
    ds = ds.with_column(
        "captions",
        aihc_udf(
            ImageCaptioning,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "Salesforce/blip2-opt-2.7b",
                "dtype": "float16",
                "caption_num": 2,
                "keep_candidate_mode": "all",
                "max_new_tokens": 32,
                "batch_size": 2,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image")),
    )
    ds.show()
```

图文匹配评分（BLIP ITM）

简介

用 BLIP 的 ITM（image-text matching）判别头给出「这张图与这段文本是否匹配」的概率分数：逐行把图像与配对文本送进判别头，输出 [0, 1] 的匹配概率。算子只产出分数，不做阈值过滤，筛样本交给 pipeline。

功能描述

- 模型走 transformers BlipProcessor + BlipForImageTextRetrieval，前向时 use_itm_head=True，取 itm_score 的 2 类 logits（[不匹配, 匹配]）做 softmax，输出匹配类概率，保留 6 位小数
- 分数量纲是判别头的概率，取值 [0, 1]；与 CLIP 双塔余弦不是同一个尺度，阈值不能互相搬用

- 只对「图像列非空 且 文本列非空（字符串去空白后非空）」的行做推理，其余行直接输出 None
- 按 batch_size 切微批推理，某个微批抛异常时只把该批的行留成 None 并记 exception 日志，不影响其他批
- 文本侧交给 processor 做 padding=True, truncation=True，超长文本按分词器上限截断
- 图像统一 convert("RGB") 后送入 processor，pixel_values 按 dtype 转精度
- 设备为 cuda:(rank % 可见卡数)（use_gpu 且 CUDA 可用）否则 cpu；CPU 上指定 float16 会打 warning 并回退 float32
- 权重目录 {model_path}/{model_name} 不存在时构造期抛 FileNotFoundError，不联网下载
- 构造期校验 image_src_type 与 dtype 取值
- 两列长度不一致时抛 ValueError

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（URL/本地或 BOS 路径 / Base64 字符串 / 二进制）
texts	文本列，与图像逐行配对；空串或空白串视为无效

输出

输出	含义
itm_score	float64 —— ITM 判别头 softmax 后的匹配概率，取值 [0, 1]；任一侧为空或该微批失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url（本地/BOS 路径）、image_base64、image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"Salesforce/blip-itm-base-coco"	相对 model_path 的权重子目录
dtype	str	"float32"	权重精度：float16 / float32 / bfloat16；CPU 上 float16 回退 float32
batch_size	int	16	算子内部推理微批大小
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- ITM 判别头与 CLIP 双塔余弦不可互换：前者是跨模态判别概率，后者是归一化向量点积，量纲与语义都不同，阈值必须分别标定。
- 权重必须是带 retrieval/ITM 头的 BLIP 权重（如 blip-itm-base-coco）；BLIP captioning 权重没有 ITM 头，加载会失败。
- 算子不带 min_score / max_score 之类的过滤参数，输出只是分数，按分数筛样本需要在 pipeline 里另加过滤步骤。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.image_text_matching_score import ImageTextMatchingScore

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

image = "bos://your-bucket/sample.jpg"
    ds = daft.from_pydict({"image": [image, image], "text": ["a photo", ""]})
    ds = ds.with_column(
        "itm_score",
        aihc_udf(
            ImageTextMatchingScore,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "Salesforce/blip-itm-base-coco",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=2,
        )(col("image"), col("text")),
    )
    ds.show()
```

图文相似度（CLIP）

简介

CLIP 双塔图文相似度算子：图像过图像塔、文本过文本塔，两侧向量 L2 归一化后取余弦。算子只产出分数，不按 min/max 过滤，过滤交给 pipeline。

功能描述

- CLIP 封装复用内部模块 multimodal/_clip_dual_tower.py（transformers CLIPProcessor + CLIPModel），文本塔与图像塔各出一次归一化向量，相似度 = 两向量逐元素乘后求和，保留 6 位小数

- 文本塔按 CLIP 的 77 token 上限做 `truncation=True, max_length=77`，超长文本只有前 77 个 token 参与
- 只对「图像列非空 且 文本列非空（字符串去空白后非空）」的行做推理，其余行输出 `None`
- 按 `batch_size` 切微批，某个微批抛异常时只把该批的行留成 `None` 并记 `exception` 日志
- 图像统一 `convert("RGB")` 后编码，`pixel_values` 按 `dtype` 转精度；相似度计算在 float32 上做
- 设备为 `cuda:(rank % 可见卡数)`（use_gpu 且 CUDA 可用）否则 cpu；CPU 上指定 `float16` 会打 warning 并回退 float32
- 权重目录 {model_path}/{model_name} 不存在时构造期抛 `FileNotFoundError`，不联网下载
- 两列长度不一致时抛 `ValueError`

算子参数

输入

输入	含义
images	图像输入列，内容形式由 image_src_type 决定（URL/本地或 BOS 路径 / Base64 字符串 / 二进制）
texts	文本列，与图像逐行配对；空串或空白串视为无效

输出

输出	含义
sim	float64 —— 图文 CLIP 余弦相似度，取值 [-1, 1]；任一侧为空或该微批失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url（本地/BOS 路径）、image_base64、image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"openai/clip-vit-base-patch32"	相对 model_path 的 CLIP 权重子目录
dtype	str	"float32"	权重精度：float16 / float32 / bfloat16；CPU 上 float16 回退 float32
batch_size	int	16	算子内部推理微批大小
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- CLIP 文本塔硬上限 77 token，长 caption 会被静默截断，长文本场景的分数不代表全文。
- 与 `ImageTextMatchingScore`（BLIP ITM 判别头）不可互换：一个是双塔归一化向量的余弦（[-1,1]），一个是判别头概率（[0,1]），阈值必须分别标定。
- 算子只出分，不带 `min_score` / `max_score` 阈值过滤，过滤在 daft 侧用 `where()` 做。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.image_text_similarity import ImageTextSimilarity

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

image = "bos://your-bucket/sample.jpg"
    ds = daft.from_pydict({"image": [image, image], "text": ["a photo", ""]})
    ds = ds.with_column(
        "sim",
        aihc_udf(
            ImageTextSimilarity,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "openai/clip-vit-base-patch32",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=2,
        )(col("image"), col("text")),
    )
    ds.show()
```

图像对差异区域（ImgDiff）

简介

给一对相似图和各自的 caption，找出「两图语义不同」的区域框，用于构造 image-diff 训练数据。整条链路分七步（整对粗筛 → 差异名词抽取 → 分割 → 裁剪 → ITM 判别 → 子图对复筛 → 去重叠），三个模型（CLIP / BLIP-ITM / FastSAM）直接在算子内加载，裁剪子图全程留在内存、不落盘。

功能描述

处理流程（每行独立执行）：

- **Step1 整对粗筛**：CLIP 图图余弦相似度不落在 [min_score_1, max_score_1] 内（差异过大或过小）的图对直接丢弃
- **Step2 差异名词**：difflib.Differ 比较两个 caption（小写、去标点、按空格拆行），取各自独有的词做 WordNet 词形还原，再用 nltk POS 只保留名词（NN/NNS/NNP/NNPS）作为 "valid object"；任一侧抽不到名词就返回空列表
- **Step3 分割**：把 image_b 双线性缩放到 image_a 尺寸，两张图分别过 FastSAM，候选框是两张图分割结果的并集，坐标统一在 image_a 坐标系
- **Step4 裁剪**：每个候选框在两张图上各裁一块子图，框按中心点 xywh 语义还原成 (left, upper, right, lower) 并裁到图像边界内，宽或高不足 1 像素的框跳过
- **Step5 ITM 判别**：BLIP-ITM 判断子图是否含有对应的 valid object，任一 object 的匹配概率落在 [itm_min_score, itm_max_score] 内即算命中；两侧同时命中的框才保留
- **Step6 子图对复筛**：对保留下来的子图对再算一次 CLIP 相似度，不落在 [min_score_2, max_score_2] 内的框剔除
- **Step7 去重叠**：iou_filter 贪心去重（面积按 +1 计算），阈值 iou_thresh

其他行为：

- 输出是 list<list<f64>>，每个框是 **中心点 xywh** 四元组，坐标系为 image_a；任何一步筛空、任一输入为 None/空 caption、或该行抛异常，都输出空列表并记日志，不中断整列
- 七步流程在单进程内串起三个模型，裁剪子图全程内存内完成、不落盘；无框时输出空列表，不用零框占位
- CLIP / BLIP / FastSAM 权重与 nltk 数据都只从本地加载，缺任何一个在构造期就抛 FileNotFoundError，不联网下载、不做运行时 pip 安装
- 设备为 cuda:(rank % 可见卡数)（use_gpu 且 CUDA 可用）否则 cpu，FastSAM 用同一张卡的编号；CPU 上指定 float16 会回退 float32
- 构造期校验 image_src_type 以及三组阈值的下界不超过上界
- 四列长度不一致时抛 ValueError

算子参数

输入

输入	含义
image_a	第一张图，内容形式由 image_src_type 决定（URL/本地或 BOS 路径 / Base64 / 二进制）
image_b	第二张图，编码形式与 image_a 相同；会被缩放到 image_a 尺寸
caption_a	image_a 的描述文本
caption_b	image_b 的描述文本

输出

输出	含义
diff_bbox	list<list> —— 差异区域框列表，每个框是 [中心x, 中心y, 宽, 高]（image_a 坐标系）；无差异或该行失败时是空列表

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	两列图像的编码形式（同一种）：image_url / image_base64 / image_binary
model_path	str	"/opt/aihc/model"	权重根目录（注意是 model 而非 models）
clip_model_name	str	"openai/clip-vit-base-patch32"	CLIP 权重子目录
blip_model_name	str	"Salesforce/blip-itm-base-coco"	BLIP-ITM 权重子目录
fastsam_model_name	str	"FastSAM-x.pt"	FastSAM 权重文件名（相对 model_path）
nltk_data_path	str	"/opt/aihc/model/nltk_data"	离线预置的 nltk 数据目录，需含 taggers 与 corpora/wordnet
dtype	str	"float32"	BLIP / CLIP 权重精度：float16 / float32 / bfloat16
min_score_1	float	0.1	整对图相似度下界（Step1）
max_score_1	float	1.0	整对图相似度上界（Step1）
min_score_2	float	0.1	子图对相似度下界（Step6）
max_score_2	float	1.0	子图对相似度上界（Step6）
itm_min_score	float	0.1	子图与 valid object 的 ITM 分数下界（Step5）
itm_max_score	float	1.0	子图与 valid object 的 ITM 分数上界（Step5）
imgsz	int	1024	FastSAM 推理分辨率
conf	float	0.05	FastSAM 置信度阈值
iou	float	0.5	FastSAM NMS IoU 阈值
iou_thresh	float	0.5	最终框去重叠的 IoU 阈值
rank	int	0	多卡场景 worker 序号

注意事项

- 输出框是**中心点 xywh**，不是左上角 + 宽高；直接当 xyxy 或 ltwh 用会画错位置。
- nltk 数据不会自动联网下载（算子内不做网络请求与安装），nltk_data_path 指向的目录缺 taggers 或 corpora/wordnet 会在构造期或 POS 阶段失败。
- Step2 的 POS 与词形还原是英文语料，中文 caption 抽不到名词，整行直接返回空列表。
- 两个 caption 必须各有对方没有的名词，caption 完全相同或差异只在虚词上时结果恒为空。
- 三个模型在同一进程同一张卡上常驻，显存占用是三者之和；imgsz 越大 FastSAM 越吃显存（测试用 640）。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.imgdiff_difference_area import ImgdiffDifferenceArea

MODEL_PATH = "/path/to/models"

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict(
        {
            "image_a": ["/tmp/aihc_imgdiff/pair_a.jpg"],
            "image_b": ["/tmp/aihc_imgdiff/pair_b.jpg"],
            "caption_a": ["a photo of a room with a window"],
            "caption_b": ["a photo of a room with a box"],
        }
    )
    ds = ds.with_column(
        "diff_bbox",
        aihc_udf(
            ImgdiffDifferenceArea,
            construct_args={
                "image_src_type": "image_url",
                "model_path": MODEL_PATH,
                "nltk_data_path": os.path.join(MODEL_PATH, "nltk_data"),
                "imgsz": 640,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image_a"), col("image_b"), col("caption_a"), col("caption_b")),
    )
    ds.show()
```

多模态大模型推理

简介

本地多模态大模型图文问答算子：把文本列当 user prompt、图像列当视觉输入，走 processor 的 chat template 生成回答。默认权重 LLaVA-1.6（llava-hf/llava-v1.6-vicuna-7b-hf），纯本地推理，不调任何在线 API。

功能描述

- 模型走 transformers `AutoProcessor` + `LlavaNextForConditionalGeneration`，权重从 {model_path}/{model_name} 本地加载，目录不存在直接抛 `FileNotFoundError`，不联网下载、不做运行时 pip 安装
- 每行构造一段单轮对话（role=user，content 为 [文本, 图像]），用 `apply_chat_template(..., add_generation_prompt=True)` 渲染成 prompt 再送 `generate`
- 生成参数 `max_new_tokens` / `temperature` / `top_p` / `num_beams` 直接透传 `generate`
- 回答写进独立的输出列，不覆盖输入的文本列
- 逐行推理，不做批合并；图像为 `None`、文本非字符串或去空白后为空的行输出 `None`
- 单行抛异常时只把该行留成 `None` 并记 error 日志，不中断整列
- 图像统一转 RGB 后送 processor；设备为 `cuda:{rank % 可见卡数}`（use_gpu 且 CUDA 可用）否则 cpu
- 构造期校验 `image_src_type` 取值；两列长度不一致时抛 `ValueError`

算子参数

输入

输入	含义
image	图像输入列，内容形式由 image_src_type 决定（URL/本地或 BOS 路径 / Base64 字符串 / 二进制）
text	问题 / 指令文本，作为 user prompt；空串或空白串该行不推理

输出

输出	含义
answer	large_string —— 模型回答文本（已 strip）；无图、无文本或该行失败时为 None

参数

参数名称	类型	默认值	描述
image_src_type	str	"image_url"	图像输入类型：image_url（本地/BOS 路径）、image_base64、image_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"llava-hf/llava-v1.6-vicuna-7b-hf"	相对 model_path 的权重子目录
dtype	str	"float16"	权重精度：float32 / float16 / bfloat16
max_new_tokens	int	256	生成长度上限
temperature	float	0.2	采样温度
top_p	float	0.9	nucleus 采样阈值
num_beams	int	1	beam 数
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 解码的是 `generate` 返回的**完整序列**（`skip_special_tokens=True`），没有裁掉输入部分，所以输出里会带 chat template 渲染出的提示文本（如 `USER: ... ASSISTANT: ...`）和原问题；只要回答正文的话需要在下游按分隔符切。
- 算子没有显式传 `do_sample`，是否采样取决于权重自带的 `generation_config`；若走贪心解码，`temperature` / `top_p` 不生效（transformers 会打 warning）。
- 模型类固定为 `LlavaNextForConditionalGeneration`，`model_name` 只能换 LLaVA-NeXT 系权重，换成其他架构会加载失败。
- 7B 级权重常驻显存，逐行推理，吞吐由生成长度决定；批量场景建议缩小 `max_new_tokens`。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.multimodal.mllm import MLlm

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict(
        {
            "image": ["bos://your-bucket/sample.jpg"],
            "text": ["Describe this image in one sentence."],
        }
    )
    ds = ds.with_column(
        "answer",
        aihc_udf(
            MLlm,
            construct_args={
                "image_src_type": "image_url",
                "model_path": "/path/to/models",
                "model_name": "llava-hf/llava-v1.6-vicuna-7b-hf",
                "dtype": "float16",
                "max_new_tokens": 64,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("image"), col("text")),
    )
    ds.show()
```

视频

视频元数据提取

简介 视频元数据提取算子，基于 FFprobe 实现，用于从视频文件中提取详细的媒体信息。

功能描述 使用 FFprobe 提取视频文件的元数据信息，包括时长、格式、编码器、分辨率、帧率、音频信息等。支持 BOS 和 S3 路径的预签名 URL 处理。

算子参数 输入 |输入|含义| |-|-| |input_path|输入视频文件路径（支持本地路径、BOS 路径或远程 URL）|

输出 |输出|含义| |-|-| |duration|视频时长（秒）| |format_name|封装格式名称| |bit_rate|比特率| |has_video|是否包含视频流| |has_audio|是否包含音频流| |video_codec|视频编码器名称| |video_width|视频宽度| |video_height|视频高度| |video_fps|视频帧率| |audio_codec|音频编码器名称| |audio_sample_rate|音频采样率| |audio_channels|音频通道数|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |timeout|int|None|提取超时时间（秒）|

调用示例

```
 from __future__ import annotations

import os
import subprocess

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_extract_metadata import VideoExtractMetadata

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # 检查 ffprobe 是否可用
    try:
        subprocess.run(["ffprobe", "-version"], capture_output=True, check=True)
    except (subprocess.CalledProcessError, FileNotFoundError):
        print("SKIP: ffprobe not available")
        exit(0)

    samples = {
        "video_path": [
            "/tmp/test_clip1.mp4",
            "/tmp/test_clip2.mov",
        ]
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "metadata",
        aihc_udf(
            VideoExtractMetadata,
            construct_args={"timeout": 60},
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_path")),
    )
    ds = ds.with_column("duration", col("metadata")["duration"])
    ds = ds.with_column("resolution", col("metadata")["video_width"])
    ds.show()
```

视频关键帧抽取

简介 视频关键帧抽取算子，基于 OpenCV 和 FFmpeg 实现，支持多种关键帧检测方法。

功能描述 从视频中提取关键帧，支持 I 帧检测、帧差异、光流直方图等多种方法。可选择输出关键帧数据、Base64 编码或上传到 BOS 存储。

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列表| |video_binaries|视频二进制数据列表| |video_formats|视频格式列表|

输出 |输出|含义| |-|-| |keyframes|关键帧数据（四维数组）| |base64|Base64 编码的关键帧图片| |timestamps|关键帧对应的时间戳（秒）| |bos_paths|BOS 存储路径|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |method|str|"I_frame"|关键帧检测方法| |img_type|str|".jpg"|输出图片格式| |threshold|float|0|检测阈值| |keyframes_cnt|int|10|最大关键帧数量| |seconds_per_frame|int|-1|每帧秒数| |output_bos_dir|str|""|BOS 输出目录| |by_count_uniform|bool|False|是否均匀采样| |return_keyframes|bool|True|是否返回关键帧数据| |return_base64|bool|True|是否返回 Base64 编码|

关键帧检测方法 |方法|说明| |-|-| |I_frame|基于 I 帧检测| |difference|基于帧差异| |optical_flow|基于光流| |histogram|基于直方图比较|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_keyframes import VideoKeyframes

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video_path": ["/tmp/test_video.mp4", "/tmp/test_video2.mp4"]
    }
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "method": "I_frame",
        "keyframes_cnt": 10,
    }
    ds = ds.with_column(
        "keyframes",
        aihc_udf(
            VideoKeyframes,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
        )(col("video_path"), None, None),
    )
    ds.show()
```

视频安全性检测

简介 视频安全性检测算子基于 Transformers 模型实现，用于检测视频中是否存在不适宜在工作场所观看的内容。

功能描述 对视频内容进行安全性检测，通过采样视频帧并使用图像分类模型进行推理，返回安全性分数。支持多种视频输入方式和采样模式。分数越高表示内容越可能不适宜公开播放。

算子参数 输入 |输入|含义| |-|-| |videos|视频路径/Base64/二进制列表|

输出 |输出|含义| |-|-| |score|安全性检测分数（0-1 之间，1 表示高风险）|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型| |model_path|str|"/opt/aihc/models"|模型路径| |model_name|str|"Falconsai/nsfw_image_detection"|模型名称| |dtype|str|"float16"|数据类型| |sample_mode|str|"by_count_uniform"|采样模式| |start_time_sec|float|0.0|起始时间（秒）| |end_time_sec|float|None|结束时间（秒）| |count_k|int|None|采样帧数| |interval_sec|float|None|采样间隔（秒）| |interval_frames|int|None|帧间隔| |target_fps|float|None|目标帧率| |timestamps_sec|list[float]|None|指定时间戳列表| |max_frames|int|None|最大采样帧数| |reduce_mode|str|"avg"|结果聚合模式（avg/max/min）| |video_format|str|"mp4"|视频格式| |batch_size|int|16|批处理大小| |rank|int|0|GPU 设备编号|

视频输入类型 |类型|说明| |-|-| |video_url|视频 URL 路径| |video_base64|Base64 编码的视频数据| |video_binary|二进制视频数据|

采样模式 |模式|说明| |-|-| |by_count_uniform|等间隔均匀采样指定帧数| |by_interval_time|按时间间隔采样| |by_interval_frames|按帧间隔采样| |by_fps|按指定 FPS 采样| |by_timestamps|按指定时间戳采样|

结果聚合模式 |模式|说明| |-|-| |avg|取平均值| |max|取最大值| |min|取最小值|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_nsfw_detect import VideoNsfwDetect

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video_path": ["/tmp/test_video.mp4", "/tmp/test_video2.mp4"]
    }
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "video_src_type": "video_url",
        "sample_mode": "by_count_uniform",
        "count_k": 5,
        "reduce_mode": "avg",
    }
    ds = ds.with_column(
        "safety_score",
        aihc_udf(
            VideoNsfwDetect,
            construct_args=constructor_kwargs,
            num_cpus=2,
            concurrency=2,
            use_gpu=True,
        )(col("video_path")),
    )
    ds.show()
```

视频帧采样

简介

视频帧采样算子，基于 OpenCV 实现，支持多种采样模式从视频中提取帧。

功能描述 从视频中按指定规则采样帧，支持多种采样模式：等间隔采样、按帧数均匀采样、按 FPS 采样、按时间戳采样、按帧间隔采样。可选输出帧数据、Base64 编码或上传到 BOS 存储。

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列表| |video_binaries|视频二进制数据列表| |video_formats|视频格式列表| |video_durations|视频时长列表|

输出 |输出|含义| |-|-| |frames|帧数据（四维数组）| |base64|Base64 编码的帧图片| |timestamps|帧对应的时间戳（秒）| |frame_indices|帧索引列表| |bos_paths|BOS 存储路径|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |sample_mode|str|"by_count_uniform"|采样模式| |start_time_sec|float|0.0|起始时间（秒）| |end_time_sec|float|None|结束时间（秒）| |count_k|int|None|采样帧数（by_count_uniform模式）| |interval_sec|float|None|采样间隔（秒，by_interval_time模式）| |interval_frames|int|None|帧间隔（by_interval_frames模式）| |target_fps|float|None|目标帧率（by_fps模式）| |timestamps_sec|list[float]|None|指定时间戳列表（by_timestamps模式）| |img_type|str|".jpg"|输出图片格式| |output_bos_dir|str|""|BOS 输出目录| |max_frames|int|None|最大采样帧数| |seed|int|42|随机种子| |output_frames|bool|True|是否输出帧数据| |output_base64|bool|True|是否输出 Base64 编码|

采样模式 |模式|说明| |-|-| |by_count_uniform|等间隔均匀采样指定帧数| |by_interval_time|按时间间隔采样| |by_interval_frames|按帧间隔采样| |by_fps|按指定 FPS 采样| |by_timestamps|按指定时间戳采样|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_frame_sampler import VideoFrameSampler

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video_path": ["/tmp/test_video.mp4", "/tmp/test_video2.mp4"]
    }
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "sample_mode": "by_count_uniform",
        "count_k": 10,
        "output_base64": True,
    }
    ds = ds.with_column(
        "frames",
        aihc_udf(
            VideoFrameSampler,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
        )(col("video_path"), None, None, None),
    )
    ds.show()
```

视频片段切分-关键帧

简介 视频按关键帧分割算子，基于 FFmpeg 实现，在关键帧位置将视频分割成多个片段。

功能描述 检测视频中的关键帧位置，在关键帧处将视频分割成多个片段。可设置分割数量，使用均匀采样选择关键帧，支持输出到 BOS 存储。

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列表| |video_binaries|视频二进制数据列表| |video_formats|视频格式列表| |output_basenames|输出基础文件名列表|

输出 |输出|含义| |-|-| |segments|分割后视频片段的路径列表| |segments_binary|分割后视频片段的二进制数据列表| |video_format|各片段的视频格式列表|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |method|str|"I_frame"|关键帧检测方法| |threshold|float|0|检测阈值| |keyframes_cnt|int|10|关键帧数量| |seconds_per_frame|int|-1|每帧秒数| |output_bos_dir|str|""|BOS 输出目录| |output_segments_binary|bool|False|是否输出二进制数据| |output_video_format|str|None|输出视频格式|

关键帧检测方法 |方法|说明| |-|-| |I_frame|基于 I 帧检测（默认）| |difference|基于帧差异| |optical_flow|基于光流| |histogram|基于直方图比较|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_split_by_keyframes import VideoSplitByKeyframes

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video_path": ["/tmp/long_video.mp4", "/tmp/long_video2.mp4"]
    }
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "method": "I_frame",
        "keyframes_cnt": 5,
    }
    ds = ds.with_column(
        "segments",
        aihc_udf(
            VideoSplitByKeyframes,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
        )(col("video_path"), None, None, None),
    )
    ds.show()
```

视频片段切分-时长

简介

视频按时间分段算子，基于 FFmpeg 实现，将视频按指定时长均匀分割成多个片段。

功能描述 将长视频按固定时长分割成多个短片段，可设置最小片段时长，支持输出到 BOS 存储。可选择输出片段路径列表或二进制数据。

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列表| |video_binaries|视频二进制数据列表| |video_formats|视频格式列表| |output_basenames|输出基础文件名列表|

输出 |输出|含义| |-|-| |segments|分割后视频片段的路径列表| |segments_binary|分割后视频片段的二进制数据列表|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |segment_duration|float|5.0|片段时长（秒）| |min_segment_duration|float|0.0|最小片段时长（秒）| |output_bosdir|str|""|BOS 输出目录| |output_segments_binary|bool|False|是否输出二进制数据| |output_video_format|str|None|输出视频格式|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_split_by_duration import VideoSplitByDuration

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "video_path": ["/tmp/long_video.mp4", "/tmp/long_video2.mp4"]
    }
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "segment_duration": 10.0,
        "min_segment_duration": 2.0,
    }
    ds = ds.with_column(
        "segments",
        aihc_udf(
            VideoSplitByDuration,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
        )(col("video_path"), None, None, None),
    )
    ds.show()
```

视频质量评分

简介 VideoQualityScore 是基于 Daft + Ray 分布式框架的视频质量评分组件，利用 CLIP 模型对视频帧进行质量评估，输出一个综合质量分数，适用于视频筛选、质量过滤等场景。视频质量评分处理器，基于 CLIP-IQA 对采样帧评分并聚合为视频质量分

功能

- 支持通过视频采样进行质量评分
- 基于 CLIP 模型的视频帧质量评估
- 支持多种帧采样策略（如均匀采样）
- 支持多种分数聚合方式（如平均值）

输入 |输入列名|说明| |-|-| |video_paths|可选，视频文件路径列（本地、BOS、HTTP等），与 video_binaries 二选一| |video_binaries|可选，视频二进制数据列，与 video_paths 二选一| |video_formats|可选，视频格式字符串列，配合 video_binaries 使用|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|/opt/aihc/models|模型文件存储路径| |clip_model_name|str|openai/clip-vit-large-patch14|CLIP 模型名称或路径| |prompt|str|quality|质量评估提示词| |device|str|cuda|设备类型，支持CPU和GPU设备默认值："cuda"可选值："cpu", "cuda", "cuda:0", "cuda:1"等| |sample_mode|str|by_count_uniform|采样模式，参考 VideoFrameSampler| |start_time_sec|float|0.0|采样起始时间（秒）| |end_time_sec|float or None||采样结束时间（秒）| |count_k|int or None|8|均匀采样帧数（by_count_uniform使用）| |interval_sec|float or None||时间间隔（秒，by_interval_time使用）| |interval_frames|int or None||解码帧间隔（by_interval_frames使用）| |target_fps|float or None||目标采样FPS（by_fps使用）| |timestamps_sec|list[float] or None||采样时间戳列表（秒，by_timestamps使用）| |max_frames|int or None||返回帧上限| |reduce_mode|str|avg|多帧聚合策略，可选 "avg"| |default_video_format|str|mp4|二进制/BASE64输入的视频格式|

输出 包含视频质量评分的浮点数组，无法评分返回null值

使用示例

```
 from __future__ import annotations

import logging
import os

import ray

import daft
from daft import col
from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_quality_score import VideoQualityScore

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    model_path = os.getenv("MODEL_PATH", "/opt/aihc/models")
    num_gpus = 1

    samples = {
        "video_path": ["https://{bucket}.bj.bcebos.com/sample.mp4"],
    }
    ds = daft.from_pydict(samples)

    constructor_kwargs = {
        "model_path": model_path,
        "clip_model_name": "openai/clip-vit-large-patch14",
        "prompt": "quality",
        "rank": 0,
        "sample_mode": "by_count_uniform",
        "count_k": 4,
        "reduce_mode": "avg",
    }

    ds = ds.with_column(
        "quality_score",
        aihc_udf(VideoQualityScore, construct_args=constructor_kwargs, num_gpus=num_gpus, batch_size=1, concurrency=1)(
            col("video_path")
        ),
    )

    ds.show()
**╭──────────────
```

```
 from __future__ import annotations

import os
import subprocess

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_convert import VideoConvert

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # 检查 ffmpeg 是否可用
    try:
        subprocess.run(["ffmpeg", "-version"], capture_output=True, check=True)
    except (subprocess.CalledProcessError, FileNotFoundError):
        print("SKIP: ffmpeg not available")
        exit(0)

    samples = {
        "input_path": [
            "/tmp/test_raw1.avi",
            "/tmp/test_raw2.mov",
        ],
        "output_path": [
            "/tmp/test_output_raw1.mp4",
            "/tmp/test_output_raw2.mp4",
        ],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_format": "mp4",
        "timeout": 120,
    }
    ds = ds.with_column(
        "convert_result",
        aihc_udf(
            VideoConvert,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=2,
            batch_size=4,
        )(col("input_path"), col("output_path")),
    )
    ds.show()
```

视频人脸脱敏

简介 视频人脸模糊处理算子

功能描述

- 自动检测视频帧中的人脸并进行模糊处理
- 支持多种模糊类型（均值、盒式、高斯）
- 支持本地路径、远程 URL/BOS 路径以及二进制/BASE64 视频输入
- 统一输出模糊后的视频路径（即使未检测到人脸也会重新编码输出）
- cpu 和 gpu 环境均可运行
- 隐私保护：对视频中的人脸信息进行脱敏处理，防止个人隐私泄露
- 数据发布前的隐私清洗：在发布视频数据前，对其中包含的人脸进行模糊处理，保护用户隐私
- 训练/评测数据集中的人脸模糊预处理：在训练/评测模型时，对数据集中的视频帧进行人脸模糊处理，提高模型泛化能力
- 人脸检测基于 InsightFace 模型，对光照、姿态变化较敏感
- 输入视频需为 PyAV/FFmpeg 可解码的常见格式（如 mp4、mov、avi 等）
- 单条样本处理失败时，该条结果返回空字符串，不影响批次内其他样本

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP 等），与 video_binaries 二选一。| |video_binaries|视频二进制数据或 BASE64 字符串列，与 video_paths 二选一。| |video_formats|视频格式字符串列（如 "mp4"、"mov"），配合 video_binaries 使用。| |output_basenames|输出文件基础名称列（不含扩展名），用于自定义输出文件名。|

输出 |输出|含义| |-|-| |result|模糊处理后的视频路径列。|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_dir|str|''|将模糊处理后的视频保存到该目录中，目前支持上传到 BOS 或本地目录。 格式："bos://bucket/path/" 或 "/local/path/" 默认值：""| |model_path|str|'/opt/aihc/models'|人脸检测模型的基础目录路径，用于 InsightFace 模型加载。 默认值："/opt/aihc/models"。| |model_name|str|'insightface'|InsightFace 模型子目录名称，用于拼接实际模型路径 model_path/model_name。 默认值："insightface"。| |blur_type|str|'gaussian'|模糊方式。 可选值：["mean", "box", "gaussian"] - "mean": 使用均值模糊（ImageFilter.BLUR） - "box": 使用盒式模糊（ImageFilter.BoxBlur） - "gaussian": 使用高斯模糊（ImageFilter.GaussianBlur） 默认值："gaussian"| |radius|float|10.0|模糊半径（盒式/高斯模糊生效）。 要求：radius >= 0，建议：根据模糊效果调整，一般 5-20 之间即可。 默认值：10.0| |det_thresh|float|0.5|人脸检测置信度阈值。隐私打码（宁可杀错不可放过）：建议设低一点，例如 0.3 - 0.4。防止漏掉人脸导致隐私泄露。精细修图：建议设高一点，例如 0.6 - 0.7，只处理正脸。 默认值：0.5| |det_size|tuple[int, int]|(640, 640)|人脸检测输入尺寸 (width, height)。普通网络视频/电影：(640, 640) 是性价比最高的默认值。监控录像/广场全景（人脸很小）：必须调大，建议 (1280, 1280) 甚至更高，否则远处的人脸会漏检。 默认值：(640, 640)|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_face_blur import VideoFaceBlur

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "video_paths": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_dir": '',
        "model_path": '/opt/aihc/models',
        "model_name": 'insightface',
        "blur_type": 'gaussian',
        "radius": 10.0,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoFaceBlur,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频区域修复

简介 视频区域修复

功能描述

- 智能修复：基于STTN时空记忆网络进行视频内容修复（去除水印、字幕等外来内容），支持设置默认修复区域
- 多区域支持：支持同时修复一个、多个指定区域或全屏处理
- 音频保留：视频修复不影响原始音频
- GPU加速：支持CUDA加速提升处理效率
- 多输入支持：支持路径输入和二进制输入
- 适用性强：支持音视频流不对齐场景
- 建议处理分辨率不超过1080p的视频
- 修复区域越小处理速度越快
- 长视频建议预先分段处理
- 时空一致性：保证修复区域的时间连续性
- 格式兼容：输出标准MP4格式视频

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用| |inpaint_areas|修复区域坐标列，List格式： [{'ymin': int, 'ymax': int, 'xmin': int, 'xmax': int}, ...] 与video_watermark_detect算子输出格式兼容 若为None则进行全屏修复| |output_basenames|输出文件基础名称列（不含扩展名） 若为None则使用原文件名加"_inpainted"后缀|

输出 |输出|含义| |-|-| |output_path|修复后视频的BOS存储路径；| |processed_frames|实际处理的视频帧数；| |processed_resolution|处理后视频的分辨率[width, height]|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_tos_dir|str|必填|BOS输出目录（必需） 修复后视频的BOS存储路径 格式："bos://bucket/path"| |model_path|str|'/opt/aihc/models'|模型存储路径 默认值："/opt/aihc/models"| |model_name|str|'researchmm/STTN'|模型名称 默认值："researchmm/STTN"| |neighbor_stride|int|5|相邻帧步长 选择参考帧的间隔距离 较小值(3-5)适合慢动作视频，较大值(8-10)适合快动作视频 默认值：5| |reference_length|int|10|参考帧数量 用于修复当前帧的参考帧总数 提供时序上下文信息，值越大修复质量越好但计算量增加 默认值：10| |max_load_num|int|50|最大加载帧数 每次处理时最多同时加载到内存的视频帧数量 控制内存使用，避免长视频导致内存溢出 约束：max_load_num >= reference_length * neighbor_stride 默认值：50| |is_keep_audio|bool|True|是否保留音频 修复视频时是否保留原始音频 默认值：True| |is_set_default_inpaint_areas|bool|False|是否设置默认修复区域 是否根据输入参数默认设置修复区域为画面最下方30% 默认值：False| |rank|int|None|GPU设备编号 指定使用的GPU设备ID（多卡环境生效） None表示自动选择可用GPU 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_sttn_inpaint import VideoSttnInpaint

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "video_paths": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'researchmm/STTN',
        "neighbor_stride": 5,
        "reference_length": 10,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoSttnInpaint,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频黑边裁剪

简介 视频黑边裁剪算子，自动检测并裁剪视频画面四周的黑色边框区域，提升画面利用率和视觉一致性。支持多种检测算法，适用于老片修复、内容上传等场景。

功能描述

- 多种检测算法：支持阈值比例、边缘检测、直方图三种黑边检测算法，适应不同视频内容
- 智能核心区域保护：自动保护视频中心区域，避免误裁剪重要内容
- 连续黑边过滤：可配置连续黑边行/列阈值，过滤掉小范围黑色区域
- 音频保留：可选择是否保留原始音频流
- 批量处理：支持对多个视频文件并行处理
- CPU环境即可运行，无需GPU
- 视频上传前的预处理（去除平台黑边）
- 老旧视频修复（去除胶片黑边）
- 内容平台适配（统一画面比例）
- 视频分析预处理（减少无效区域干扰）
- 视频压缩优化（去除冗余黑边区域）
- 帧处理仅做统计与坐标计算，不会修改原始文件
- 视频裁剪通过 FFmpeg 完成，编码格式为 H.264 (libx264)
- 原始音频流会被保留（如启用）或被丢弃
- 复杂场景（深色视频、渐变背景）可能需要调整参数以获得最佳效果

算子参数 输入 |输入|含义| |-|-| |input_path|输入视频路径（BOS路径、本地路径或HTTP URL）| |output_path|输出视频路径（BOS路径或本地路径）|

输出 |输出|含义| |-|-| |result|处理后的视频路径（BOS路径），失败时返回 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |detection_method|str|'threshold_ratio'|黑边检测算法。可选值：threshold_ratio: 阈值比例法，适用于大部分视频 edge_detection: 边缘检测法，适用于内容对比明显的视频 histogram: 直方图法，适用于渐变场景| |black_threshold|int|10|判定为黑色的像素亮度上限（0-255）。值越小判定越严格 默认值：10| |valid_pixel_ratio|float|0.1|行/列中非黑像素的最小有效比例。值越小保留区域越多 默认值：0.1| |sample_frames|int|20|用于检测的采样帧数。值越大检测越精确但速度越慢 默认值：20| |is_keep_audio|bool|True|是否保留原始音频流。True: 保留音频；False: 去除音频 默认值：True| |timeout|int | None|None|FFmpeg处理超时时间（秒），None表示不限制 默认值：None| |core_region_ratio|float|0.5|中心保护区域占视频尺寸的比例，范围 (0, 1)。该区域内不会被裁剪 默认值：0.5| |continuous_black_rows|int|3|行方向上连续黑边的最小行数阈值，少于该值不视为黑边 默认值：3| |continuous_black_cols|int|3|列方向上连续黑边的最小列数阈值，少于该值不视为黑边 默认值：3| |dark_region_brightness|int|50|判定为暗场景的亮度均值阈值。低于该值时使用更宽松的检测比例 默认值：50| |edge_sensitivity|float|1.0|边缘检测的灵敏度系数（仅在 edge_detection 算法下生效）。值越大检测越敏感 默认值：1.0|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_blackborder_crop import VideoBlackBorderCrop

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "input_path": [
            "bos://storage-aihc/test/aihc/test_data/test_video_with_black_border.mp4"
        ],
        "output_path": [
            "bos://storage-aihc/test/aihc/test_data/test_video_cropped.mp4"
        ],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "detection_method": 'threshold_ratio',
        "black_threshold": 10,
        "valid_pixel_ratio": 0.1,
        "sample_frames": 20,
        "is_keep_audio": True,
        "core_region_ratio": 0.5,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoBlackBorderCrop,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("input_path"), col("output_path")),
    )
    ds.show()
```

视频运动分计算

简介 视频运动分计算

功能描述

- 多算法支持：支持多种光流计算算法，适应不同场景和性能需求：
  - Farneback：经典稠密光流算法，精度与速度平衡。
  - TV-L1：基于全变分的光流算法，对光照变化鲁棒。
  - DIS (Dense Inverse Search)：快速光流算法，支持多种预设（ULTRAFAST, FAST, MEDIUM, accurate）
  - MEMFOF（深度学习光流）：基于深度学习的光流估计算法，精度最高
- 多指标支持：返回多种多层次计算指标，均值、中位数、95分位数、尺寸自适应均值、尺寸自适应95分位
- 密度指标: 运动密度分，反映运动区域占比的密度分。

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用|

输出 |输出|含义| |-|-| |standardized_score|标准化分数| |motion_pattern|运动模式分类| |mean_score|原始均值| |median_score|中位数| |dynamic_mean_score|尺寸自适应均值| |high_percentile_score|原始95分位| |dynamic_high_percentile_score|尺寸自适应95分位| |density_score|运动密度分| |video_resolution|视频分辨率[width, height]| |total_frames|视频总帧数| |sample_frames_count|实际采样帧数| |used_algorithm|使用的光流算法| |used_metric|使用的运动分指标| |status|处理状态|

参数 |参数名称|类型|默认值|描述| |-|-|-| |optical_flow_algorithm|str|'farneback'|稠密光流算法类型 可选值：["farneback", "tv-l1", "dis-ultrafast", "dis-fast", "dis-medium", "dis-accurate", "memfof"] 默认值："farneback"| |sample_ratio|float|0.125|视频采样帧数比例，用于平衡计算效率与精度 默认值：0.125| |mag_threshold|float|0.01|运动幅值阈值，过滤微小噪声位移（像素） 默认值：1.5| |flow_threshold|float|6|-| |smooth_window|int|5|-| |downsample_ratio|float|1.0|帧降采样比例（0.0-1.0），1.0为原始分辨率 默认值：1.0| |high_motion_threshold|float|0.02|高动态视频判断阈值 默认值：0.02| |batch_size|int|10|批处理大小 默认值: 10| |model_path|str|'/opt/aihc/models'|模型文件存储路径（用于MEMFOF） 默认值: "/opt/aihc/models"| |algorithm_params|dict[str, Any]|None|算法特定参数字典 默认值: None| |rank|int|0|指定使用的GPU设备编号（多卡环境有效） 默认值: 0| |num_workers|int|4|多线程工作进程数 默认值: 4| |use_cuda|bool|True|是否启用CUDA加速 默认值: True| |max_gpu_memory|float|0.8|GPU最大内存使用率限制（0.0-1.0） 默认值: 0.8| |min_frame_size|int|32|最小帧尺寸（宽/高），低于此值会被缩放 默认值: 32|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_motion_score import VideoMotionScore

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "video_paths": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "optical_flow_algorithm": 'farneback',
        "sample_ratio": 0.125,
        "mag_threshold": 0.01,
        "flow_threshold": 6,
        "smooth_window": 5,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoMotionScore,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频首帧提取

简介 视频起始帧识别

功能描述

- 从视频中提取第一帧作为封面图片
- 支持多种图片格式输出
- 高效处理，只读取视频头部信息
- 支持路径输入、二进制输入和BOS输出
- 输入：MP4 (.mp4), AVI (.avi), MOV (.mov), MKV (.mkv)等常见视频格式
- 输出：JPEG (.jpg), PNG (.png), BMP (.bmp)等常见图片格式

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用| |output_basenames|输出文件基础名称列（不含扩展名）|

输出 |输出|含义| |-|-| |result|提取的起始帧图片路径列|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_tos_dir|str|''|将提取的起始帧图片保存到该BOS目录中，如果为空则不保存到BOS。 格式："bos://bucket/path/" 默认值："" | |output_format|str|'jpg'|输出图片的格式。 可选值：["jpg", "jpeg", "png", "bmp"] 默认值："jpg" | |rank|int|None|指定使用的GPU设备编号（多卡环境有效）。 说明：0表示第一张GPU，1表示第二张GPU，None表示自动选择 适用：仅在GPU处理时有效 默认值：None| |skip_black_frames|bool|False|是否跳过黑屏帧，提取第一个非黑屏帧。 默认值：False| |black_threshold|float|0.1|黑屏检测阈值（像素级阈值），值越小检测越严格。 默认值：0.1|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_first_frame import VideoFirstFrame

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_tos_dir": '',
        "output_format": 'jpg',
        "rank": None,
        "skip_black_frames": False,
        "black_threshold": 0.1,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoFirstFrame,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频移除音轨

简介 视频音轨移除处理器

功能描述

- 使用 ffmpeg subprocess 方式移除视频音轨
- 保留原始视频编码和质量（无需重编码）
- 自动下载远程文件 / 上传结果
- 支持超时控制
- 支持多种视频格式

算子参数 输入 |输入|含义| |-|-| |input_col|输入视频路径列| |output_col|输出视频路径列|

输出 |输出|含义| |-|-| |result|输出路径的 PyArrow Array|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-| |output_format|str|None|None|输出视频格式，如"mp4"、"avi"、"mkv"，为None时保持原格式 默认值：None| |extra_params|list[str]|None|None|额外的ffmpeg参数列表，如["-preset", "fast"] 默认值：None| |timeout|int|None|None|ffmpeg执行超时时间（秒），默认为None（无超时）|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_remove_audio import VideoRemoveAudio

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"input_col": [...], "output_col": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_format": None,
        "extra_params": None,
        "timeout": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoRemoveAudio,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("input_col"), col("output_col")),
    )
    ds.show()
```

视频清晰度评分

简介 视频清晰度计算处理器，支持多种清晰度评估方法和抽帧策略。

功能描述

- 视频抽帧：支持基于FPS的抽帧策略
- 拉普拉斯方差法（laplacian） - 基于二阶导数的边缘检测
- Tenengrad法（tenengrad） - 基于Sobel梯度的边缘强度
- Brenner法（brenner） - 基于相邻像素灰度差的简单计算
- FFT高频能量法（fft_highfreq） - 基于频域分析的清晰度评估
- 池化功能：返回所有池化方法的结果（均值、中位数、最大值、最小值、标准差）
- URL地址
- 二进制流

算子参数 输入 |输入|含义| |-|-| |video_paths|输入视频路径列，类型为数组。 默认值：None| |video_binaries|输入视频二进制数据列，类型为数组。 默认值：None| |video_formats|输入视频格式列（如 'mp4'、'avi' 等），类型为数组。 默认值：None|

输出 |输出|含义| |-|-| |mean|清晰度均值| |median|清晰度中位数| |max|清晰度最大值| |min|清晰度最小值| |std|清晰度标准差|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |fps|float|2.0|每秒抽帧数量，默认为2.0 描述: 从视频中每秒抽取的帧数，用于计算清晰度 默认值: 2.0| |method|str|'laplacian'|清晰度计算方法 描述: 清晰度计算方法，支持laplacian(拉普拉斯方差)、tenengrad(Tenengrad梯度)、brenner(Brenner梯度)、fft_highfreq(FFT高频能量)，默认为 laplacian 可选值: ["laplacian", "tenengrad", "brenner", "fft_highfreq"] 默认值: "laplacian"| |max_frames|int|100|最大抽帧数量，默认为100 描述: 限制抽帧的最大数量，防止处理长视频时陷入无限循环 默认值: 100|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_sharpness import VideoSharpness

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "fps": 2.0,
        "method": 'laplacian',
        "max_frames": 100,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoSharpness,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频宽高比调整

简介 视频宽高比调整处理器

功能描述

- 支持将视频调整到指定宽高比的画布尺寸
- 支持两种策略：padding(留边) 或 crop(裁剪)
- 支持本地/远端路径输入与输出路径指定

算子参数 输入 |输入|含义| |-|-| |input_col|-| |output_col|-|

输出 |输出|含义| |-|-| |result|算子输出|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |target_width|int|必填|输出视频宽度（像素）| |target_height|int|必填|输出视频高度（像素）| |mode|str|'pad'|处理模式，可选 "pad" 或 "crop"| |video_codec|str|'libx264'|视频编码器，默认 libx264| |audio_codec|str|'aac'|音频编码器，默认 aac| |extra_params|list[str]|None|额外 ffmpeg 参数列表| |timeout|int|None|ffmpeg 执行超时时间（秒）|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_aspect_ratio_adjust import VideoAspectRatioAdjust

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"input_col": [...], "output_col": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "mode": 'pad',
        "video_codec": 'libx264',
        "audio_codec": 'aac',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoAspectRatioAdjust,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("input_col"), col("output_col")),
    )
    ds.show()
```

视频按时间戳切分

简介 视频时间戳切分处理器，支持按指定时间范围分割

功能描述

- 按给定的时间戳区间切分视频
- 支持片段二进制输出或BOS存储
- 提供格式自动推断与自定义
- MP4 (.mp4)
- AVI (.avi)
- MOV (.mov)
- MKV (.mkv)
- 其他常见视频格式

算子参数 输入 |输入|含义| |-|-| |video_paths|包含输入视频路径的数组 默认值：None| |video_binaries|包含视频二进制数据的数组 默认值：None| |video_formats|包含输入视频格式（如 'mp4'、'avi' 等）的数组，指定 video_binaries 时可以提供格式信息 默认值：None| |timestamp_ranges|每行对应的时间戳范围列表，支持格式： - [(start,end), (start,end)] 或 [[start,end], [start,end]] - 单个 (start,end) 或 [start,end]| |output_basenames|可选，输出子目录名（文件名）数组|

输出 |输出|含义| |-|-| |segments|list[str]，切分后视频片段的路径列表| |segments_binary|list[bytes]，切分后视频片段的二进制数据列表|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_tos_dir|str|''|保存视频片段的BOS路径，若为空字符串则不上传 默认值：""| |output_segments_binary|bool|False|是否输出视频片段的二进制数据 默认值：False| |output_video_format|str|None|全局指定输出视频格式（如"mp4"、"avi"等），优先级高于输入文件后缀和video_format列|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_split_by_timestamps import VideoSplitByTimestamps

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_tos_dir": '',
        "output_segments_binary": False,
        "output_video_format": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoSplitByTimestamps,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频拼接

简介 视频拼接处理器，支持将多个视频文件拼接成一个视频

功能描述

- 使用 ffmpeg concat filter 拼接视频并重编码
- 支持多输入格式（本地/BOS/HTTP）
- 支持输出到指定路径
- 可配置视频/音频编码参数

算子参数 输入 |输入|含义| |-|-| |video_paths_list|包含视频路径列表的列，每个元素是一个字符串列表| |output_col|输出视频路径列|

输出 |输出|含义| |-|-| |result|算子输出|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_format|str|'mp4'|输出视频格式（仅用于临时文件扩展名）| |video_codec|str|'libx264'|视频编码器| |audio_codec|str|'aac'|音频编码器| |timeout|int|None|ffmpeg执行超时时间（秒）| |extra_params|list[str]|None|额外的ffmpeg参数列表|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_concat import VideoConcat

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths_list": [...], "output_col": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_format": 'mp4',
        "video_codec": 'libx264',
        "audio_codec": 'aac',
        "timeout": None,
        "extra_params": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoConcat,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths_list"), col("output_col")),
    )
    ds.show()
```

视频水印检测

简介 视频水印检测

功能描述

- 智能检测：基于一致性分析识别视频中的固定文本水印区域
- 采样优化：通过智能采样策略提升检测效率和准确性
- 格式兼容：输出格式与视频修复算子完全兼容
- CPU处理：当前版本仅支持CPU模式处理
- 多输入支持：支持路径输入和二进制输入
- 建议处理分辨率不超过1080p的视频
- 采样帧数可根据视频长度调整
- 一致性阈值建议保持在0.7-0.9之间
- 位置一致性：检测在多帧中位置稳定的文本区域
- 坐标标准化：输出标准化的区域坐标格式
- 检测加去除：与视频修复算子组合使用，实现水印去除

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用|

输出 |输出|含义| |-|-| |watermark_regions|检测到的水印区域列表| |video_resolution|视频分辨率[width, height]| |total_frames|视频总帧数|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|'/opt/aihc/models'|模型文件存储的根目录路径 默认值："/opt/aihc/models"| |model_name|str|'PP-OCRv4/ch_det'|使用的OCR检测模型名称 默认值："PP-OCRv4/ch_det"| |sample_count|int|15|从视频中采样的帧数，用于水印一致性检测 默认值：15| |consistency_threshold|float|0.8|水印位置一致性阈值，范围0-1，值越高要求越严格 默认值：0.8| |position_tolerance|int|15|位置容差像素值，用于判断多帧间文本位置是否一致 默认值：15|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_watermark_detect import VideoWatermarkDetect

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "video_paths": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "model_path": '/opt/aihc/models',
        "model_name": 'PP-OCRv4/ch_det',
        "sample_count": 15,
        "consistency_threshold": 0.8,
        "position_tolerance": 15,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoWatermarkDetect,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频裁剪

简介 视频裁剪

功能描述

- 支持多种裁剪模式：指定分辨率、宽高比、中心正方形、横图、竖图
- 支持基于边界框(bbox)的精确裁剪
- 可控制视频质量和编码参数
- 保持音频流不受影响
- 支持路径输入、二进制输入和BOS输出
- MP4 (.mp4)
- AVI (.avi)
- MOV (.mov)
- MKV (.mkv)
- 其他常见视频格式
- CPU编码 (libx264)**：质量优先，压缩效率高，速度较慢
- GPU编码 (h264_nvenc)**：速度快，适合批量处理，质量略低于libx264（相同码率下）
- 边界框(bbox) > 目标分辨率(target_width/target_height) > 宽高比(aspect_ratio)

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用| |output_basenames|输出文件基础名称列（不含扩展名）|

输出 |输出|含义| |-|-| |result|裁剪后的视频路径列|

参数 |参数名称|类型|默认值|描述| |-|-|-| |output_tos_dir|str|''|将裁剪后的视频保存到该BOS目录中，如果为空则不保存。 格式："bos://bucket/path/" 默认值："""| |target_width|int|None|目标视频宽度（像素）。 与target_height配合使用，指定具体分辨率 优先级：bbox > target_width/target_height > aspect_ratio 默认值：None| |target_height|int|None|目标视频高度（像素）。 与target_width配合使用，指定具体分辨率 优先级：bbox > target_width/target_height > aspect_ratio 默认值：None| |aspect_ratio|float|None|目标宽高比。 格式：宽度/高度（如16/9=1.7778） 优先级：bbox > target_width/target_height > aspect_ratio 默认值：None| |bbox|tuple[int, int, int, int]|None|边界框参数，格式为(x, y, width, height)。 x, y: 裁剪区域左上角坐标 width, height: 裁剪区域的宽度和高度 优先级高于target_width/target_height和aspect_ratio 注意：bbox参数会覆盖其他裁剪参数 默认值：None| |crop_mode|str|'center'|裁剪模式。 center: 中心裁剪 top: 顶部裁剪 bottom: 底部裁剪 left: 左侧裁剪 right: 右侧裁剪 可选值：["center", "top", "bottom", "left", "right"] 仅在不使用bbox参数时生效 默认值："center"| |force_divisible_by|int|2|像素对齐步长，确保宽高能被该值整除。 默认值：2| |crf|float|23.0|libx264编码器的恒定质量因子。 适用：仅在CPU编码(libx264)时使用 范围：0.0-51.0，数值越小质量越高文件越大 推荐：18(高质量) 23(平衡) 28(压缩优先) 默认值：23.0| |preset|str|'medium'|libx264编码器的编码速度预设。 适用：仅在CPU编码(libx264)时使用 权衡：速度 ↔ 压缩效率 可选值：["ultrafast", "superfast", "veryfast", "faster", "fast", "medium", "slow", "slower", "veryslow"] 推荐："medium"(平衡) "fast"(速度优先) "slow"(质量优先) 默认值："medium"| |cq|float|0|NVENC编码器的质量控制参数。 适用：仅在GPU编码(h264_nvenc)时使用 范围：0-51，0表示自动质量控制 推荐：0(自动) 或 18-28(手动控制) 默认值：0| |rc|str|'vbr'|NVENC编码器的码率控制模式。 适用：仅在GPU编码(h264_nvenc)时使用 constqp: 恒定量化参数，质量稳定 vbr: 变码率，平衡质量和文件大小 cbr: 恒定码率，适合流媒体传输 推荐："vbr"(通用) "cbr"(直播) 默认值："vbr"| |rank|int|None|指定使用的GPU设备编号（多卡环境有效）。 说明：0表示第一张GPU，1表示第二张GPU，None表示自动选择 适用：仅在GPU编码时有效 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_crop import VideoCrop

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_tos_dir": '',
        "target_width": None,
        "target_height": None,
        "aspect_ratio": None,
        "bbox": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoCrop,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频提取音频

简介 视频音频抽取处理器，支持多流分离

功能描述

- 支持从本地、BOS、HTTP等多种路径或二进制输入的视频中抽取音频流
- 支持多音频流选择、只取第一个流或全部流
- 支持输出音频到BOS、返回二进制、采样率等
- 支持抽取指定时间区间（start_second, end_second）
- 所有输出音频均为 mp3 格式

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径数组（本地、BOS、HTTP等）| |video_binaries|视频二进制数据数组（可选）| |video_formats|视频格式字符串数组（可选）| |output_basenames|可选，输出子目录名（文件名）数组|

输出 |输出|含义| |-|-| |audio_paths|包含音频 BOS 路径的列| |audio_arrays|包含音频 array 数据的列（可选）| |binaries|包含音频二进制数据的列（可选）| |original_audio_sampling_rates|始终返回，包含每个音频流的原始采样率，和audio_paths/binaries一一对应|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_tos_dir|str|''|将从视频中抽取出的音频保存到该 BOS 目录中，如果为空，则不保存音频| |output_audio_binary|bool|False|是否返回音频的二进制内容，默认为 False| |output_audio_array|bool|False|是否返回音频的 numpy array 格式，默认为 False| |stream_indexes|list[int]|None|指定抽取哪些音频流，默认全部。超出范围的索引会被自动忽略| |return_first_stream|bool|True|如果为 True，则只返回第一个音频流，否则返回所有流的列表，默认为 True| |start_second|float|None|从视频的第几秒开始抽取音频，None 表示从头开始，单位为秒| |end_second|float|None|到视频的第几秒结束抽取音频，None 表示到结尾，单位为秒| |output_format|str|'mp3'|输出音频格式，默认 "mp3"| |output_sample_rate|int|48000|输出音频采样率，默认 48000|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_extract_audio import VideoExtractAudio

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_tos_dir": '',
        "output_audio_binary": False,
        "output_audio_array": False,
        "stream_indexes": None,
        "return_first_stream": True,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoExtractAudio,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频自适应关键帧抽取

简介 视频自适应关键帧抽取处理器，使用CLIP和VideoAdaptiveKeyframesSampling算法进行智能抽帧

功能描述

- 基于CLIP的语义理解：利用CLIP模型计算视频帧与文本的语义相似度
- VideoAdaptiveKeyframesSampling自适应分段算法：动态调整关键帧分布，确保重要帧不丢失
- 批量处理：支持高效的批量视频处理
- 多格式支持：兼容多种视频输入格式
- 灵活配置：支持自定义采样率、关键帧数量、模型参数等
- 输入：支持视频URL、BOS地址、本地文件路径等
- 输出：包含关键帧信息的字典数组
- 视频格式：MP4、AVI、MOV等常见视频格式
- 视频帧采样：按指定帧率采样视频帧
- CLIP特征提取：提取视频帧和文本的特征向量
- 相似度计算：计算帧与文本的余弦相似度
- VideoAdaptiveKeyframesSampling关键帧选择：基于相似度分数进行自适应关键帧选择

算子参数 输入 |输入|含义| |-|-| |videos|输入视频列，类型依据 video_src_type（url/base64/binary）。|

输出 |输出|含义| |-|-| |result|pyarrow.Array: 包含关键帧信息的字典数组|

参数 |参数名称|类型|默认值|描述| |-|-|-| |text|str|''|用于计算语义相似度的文本描述。 默认值："" |model_path|str|'/opt/aihc/models'|模型基础路径。如果为空，则从Hugging Face Hub下载模型。 默认值：'' |clip_model_name|str|'openai/clip-vit-base-patch32'|CLIP模型名称/子目录。 默认值：'openai/clip-vit-base-patch32' |dtype|str|'float16'|模型推理精度选择： - float16: 更快推理 - float32: 更高精度 可选值：["float16", "float32"] 默认值："float16" |video_src_type|str|'video_url'|输入视频的格式类型，支持： - 路径/URL（video_url） - base64 编码（video_base64） - 二进制流（video_binary） 可选值: ["video_url", "video_base64", "video_binary"] 默认值: "video_url" |fps|float|1.0|候选帧采样率（按fps均匀采样）。 默认值：1.0| |max_num_frames|int|32|VideoAdaptiveKeyframesSampling最终要选出的关键帧数。 默认值：32| |t1|float|0.8|判断是否有明显峰值的阈值（mean_top - mean）。 默认值：0.8| |t2|float|-100|判断是否继续分割的std阈值。 默认值：-100| |all_depth|int|5|最大递归深度。 默认值：5| |img_type|str|'.jpg'|输出关键帧图片格式，支持 ".jpg"、".png"。 可选值：[".jpg", ".png"] 默认值：".jpg"| |output_tos_dir|str|''|保存关键帧图片到 BOS 的目标路径，若为空字符串则不上传。 默认值："" |return_keyframes|bool|True|是否返回关键帧图片的array数据。 默认值：True| |batch_size|int|16|批量推理的帧数量。 默认值：16| |rank|int|0|GPU编号。 默认值：0| |video_format|str|'mp4'|二进制/base64输入的视频格式（如mp4、mov）。 默认值："mp4"|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_adaptive_keyframes_sampling import VideoAdaptiveKeyframesSampling

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "videos": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "text": '',
        "model_path": '/opt/aihc/models',
        "clip_model_name": 'openai/clip-vit-base-patch32',
        "dtype": 'float16',
        "video_src_type": 'video_url',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoAdaptiveKeyframesSampling,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("videos")),
    )
    ds.show()
```

视频分辨率调整

简介 视频分辨率调整

功能描述

- 智能分辨率调整到指定范围内
- 支持多种宽高比保持策略
- 可控制视频质量和编码参数
- 保持音频流不受影响
- 支持路径输入、二进制输入和BOS/本地输出
- MP4 (.mp4)
- AVI (.avi)
- MOV (.mov)
- MKV (.mkv)
- 其他常见视频格式
- CPU编码 (libx264)**：质量优先，压缩效率高，速度较慢
- GPU编码 (h264_nvenc)**：速度快，适合批量处理，质量略低于libx264（相同码率下），压缩效率稍差，文件可能更大

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用| |output_basenames|输出文件基础名称列（不含扩展名）|

输出 |输出|含义| |-|-| |result|调整后的视频路径列|

参数 |参数名称|类型|默认值|描述| |-|-|-| |output_tos_dir|str|''|将分辨率调整后的视频保存到该BOS或者本地目录中，必填。 格式："bos://bucket/path/"，或者本地目录路径 默认值：""| |min_width|int|1280|视频最小宽度，小于该值时将被调整。 单位：像素 默认值：1280| |max_width|int|2560|视频最大宽度，大于该值时将被调整。 单位：像素 默认值：2560| |min_height|int|1280|视频最小高度，小于该值时将被调整。 单位：像素 默认值：1280| |max_height|int|2560|视频最大高度，大于该值时将被调整。 单位：像素 默认值：2560| |force_original_aspect_ratio_type|str|'disable'|宽高比保持策略。 disable: 不强制保持原始宽高比，可能会拉伸变形 increase: 保持宽高比，调整到大于等于目标尺寸 decrease: 保持宽高比，调整到小于等于目标尺寸 可选值：["disable", "increase", "decrease"] 默认值："disable"| |force_divisible_by|int|2|像素对齐步长，确保宽高能被该值整除。 默认值：2| |crf|float|23.0|libx264编码器的恒定质量因子。 适用：仅在CPU编码(libx264)时使用 范围：0.0-51.0，数值越小质量越高文件越大 推荐：18(高质量) 23(平衡) 28(压缩优先) 默认值：23.0| |preset|str|'medium'|libx264编码器的编码速度预设。 适用：仅在CPU编码(libx264)时使用 权衡：速度 ↔ 压缩效率 可选值：["ultrafast", "superfast", "veryfast", "faster", "fast", "medium", "slow", "slower", "veryslow"] 推荐："medium"(平衡) "fast"(速度优先) "slow"(质量优先) 默认值："medium"| |cq|float|0|NVENC编码器的质量控制参数。 适用：仅在GPU编码(h264_nvenc)时使用 范围：0-51，0表示自动质量控制 推荐：0(自动) 或 18-28(手动控制) 默认值：0| |rc|str|'vbr'|NVENC编码器的码率控制模式。 适用：仅在GPU编码(h264_nvenc)时使用 constqp: 恒定量化参数，质量稳定 vbr: 变码率，平衡质量和文件大小 cbr: 恒定码率，适合流媒体传输 推荐："vbr"(通用) "cbr"(直播) 默认值："vbr"| |rank|int|None|指定使用的GPU设备编号（多卡环境有效）。 说明：0表示第一张GPU，1表示第二张GPU，None表示自动选择 适用：仅在GPU编码时有效 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_resize_resolution import VideoResizeResolution

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_tos_dir": '',
        "min_width": 1280,
        "max_width": 2560,
        "min_height": 1280,
        "max_height": 2560,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoResizeResolution,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频音频流检测

简介 视频音频检测处理器

功能描述

- 使用 ffprobe 检测视频中是否存在音频流
- 自动下载远程文件
- 支持超时控制
- 支持多种视频格式
- 返回布尔值表示是否存在音频

算子参数 输入 |输入|含义| |-|-| |input_col|输入视频路径列|

输出 |输出|含义| |-|-| |result|布尔值的 PyArrow Array，True 表示存在音频，False 表示不存在，None 表示检测失败|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |timeout|int|None|ffprobe执行超时时间（秒），默认为None（无超时）|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_detect_audio import VideoDetectAudio

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "input_paths": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "timeout": None,
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoDetectAudio,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("input_col")),
    )
    ds.show()
```

视频转 MP4

简介 视频格式转换处理器，将各种视频格式转换为MP4

功能描述

- 支持多种视频格式转换为MP4
- 自动选择第一个音轨
- 视频质量和编码参数自定义
- 支持视频高度限制和缩放
- 音频编码参数精细控制
- 输入：AVI、MOV、MKV、FLV、WMV、3GP等主流视频格式
- 输出：MP4 (.mp4)
- 视频编解码器：H.264/H.265等
- 音频编解码器：AAC、MP3等

算子参数 输入 |输入|含义| |-|-| |input_col|包含输入视频路径的数组（支持本地路径、HTTP/HTTPS URL、BOS/S3 URL）| |output_col|包含输出MP4文件路径的数组|

输出 |输出|含义| |-|-| |result|pa.Array: 包含转换结果路径的数组，成功返回输出路径，失败返回None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_codec|str|'libx264'|视频编码器，支持libx264、libx265等 默认值："libx264"| |crf|int|23|视频质量控制，取值范围0-51，越小质量越好 默认值：23| |preset|str|'medium'|编码速度预设，支持ultrafast、superfast、veryfast、faster、fast、medium、slow、slower、veryslow 默认值："medium"| |max_height|int|None|视频最大高度限制，超过时自动缩放，为None时不限制 默认值：None| |audio_codec|str|'aac'|音频编码器，支持aac等 默认值："aac"| |audio_bitrate|str|'192k'|音频码率，如"192k"、"128k" 默认值："192k"| |audio_sample_rate|int|None|音频采样率，如44100。48000，为None时保持原始采样率 默认值：None| |extra_params|list[str]|None|额外的ffmpeg参数列表，如["-movflags", "+faststart"] 默认值：None| |timeout|int|None|单个视频处理超时时间（秒），为None时不限制 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_convert_to_mp4 import VideoConvertToMp4

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {
        "input_paths": ["bos://bucket/test/aihc/test_data/test_video.mp4"],
        "output_paths": ["bos://bucket/test/aihc/test_data_outputs/converted.mp4"],
    }
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "video_codec": 'libx264',
        "crf": 23,
        "preset": 'medium',
        "max_height": None,
        "audio_codec": 'aac',
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoConvertToMp4,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("input_col"), col("output_col")),
    )
    ds.show()
```

视频自适应压缩

简介 视频自适应压缩

功能描述

- 根据目标文件大小自适应压缩视频
- 多级压缩策略：帧率调整 -> 分辨率调整 -> 码率控制
- 保持视频质量的前提下尽可能压缩文件大小
- 支持CPU和GPU编码
- 支持路径输入、二进制输入和BOS输出
- MP4 (.mp4)
- AVI (.avi)
- MOV (.mov)
- MKV (.mkv)
- 其他常见视频格式
- CPU编码 (libx264)**：质量优先，压缩效率高，速度较慢，成本低
- GPU编码 (h264_nvenc)**：速度快，成本相对高

算子参数 输入 |输入|含义| |-|-| |video_paths|视频文件路径列（本地、BOS、HTTP等），与video_binaries二选一| |video_binaries|视频二进制数据列，与video_paths二选一| |video_formats|视频格式字符串列，配合video_binaries使用| |output_basenames|输出文件基础名称列（不含扩展名）|

输出 |输出|含义| |-|-| |result|压缩后的视频路径列|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |output_tos_dir|str|''|将压缩后的视频保存到该BOS目录中，如果为空则不保存。 格式："bos://bucket/path/" 默认值："'| |max_output_size_mb|float|50.0|最大输出文件大小（MB）。 默认值：50.0| |target_fps|float|5.0|帧率调整的目标值。 默认值：5.0| |min_resolution_height|int|360|最小分辨率高度（像素），保持宽高比。 默认值：360| |allowed_formats|list[str]|['mp4', 'avi', 'mov']|当视频大小满足时，允许直接输出的视频格式列表。 默认值：["mp4", "avi", "mov"]| |rank|int|None|指定使用的GPU设备编号（多卡环境有效）。 默认值：None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_adaptive_compress import VideoAdaptiveCompress

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # TODO: 根据实际场景准备样本数据
    samples = {"video_paths": [...], "video_binaries": [...]}
    ds = daft.from_pydict(samples)
    constructor_kwargs = {
        "output_tos_dir": '',
        "max_output_size_mb": 50.0,
        "target_fps": 5.0,
        "min_resolution_height": 360,
        "allowed_formats": ['mp4', 'avi', 'mov'],
    }
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoAdaptiveCompress,
            construct_args=constructor_kwargs,
            num_cpus=1,
            concurrency=4,
            batch_size=8,
        )(col("video_paths"), col("video_binaries"), col("video_formats")),
    )
    ds.show()
```

视频训练适用性评分（VTSS）

简介 视频训练适宜性评分（VTSS）算子，是 SILA 论文 VTSS 的冷启动版（VTSS 权重未开源）。用 CLIP-IQA + Farneback 光流 + Laplacian 清晰度组合规则打 0-9 综合分。用于 SILA §5 步骤 ④ 训练适宜性过滤。

功能描述 • 组合公式：Q = w_iqa × CLIP_IQA + w_motion × motion_norm + w_sharp × sharpness_norm，再线性映射到 [0, 9]。

• CLIP-IQA 用 "a high quality photo" vs "a low quality photo" prompt 对求高质量概率；motion 用 Farneback 光流平均模长归一化；sharpness 用 Laplacian 方差归一化。

• 三项权重须和为 1.0；motion/sharpness 用 min/max clip 归一化到 [0,1]。

• CLIP 权重需预置于 `{model_path}/{model_name}`，不自动下载；支持 GPU/CPU 自动选择，异常时按 `fail_on_error` 决定抛异常或返回 None。

算子参数 输入 |输入|含义| |-|-| |videos|视频（URL/base64/binary，由 `video_src_type` 决定），单个输入列|

输出 |输出|含义| |-|-| |result| `pa.float32()`，训练适宜性综合分，取值 0-9；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 "video_url"/"video_base64"/"video_binary"| |model_path|str|"/opt/aihc/models"|CLIP 权重根目录| |model_name|str|"openai/clip-vit-base-patch16"|CLIP 权重子目录（用于 CLIP-IQA 打分）| |num_frames|int|8|抽帧数量| |w_iqa|float|0.4|CLIP-IQA 项权重，三权重需和为 1.0| |w_motion|float|0.3|运动项权重，三权重需和为 1.0| |w_sharp|float|0.3|清晰度项权重，三权重需和为 1.0| |motion_min|float|0.05|Farneback 光流平均模长归一化 min clip| |motion_max|float|6.0|Farneback 光流平均模长归一化 max clip| |sharp_min|float|30.0|Laplacian 方差归一化 min clip| |sharp_max|float|800.0|Laplacian 方差归一化 max clip| |rank|int|0|多卡 rank| |fail_on_error|bool|False|单行出错是否立即抛异常，默认 False 返回 None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.vtss_video_training_suitability import (
    VTSSVideoTrainingSuitability,
)

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            VTSSVideoTrainingSuitability,
            construct_args={
                "video_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "openai/clip-vit-base-patch32",
                "num_frames": 8,
                "w_iqa": 0.4,
                "w_motion": 0.3,
                "w_sharp": 0.3,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频美学评分（DOVER）

简介 DOVER 视频美学质量评分算子，复用 DOVER 官方美学分支（aesthetic branch），对输入视频输出 0-9 分的美学质量分数。用于 SILA §5 步骤 ④ 视频美学质量过滤。

功能描述 • 从视频中按 num_clips × num_frames_per_clip 均匀采样帧，送入 DOVER 美学分支推理。

• 输出 pa.float32() 0-9 分（DOVER 原生 0-1 输出经 sigmoid 标准化后线性映射到 0-9，对齐 SILA 论文口径）。

• 支持 GPU/CPU 自动选择，支持 float16/bfloat16/float32 多种精度。

• 权重需预置于 {model_path}/{model_name}，不自动下载；单行推理失败时按 fail_on_error 决定抛异常或返回 None。

算子参数 输入 |输入|含义| |-|-| |videos|视频（URL/base64/binary，由 video_src_type 决定），单个输入列|

输出 |输出|含义| |-|-| |result| pa.float32()，视频美学质量分数，取值 0-9；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 "video_url"/"video_base64"/"video_binary"| |model_path|str|"/opt/aihc/models"|DOVER 权重根目录| |model_name|str|"dover/dover-aesthetic"|DOVER 美学分支权重子目录（相对 model_path）| |num_clips|int|1|采样 clip 数（对齐 DOVER 默认 1×32）| |num_frames_per_clip|int|32|每个 clip 采样的帧数| |dtype|str|"float32"|推理精度，可选 "float16"/"bfloat16"/"float32"| |rank|int|0|多卡 rank| |fail_on_error|bool|False|单行出错是否立即抛异常，默认 False 返回 None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.dover_video_aesthetic import DOVERVideoAesthetic

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            DOVERVideoAesthetic,
            construct_args={
                "video_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "dover/dover-aesthetic",
                "num_clips": 1,
                "num_frames_per_clip": 32,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频相机标定（静态_MoGe）

简介 静态相机内参 / FOV 标定算子，基于 MoGe-2 对静态相机视频估计相机内参矩阵 K(3x3) 与水平 / 垂直 FOV（比 DeepCalib 更准）。

功能描述 • 从视频用 opencv 均匀抽取 `frame_num` 帧，逐帧用 MoGe-2 推理归一化内参并还原到像素单位。

• 对逐帧内参取中位数作为代表内参，并据此换算水平 / 垂直 FOV（弧度）。

• 输出包含中位数内参、逐帧内参、hfov/vfov、宽高的结构体。

• 支持 GPU/CPU 自动选择；推理异常时按 `fail_on_error` 决定抛异常或返回空内参结果。

算子参数 输入 |输入|含义| |-|-| |video|视频（本地路径 / bos 路径 / http url，经 run_on_local_path 落地），单个输入列|

输出 |输出|含义| |-|-| |result|`struct{intrinsics: list<list<float64>>(3x3), hfov: float64, vfov: float64, width: int32, height: int32, per_frame_intrinsics: list<list<list<float64>>>}`；intrinsics 为逐帧内参中位数（像素单位），hfov/vfov 单位为弧度|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|"/opt/aihc/models"|模型所在目录| |model_name|str|"moge-2-vitl"|MoGe-2 所在路径| |frame_num|int|3|均匀采样帧数，须 >= 1| |fail_on_error|bool|False|推理异常时是否立即抛出；默认 False 返回空内参|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_camera_calibration_static_moge import (
    VideoCameraCalibrationStaticMoge,
)

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoCameraCalibrationStaticMoge,
            construct_args={
                "model_path": MODEL_PATH,
                "model_name": "moge-2-vitl"
                "frame_num": 3,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频按场景切分

简介 视频按场景切分算子，用 scenedetect 检测场景切换并把视频切分为多个片段。

功能描述 • 支持 ContentDetector / AdaptiveDetector / ThresholdDetector 三种场景检测器，可通过 `**kwargs` 透传各检测器专属参数。

• 用 scenedetect 检测场景列表后调用 ffmpeg 切分；无场景切换时复制原视频到输出目录。

• 输出每个视频切出的片段路径列表（本地路径），可选回传到 BOS。

• 内置提交 / 成功 / 失败计数器；单行失败时按 `fail_on_error` 决定抛异常或返回空列表。

算子参数 输入 |输入|含义| |-|-| |video|视频（bos/http url、base64 或 binary，由 `video_src_type` 决定），单个输入列|

输出 |输出|含义| |-|-| |result|`pa.list_(pa.large_string())`，每个视频切出的片段路径列表（本地或 BOS）|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 "video_url"/"video_base64"/"video_binary"| |detector|str|"ContentDetector"|场景检测器，可选 "ContentDetector"/"AdaptiveDetector"/"ThresholdDetector"| |threshold|float|27.0|场景检测阈值| |min_scene_len|int|15|最小场景长度（帧）| |show_progress|bool|False|是否显示进度条| |output_dir|str|"/tmp/aihc_scene_split"|切分片段的本地输出目录| |output_bosdir|str|""|非空时片段回传到该 BOS 目录并返回 BOS 路径| |ffmpeg_extra_args|str|"-movflags frag_keyframe+empty_moov"|传给 ffmpeg 的额外参数| |fail_on_error|bool|False|单行出错是否立即抛异常，默认 False 返回空列表|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_split_by_scene import VideoSplitByScene

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
OUTPUT_DIR = "/tmp/aihc_scene_split"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoSplitByScene,
            construct_args={
                "video_src_type": "video_url",
                "detector": "ContentDetector",
                "threshold": 27.0,
                "min_scene_len": 15,
                "output_dir": OUTPUT_DIR,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频技术质量评分（DOVER）

简介 DOVER 视频技术质量评分算子，继承自 `DOVERVideoAesthetic`，只切换到 DOVER 官方 technical 分支权重，用于评估噪声、压缩失真、抖动等技术质量维度，输出 0-9 分。用于 SILA §5 步骤 ④ 视频技术质量过滤。

功能描述 • 从视频中按 `num_clips` × `num_frames_per_clip` 均匀采样帧，送入 DOVER 技术分支推理。

• 与 `DOVERVideoAesthetic` 共用 backbone，参数完全同构，仅加载 `technical` 分支权重。

• 输出 `pa.float32()` 0-9 分（DOVER 原生输出经 sigmoid 标准化后线性映射到 0-9）。

• 权重需预置于 `{model_path}/{model_name}`，不自动下载；单行推理失败时按 `fail_on_error` 决定抛异常或返回 None。

算子参数 输入 |输入|含义| |-|-| |videos|视频（URL/base64/binary，由 `video_src_type` 决定），单个输入列|

输出 |输出|含义| |-|-| |result|`pa.float32()`，视频技术质量分数，取值 0-9；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 "video_url"/"video_base64"/"video_binary"| |model_path|str|"/opt/aihc/models"|DOVER 权重根目录| |model_name|str|"dover/dover-technical"|DOVER 技术分支权重子目录（相对 model_path）| |num_clips|int|1|采样 clip 数（对齐 DOVER 默认 1×32）| |num_frames_per_clip|int|32|每个 clip 采样的帧数| |dtype|str|"float32"|推理精度，可选 "float16"/"bfloat16"/"float32"| |rank|int|0|多卡 rank| |fail_on_error|bool|False|单行出错是否立即抛异常，默认 False 返回 None|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.dover_video_technical import DOVERVideoTechnical

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            DOVERVideoTechnical,
            construct_args={
                "video_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "dover/dover-technical",
                "num_clips": 1,
                "num_frames_per_clip": 32,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频手部重建（HaWoR）

简介 视频 3D 手部重建算子，对视频重建双手 3D 位姿，并直接产出下游 `HandActionCompute` 可消费的 `hand_pose_seq`，闭合「人类视频 → 动作」链路。离线 GPU 运行。

功能描述 • 从视频均匀抽取 `frame_num` 帧，用 MoGe-2 估计 FoV 换算 img_focal，用 YOLO 检测器做手部检测与跟踪。

• 用 HaWoR 做逐帧 3D 手部重建，模型前向得到关节点，产出所选手（默认右手，缺则左手）逐帧 6D 位姿。

• 输出 `hand_pose_seq`（position/rotation/gripper 逐帧），rotation 为 3x3 展平(9)，gripper 为拇指-食指指尖距离（原始尺度）。

算子参数 输入 |输入|含义| |-|-| |video|视频（本地 / bos / http，经 run_on_local_path 落地），单个输入列|

输出 |输出|含义| |-|-| |result|`struct{hand_pose_seq: list<struct{position: list<float64>, rotation: list<float64>, gripper: float64}>, hand: string, num_frames: int32, fov_x: float64}`|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |hawor_model_path|str|"/opt/aihc/models"|HaWoR 权重根目录，与 `hawor_model_name` 拼接为 checkpoint 文件路径| |hawor_model_name|str|"HaWoR-weights/hawor/checkpoints/hawor.ckpt"|HaWoR checkpoint 相对路径| |hawor_config_path|str|"/opt/aihc/models"|HaWoR 配置根目录，与 `hawor_config_name` 拼接| |hawor_config_name|str|"HaWoR-weights/hawor/model_config.yaml"|HaWoR 配置文件相对路径| |hawor_detector_path|str|"/opt/aihc/models"|YOLO 检测器根目录，与 `hawor_detector_name` 拼接| |hawor_detector_name|str|"HaWoR-weights/external/detector.pt"|YOLO 检测器权重相对路径| |moge_model_path|str|"/opt/aihc/models"|MoGe-2 权重根目录，与 `moge_model_name/model.pt` 拼接| |moge_model_name|str|"moge-2-vitl"|MoGe-2 模型目录名| |hawor_repo_dir|str|"/opt/aihc/models/HaWoR-repo"|HaWoR 仓库目录，不存在时运行时 git clone| |frame_num|int|16|均匀采样帧数，须 >= 1| |thresh|float|0.2|手部检测置信度阈值| |prefer_hand|str|"right"|优先选取的手，可选 "right"/"left"| |fail_on_error|bool|False|单行出错是否立即抛异常，默认 False 返回空结果|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_hand_reconstruction_hawor import (
    VideoHandReconstructionHawor,
)


VIDEO = "bos://your-bucket/sample.mp4"  # MOCK

MODEL_PATH = "/opt/aihc/models"  # MOCK

**HaWoR checkpoint**
HAWOR_MODEL_PATH = MODEL_PATH
HAWOR_MODEL_NAME = "HaWoR-weights/hawor/checkpoints/hawor.ckpt"

**HaWoR 配置**
HAWOR_CONFIG_PATH = MODEL_PATH
HAWOR_CONFIG_NAME = "HaWoR-weights/hawor/model_config.yaml"

**YOLO 手部检测器**
HAWOR_DETECTOR_PATH = MODEL_PATH
HAWOR_DETECTOR_NAME = "HaWoR-weights/external/detector.pt"

**MoGe-2**
MOGE_MODEL_PATH = MODEL_PATH
MOGE_MODEL_NAME = "moge-2-vitl"

MANO_RIGHT_PATH = f"{MODEL_PATH}/xxx.pkl"

**HaWoR 源码仓库**
HAWOR_REPO_DIR = f"{MODEL_PATH}/HaWoR-repo"


os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")


if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()

    daft.set_execution_config(
        actor_udf_ready_timeout=6000,
        min_cpu_per_task=0,
    )

    construct_args = {
        "hawor_model_path": HAWOR_MODEL_PATH,
        "hawor_model_name": HAWOR_MODEL_NAME,
        "hawor_config_path": HAWOR_CONFIG_PATH,
        "hawor_config_name": HAWOR_CONFIG_NAME,
        "hawor_detector_path": HAWOR_DETECTOR_PATH,
        "hawor_detector_name": HAWOR_DETECTOR_NAME,
        "moge_model_path": MOGE_MODEL_PATH,
        "moge_model_name": MOGE_MODEL_NAME,
        "hawor_repo_dir": HAWOR_REPO_DIR,
        "frame_num": 16,
        "thresh": 0.2,
        "prefer_hand": "right",
        "fail_on_error": False,
    }

    samples = {
        "video": [VIDEO],
    }

    ds = daft.from_pydict(samples)

    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoHandReconstructionHawor,
            construct_args=construct_args,
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )

    ds.show()
```

视频去畸变

简介 视频去畸变算子，用标准 opencv 通用畸变模型（针孔 pinhole + 鱼眼 fisheye）对视频逐帧去畸变。兼容性优先，无 opencv-contrib 依赖，可与 AIHC 现有算子共存于统一环境。

功能描述 • 支持 pinhole（针孔，径向 + 切向畸变）与 fisheye（鱼眼）两种相机模型，覆盖绝大多数客户标定场景。

• 从输入内参 K、畸变系数（可选新内参）构建去畸变重映射表，用 cv2.remap 逐帧去畸变。

• 通过 subprocess 调系统 ffmpeg 分片编码（libx264），再 concat 合成 mp4，无 ffmpeg-python 依赖。

• 输出去畸变后视频路径，可选回传 BOS；内置计数器，单行失败时按 fail_on_error 决定抛异常或返回 None。

算子参数 输入 transform 入参顺序：(video, intrinsics, distortion_coefficients, intrinsics_new)。

输入	含义
video	视频（bos/http url、base64 或 binary，由 video_src_type 决定）
intrinsics	3x3 相机内参矩阵 K
distortion_coefficients	畸变系数；pinhole 为 [k1,k2,p1,p2(,k3)]（长度 4 或 5），fisheye 为 [k1,k2,k3,k4]（长度 4）。可选，缺省为零畸变
intrinsics_new	新相机内参（去畸变后目标内参）；可选，None 时用原 K

输出 |输出|含义| |-|-| |result| pa.large_string()，去畸变后视频的路径（本地或 BOS）；失败为 None|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 "video_url"/"video_base64"/"video_binary"| |camera_model|str|"pinhole"|相机畸变模型，可选 "pinhole"（针孔）/"fisheye"（鱼眼）| |output_video_dir|str|"/tmp/aihc_undistort"|去畸变视频的本地输出目录| |output_bosdir|str|""|非空时去畸变视频回传该 BOS 目录并返回 BOS 路径| |batch_size_each_video|int|1000|每个临时 TS 分片的帧数| |crf|int|22|FFmpeg 编码质量（值越小质量越高）| |fail_on_error|bool|False|单行失败时是否立即抛异常，默认 False 返回 None；pipeline 建议显式设 True|

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_undistort import VideoUndistort

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
OUTPUT_DIR = "/tmp/aihc_undistort"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    # 3x3 内参 + 畸变系数（示例值，真实使用替换为标定结果）
    # pinhole: distortion = [k1, k2, p1, p2(, k3)]
    samples = {
        "video": [VIDEO],
        "intrinsics": [[[300.0, 0.0, 320.0], [0.0, 300.0, 240.0], [0.0, 0.0, 1.0]]],
        "distortion_coefficients": [[-0.2, 0.05, 0.0, 0.0, 0.0]],
        "intrinsics_new": [None],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoUndistort,
            construct_args={
                "video_src_type": "video_url",
                "camera_model": "pinhole",
                "crf": 22,
                "output_video_dir": OUTPUT_DIR,
            },
            num_cpus=1,
            concurrency=1,
            batch_size=1,
        )(
            col("video"),
            col("intrinsics"),
            col("distortion_coefficients"),
            col("intrinsics_new"),
        ),
    )
    ds.show()
```

视频全身姿态估计

简介 视频全身 2D 关键点估计算子，用 DWPose（ONNX：yolox_l 检测 + dw-ll_ucoco_384 姿态）对视频抽帧估计全身关键点（body/hands/faces/foot）。离线 GPU 运行（CPU 亦可，慢）。

功能描述 • 从视频均匀抽取 frame_num 帧（BGR），用 DWPose 逐帧推理全身关键点。

• 输出逐帧的 body 与 hands 关键点（外层为帧，中层为点，内层为 [x,y]，归一化到 [0,1]）。

• 依赖 onnxruntime（GPU 版 onnxruntime-gpu）+ opencv-contrib-python 与两个 ONNX 权重（yolox_l.onnx / dw-ll_ucoco_384.onnx）。

• 支持 GPU（需 CUDAExecutionProvider）/CPU 自动选择；单帧推理失败时该帧返回空点集，权重需预置不自动下载。

算子参数 输入 |输入|含义| |-|-| |video|视频（本地 / bos / http，经 run_on_local_path 落地），单个输入列|

输出 |输出|含义| |-|-| |result| struct{num_frames: int32, body: list<list<list<float64>>>, hands: list<list<list<float64>>>}；body/hands 为逐帧关键点，内层 [x,y] 归一化到 [0,1]|

参数 |参数名称|类型|默认值|描述| |-|-|-|-| |model_path|str|"/opt/aihc/models/dwpose"|存放两个 ONNX 权重的目录| |det_onnx|str|"yolox_l.onnx"|检测 ONNX 文件名| |pose_onnx|str|"dw-ll_ucoco_384.onnx"|姿态 ONNX 文件名| |frame_num|int|3|均匀采样帧数，须 >= 1|

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_whole_body_pose_estimation import (
    VideoWholeBodyPoseEstimation,
)

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
MODEL_PATH = "/opt/aihc/models/dwpose"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoWholeBodyPoseEstimation,
            construct_args={
                "model_path": MODEL_PATH,
                "det_onnx": "yolox_l.onnx",
                "pose_onnx": "dw-ll_ucoco_384.onnx",
                "frame_num": 3,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频二元瑕疵标注

简介 视频二元瑕疵标注算子，基于 QwenVL 多模态大模型 + 结构化 JSON prompt 对视频做多标签二值打标（每个 artifact 标签输出 0/1），是 SILA 自训 tagger 的冷启动版。用于 SILA §5 步骤 ④ 100 binary tags 过滤。

功能描述 • 默认给出 20 个高价值 artifact 标签（split_screen / rotation / static_camera / text_overlay / motion_blur 等），可通过`tags` 参数扩展到 100 个。

• 用 Qwen2.5-VL 对视频抽帧推理，要求模型输出一行 JSON（key 为标签名、value 为 0/1），再解析为标签 → 布尔映射。

• 支持 vLLM / transformers 双后端，`backend="auto"` 时按 GPU 算力自动选择（sm_120 及以上回退 transformers）。

• 需 CUDA GPU，权重需预置于 `{model_path}/{model_name}`，不自动下载；解析失败或异常时返回全 False。

算子参数 输入 |输入|含义| |-|-| |videos|视频（URL/base64/binary，由 `video_src_type` 决定），单个输入列|

输出 |输出|含义| |-|-| |result|`pa.map_(pa.string(), pa.bool_())`，标签名 → 0/1 的映射|

参数 |参数名称|类型|默认值|描述|| |-|-|-|-|-| |video_src_type|str|"video_url"|视频输入类型，可选 "video_url"/"video_base64"/"video_binary"|| |model_path|str|"/opt/aihc/models"|模型权重根目录|| |model_name|str|"Qwen/Qwen2.5-VL-7B-Instruct"|VLM 权重子目录（相对 model_path）|| |tags|list[str] |None|None|artifact 标签列表；None 时使用内置 20 个默认标签，须非空| |dtype|str|"auto"|推理精度，"auto" 时按 bf16 支持情况选 bfloat16/float16|| |max_model_len|int|8000|模型最大上下文长度|| |max_pixels|int |None|1280000|视频帧最大像素数| |fps|float |None|1.0|视频抽帧帧率| |tensor_parallel_size|int|1|张量并行数（vLLM）|| |gpu_memory_utilization|float|0.85|vLLM 显存占用比例|| |enforce_eager|bool|False|vLLM 是否强制 eager 模式|| |temperature|float|0.0|采样温度|| |top_p|float|0.95|采样 top_p|| |max_tokens|int|512|生成最大 token 数|| |seed|int|42|随机种子|| |backend|str|"auto"|后端，可选 "auto"/"vllm"/"transformers"|| |fail_on_error|bool|False|单行出错是否立即抛异常，默认 False 返回全 False||

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_binary_artifact_tagger import VideoBinaryArtifactTagger

VIDEO = "bos://your-bucket/sample.mp4"  # MOCK
MODEL_PATH = "/opt/aihc/models"  # MOCK

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray

        ray.init(ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {"video": [VIDEO]}
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "result",
        aihc_udf(
            VideoBinaryArtifactTagger,
            construct_args={
                "video_src_type": "video_url",
                "model_path": MODEL_PATH,
                "model_name": "Qwen/Qwen2.5-VL-7B-Instruct",
                "fps": 1.0,
                "backend": "auto",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频美学评分（帧级）

简介

视频美学评分算子：抽帧后逐帧过 LAION 美学预测器（CLIP 主干 + 线性回归头），再按 reduce_mode 聚合成一个视频级分数。与已有 DOVERVideoAesthetic（时空双分支的视频质量网络）方法不同，分值量纲不可互比。

功能描述

- 抽帧内置 VideoFrameSampler，默认 by_count_uniform + count_k=3；sample_mode 与时间窗、间隔、时间戳、max_frames 参数全部透传
- 按权重目录名选预测器实现：名字含 v1 用 AestheticsPredictorV1，含 v2 且含 linear 用 AestheticsPredictorV2Linear，含 v2 且含 relu 用 AestheticsPredictorV2ReLU，都不匹配时初始化抛 ValueError
- 归一化：normalize_by_ten=None 时，模型名含 aesthetics-predictor 就把原始 [0,10] 分除以 10，输出落在 [0,1]；显式传 True/False 可覆盖
- VideoFrameSampler 输出 BGR 帧，算子内部转 RGB 后再交给 CLIPProcessor
- 聚合支持 avg / max / min，结果保留 6 位小数
- 逐帧分数按 batch_size 微批推理，单个微批推理异常只丢这批帧并打日志，不让整列失败
- 抽帧口径由 VideoFrameSampler 的 by_count_uniform 决定：在抽帧时间窗内按时间均匀取点
- 整列抽帧失败、帧形状非法、或该视频所有帧都推理失败时，该行返回 None
- 权重只从 model_path/model_name 本地目录加载，初始化时校验目录存在，不联网下载

算子参数

输入

输入	含义
videos	视频列，内容形式由 video_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制）

输出

输出	含义
aesthetic_score	float64，逐帧美学分按 reduce_mode 聚合后的视频级分数；默认权重下已除以 10，取值 [0,1]。无有效帧或全部帧推理失败返回 None

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	视频输入类型：video_url（本地/BOS/HTTP 路径）、video_base64、video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"shunk031/aesthetics-predictor-v2-sac-logos-ava1-l14-linearMSE"	相对 model_path 的权重子目录
sample_mode	str	"by_count_uniform"	抽帧模式，透传 VideoFrameSampler：by_count_uniform / by_interval_time / by_interval_frames / by_fps / by_timestamps
start_time_sec	float	0.0	抽帧起始秒
end_time_sec	float 或 None	None	抽帧结束秒，None 表示到视频末尾
count_k	int 或 None	3	by_count_uniform 模式的抽帧数
interval_sec	float 或 None	None	by_interval_time 模式的间隔秒
interval_frames	int 或 None	None	by_interval_frames 模式的间隔帧
target_fps	float 或 None	None	by_fps 模式的目标帧率
timestamps_sec	list[float] 或 None	None	by_timestamps 模式的时间戳列表（秒）
max_frames	int 或 None	None	抽帧数上限，None 不限
reduce_mode	str	"avg"	帧分数聚合方式：avg / max / min
normalize_by_ten	bool 或 None	None	是否把分数除以 10；None 时按模型名是否含 "aesthetics-predictor" 自动判断
video_format	str	"mp4"	video_binary / video_base64 输入时的容器格式
batch_size	int	16	算子内部推理微批大小（与 aihc_udf 的 batch_size 是两个参数）
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 与 `ImageAestheticScore` 虽是同一份 LAION MLP 权重，但封装形式不同（HF 模型 vs CLIP-L14 + 裸 .pth），且本算子默认再除以 10，两者分值不能直接比较；与 `DOVERVideoAesthetic` 是不同方法，更不可比。
- 本算子只出分、不做样本过滤，`min_score` / `max_score` 之类的阈值判断在 daft 侧用 `where()` 做。
- 一次 `transform` 会先把该批次所有视频抽出的帧全部转成 PIL 图像再分批推理，峰值内存约等于 UDF `batch_size` × 单视频抽帧数 × 帧尺寸；长视频或大 `count_k` 时要压小 UDF `batch_size`。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_aesthetic_score import VideoAestheticScore

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "aesthetic_score",
        aihc_udf(
            VideoAestheticScore,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "shunk031/aesthetics-predictor-v2-sac-logos-ava1-l14-linearMSE",
                "sample_mode": "by_count_uniform",
                "count_k": 3,
                "reduce_mode": "avg",
                "batch_size": 16,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频描述生成（抽帧）

简介

视频描述生成算子（抽帧路线）：按 `all_keyframes` / `uniform` 抽帧，每帧单独过 BLIP2 生成 caption，同一候选轮次内各帧的 caption 用 ". " 拼成一条视频级 caption，生成 `caption_num` 条候选后按 `keep_candidate_mode` 归约。

功能描述

- 抽帧语义：`all_keyframes` 只解 I 帧（`skip_frame="NONKEY"`）；`uniform` 在 `frame_num=1` 时取中间帧、`=2` 时取首尾帧、`>2` 时含首尾均匀取
- `max_frames` 上限：关键帧数超上限时再按 uniform 语义抽稀，防止长视频逐帧生成把时间和显存打满

- 抽帧只用 PyAV，不走仓库里依赖 torchcodec 的解码路径（torchcodec 需要 CUDA NPP 的 libnppicc.so，镜像里缺库会导致模块 import 失败）；本地路径直接交给 av，远端对象先下载到内存再解
- 候选拼接：per_frame[i][j] 是第 i 帧的第 j 条候选，按候选轮次 j 把各帧结果 ". " 拼接，得到第 j 条视频级 caption；空串被跳过
- keep_candidate_mode 三种取值：random_any 随机留 1 条、similar_one_simhash 留与参考文本 simhash 汉明距离最近的 1 条、all 全部保留
- similar_one_simhash 的参考文本作为可选第二列 reference_text 传入；该列缺失或该行不是字符串时退化为 random_any
- simhash 用自带的 64 位实现（window_size=2 的 shingle + 汉明距离），不引入 simhash-pybind C 扩展；候选之间差异明显时选中结果稳定，极端相近的候选汉明距离容易并列，选中哪一条不保证唯一
- 行数恒定：本算子是列式算子，直接输出归约后的 caption 列表，不增删数据行、也不把 caption 回填到原文
- 输入为 None、抽不到帧、或该行生成异常时返回空列表（不是 None），异常只记日志不打断整列
- 权重只从 model_path/model_name 本地目录加载，初始化时校验目录存在，不联网下载、不做运行时 pip 安装

算子参数

输入

输入	含义
video	视频列，内容形式由 video_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制）
reference_text	可选，参考文本列，仅 keep_candidate_mode="similar_one_simhash" 时使用

输出

输出	含义
captions	list<large_string>，归约后的视频级 caption；random_any / similar_one_simhash 返回 1 条，all 返回 caption_num 条，失败或抽不到帧返回空列表

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	视频输入类型：video_url（本地/BOS/HTTP 路径）、video_base64、video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"Salesforce/blip2-opt-2.7b"	相对 model_path 的 BLIP2 权重子目录
dtype	str	"float16"	权重精度：float16 / float32 / bfloat16
frame_sampling_method	str	"all_keyframes"	抽帧方式：all_keyframes（只解 I 帧）或 uniform（均匀抽 frame_num 帧）
frame_num	int	3	uniform 模式抽帧数
max_frames	int	16	单视频抽帧上限，超出按 uniform 语义抽稀；<=0 表示不限
caption_num	int	1	每个视频生成的候选 caption 条数
keep_candidate_mode	str	"random_any"	候选归约方式：random_any / similar_one_simhash / all
prompt	str 或 None	None	全局提示词，非空时对每帧都带同一条提示
max_new_tokens	int	128	单帧生成长度上限
do_sample	bool	True	是否采样；caption_num > 1 时必须为 True 才有多样性
batch_size	int	8	帧推理微批大小（与 aihc_udf 的 batch_size 是两个参数）
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 本算子不提供帧翻转参数，也不做样本增删与原文回填：输出严格逐行对应输入。
- all_keyframes 下抽帧数取决于视频的 I 帧密度，逐帧生成 caption 的耗时随帧数线性增长；批量场景建议改用 uniform + 明确的 frame_num。
- BLIP2-OPT 权重内嵌 Meta OPT，商用前需确认许可；换 BLIP-2 Flan-T5 版可规避。
- 与「视频描述生成（整段视频）」的差别：本算子逐帧生成再拼接，不建模帧间时序；后者把整段帧序列一次喂给 Video-BLIP。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_captioning_from_frames import VideoCaptioningFromFrames

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "captions",
        aihc_udf(
            VideoCaptioningFromFrames,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "Salesforce/blip2-opt-2.7b",
                "dtype": "float16",
                "frame_sampling_method": "uniform",
                "frame_num": 2,
                "caption_num": 1,
                "keep_candidate_mode": "random_any",
                "max_new_tokens": 24,
                "batch_size": 2,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频描述生成（整段视频）

简介

视频描述生成算子（整段视频路线）：抽帧后把整段帧序列（tchw 转成 bcthw）一次性喂给 Video-BLIP，生成 caption_num 条候选，再按 keep_candidate_mode 归约。生成参数默认为 num_beams=4、temperature=0.7、top_p=0.9、repetition_penalty=1.5、do_sample=True。

功能描述

- 模型侧在 video/_video_blip.py 里定义两个子类：VideoBlipVisionModel 把 (b, c, t, h, w) 摊成 (b*t, c, h, w) 过视觉塔再还原时间维；VideoBlipForConditionalGeneration 跳过 Blip2ForConditionalGeneration.__init__、直接调祖父类，才能把 vision_model 换成支持时间维的版本
- 加载后固定设置 processor.num_query_tokens=32 与 model.config.image_token_index=50265；早期 checkpoint 的 processor 配置里没有 num_query_tokens，而 transformers>=4.47 的 Blip2Processor.__call__ 会拿它去减 max_length，为 None 时直接抛 TypeError，算子初始化时会用 model.config.num_query_tokens 补齐
- 抽帧语义：all_keyframes 只解 I 帧；uniform 在 frame_num=1 时取中间帧、=2 时取首尾帧、>2 时含首尾均匀取；抽帧只用 PyAV，不走依赖 torchcodec 的解码路径
- max_frames 上限（默认 8）：Video-BLIP 一次吃整段帧序列，显存随帧数线性上涨，必须兜底；超限时按 uniform 语义抽稀
- caption_num 条候选是重复调 generate 得到的（do_sample=True 固定开启），每条候选把 batch_decode 出的文本用 ". " 拼接
- keep_candidate_mode 三种取值：random_any 随机留 1 条、similar_one_simhash 留与参考文本 simhash 汉明距离最近的 1 条、all 全部保留；参考文本作为可选第二列 reference_text 传入，缺失时退化为 random_any
- simhash 用自带的 64 位实现（window_size=2 的 shingle + 汉明距离），不引入 simhash-pybind；候选之间差异明显时选中结果稳定，极端相近的候选汉明距离容易并列，选中哪一条不保证唯一
- 行数恒定：不做样本增删与占位符回填
- 输入为 None、抽不到帧、或该行生成异常时返回空列表，异常只记日志不打断整列
- 权重只从 model_path/model_name 本地目录加载，初始化时校验目录存在，不联网下载、不做运行时 pip 安装

算子参数

输入

输入	含义
video	视频列，内容形式由 video_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制）
reference_text	可选，参考文本列，仅 keep_candidate_mode="similar_one_simhash" 时使用

输出

输出	含义
captions	list<large_string>，归约后的视频级 caption；random_any / similar_one_simhash 返回 1 条，all 返回 caption_num 条，失败或抽不到帧返回空列表

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	视频输入类型：video_url（本地/BOS/HTTP 路径）、video_base64、video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"kpyu/video-blip-opt-2.7b-ego4d"	相对 model_path 的 Video-BLIP 权重子目录
dtype	str	"float16"	权重精度：float16 / float32 / bfloat16
frame_sampling_method	str	"all_keyframes"	抽帧方式：all_keyframes（只解 I 帧）或 uniform（均匀抽 frame_num 帧）
frame_num	int	3	uniform 模式抽帧数
max_frames	int	8	单视频帧上限，超出按 uniform 语义抽稀；<=0 表示不限
caption_num	int	1	每个视频生成的候选 caption 条数
keep_candidate_mode	str	"random_any"	候选归约方式：random_any / similar_one_simhash / all
prompt	str 或 None	None	全局提示词，非空时作为生成的文本前缀
max_new_tokens	int	128	生成长度上限
num_beams	int	4	beam 数
temperature	float	0.7	采样温度
top_p	float	0.9	nucleus 采样阈值
repetition_penalty	float	1.5	重复惩罚
horizontal_flip	bool	False	是否水平翻转每帧
vertical_flip	bool	False	是否垂直翻转每帧
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- `do_sample` 在本算子里固定为 True，没有开关；因此同一视频多次运行的 caption 不完全可复现。
- 帧数直接决定显存：整段帧序列一次进视觉塔，`max_frames` 调大前先确认显存。
- 默认权重名里的 `ego4d` 指其在 Ego4D 第一人称视频上微调，素材域与之差距大时描述质量会下降，换权重需自行评估。
- 与「视频描述生成（抽帧）」的差别：本算子把帧序列作为一个整体输入，能利用帧间信息；后者逐帧生成再用 `". "` 拼接。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_captioning_from_video import VideoCaptioningFromVideo

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "captions",
        aihc_udf(
            VideoCaptioningFromVideo,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "kpyu/video-blip-opt-2.7b-ego4d",
                "dtype": "float16",
                "frame_sampling_method": "uniform",
                "frame_num": 4,
                "caption_num": 1,
                "max_new_tokens": 32,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频帧图文相似度

简介

视频帧与文本的相似度算子：视频抽帧过 CLIP 图像塔、文本过 CLIP 文本塔，逐帧算余弦相似度后按 `reduce_mode` 聚合。CLIP 编码复用 `multimodal/_clip_dual_tower.py`，与 `ClipSimilarityScore` 同一份实现。

功能描述

- 双输入列：视频列 + 文本列，两列必须等长，不等长直接抛 `ValueError`
- 文本按整列一次性编码（跳过 None 与空白串），帧按 `batch_size` 微批过图像塔，避免逐帧 batch=1 的吞吐损失
- 图像与文本向量都做 L2 归一化后点积，即标准余弦相似度，取值 [-1,1]

- 抽帧内置 VideoFrameSampler，默认 by_count_uniform + count_k=3；sample_mode 与时间窗、间隔、时间戳、max_frames 参数全部透传
- 抽帧默认走均匀抽取而不是解全部关键帧，帧数可控、代价可预期
- VideoFrameSampler 输出 BGR 帧，算子内部转 RGB 后再交给 CLIPProcessor
- 文本编码失败（整列）或该行文本为空时不做图像编码，该行直接返回 None
- 聚合支持 avg / max / min，结果保留 6 位小数；某个微批推理异常只丢这批帧并打日志
- CPU 上强制把 float16 回退成 float32（CPU 的 fp16 矩阵乘支持不完整）
- 权重只从 model_path/model_name 本地目录加载，初始化时校验目录存在，不联网下载

算子参数

输入

输入	含义
videos	视频列，内容形式由 video_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制）
texts	文本列，与视频列逐行对应；None 或空白串该行输出 None

输出

输出	含义
frames_text_sim	float64，逐帧图文余弦相似度按 reduce_mode 聚合后的结果，取值 [-1,1]；抽不到帧、文本为空或推理失败返回 None

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	视频输入类型：video_url（本地/BOS/HTTP 路径）、video_base64、video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"openai/clip-vit-base-patch32"	相对 model_path 的 CLIP 权重子目录
dtype	str	"float32"	权重精度：float16 / float32 / bfloat16；CPU 上 float16 自动回退 float32
sample_mode	str	"by_count_uniform"	抽帧模式，透传 VideoFrameSampler：by_count_uniform / by_interval_time / by_interval_frames / by_fps / by_timestamps
start_time_sec	float	0.0	抽帧起始秒
end_time_sec	float 或 None	None	抽帧结束秒，None 表示到视频末尾
count_k	int 或 None	3	by_count_uniform 模式的抽帧数
interval_sec	float 或 None	None	by_interval_time 模式的间隔秒
interval_frames	int 或 None	None	by_interval_frames 模式的间隔帧
target_fps	float 或 None	None	by_fps 模式的目标帧率
timestamps_sec	list[float] 或 None	None	by_timestamps 模式的时间戳列表（秒）
max_frames	int 或 None	None	抽帧数上限，None 不限
reduce_mode	str	"avg"	帧相似度聚合方式：avg / max / min
video_format	str	"mp4"	video_binary / video_base64 输入时的容器格式
batch_size	int	16	图像塔推理微批大小（与 aihc_udf 的 batch_size 是两个参数）
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- CLIP 文本塔上限 77 token，更长的文本会被截断。
- 本算子不提供 horizontal_flip / vertical_flip 帧翻转参数。
- 本算子只出分、不带阈值过滤，过滤在 daft 侧用 where() 做。
- 一次 transform 会先把该批次所有视频抽出的帧全部转成 PIL 图像再分批推理，峰值内存约等于 UDF batch_size × 单视频抽帧数 × 帧尺寸。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_frames_text_similarity import VideoFramesTextSimilarity

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

samples = {
        "video": ["bos://your-bucket/sample.mp4"],
        "text": ["a video clip"],
    }
    ds = daft.from_pydict(samples)
    ds = ds.with_column(
        "frames_text_sim",
        aihc_udf(
            VideoFramesTextSimilarity,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "openai/clip-vit-base-patch32",
                "sample_mode": "by_count_uniform",
                "count_k": 3,
                "reduce_mode": "avg",
                "batch_size": 16,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video"), col("text")),
    )
    ds.show()
```

视频水印概率

简介

视频水印概率算子：抽帧后逐帧过 HuggingFace 水印图像分类器，取「有水印」类别的 softmax 概率，再按 `reduce_mode` 聚合成视频级概率。只出概率，不给水印位置。

功能描述

- 抽帧内置 `VideoFrameSampler`，默认 `by_count_uniform` + `count_k=3`；`sample_mode` 与时间窗、间隔、时间戳、`max_frames` 参数全部透传
- 逐帧 `softmax(logits)` 后取「有水印」类别的概率：类别下标从 `model.config.id2label` 里找含 `watermark` 且不含 `no` 的那一项，找不到时回落到下标 1（不同权重的 label 命名习惯不同）
- `VideoFrameSampler` 输出 BGR 帧，算子内部转 RGB 后再交给 `AutoImageProcessor`
- 聚合支持 `avg` / `max` / `min`，结果保留 6 位小数
- 按 `batch_size` 微批推理，单个微批异常只丢这批帧并打日志，不让整列失败
- 抽帧默认均匀抽 3 帧，吞吐可控但覆盖面有限；水印通常整片存在，抽样代价可接受，需要更全的覆盖可调大 `count_k` 或换 `by_fps` / `by_interval_*` 模式
- 整列抽帧失败、帧形状非法、或该视频所有帧都推理失败时，该行返回 None
- 权重只从 `model_path/model_name` 本地目录加载，初始化时校验目录存在，不联网下载

算子参数

输入

输入	含义
videos	视频列，内容形式由 video_src_type 决定（本地/BOS/HTTP 路径、Base64 字符串、二进制）

输出

输出	含义
watermark_prob	float64，逐帧水印概率按 reduce_mode 聚合后的结果，取值 [0,1]；无有效帧或全部帧推理失败返回 None

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	视频输入类型：video_url（本地/BOS/HTTP 路径）、video_base64、video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"amrul-hzz/watermark_detector"	相对 model_path 的权重子目录
sample_mode	str	"by_count_uniform"	抽帧模式，透传 VideoFrameSampler：by_count_uniform / by_interval_time / by_interval_frames / by_fps / by_timestamps
start_time_sec	float	0.0	抽帧起始秒
end_time_sec	float 或 None	None	抽帧结束秒，None 表示到视频末尾
count_k	int 或 None	3	by_count_uniform 模式的抽帧数
interval_sec	float 或 None	None	by_interval_time 模式的间隔秒
interval_frames	int 或 None	None	by_interval_frames 模式的间隔帧
target_fps	float 或 None	None	by_fps 模式的目标帧率
timestamps_sec	list[float] 或 None	None	by_timestamps 模式的时间戳列表（秒）
max_frames	int 或 None	None	抽帧数上限，None 不限
reduce_mode	str	"avg"	帧概率聚合方式：avg / max / min
video_format	str	"mp4"	video_binary / video_base64 输入时的容器格式
batch_size	int	16	算子内部推理微批大小（与 aihc_udf 的 batch_size 是两个参数）
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 与存量的 VideoWatermarkDetect 是两个不同算子：后者走 PP-OCRv4 文本检测 + 跨帧位置一致性，输出水印**区域**；本算子是逐帧图像分类器，输出水印**概率**，不给位置。目标相同、方法不同，按需选用。
- 本算子只出概率、不做样本过滤，例如 prob_threshold=0.8 这类阈值判断在 daft 侧用 where() 做。
- 一次 transform 会先把该批次所有视频抽出的帧全部转成 PIL 图像再分批推理，峰值内存约等于 UDF batch_size × 单视频抽帧数 × 帧尺寸。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_watermark_prob import VideoWatermarkProb

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "watermark_prob",
        aihc_udf(
            VideoWatermarkProb,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "amrul-hzz/watermark_detector",
                "sample_mode": "by_count_uniform",
                "count_k": 3,
                "reduce_mode": "avg",
                "batch_size": 16,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    # 过滤水印概率高于 0.8 的视频
    ds = ds.where(col("watermark_prob") < 0.8)
    ds.show()
```

视频 OCR 文字面积占比

简介

视频文字面积占比算子：抽帧 → EasyOCR 的 CRAFT 检测器找文本框 → 计算文字面积占帧面积的比例 → 多帧聚合成一个视频级数值。只产出占比，过滤交给下游表达式。

功能描述

- 抽帧复用存量 VideoFrameSampler（output_frames=True、output_base64=False），抽帧参数（模式、时间窗、上限等）由本算子透传，默认 by_count_uniform + count_k=3
- 只加载 EasyOCR 的检测器（recognizer=False），不加载识别模型，因此不产出文字内容；download_enabled=False，权重必须离线预置
- 单帧占比按文本框**并集**面积 / 帧面积计算：把所有框画进一张布尔掩码再数像素，取值恒在 [0, 1]，框重叠不会重复计数
- 水平框直接用 [x_min, x_max, y_min, y_max]（会先裁到画面内并纠正大小颠倒）；自由四边形框用外接矩形近似
- 多帧聚合：reduce_mode 取 avg / max / min，结果保留 6 位小数

- 整列抽帧失败时该列全部按空帧处理；单帧 OCR 异常只记日志并跳过该帧，不影响同视频其它帧
- 一个视频没有任何有效帧（或全部帧检测失败）时返回 None，不是 0
- 支持三种输入形式：`video_url`（本地/BOS 路径）、`video_base64`、`video_binary`；后两种会先解码成字节再连同 `video_format` 交给抽帧算子

算子参数

输入

输入	含义
videos	视频输入数组，内容类型由 video_src_type 决定（路径 / Base64 字符串 / bytes）

输出

输出	含义
ocr_area_ratio	float64，文字面积占比，取值 [0, 1]；无有效帧时为 None

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	输入形式：video_url / video_base64 / video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"EasyOCR"	相对 model_path 的 EasyOCR 权重目录
lang_list	list[str] 或 None	None	检测语言列表，None 时取 ["ch_sim", "en"]
sample_mode	str	"by_count_uniform"	抽帧模式，透传 VideoFrameSampler：by_count_uniform / by_interval_time / by_interval_frames / by_fps / by_timestamps
start_time_sec	float	0.0	抽帧起始秒
end_time_sec	float 或 None	None	抽帧结束秒，None 表示到视频末尾
count_k	int 或 None	3	by_count_uniform 模式的抽帧数
interval_sec	float 或 None	None	by_interval_time 模式的间隔秒
interval_frames	int 或 None	None	by_interval_frames 模式的间隔帧
target_fps	float 或 None	None	by_fps 模式的目标帧率
timestamps_sec	list[float] 或 None	None	by_timestamps 模式的时间戳列表
max_frames	int 或 None	None	抽帧数上限
reduce_mode	str	"avg"	多帧聚合方式：avg / max / min
video_format	str	"mp4"	二进制 / Base64 输入的容器格式
rank	int	0	多卡场景 worker 序号，当前只记录不参与设备选择

注意事项

- 面积口径是设计如此：按文本框并集面积统计，重叠框不重复计数；自由四边形按外接矩形近似，斜排文字的占比会偏大。
- 抽帧策略：走 `VideoFrameSampler` 的 `by_count_uniform`，在 [start, end] 上按时间均匀取点（`count_k=1` 取起始位置），视频时长拿不到时退化为固定随机种子的蓄水池采样。
- 送检的帧是 OpenCV 的 BGR 通道序。
- EasyOCR 权重目录缺失或 Reader 初始化失败时 `__init__` 直接抛错，不会退化成联网下载。
- 是否用 GPU 由 aihc_udf 的 `num_gpus` 推导（基类按 `num_gpus > 0` 置 `use_gpu`），传给 EasyOCR 的 `gpu=` 参数；`rank` 不参与选卡。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_ocr_area_ratio import VideoOcrAreaRatio

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "ocr_area_ratio",
        aihc_udf(
            VideoOcrAreaRatio,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "EasyOCR",
                "count_k": 3,
                "reduce_mode": "avg",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    # 只保留文字面积占比低于 0.3 的视频
    ds = ds.where(col("ocr_area_ratio") < 0.3)
    ds.show()
```

视频音频标签

简介

视频音轨打标算子：抽出视频的第一条音轨，重采样到 16k，过 Hugging Face 的 Audio Spectrogram Transformer（AST，AudioSet 微调）分类，取 logit 最大的标签作为视频的音频标签。

功能描述

- 用 PyAV 打开容器、只取 `streams.audio[0]` 解码；多声道按通道取均值压成单声道
- 最多解码 `max_seconds` 秒的音频（默认 30s）；源码注明 AST 是固定 1024 帧输入，长音频截断不影响结果
- 采样率不等于 16000 时重采样：优先 `torchaudio.transforms.Resample`，没装 torchaudio 时退化为 `numpy` 线性插值
- 单标签输出：`argmax(logits)` 后取 `model.config.id2label`
- 无音轨、采样率取不到、解码不出数据，以及输入为 null 的行，统一返回 `EMPTY`，便于下游用同一个判断处理
- 单行推理异常只记日志并返回 None，不影响同批其它行
- 权重只从本地 `<model_path>/<model_name>` 加载，目录缺失时 `__init__` 直接抛 `FileNotFoundError`，不联网
- 解码走 PyAV 而不是 torchcodec：torchcodec 依赖 CUDA NPP 动态库（`libnppicc.so`），镜像缺库时整个模块 import 就会失败

算子参数

输入

输入	含义
video	视频输入数组，内容类型由 video_src_type 决定（本地/BOS 路径 / Base64 字符串 / bytes）

输出

输出	含义
audio_tag	large_string，AST 预测的音频标签；无音轨返回 EMPTY；该行处理失败返回 None

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	输入形式：video_url / video_base64 / video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"MIT/ast-finetuned-audioset-10-10-0.4593"	相对 model_path 的 AST 权重子目录
max_seconds	float	30.0	最多送入模型的音频秒数，<= 0 表示不截断
rank	int	0	多卡场景 worker 序号，设备取 cuda:<rank % 可见卡数>

注意事项

- 只看第一条音轨（固定 `stream_indexes=[0]`）。多音轨视频（多语言配音等）不会遍历。
- 没装 torchaudio 时用线性插值兜底重采样，重采样质量与 torchaudio 不同，个别边界样本的标签可能因此不同。
- 输出是单个标签，不是标签列表；需要多标签时要另接阈值化的分类算子。

- 是否用 GPU 由 aihc_udf 的 num_gpus 推导（基类按 num_gpus > 0 置 use_gpu）。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_tagging_from_audio import VideoTaggingFromAudio

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "audio_tag",
        aihc_udf(
            VideoTaggingFromAudio,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "MIT/ast-finetuned-audioset-10-10-0.4593",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

视频深度估计

简介

视频逐帧深度估计算子：抽帧后逐帧过 Depth-Anything V2 得到相对深度图，输出逐帧深度统计量，可选把原始深度图落盘成 .npz。模型用 transformers 内置的 Depth-Anything V2 做逐帧推理，权重全部本地加载，算子内不做网络请求与安装。

功能描述

- 抽帧走 OpenCV（cv2.VideoCapture 按帧号 seek），两种口径：sample_fps > 0 且源 fps 可读时按 round(src_fps / sample_fps) 的步长等间隔抽（产出时间轴上的深度曲线）；否则按 frame_num 均匀抽（frame_num=1 取中间帧，否则在 [0, 总帧数-1] 上 linspace）
- 源 fps 读不到时即便设了 sample_fps 也会回落到 frame_num 口径
- 长边超过 max_res 的帧先按 INTER_AREA 等比缩小（max_res 默认 1280）
- 抽出的帧 BGR 转 RGB 后逐帧推理，predicted_depth 用 bicubic 插值回该帧原尺寸
- 输出是 struct：逐帧的 depth_min / depth_max / depth_mean 与 frame_indices 等长、逐帧对应；默认只出统计量，因为整段逐帧 float32 深度图在列式场景下会是 GB 级的列，需要原始深度图请开 save_depth 落盘
- save_depth=True 时把逐帧深度堆成一个数组存进 {output_dir}/{视频文件名}_depth.npz（含 depth、frame_indices 两个数组），列里返回该路径；save_depth=False 时 depth_npz 为空串
- 远端路径（BOS / HTTP）经 run_on_local_path 先下载到临时目录再处理，函数返回后临时文件即清理
- 输入为 None、解不出任何帧、或该行处理异常时返回全空 struct（num_frames=0、fps=-1.0、三个列表为空、depth_npz=""），异常只记日志不打断整列
- CPU 上 dtype="float16" 自动回退 float32
- 权重只从 model_path/model_name 本地目录加载，初始化时校验目录存在，不联网下载

算子参数

输入

输入	含义
video	视频列，支持本地路径、BOS、HTTP；远端路径先落地到临时文件再解码

输出

输出	含义
depth	struct，字段见下
depth.num_frames	int32，实际完成推理的帧数
depth.fps	float64，sample_fps > 0 时为 sample_fps，否则为源视频 fps；失败为 -1.0
depth.frame_indices	list，各帧在源视频中的帧号
depth.depth_min	list，逐帧深度最小值（相对深度，无物理单位）
depth.depth_max	list，逐帧深度最大值
depth.depth_mean	list，逐帧深度均值
depth.depth_npz	large_string，逐帧深度图的 npz 路径，仅 save_depth=True 时非空

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"depth-anything/Depth-Anything-V2-Small-hf"	相对 model_path 的 Depth-Anything V2 权重子目录
dtype	str	"float16"	推理精度：float16 / float32 / bfloat16；CPU 上 float16 自动回退 float32
frame_num	int	8	均匀采样帧数，必须 >=1；sample_fps > 0 时本参数被忽略
sample_fps	float	0.0	按固定时间间隔采样的频率（帧/秒），>0 时优先于 frame_num
max_res	int	1280	长边超过该阈值的帧先等比缩小，必须 >=64
save_depth	bool	False	是否把逐帧深度图存成 npz 并在列里返回路径
output_dir	str	"/tmp/aihc_video_depth"	npz 输出目录，仅 save_depth=True 时使用
rank	int	0	多卡场景 worker 序号，设备取 cuda:(rank % 可见卡数)

注意事项

- 本算子是逐帧独立推理，没有 Video-Depth-Anything 的时序模块，长视频上帧间深度可能有轻微闪烁；需要严格时序一致的场景请走离线 VDA 流程。
- 输出是相对深度，没有物理尺度，不同帧、不同视频之间的数值不保证同一尺度，不要直接当米制距离比较。本算子也不提供 metric 深度模式（点云 ply + open3d），那需要 metric 权重。
- npz 文件名只取视频文件名的 stem：同一 output_dir 下不同目录的同名视频会互相覆盖，批量落盘时要按数据源分开 output_dir。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_depth_estimation import VideoDepthEstimation

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "depth",
        aihc_udf(
            VideoDepthEstimation,
            construct_args={
                "model_path": "/path/to/models",
                "model_name": "depth-anything/Depth-Anything-V2-Small-hf",
                "frame_num": 4,
                "max_res": 640,
                "save_depth": True,
                "output_dir": "/tmp/aihc_video_depth",
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds = ds.with_column("depth_mean", col("depth")["depth_mean"])
    ds = ds.with_column("depth_npz", col("depth")["depth_npz"])
    ds.show()
```

视频运动分（RAFT 光流）

简介

视频运动分算子的 RAFT 稠密光流后端，以子类形式扩展存量 VideoMotionScore（daft/aihc/functions/video/video_motion_score.py），不改父算子代码，只覆写预处理与光流计算两个钩子：用 torchvision 的 RAFT-Large 估计相邻采样帧的稠密光流，抽帧、统计与打分沿用父算子。输出 schema 与父算子完全一致，used_algorithm 字段固定为 raft。

功能描述

- 抽帧：间隔为 max(1, round(1 / sample_ratio)) 帧（默认 sample_ratio=0.125 即每 8 帧取 1 帧），逐帧读完整段视频后按间隔留帧
- 预处理整段覆写父类：先按 min_frame_size 放大过小帧，再按 downsample_ratio 缩放，最后把边长补到 8 的倍数，并把 BGR 转 RGB（父类的 OpenCV 后端是转灰度）
- 光流：像素归一到 [-1, 1] 后做两帧前向，取最后一次迭代的输出（num_flow_updates=iters，默认 12）
- 每对相邻采样帧算 6 个统计量并对全片取平均：幅值均值、中位数、中位数/帧对角线、95 分位以上幅值的均值、该值/帧对角线、有效幅值密度（幅值 > mag_threshold 的像素数 × 其均值 / 像素总数）
- 综合分：均值 / 95 分位 / 密度三项按本算子自带的标定常量做 (x - mean) / std 标准化并裁剪到 [0, 1]，再按 0.4 / 0.4 / 0.2 加权；按阈值 0.2 / 0.4 / 0.6 / 0.8 / 1.0 归入 static / low / medium / high / extreme 五档
- 权重离线加载：raft_large(weights=None) 建网后 torch.load(ckpt_path)，torchvision 的自动下载被关掉，__init__ 阶段就校验 ckpt 存在
- 只打分不过滤：抽帧按 sample_ratio，输出 18 字段 struct
- batch_size 只把帧对循环切块，RAFT 仍逐对前向，不做批推理（父类只有 memfof 后端走真正的批推理）
- 异常兜底：采样帧不足 2 帧、解码失败或运行期异常时返回全 0 的空结果，status 写 empty 或 error: <信息>，不抛给调用方；一列里单个视频失败不影响其它行

- 每次 transform 结束（含异常路径）清一次 CUDA 缓存

算子参数

输入

输入	含义
video_paths	视频路径数组，支持本地路径与 BOS 路径（远端路径先下载到临时目录）
video_binaries	视频二进制数组（可选），路径为空时使用
video_formats	视频容器格式数组（可选），二进制输入时决定临时文件后缀，缺省按 .mp4

输出

输出为一个 struct 列，字段与父算子 VideoMotionScore 完全一致：

输出	含义
standardized_score	float32，综合运动分，取值 [0, 1]
motion_pattern	string，运动等级：static / low / medium / high / extreme
mean_score	float32，全片平均的光流幅值均值
median_score	float32，全片平均的光流幅值中位数
dynamic_mean_score	float32，中位数 / 帧对角线长度
high_percentile_score	float32，95 分位以上幅值的均值
dynamic_high_percentile_score	float32，上一项 / 帧对角线长度
density_score	float32，有效光流像素密度
video_resolution	list，[宽, 高]（取自解码器元数据）
total_frames	int64，视频总帧数（解码器元数据）
sample_frames_count	int64，实际采样帧数
used_algorithm	string，固定为 raft
used_metric	string，固定为 composite
status	string，success / empty / error: <信息>
total_process_time	float64，单视频总耗时（秒）
avg_frame_process_time	float64，单帧对光流平均耗时（秒）
max_frame_process_time	float64，单帧对光流最大耗时（秒）
min_frame_process_time	float64，单帧对光流最小耗时（秒）

参数

下表前 3 个是本算子自有参数，其余为父类 VideoMotionScore 的参数（通过 **kwargs 透传，默认值与父类一致）。

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
ckpt_path	str 或 None	None	RAFT 权重文件路径；None 时取 <model_path>/raft/raft_large_C_T_SKHT_V2-ff5fadd5.pth
iters	int	12	RAFT 迭代次数（num_flow_updates）
optical_flow_algorithm	str	-	父类参数，构造时被强制为 raft，传入值会被丢弃
sample_ratio	float	0.125	抽帧比例，实际抽帧间隔 max(1, round(1 / sample_ratio)) 帧
mag_threshold	float	0.01	算密度分时的有效光流幅值下限
flow_threshold	float	6	父类参数，仅保存不参与计算
smooth_window	int	5	父类参数，未被使用
downsample_ratio	float	1.0	帧缩放比例，取值会被夹到 [0.1, 1.0]
high_motion_threshold	float	0.02	父类参数，仅保存不参与计算
batch_size	int	10	帧对循环的切块大小；RAFT 逐对前向，不改变显存占用
algorithm_params	dict 或 None	None	父类给 OpenCV 后端传算法参数用；本子类覆写了估计器构建，不生效
rank	int	0	GPU 序号，推理设备取 cuda:
num_workers	int	4	父类参数，未被使用
use_cuda	bool	True	是否用 GPU；torch.cuda 不可用时退回 CPU
max_gpu_memory	float	0.8	父类只在 memfof 后端用它设显存占用比例，对本子类不生效
min_frame_size	int	32	帧最短边下限，低于该值先等比放大

注意事项

- 标定常量为 (45.5434, 45.8182, 91.115, 51.273, 45.5434, 45.8182)，源码注释说明实测自 8 段视频、口径是 sample_ratio=0.03 + downsample_ratio=0.5，样本量偏小且与默认参数口径不同。standardized_score 只在同一后端、同一抽帧/缩放参数下可比，不能与 VideoMotionScore 的 farneback / dis / tv-l1 / memfof 分数横向比较（每个后端一套标定）。
- 密度项的标定常量与均值项相同（6 元组第 5、6 位复用了第 1、2 位）。
- 权重缺失时 __init__ 直接抛 FileNotFoundError，不会退化成联网下载。
- 抽帧要把整段视频解完（逐帧 cap.read() 再按间隔留帧），解码耗时随总帧数增长；调小 sample_ratio 只减少光流计算量。
- 采样帧会被 resize（min_frame_size 放大、downsample_ratio 缩放、8 的倍数对齐），光流幅值与分辨率相关，改这两个参数会改变分数。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_motion_score_raft import VideoMotionScoreRaft

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "motion",
        aihc_udf(
            VideoMotionScoreRaft,
            construct_args={
                "model_path": "/path/to/models",
                "sample_ratio": 0.05,
                "batch_size": 4,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds = ds.with_column("used_algorithm", col("motion")["used_algorithm"])
    ds = ds.with_column("standardized_score", col("motion")["standardized_score"])
    ds.show()
```

视频运动分（ptlflow 光流）

简介

视频运动分算子的 ptlflow 稠密光流后端（默认 dpflow 模型），以子类形式扩展存量 `VideoMotionScore`（`daft/aihc/functions/video/video_motion_score.py`），不改父算子代码，只覆写预处理与光流计算两个钩子；抽帧、统计与打分沿用父算子，输出 schema 与父算子完全一致，`used_algorithm` 字段固定为 `ptlflow`。

功能描述

- 抽帧：间隔为 `max(1, round(1 / sample_ratio))` 帧（默认 `sample_ratio=0.125` 即每 8 帧取 1 帧），逐帧读完整段视频后按间隔留帧
- 预处理整段覆写父类：先按 `min_frame_size` 放大过小帧，再按 `downsample_ratio` 缩放，最后 BGR 转 RGB（父类的 OpenCV 后端是转灰度）；不做 8 的倍数对齐，对齐交给 ptlflow 的 `IOAdapter` 做 padding
- 光流：`IOAdapter` 按模型的 `output_stride` padding 后两帧前向，输出再 `unscale` 并手动裁回原始高宽（该版本 `IOAdapter` 只提供 `unscale`，不负责去 padding）；dpflow 的 `output_stride` 属性为 None，代码取 32 兜底
- `IOAdapter.prepare_inputs` 不保证把张量放到模型所在设备，算子显式搬一次，避免 CPU 输入撞 CUDA 权重
- 每对相邻采样帧算 6 个统计量并对全片取平均：幅值均值、中位数、中位数/帧对角线、95 分位以上幅值的均值、该值/帧对角线、有效幅值密度（幅值 > `mag_threshold` 的像素数 × 其均值 / 像素总数）
- 综合分：均值 / 95 分位 / 密度三项按本算子自带的标定常量做 `(x - mean) / std` 标准化并裁剪到 [0, 1]，再按 0.4 / 0.4 / 0.2 加权；按阈值 0.2 / 0.4 / 0.6 / 0.8 / 1.0 归入 static / low / medium / high / extreme 五档
- 权重离线加载：直接把预置 ckpt 传给 `ptlflow.get_model`，ptlflow 的首次运行自动下载被绕开；`ckpt_path` 允许传目录，目录内按名称排序取第一个 `.ckpt`；`__init__` 阶段就校验路径存在
- 只打分不过滤：抽帧按 `sample_ratio`，输出 18 字段 struct
- `batch_size` 只把帧对循环切块，仍逐对前向，不做批推理（父类只有 memfof 后端走真正的批推理）
- 异常兜底：采样帧不足 2 帧、解码失败或运行期异常时返回全 0 的空结果，`status` 写 `empty` 或 `error: <信息>`，不抛给调用方；一列里单个视频失败不影响其它行
- 每次 `transform` 结束（含异常路径）清一次 CUDA 缓存

算子参数

输入

输入	含义
video_paths	视频路径数组，支持本地路径与 BOS 路径（远端路径先下载到临时目录）
video_binaries	视频二进制数组（可选），路径为空时使用
video_formats	视频容器格式数组（可选），二进制输入时决定临时文件后缀，缺省按 .mp4

输出

输出为一个 struct 列，字段与父算子 `VideoMotionScore` 完全一致：

输出	含义
standardized_score	float32，综合运动分，取值 [0, 1]
motion_pattern	string，运动等级：static / low / medium / high / extreme
mean_score	float32，全片平均的光流幅值均值
median_score	float32，全片平均的光流幅值中位数
dynamic_mean_score	float32，中位数 / 帧对角线长度
high_percentile_score	float32，95 分位以上幅值的均值
dynamic_high_percentile_score	float32，上一项 / 帧对角线长度
density_score	float32，有效光流像素密度
video_resolution	list，[宽, 高]（取自解码器元数据）
total_frames	int64，视频总帧数（解码器元数据）
sample_frames_count	int64，实际采样帧数
used_algorithm	string，固定为 ptlflow
used_metric	string，固定为 composite
status	string，success / empty / error: <信息>
total_process_time	float64，单视频总耗时（秒）
avg_frame_process_time	float64，单帧对光流平均耗时（秒）
max_frame_process_time	float64，单帧对光流最大耗时（秒）
min_frame_process_time	float64，单帧对光流最小耗时（秒）

参数

下表前 4 个是本算子自有参数，其余为父类 VideoMotionScore 的参数（通过 `**kwargs` 透传，默认值与父类一致）。

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
ptlflow_model_name	str	"dpflow"	ptlflow 模型名
ckpt_path	str 或 None	None	ckpt 文件或所在目录；None 时取 <model_path>/ptlflow/<ptlflow_model_name>，传目录则取目录内排序后的第一个 .ckpt
get_model_args	dict 或 None	None	透传给 ptlflow.get_model 的额外参数（dict 形态）
optical_flow_algorithm	str	-	父类参数，构造时被强制为 ptlflow，传入值会被丢弃
sample_ratio	float	0.125	抽帧比例，实际抽帧间隔 max(1, round(1 / sample_ratio)) 帧
mag_threshold	float	0.01	算密度分时的有效光流幅值下限
flow_threshold	float	6	父类参数，仅保存不参与计算
smooth_window	int	5	父类参数，未被使用
downsample_ratio	float	1.0	帧缩放比例，取值会被夹到 [0.1, 1.0]
high_motion_threshold	float	0.02	父类参数，仅保存不参与计算
batch_size	int	10	帧对循环的切块大小；逐对前向，不改变显存占用
algorithm_params	dict 或 None	None	父类给 OpenCV 后端传算法参数用；本子类覆写了估计器构建，不生效
rank	int	0	GPU 序号，推理设备取 cuda:
num_workers	int	4	父类参数，未被使用
use_cuda	bool	True	是否用 GPU；torch.cuda 不可用时退回 CPU
max_gpu_memory	float	0.8	父类只在 memfof 后端用它设显存占用比例，对本子类不生效
min_frame_size	int	32	帧最短边下限，低于该值先等比放大

注意事项

- 标定常量为 (42.3111, 42.926, 85.8527, 50.6632, 42.3111, 42.926)，源码注释说明实测自 8 段视频、口径是 sample_ratio=0.03 + downsample_ratio=0.5，样本量偏小且与默认参数口径不同。standardized_score 只在同一后端、同一抽帧/缩放参数下可比，不能与 VideoMotionScore 的 farneback / dis / tv-l1 / memfof 或 RAFT 后端横向比较（每个后端一套标定）。
- 密度项的标定常量与均值项相同（6 元组第 5、6 位复用了第 1、2 位）。
- ckpt 缺失时 __init__ 直接抛 FileNotFoundError，不会退化成 ptlflow 的自动下载；换 ptlflow_model_name 时 ckpt_path 要同步换成该模型对应的 ckpt。
- 抽帧要把整段视频解完（逐帧 cap.read() 再按间隔留帧），解码耗时随总帧数增长；调小 sample_ratio 只减少光流计算量。
- 采样帧会被 resize（min_frame_size 放大、downsample_ratio 缩放），光流幅值与分辨率相关，改这两个参数会改变分数。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_motion_score_ptlflow import VideoMotionScorePtlflow

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "motion",
        aihc_udf(
            VideoMotionScorePtlflow,
            construct_args={
                "model_path": "/path/to/models",
                "ptlflow_model_name": "dpflow",
                "sample_ratio": 0.05,
                "batch_size": 4,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds = ds.with_column("used_algorithm", col("motion")["used_algorithm"])
    ds = ds.with_column("standardized_score", col("motion")["standardized_score"])
    ds.show()
```

视频目标分割

简介

文本引导的视频目标分割算子：首帧用 YOLOE 按文本类别做开集检测，检测框作为提示交给 SAM2.1 在整段视频里传播，得到逐帧逐目标的分割掩码。掩码以 RLE 形式输出。

功能描述

- 首帧检测：YOLOE.set_classes(class_names, get_text_pe(class_names)) 后 predict(conf=yoloe_conf)，输出框（xyxy 取整）、类别 id、置信度
- 加载 YOLOE 文本提示编码前会临时 chdir 到权重所在目录：ultralytics 按相对路径找 mobileclip_blt.pt
- 整段传播：Sam2VideoProcessor.init_video_session + add_inputs_to_inference_session（首帧、全部框），先显式对首帧推理一次，再走 propagate_in_video_iterator 逐帧取掩码
- 目标 id 以 session.obj_ids 为准（add_inputs_to_inference_session 会消费掉传入的 obj_ids 列表）
- 掩码后处理 binarize=False，算子自己按 binarize_threshold 二值化（logit > 阈值为前景），再压成 RLE：rle_counts 是交替的 0/1 段长、约定从 0 段开始，首段为前景时补一个长度 0 的 0 段；段长之和等于 shape 的元素总数
- 掩码按 RLE 存储而不是把每帧每目标的位图全量入内存，避免长视频 OOM
- 算子内不做网络请求与安装，transformers 版本不足时只在 import 失败时报错要求升级镜像
- 权重全部从本地预置目录加载，不联网下载；SAM2 目录或 YOLOE 权重缺失时 __init__ 直接失败
- max_frames 截断：SAM2 会把帧全部载入内存，超过上限只处理前 N 帧
- 类别列为空时回落 default_class_names；两者都为空该行按错误处理
- 首帧检不到目标时返回空 segment_data，status 为 success: no object detected，cls_id_dict 仍带出 YOLOE 的类别表
- 单行异常只记日志并写 status 为 error: <信息>，不影响同批其它行
- CPU 环境下推理精度强制为 float32（dtype 只在 GPU 生效）

算子参数

输入

输入	含义
videos	视频路径数组，支持本地路径与 BOS 路径（远端路径先下载到临时目录）
class_names	目标类别名数组（list），可选；为空时回落 default_class_names

输出

输出为一个 struct 列，字段如下：

输出	含义
segment_data	list，逐帧掩码。每项为 {frame_index: int32, masks: list}，masks 每项为 {obj_id: int32, shape: list, rle_counts: list}
cls_id_dict	list，YOLOE 本次推理的类别名表（results[0].names），object_cls_list 里的类别 id 按下标索引它
object_cls_list	list，首帧每个检测框的类别 id
yoloe_conf_list	list，首帧每个检测框的置信度
frame_count	int32，实际处理的帧数（已按 max_frames 截断）
status	string，success / success: no object detected / error: no frames / error: <信息>

参数

参数名称	类型	默认值	描述
model_path	str	"/opt/aihc/model"	权重根目录
sam2_model_name	str	"facebook/sam2.1-hiera-tiny"	相对 model_path 的 SAM2 权重子目录
yoloe_model_name	str	"yoloe/yoloe-11l-seg.pt"	相对 model_path 的 YOLOE 权重文件，同目录需有 mobileclip_blt.pt
default_class_names	list[str] 或 None	None	输入类别列为空时使用的兜底类别名
yoloe_conf	float	0.5	YOLOE 检测置信度阈值
dtype	str	"bfloat16"	SAM2 推理精度，可选 bfloat16 / float16 / float32；CPU 上强制 float32
max_frames	int 或 None	300	单视频最多处理帧数，None 表示不截断
binarize_threshold	float	0.0	掩码二值化阈值，logit 大于该值视为前景
rank	int	0	多卡场景 worker 序号，设备取 cuda:<rank % 可见卡数>

注意事项

- 掩码是 RLE，不是位图：按 `shape` reshape 前先把 `rle_counts` 展开成 0/1 序列（首段是背景段，长度可能为 0）。校验方式是段长之和等于 `shape` 各维乘积。
- `segment_data` 的第一项来自对首帧的显式推理，其后各项来自 `propagate_in_video_iterator` 的产出，`frame_index` 取的是迭代序号而非 SAM2 返回的帧号。
- `transformers` 必须 >= 4.56（`Sam2VideoModel` / `Sam2VideoProcessor` 自该版本起是正式 API）。算子不会在运行时安装依赖，版本不足只报错。
- YOLOE 的文本提示依赖 `mobileclip_blt.pt` 与权重同目录，缺失时只打 warning，实际会在 `get_text_pe` 阶段失败。
- 长视频显存/内存占用由 `max_frames` 决定，SAM2 的 session 会持有全部帧，调大要同步评估显存。
- 是否用 GPU 由 `aihc_udf` 的 `num_gpus` 推导（基类按 `num_gpus > 0` 置 `use_gpu`），`rank` 只用于多卡取模选卡。

调用示例

```
 from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_object_segment import VideoObjectSegment

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict(
        {
            "video": ["bos://your-bucket/sample.mp4"],
            "classes": [["person", "cat", "dog", "car"]],
        }
    )
    ds = ds.with_column(
        "segment",
        aihc_udf(
            VideoObjectSegment,
            construct_args={
                "model_path": "/path/to/models",
                "sam2_model_name": "facebook/sam2.1-hiera-tiny",
                "yoloe_model_name": "yoloe/yoloe-11l-seg.pt",
                "yoloe_conf": 0.25,
                "max_frames": 30,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video"), col("classes")),
    )
    ds = ds.with_column("status", col("segment")["status"])
    ds = ds.with_column("frame_count", col("segment")["frame_count"])
    ds.show()
```

视频帧标签

简介

视频抽帧打标算子：按 all_keyframes 或 uniform 抽帧，逐帧过 RAM++（recognize-anything plus）打标，再按标签出现频次降序合并成一条视频级标签列表。

功能描述

- 抽帧语义：all_keyframes 只解关键帧（I 帧，PyAV 的 skip_frame="NONKEY"）；uniform 下 frame_num=1 取中间帧、=2 取首尾帧、>2 在时长内均匀取（含首尾）
- max_frames（默认 64）是显存兜底：抽出的帧超过上限时再按同一均匀规则抽稀，否则长视频关键帧过多会打爆显存
- 打标按 batch_size 分批，且整列的 (行号, 帧) 会先展平再统一分批，一个 batch 可以跨行，避免逐视频起停模型
- 帧级标签先在帧内去重（保序），再跨帧计频；因此最终频次等于「出现该标签的帧数」，同一帧内重复出现的标签不重复计数
- 视频级输出按频次降序、已去重（同频次保持首次出现顺序）
- 权重只从本地 <model_path>/<model_name> 加载，文件缺失时 __init__ 抛 FileNotFoundError，不联网、不做运行时 pip 安装
- 兼容处理：RAM++ 的 BERT 代码从 transformers.modeling_utils 导三个已迁走的函数，算子加载前会从 transformers.pytorch_utils 回填同名别名，不需要为它降级整个环境的 transformers
- 解码走 PyAV 而不是 torchcodec：torchcodec 依赖 CUDA NPP 动态库（libnppicc.so），镜像缺库时整个模块 import 就会失败
- 单行抽帧失败或某个 batch 推理失败只记日志，对应行返回空列表，不影响同批其它行

算子参数

输入

输入	含义
video	视频输入数组，内容类型由 video_src_type 决定（本地/BOS 路径 / Base64 字符串 / bytes）

输出

输出	含义
frame_tags	list<large_string>，视频级标签，按出现频次降序去重；输入为 null 或该行失败时为空列表

参数

参数名称	类型	默认值	描述
video_src_type	str	"video_url"	输入形式：video_url / video_base64 / video_binary
model_path	str	"/opt/aihc/model"	权重根目录
model_name	str	"ram_plus_swin_large_14m.pth"	相对 model_path 的 RAM++ 权重文件名
frame_sampling_method	str	"all_keyframes"	抽帧方式：all_keyframes / uniform
frame_num	int	3	uniform 模式抽帧数，必须为正
input_size	int	384	模型输入分辨率
max_frames	int	64	单视频抽帧上限，<= 0 表示不限
batch_size	int	8	打标微批大小，<= 1 时按 1 处理
rank	int	0	多卡场景 worker 序号，设备取 cuda:<rank % 可见卡数>

注意事项

- 标签频次是帧内去重后的计频（等于「出现该标签的帧数」），同频次标签按首次出现顺序排列，因此同频次标签之间的先后不代表置信度差异。
- all_keyframes 抽出的帧数取决于视频的 GOP 结构，不可控；需要固定帧数就用 uniform + frame_num。
- backbone 固定为 swin_l（与默认权重匹配），换权重要确认结构一致；input_size 需与权重训练分辨率匹配。
- ram（recognize-anything）包是运行依赖，算子不会在运行时安装它，缺失时 import 直接失败。
- 是否用 GPU 由 aihc_udf 的 num_gpus 推导（基类按 num_gpus > 0 置 use_gpu）。

调用示例

```
from __future__ import annotations

import os

import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.video.video_tagging_from_frames import VideoTaggingFromFrames

os.environ.setdefault("BOS_ENDPOINT", "http://bj.bcebos.com")
os.environ.setdefault("BOS_REGION", "bj")

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

ds = daft.from_pydict({"video": ["bos://your-bucket/sample.mp4"]})
    ds = ds.with_column(
        "frame_tags",
        aihc_udf(
            VideoTaggingFromFrames,
            construct_args={
                "video_src_type": "video_url",
                "model_path": "/path/to/models",
                "model_name": "ram_plus_swin_large_14m.pth",
                "frame_sampling_method": "uniform",
                "frame_num": 3,
                "batch_size": 4,
            },
            num_cpus=1,
            num_gpus=1,
            concurrency=1,
            batch_size=1,
        )(col("video")),
    )
    ds.show()
```

AIHC-Daft算子开发使用指南

数据处理算子可使用百舸提供的 aihc-daft 包进行开发。aihc-daft 是百度 AI 异构计算平台（AIHC）推出的多模态 AI 数据处理框架，基于Daft 构建，提供开箱即用的数据处理算子库，支持单机多核到多机分布式的弹性扩展，面向 AI 训练数据生产场景。

Daft 核心特性

- **弹性分布式执行**。支持单机多核（Native Runner）与多机集群（Ray Runner）两种执行模式，业务代码无需修改，一行配置即可从单机无缝扩展至百节点集群，满足从开发调试到 PB 级数据生产的全场景需求。
- **惰性求值与查询优化**。 采用惰性执行模型，所有数据变换操作仅构建逻辑计划，在触发 `collect()` / `show()` / `write_*()` 时统一优化执行，自动进行谓词下推、列裁剪等优化，减少不必要的 I/O 和计算开销。
- **丰富的数据格式支持**。原生支持主流数据格式的读写，包括：
  - 结构化数据：Parquet、CSV、JSON、SQL 数据库
  - 数据湖格式：Delta Lake、Apache Iceberg、Apache Hudi、Lance
  - AI 数据集：HuggingFace Hub 数据集
  - 多媒体：视频帧序列、WARC 网页归档、MCAP 机器人传感器数据
- **多模态数据类型原生支持**内置 `Image`、`Video`、`Audio` 等多媒体数据类型，支持直接在 DataFrame 列中存储和处理图像、音视频数据，无需手动序列化。
- **灵活的 UDF（用户自定义函数）系统** 提供完整的 UDF 开发框架，支持声明 CPU、GPU、内存等资源需求，框架自动完成任务调度与资源分配。支持批处理模式、并发控制、进程/线程隔离，满足 CPU 密集与 GPU 推理等不同场景。
- **GPU 原生调度支持** UDF 可声明所需 GPU 资源（支持小数，如 `num_gpus=0.5`），框架与 Ray 协同完成 GPU 感知调度，天然适配深度学习推理、向量化等 GPU 密集型算子。
- **SQL 查询支持**支持直接使用 SQL 语法对 DataFrame 进行查询（`daft.sql()`），降低数据处理门槛，兼容熟悉 SQL 的用户习惯。
- **多种存储后端统一接入**通过统一的存储抽象层，支持本地文件系统、百度对象存储（BOS）、AWS S3、HTTP(S) 等多种存储后端，使用相同 API 访问不同存储，路径前缀自动路由。
- **DataFrame API 简洁直观**提供类 Pandas 的 DataFrame 操作接口，支持 `select`、`filter`、`groupby`、`join`、`sort`、`limit` 等常用操作，以及窗口函数（`Window`），学习成本低。
- **数据湖 Catalog 集成**支持与 Apache Iceberg、Apache Gravitino、Unity Catalog 等主流数据湖 Catalog 集成，实现数据治理、表版本管理与跨平台数据共享。

集成aihc-daft方式

你可以通过镜像或者 pip 包的方式，集成 aihc-daft。

1. 镜像

```
ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.5.3-cu12.9-py3.12-ubuntu22.04
```

镜像已经预置aihc-Daft 以及多模态算子相关的依赖以及运行环境，比如 Cuda、Conda、Ray 等，推荐直接使用

2. pip包离线安装
   - Aihc-daft 尚未在pypi 仓库发布，用户需要下载 aihc-daft 的离线安装包。点击下载
   - 执行以下命令安装

```
pip install aihc_daft-0.5.3-cp312-abi3-manylinux_2_35_x86_64.whl
```

aihc-daft内置算子示例

这里以图片哈希计算处理器算子为例

test_image_hash.py脚本如下：

```
from __future__ import annotations

import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.image.image_hash import ImageHash

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    samples = {
        "image": [
            "file:///local/sample_1.jpg",
            "file:///mnt/pfs/sample_2.jpg",
            "file:///mnt/bos/sample_3.jpg",
        ]
    }

    num_datasets = len(samples["image"])
    ds = daft.from_pydict(samples).into_partitions(num_datasets) #强制分布式切分partitions
    ds = ds.with_column(
        "image_hash",
        aihc_udf(
            ImageHash,
            construct_args={
                "image_src_type": "image_url",
                "method": "phash",
            },
            num_cpus=0.5,
            batch_size=1,
            concurrency=num_datasets, # 多个数据集并发执行
        )(col("image")),
    )
    ds.show()
```

分布式数据处理执行命令

```
##### 使用DAFT_RUNNER=ray基于ray分布式执行
DAFT_RUNNER=ray python test_image_hash.py

##### 单机执行
python test_image_hash.py
```

aihc-daft 基础参数说明

aihc_udf 参数说明

参数	说明	默认值	示例
operator	算子类（必填）	—	ImageHash
construct_args	传给算子初始化的参数	{}	construct_args={ "image_src_type": "image_url", "method": "phash", }
num_cpus	每实例占用 CPU 核数	None（由调度器自动分配）	2
num_gpus	每实例占用 GPU 卡数	None（不使用 GPU）	1 或 0.5
memory_bytes	每实例内存上限（字节）	None（不限制）	2 * 1024^3（2GB）
batch_size	每次处理的数据条数	None（由框架自动决定）	64
concurrency	同时运行的实例数	None（由框架自动决定）	8
use_process	是否使用进程隔离（CPU 密集时建议开启）	False（使用线程）	True

数据读写方式

支持本地文件/挂载目录文件/BOS/HTTP等多种方式:

```
    samples = {
        "image": [
            "file:///local/sample_1.jpg",      #本地数据
            "file:///mnt/pfs/sample_2.jpg",    #pfs挂载点内数据
            "file:///mnt/bos/sample_3.jpg",    #bos挂载点内数据
            "bos://bucket/path/sample_4.jpg"   #bos直接抓取数据
            "http://url/sample_5.jpg"          #http抓取数据
        ]
    }
```

若使用BOS直接抓取数据的方式, 需要在数据处理代码中注入BOS相关环境变量, 如下：

```
os.environ["BOS_ENDPOINT"] = "http://bj.bcebos.com"     #endpoint
os.environ["BOS_ACCESS_KEY_ID"] = ""
os.environ["BOS_SECRET_ACCESS_KEY"] = ""
os.environ["BOS_REGION"] = "bj"
```

最佳实践

这里我们以具身数据格式转换为例，指导用户基于百舸平台的数据处理算子，通过开发机/分布式训练任务，实现 LerobotV2.1 数据集向 V3.0 版本的格式转换。准备工作

环境准备

1. 这里我们可以使用开发机来开发调试代码，关于aihc-daft集成，你可以直接使用 aihc-daft 的镜像启动开发机，也可以使用自己的镜像启动，手动安装 aihc-daft 的包
2. 数据准备
   - 这里我们以huggingface中开源测试数据集lerobot/pusht/ 、dataset/lerobot/pusht2/和 lerobot/aloha_sim_insertion_human/ 为例
   - 我们提供了打包的数据集，点击下载。 原始数据集分布为:
     - dataset/lerobot/aloha_sim_insertion_human/
     - dataset/lerobot/pusht/
     - dataset/lerobot/pusht2/

算子开发

1. 这里我们基于原始的TarUncompress算子，实现了递归目录下Tar 文件解压等能力。脚本如下：

```
 import json
import os
import tarfile
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.process.tar_extractor_udf import TarUncompress
from daft.aihc.functions.process.tar_extractor_udf import discover_datasets
from daft.aihc.functions.process.tar_extractor_udf import create_tasks_from_datasets

TAR_EXTENSIONS = (".tar", ".tar.gz", ".tgz", ".tar.bz2", ".tbz2", ".tar.xz", ".txz", ".tar.zst")


def is_tar_file(filepath: str) -> bool:
    lower = filepath.lower()
    if not any(lower.endswith(ext) for ext in TAR_EXTENSIONS):
        return False
    return tarfile.is_tarfile(filepath)


def find_tar_files(directory: str) -> list[str]:
    tar_files = []
    for root, _dirs, files in os.walk(directory):
        for f in files:
            full = os.path.join(root, f)
            if is_tar_file(full):
                tar_files.append(full)
    return tar_files


def safe_members(tf: tarfile.TarFile) -> list[tarfile.TarInfo]:
    return [m for m in tf.getmembers() if not m.name.startswith("/") and ".." not in m.name]


def extract_recursive(tar_path: str, output_dir: str) -> list[str]:
    all_extracted = []

    # 第一次解压
    with tarfile.open(tar_path) as tf:
        members = safe_members(tf)
        tf.extractall(path=output_dir, members=members)
        all_extracted.extend([os.path.join(output_dir, m.name) for m in members])

    # 持续扫描并解压新出现的 tar 文件
    pending = find_tar_files(output_dir)
    processed = set()

    while pending:
        current = pending.pop(0)
        if current in processed:
            continue
        processed.add(current)

        extract_dir = os.path.dirname(current)
        with tarfile.open(current) as tf:
            members = safe_members(tf)
            tf.extractall(path=extract_dir, members=members)
            all_extracted.extend([os.path.join(extract_dir, m.name) for m in members])

        # 删除已解压的内层 tar 包（如需保留，注释掉下面这行）
        # os.remove(current)

        new_tars = find_tar_files(output_dir)
        for t in new_tars:
            if t not in processed and t not in pending:
                pending.append(t)

    return all_extracted


class RecursiveTarUncompress(TarUncompress):
    """递归解压多层嵌套 tar 包的 UDF。"""

    def __call__(self, input_path, output_path):
        input_list = input_path.to_pylist()
        output_list = output_path.to_pylist()
```

```
        output_list = output_path.to_pylist()

        results = []
        for inp, outp in zip(input_list, output_list):
            os.makedirs(outp, exist_ok=True)
            all_files = extract_recursive(inp, outp)
            # 返回 JSON 字符串，与父类 TarUncompress 的 __return_column_type__(String) 一致
            results.append(json.dumps({
                "status": "success",
                "input": inp,
                "output": outp,
                "extracted_files": all_files,
                "extracted_count": len(all_files),
            }))
        return results


if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    base_path = "/mnt/pfs/xx"  # 【用户需替换】实际存放tar包的目录

    # 直接扫描 base_path 下的 tar 文件
    tar_files = find_tar_files(base_path)
    if not tar_files:
        raise ValueError(f"未在 {base_path} 下发现任何 tar 文件，请检查路径")

    # 去除所有 tar 后缀作为输出目录，如 test_dataset.tar.gz -> test_dataset
    def strip_tar_ext(path: str) -> str:
        while True:
            base, ext = os.path.splitext(path)
            if ext.lower() in (".gz", ".bz2", ".xz", ".zst", ".tar", ".tgz", ".tbz2", ".txz"):
                path = base
            else:
                break
        return path

    tasks = {
        "input_path": tar_files,
        "output_path": [strip_tar_ext(t) for t in tar_files],
    }

    num_tasks = len(tasks["input_path"])
    concurrency = max(num_tasks, 1)

    ds = daft.from_pydict(tasks)
    ds = ds.into_partitions(num_tasks)

    ds = ds.with_column(
        "result",
        aihc_udf(
            RecursiveTarUncompress,
            construct_args={},
            num_cpus=1,
            num_gpus=0,
            batch_size=1,
            concurrency=concurrency,
            use_process=True,
        )(col("input_path"), col("output_path")),
    )
    ds.show()
```

2. 基于ConvertDatasetV21ToV30算子，将lerobotV2.1数据集格式 转换为 lerobotV3.0数据集格式，脚本如下：

```
 import os
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.embodied.convert_dataset_v21_to_v30_udf import ConvertDatasetV21ToV30

if __name__ == "__main__":
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    tasks = {
        "input_repoid": [
            "lerobot/aloha_sim_insertion_human/",
            "lerobot/pusht/",
            "lerobot/pusht2/"
        ],
        "input_path": ["/mnt/pfs/xx/test_dataset/dataset/"] * 3,
        "output_path": ["/mnt/pfs/xx/lerobotv3"] * 3  # 【用户需替换】格式转换后的输出目录
    }
    num_datasets = len(tasks["input_repoid"])
    ds = daft.from_pydict(tasks).into_partitions(num_datasets)

    ds = ds.with_column(
        "convert_result",
        aihc_udf(
            ConvertDatasetV21ToV30,
            construct_args={
            },
            num_cpus=0.1,
            batch_size=1,
            concurrency=num_datasets,
            use_process=True
        )(col("input_repoid"), col("input_path"), col("output_path")),
    )
    ds.show()
```

整体处理流程:

运行pipiline.py

```
 import json
import os
import tarfile
import daft
from daft import col

from daft.aihc.common.udf import aihc_udf
from daft.aihc.functions.process.tar_extractor_udf import TarUncompress
from daft.aihc.functions.embodied.convert_dataset_v21_to_v30_udf import ConvertDatasetV21ToV30

##### ====================== 直接复用 data_convert.py 全部代码 ======================
TAR_EXTENSIONS = (".tar", ".tar.gz", ".tgz", ".tar.bz2", ".tbz2", ".tar.xz", ".txz", ".tar.zst")

def is_tar_file(filepath: str) -> bool:
    lower = filepath.lower()
    if not any(lower.endswith(ext) for ext in TAR_EXTENSIONS):
        return False
    return tarfile.is_tarfile(filepath)

def find_tar_files(directory: str) -> list[str]:
    tar_files = []
    for root, _dirs, files in os.walk(directory):
        for f in files:
            full = os.path.join(root, f)
            if is_tar_file(full):
                tar_files.append(full)
    return tar_files

def safe_members(tf: tarfile.TarFile) -> list[tarfile.TarInfo]:
    return [m for m in tf.getmembers() if not m.name.startswith("/") and ".." not in m.name]

def extract_recursive(tar_path: str, output_dir: str) -> list[str]:
    all_extracted = []
    with tarfile.open(tar_path) as tf:
        members = safe_members(tf)
        tf.extractall(path=output_dir, members=members)
        all_extracted.extend([os.path.join(output_dir, m.name) for m in members])

    pending = find_tar_files(output_dir)
    processed = set()

    while pending:
        current = pending.pop(0)
        if current in processed:
            continue
        processed.add(current)

        extract_dir = os.path.dirname(current)
        with tarfile.open(current) as tf:
            members = safe_members(tf)
            tf.extractall(path=extract_dir, members=members)
            all_extracted.extend([os.path.join(extract_dir, m.name) for m in members])

        new_tars = find_tar_files(output_dir)
```

```
        for t in new_tars:
            if t not in processed and t not in pending:
                pending.append(t)

    return all_extracted

class RecursiveTarUncompress(TarUncompress):
    """递归解压多层嵌套 tar 包的 UDF。"""
    def __call__(self, input_path, output_path):
        input_list = input_path.to_pylist()
        output_list = output_path.to_pylist()
        results = []
        for inp, outp in zip(input_list, output_list):
            os.makedirs(outp, exist_ok=True)
            all_files = extract_recursive(inp, outp)
            results.append(json.dumps({
                "status": "success",
                "input": inp,
                "output": outp,
                "extracted_files": all_files,
                "extracted_count": len(all_files),
            }))
        return results

##### ====================== 主 pipeline 工作流 ======================
if __name__ == "__main__":
    # 统一环境初始化
    if os.getenv("DAFT_RUNNER", "native") == "ray":
        import ray
        ray.init(dashboard_host="0.0.0.0", ignore_reinit_error=True)
        daft.set_runner_ray()
    daft.set_execution_config(actor_udf_ready_timeout=6000, min_cpu_per_task=0)

    base_path = "/mnt/pfs/xx"
    convert_output_root = "/mnt/pfs/xx/lerobotv3"

    # ====================== 步骤1：执行解压（原 data_convert.py） ======================
    print("=== 步骤1：开始递归解压 tar 文件 ===")
    tar_files = find_tar_files(base_path)
    if not tar_files:
        raise ValueError(f"未在 {base_path} 下发现任何 tar 文件")

    def strip_tar_ext(path: str) -> str:
        while True:
            base, ext = os.path.splitext(path)
            if ext.lower() in (".gz", ".bz2", ".xz", ".zst", ".tar", ".tgz", ".tbz2", ".txz"):
                path = base
            else:
                break
        return path

    tasks_extract = {
        "input_path": tar_files,
        "output_path": [strip_tar_ext(t) for t in tar_files],
    }
    num_tasks = len(tasks_extract["input_path"])
    concurrency = max(num_tasks, 1)

    ds = daft.from_pydict(tasks_extract)
    ds = ds.into_partitions(num_tasks)

    ds = ds.with_column(
        "result",
        aihc_udf(
            RecursiveTarUncompress,
            construct_args={},
            num_cpus=1,
            num_gpus=0,
            batch_size=1,
            concurrency=concurrency,
            use_process=True,
        )(col("input_path"), col("output_path")),
    )
    df_extract = ds.collect()
    print("=== 解压完成 ===")

    # ====================== 步骤2：执行 v21 → v30 转换（原 lerobotv21-30.py） ======================
    print("=== 步骤2：开始格式转换 ===")
    tasks_convert = {
        "input_repoid": [
            "lerobot/aloha_sim_insertion_human/",
            "lerobot/pusht/",
            "lerobot/pusht2/"
        ],
        "input_path": ["/mnt/pfs/xx/test_dataset/dataset/"] * 3,
        "output_path": [convert_output_root] * 3
    }
    num_datasets = len(tasks_convert["input_repoid"])

    ds_convert = daft.from_pydict(tasks_convert).into_partitions(num_datasets)

    ds_convert = ds_convert.with_column(
        "convert_result",
        aihc_udf(
            ConvertDatasetV21ToV30,
            construct_args={},
            num_cpus=0.1,
            batch_size=1,
            concurrency=num_datasets,
            use_process=True
```

```
    )(col("input_repoid"), col("input_path"), col("output_path")),
)
ds_convert.show()
print("=== 全部 pipeline 执行完成 ===")
```

分布式数据处理

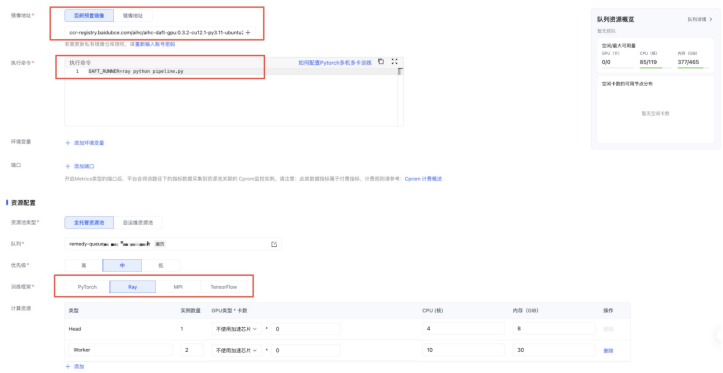
在分布式训练模块中，基于上述开发的算子代码，使用 Ray 计算引擎进行分布式处理数据。

> 也可以直接使用开发机进行单机的数据处理

Rayjob的提交可参考 快速提交Ray任务，关键参数如下：

1. 镜像地址：使用百舸预置镜像，选择aihc-daft预置镜像
2. 执行命令：`DAFT_RUNNER=ray python /mnt/pfs/xx/pipeline.py`，其中`pipeline.py`是上面开发的算子代码。
3. 计算框架：选择 Ray
4. 计算资源：可设置多个 worker 实例并行执行。在任务执行时，**Daft + Ray** 会自动调度、自动负载均衡、用满集群资源。
5. 存储挂载：将源数据所在存储实例挂载到容器内

提交任务即可进行数据处理，可通过 submitter 节点的日志查询数据处理的进度



数据集管理

创建数据集

在进行模型开发、训练、推理之前，一般需要准备好指定的数据集。数据集管理提供了强大的数据集管理功能，允许您创建和管理数据集及其多个版本。

通过数据集版本管理，可以精确复现实验、追踪数据版本、控制数据权限，以及在新版本出现问题时快速切换至旧版本，从而确保业务的连续性。

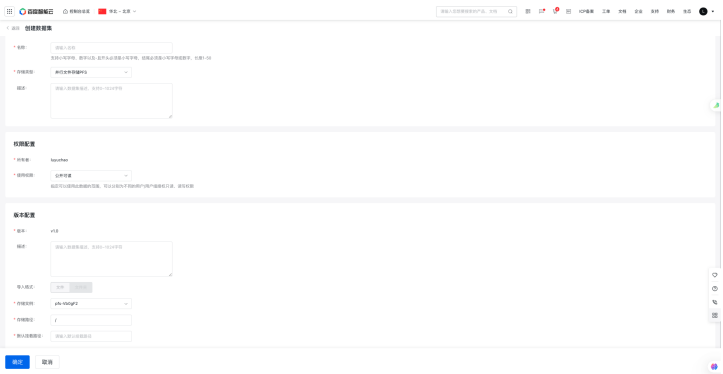
数据集管理功能支持对用户数据集进行元数据管理和权限管理。

进入数据集管理

1. 登录百舸控制台
2. 在左侧导航栏选择 AI资产管理 > 数据集

创建数据集

在自定义数据集页签下单击【创建数据集】。



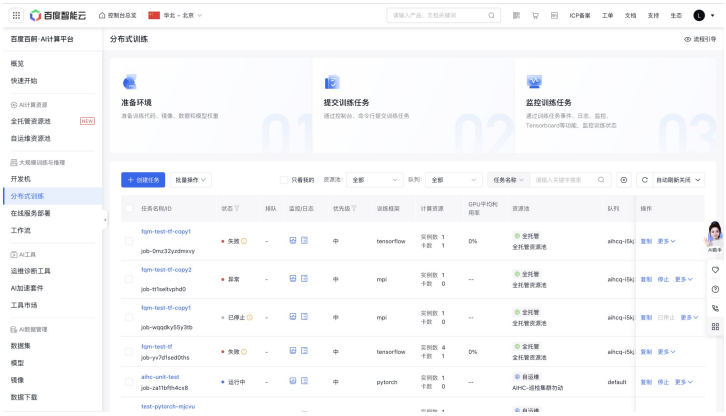
字段	说明
名称	必填，数据集的名称
存储类型	必选，存储类型支持对象存储（BOS）、并行文件存储（PFS）、数据湖存储加速工具（RapidFS）
存储实例	必选，BOS存储桶/PFS实例/RapidFS实例
导入格式	必选，支持文件和文件夹 当导入格式为文件时，存储路径需选择一个文件，创建好的数据集将与此文件相对应，常用于直接指定到某个具体文件的数据集创建。 当导入格式为文件夹时，存储路径需选择一个文件夹路径，可被挂载在容器中，常用于训练、推理或开发机的数据卷。
描述	选填，对数据集的描述信息
所有者	必选，创建时可以指定一个所有者，所有者对该数据集有管理权限（编辑、删除、创建版本等）
使用权限	必选，有三种数据权限可选:公开可读、仅所有者可读写、指定范围可用 公开可读：所有子用户可以以只读方式使用此数据集，仅所有者可读写方式使用 仅所有者可读写：仅数据集的所有者可以读写 指定范围可用：指定用户/用户组可以使用过，且根据拥有的权限读写 以上所述 可读、读写 指在使用数据集时用户创建的负载可以以哪种方式使用，比如在训练任务中一般需要使用读写权限的数据集保存权重、推理服务中如无写入操作则可以直接使用仅可读的数据集
版本	必选，系统自动生成
版本描述	选填，对版本的描述
存储路径	必选，数据集在存储中对应的路径，存储类型为BOS/PFS时须填写
数据源	必选，数据集对应的RapidFS数据源，需要先在RapidFS控制台创建后选择
默认挂载路径	必选，数据集挂载到负载上时的挂载路径，在挂载时可以修改

使用数据集创建业务负载

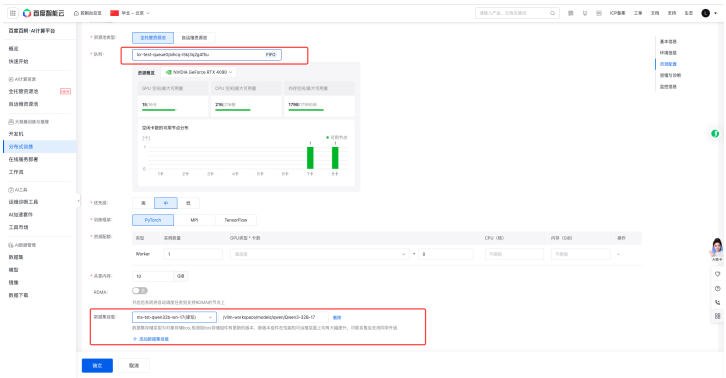
数据集创建成功后，可以在创建分布式训练任务、创建开发机实例、部署在线服务时使用数据集。

以下以创建分布式训练任务为例：

1. 在左侧菜单选择【分布式训练任务】，在页面中点击【创建任务】



2. 选中需要使用的资源队列后，在【数据集挂载】表单项下选择【添加数据集挂载】选择数据集



管理数据集

概述

本文档介绍如何在百舸控制台管理已经创建的数据集，包括编辑数据集、新建数据集版本和删除数据集。

前提条件

- 已登录百度智能云控制台。
- 已进入 **百度百舸·AI 计算平台->AI 资产管理->数据集** 页面。
- 已创建自定义数据集。## 角色权限

AI 资产管理员：可以对数据集进行管理操作，可以查看版本列表、新建版本、编辑数据集和删除等操作。

数据集所有者：可以对数据集进行管理操作，可以查看版本列表、新建版本、编辑数据集和删除等操作。

普通成员：可查看和使用自己有使用权限的数据集。

管理数据集

编辑数据集

支持对数据集名称、描述、使用权限进行修改，在数据集权限发生变更时，运行中的负载不会受到影响。

新建版本

同一个数据集下可以创建多个数据版本，数据版本不可以编辑。

删除数据集

删除数据集时会删除数据集下所有的版本，存储中的数据不会被删除。

导入数据到数据集

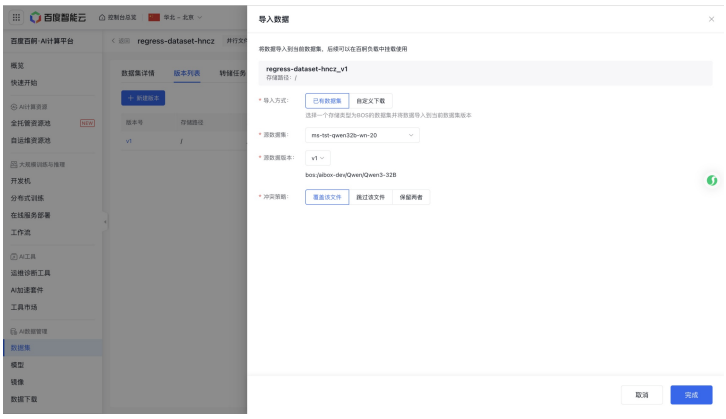
数据集提供数据导入能力，帮助用户将数据导入到已有数据集中，实现数据集、权重等文件的上传到百舸进行训练、开发等业务操作。

目前数据集支持已有数据集导入和自定义下载导入：

导入方式	概述	数据集存储类型支持
已有数据集导入	数据集管理提供存储类型为对象存储BOS和并行文件存储PFS（L2）数据集之间相互导入数据，通过已有数据导入功能可以方便的将数据在PFS和BOS之间进行转存。	PFS（极速型L2）BOS
自定义下载	通过编写自定义下载任务脚本，从任意百舸资源池网络可达的数据源下载或转储数据到数据集。	数据集支持的全部存储类型

进入数据集详情页，点击版本列表Tab，在版本列表中操作【导入数据】

已有数据集导入



限制条件

当前仅支持PFS L2与BOS之间进行互相转储。

操作步骤

1. 将需要导入数据的目录（目标数据集）和被导入目录（源数据集）分别创建为数据集，数据集创建完成后在数据集列表中找到对应数据集，在操作列选择【导入数据】
2. 在创建转储任务弹窗内选择源数据集及版本等信息。

> 例如将数据集B（BOS类型）的v1版本（存储路径/dataset/test）数据导入到数据集A（PFS类型）的v1版本（存储路径/dataset/test/v1），则在数据集A操作【数据导入】

参数	说明
任务名称	转储任务的名称，字母、数字、下划线_、中划线-，以字母开头，1~128个字符
源数据集	要导入的数据集，可选择当前用户有读权限的数据集
源数据版本	源数据集的数据版本，每个版本对应一个存储路径
冲突策略	覆盖该文件：保留源数据集中的文件

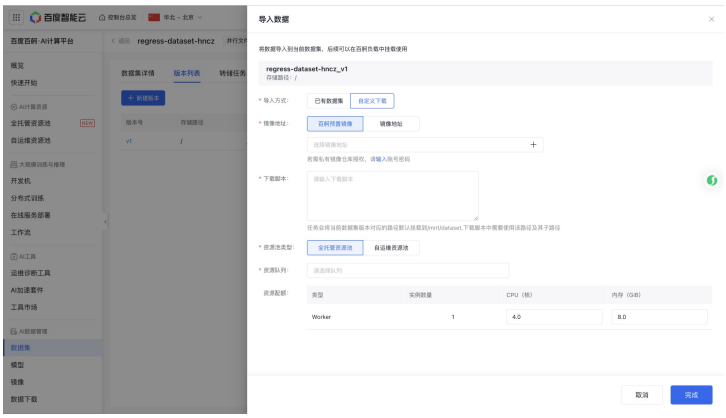
跳过该文件：不导入该文件 保留两者：在目标数据集中同时保存两个同名数据文件| |目标数据集|当前操作的数据集| |目标数据版本|导入数据存储的版本，每个版本对应一个存储路径|

3. 提交转储任务,在数据集详情的【转储任务】Tab页可以查看转储任务记录及状态。

自定义下载导入

自定义下载是利用用户自有资源池以任务的形式下载网络上的数据到当前数据集。

1. 通过编写合适的启动命令，可以实现从任意远程数据源下载数据；
2. 任务使用资源池的CPU和内存资源，不需要使用GPU资源。



限制条件

需要资源池可以访问需要下载的远程数据源，下载HuggingFace、GitHub等国外网站资源需先自行设置资源池可以访问外网或使用国内源。

操作步骤

1. 填写下载任务表单信息

参数	说明
镜像地址	下载数据需要使用的镜像，支持百舸预置镜像和自定义镜像一般情况如无特殊的需求，选择任意百舸预置镜像均可以完成下载
下载脚本	下载任务的执行脚本，脚本中包含下载数据的全部逻辑，包括鉴权、端点续传等 特别注意： 任务会将当前数据集版本对应的路径默认挂载到/mnt/dataset,下载脚本中需要使用该路径及其子路径保存下载的数据，同时默认注入环境变量DATASET_PATH=/mnt/dataset，也可以通过引用DATASET_PATH使用默认挂载路径
资源池类型	运行下载任务的资源池类型
资源队列	运行下载任务的资源队列
资源配额	运行任务的资源限制，默认值4核8G，可以根据下载脚本设置的并发数进行调整，如果不设置则不会限制

2. 提交下载任务成功后，在【分布式训练】任务列表中可以查看任务的日志和事件观察进度和异常。

下载脚本示例

> 特别注意：
>
> 1. 以下示例均在网络可达的前提条件下验证，使用前请先验证资源池可以访问源数据地址
> 2. 验证使用镜像registry.baidubce.com/aihcp-public/pytorch:2.7.0-cu12.8.61-py3.12-ubuntu24.04，自定义其他镜像时可根据事件信息进行脚本调试
> 3. 示例脚本只是方便用户理解下载方法，实际使用时需要根据需求对脚本进行修改

从魔搭下载数据集、权重

```
 ##### !/bin/sh

##### 设置数据集名称
DATASET_NAME="liucong/Chinese-DeepSeek-R1-Distill-data-110k"

##### 构建目标存储路径，以数据集根目录/数据集名称存储
TARGET_DIR="$DATASET_PATH/$DATASET_NAME_ESCAPED"

##### 检查是否已安装 modelscope
if ! command -v modelscope >/dev/null 2>&1; then
    echo "modelscope 未安装，正在安装..."
    # 使用 --user 避免权限问题（可选）
    pip install --user modelscope
    # 将 --user 安装的 bin 加入 PATH（适用于某些环境）
    export PATH="$HOME/.local/bin:$PATH"
fi

##### 使用 modelscope CLI 下载数据集（注意：使用 --dataset 而非 --model）
echo "正在下载数据集 $DATASET_NAME 到 $TARGET_DIR ..."
modelscope download --dataset "$DATASET_NAME" --revision master --local_dir "$TARGET_DIR"

if [ $? -eq 0 ]; then
    echo "数据集已成功下载至: $TARGET_DIR"
else
    echo "下载失败！请检查网络、权限或数据集是否存在。"
    exit 1
fi
```

从HuggingFace下载数据集

```
 ##### !/bin/sh

##### 配置
HF_DATASET_ID="Congliu/Chinese-DeepSeek-R1-Distill-data-110k"

##### 构建本地目标路径
LOCAL_DIR="$DATASET_PATH/$HF_DATASET_ID"
mkdir -p "$LOCAL_DIR"

##### 检查是否已存在（简单判断）
if [ -f "$LOCAL_DIR/.hf_download_complete" ]; then
    echo "数据集已下载（检测到 .hf_download_complete 标记），跳过。"
    exit 0
fi

##### 安装 huggingface_hub（如果未安装）
if ! python3 -c "import huggingface_hub" &> /dev/null; then
    echo "正在安装 huggingface_hub..."
    pip install --quiet huggingface_hub
fi

##### 创建临时 Python 脚本
PY_SCRIPT=$(cat <<EOF
from huggingface_hub import snapshot_download
import os

local_dir = os.environ['LOCAL_DIR']
repo_id = os.environ['HF_DATASET_ID']

print(f"正在从 Hugging Face 下载数据集: {repo_id}")
snapshot_download(
    repo_id=repo_id,
    local_dir=local_dir,
    repo_type="dataset",
    max_workers=8
)
print("下载完成！")
EOF
)

##### 导出环境变量供 Python 使用
export LOCAL_DIR="$LOCAL_DIR"
export HF_DATASET_ID="$HF_DATASET_ID"

##### 执行下载
echo "开始从 Hugging Face Hub 下载数据集..."
python3 -c "$PY_SCRIPT"

##### 检查是否成功并创建标记文件
if [ $? -eq 0 ]; then
    touch "$LOCAL_DIR/.hf_download_complete"
    echo "数据集已保存至: $LOCAL_DIR"
else
    echo "下载失败。请检查数据集 ID、网络或访问权限。"
    exit 1
fi
```

从开放网址下载

```
##### !/bin/bash

##### 配置：通过环境变量传入
HTTP_URL="https://example.com/data/SKYLENAGE-ReasoningMATH.zip"
DATASET_NAME="SKYLENAGE-ReasoningMATH"

##### 构建本地目标目录
LOCAL_DIR="$DATASET_PATH/Alibaba-DT/$DATASET_NAME"
mkdir -p "$LOCAL_DIR"

##### 推断文件名（从 URL 最后一段）
FILENAME=$(basename "$HTTP_URL")
TMP_FILE="$LOCAL_DIR/${FILENAME}"

##### 检查是否已下载
if [ -f "$TMP_FILE" ]; then
    echo "文件 $TMP_FILE 已存在，跳过下载。"
    exit 0
fi

##### 下载文件
echo "正在从 $HTTP_URL 下载..."
if command -v wget &> /dev/null; then
    wget -O "$TMP_FILE" --progress=bar:force:noscroll "$HTTP_URL"
elif command -v curl &> /dev/null; then
    curl -L -o "$TMP_FILE" -# "$HTTP_URL"
else
    echo "错误: 未找到 wget 或 curl，请安装其中一个工具。"
    exit 1
fi

##### 检查下载是否成功
if [ $? -ne 0 ] || [ ! -s "$TMP_FILE" ]; then
    echo "下载失败或文件为空。"
    rm -f "$TMP_FILE"
    exit 1
fi

##### 自动解压（如果为压缩包）
if [[ "$FILENAME" == *.zip ]]; then
    echo "正在解压 ZIP 文件..."
    unzip -q -o "$TMP_FILE" -d "$LOCAL_DIR"
    # 可选：删除压缩包
    # rm "$TMP_FILE"
elif [[ "$FILENAME" == *.tar.gz ]] || [[ "$FILENAME" == *.tgz ]]; then
    echo "正在解压 TAR.GZ 文件..."
    tar -xzf "$TMP_FILE" -C "$LOCAL_DIR"
    # rm "$TMP_FILE"
elif [[ "$FILENAME" == *.tar.bz2 ]]; then
    echo "正在解压 TAR.BZ2 文件..."
    tar -xjf "$TMP_FILE" -C "$LOCAL_DIR"
    # rm "$TMP_FILE"
else
    echo "文件非压缩格式，已保存至: $TMP_FILE"
fi

echo "数据集已准备完毕，路径: $LOCAL_DIR"
```

更多下载方式

不知道如何构建自定义数据源下载脚本，试试使用DeepSeek、ChatGPT等大语言模型编写，在Mac、Linux环境测试通过后即可在自定义导入任务中使用。

使用导入的数据

在分布式训练任务、在线服务部署、开发机中直接使用数据集或直接挂载数据集对应的存储路径。

管理数据集存储容量

对于PFS L2类型的存储，数据集支持通过PFS提供的Fileset能力管理存储容量配额和QOS等。

> Fileset介绍参考PFP产品文档 管理Fileset 章节

适用场景

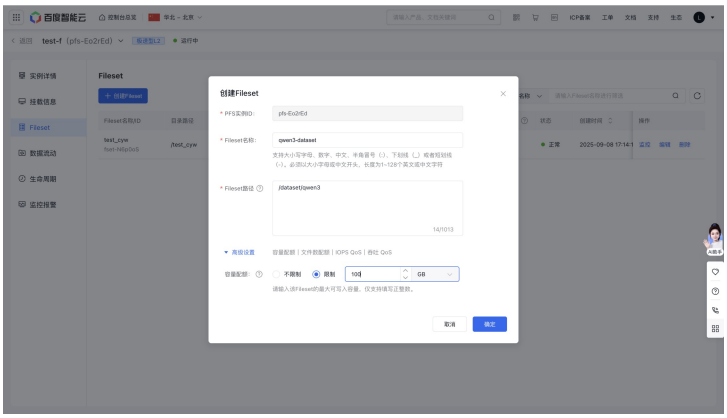
适用于需要对单个数据集或人员进行存储容量进行限制的业务场景

前置条件

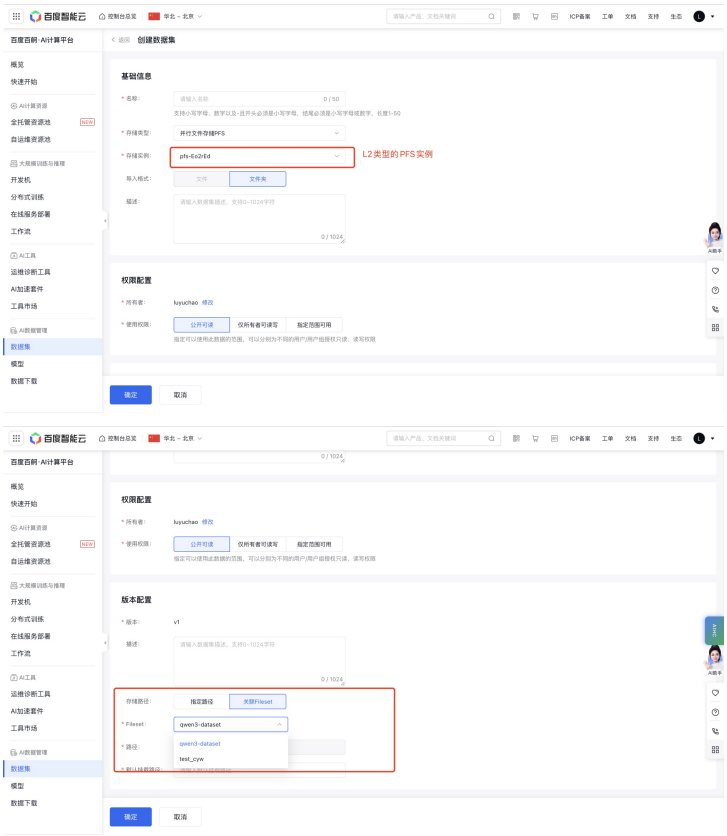
1. 仅支持PFS L2极速型
2. 创建的Fileset源路径必须是PFS中不存在的路径且父路径必须存在，例如：需要使用/dataset/qwen3路径时，/dataset必须存在且/dataset/qwen3不存在

使用Fileset创建数据集

1. 进入PFS控制台创建Fileset，详细操作参考 管理Fileset ，其中“Fileset路径”为需要创建数据集的源路径，以源路径/dataset/qwen3为例



2. 创建数据集，选择上一步中相同的PFS L2实例以及已创建的Fileset



3. 点击确定完成数据集创建，该数据集的容量配额即在Fileset设置的配额下使用。

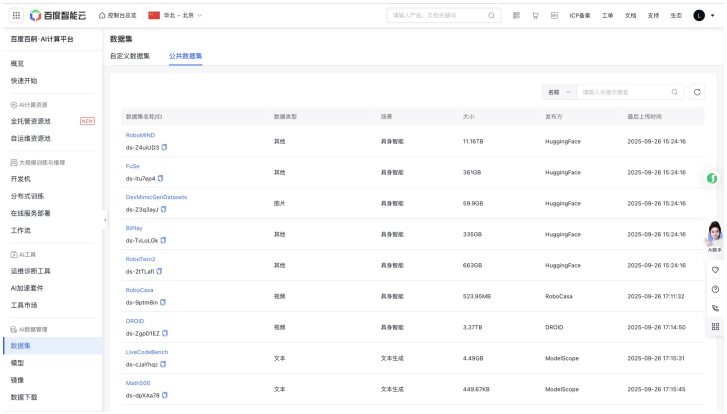
公共数据集

下载和使用公共数据集

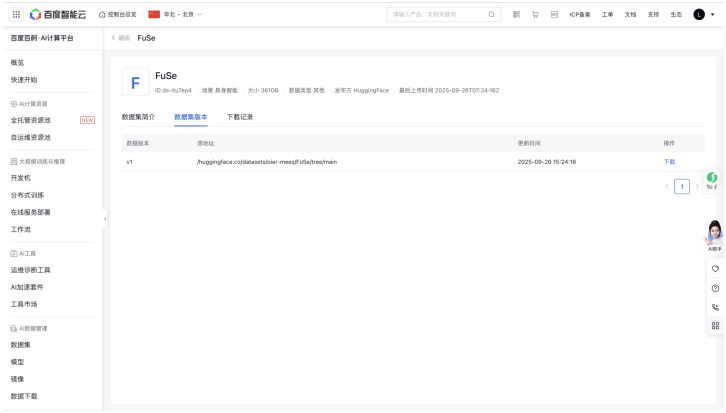
百舸平台提供开源热门数据集供用户下载使用，公共数据集来源于开源社区，用户可以将公共数据集下载（转储）到自定义BOS类型的数据集中，然后在训练任务中使用。

下载公共数据集

1. 进入【数据集】菜单，进入【公共数据集】列表

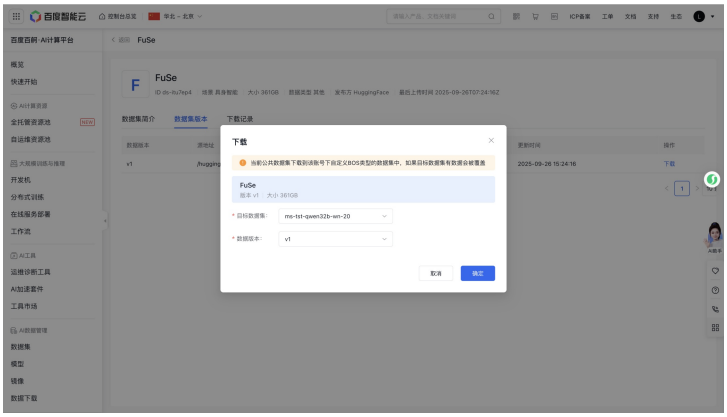


2. 选择需要下载的数据集，进入数据集版本列表



3. 点击【下载】，在弹窗中选择目标数据集，点击确定后会将当前选中版本保存到目标数据集中

> 需要先创建好目标数据集，且仅支持BOS类型的自定义数据集



使用公共数据集

在创建分布式训练任务时选择上一步设置的目标数据集作为数据集即可。

FuSe

FuSe 数据集包含在 RAIL 实验室@加州大学伯克利分校的 WidowX 机器人上收集的 26,866 条轨迹（远程操控轨迹，涵盖桌面抓取、购物袋内抓取和按钮按压三类任务）。

它包含在多个环境中收集的视觉、触觉、声音和动作数据，并使用自然语言进行标注。

机器人通过Meta Oculus Quest 2 VR头显操作，任务结合语言指令和复杂视觉遮挡，支持多传感器与语言融合的机器人策略研究。

论文：https://huggingface.co/papers/2501.04693

代码：https://github.com/oier-mees/FuSe

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

```
@inproceedings{jones25fuse,
 title={Beyond Sight: Finetuning Generalist Robot Policies with Heterogeneous Sensors via Language Grounding},
 author={Joshua Jones and Oier Mees and Carmelo Sferrazza and Kyle Stachowicz and Pieter Abbeel and Sergey Levine},
 booktitle = {Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)},
 year={2025},
 address = {Atlanta, USA}
}
```

BiPlay

BiPlay 包含 9.7 小时的双手动数据，这些数据由 aloha 机器人在美国加州大学伯克利分校的 RAIL 实验室收集。它包含 7023 个片段，2000 个语言标注和 326 个独特场景。

为了解决现有双臂数据集任务单一、环境固定的问题，BiPlay数据集采用随机物体和背景，采集多样化双臂操作轨迹。数据由多段3.5分钟的机器人操作视频拆分成7023个带语言任务描述的剪辑，总计10小时数据，支持双臂操作泛化研究。

论文：https://huggingface.co/papers/2410.10088

代码：https://github.com/sudeepdasari/dit-policy

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

```
@inproceedings{jones25fuse,
 title={Beyond Sight: Finetuning Generalist Robot Policies with Heterogeneous Sensors via Language Grounding},
 author={Joshua Jones and Oier Mees and Carmelo Sferrazza and Kyle Stachowicz and Pieter Abbeel and Sergey Levine},
 booktitle = {Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)},
 year={2025},
 address = {Atlanta, USA}
}
```

RoboTwin2

一个大型对象库，包括147个类别的731个实例，每个实例都用语义和操作相关的标签进行注释。在此基础上，我们开发了一个专家数据合成管道，该管道将多模态大型语言模型（MLLM）与循环中的仿真细化相结合，自动生成任务级执行代码。

代码：https://github.com/RoboTwin-Platform/RoboTwin

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

DexMimicGenDatasets

本仓库包含 ICRA 2025 论文"DexMimicGen: Automated Data Generation for Bimanual Dexterous Manipulation via Imitation Learning"的模拟环境和数据集的官方发布。

以RoboSuite和MimicGen为基础，在MuJoCo平台上构建的高保真双臂桌面操作任务仿真环境。DexMimicGen涵盖9类典型双臂任务，提出了增强版real2sim2real数据自动生成技术，只需60段真实人类演示便可生成2.1万条高质量仿真数据。相比原版MimicGen，该框架显著提升了数据生成效率和真实感，使机器人双臂协作任务的仿真训练更具实用性。

网站：https://dexmimicgen.github.io

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

RoboMIND

RoboMIND 概述 RoboMIND 组成

RoboMIND（多体智能规范数据集和机器人操作基准），这是一个综合数据集，包含 107k 条真实世界演示轨迹，涵盖 479 项不同任务和 96 个独特物体类别。

RoboMIND 数据集集成了来自多个机器人实体的远程操作数据，包括来自 Franka Emika Panda 单臂机器人的 52,926 条轨迹，来自 Tien Kung 仿人机器人的 19,152 条轨迹，来自 AgileX Cobot Magic V2.0 双臂机器人的 10,629 条轨迹，以及来自 UR-5e 单臂机器人的 25,170 条轨迹。

RoboMIND 为研究人员和开发者提供了一个宝贵资源，通过涵盖广泛的任务类型和多样化的物体类别，推动机器人学习和自动化技术的发展。该数据集因其巨大的规模和卓越的质量而脱颖而出，确保其在实际应用中的有效性和可靠性。

轨迹长度分布

不同的机器人形态表现出不同的轨迹长度分布。Franka 和 UR 机器人通常具有较短的轨迹，步数少于 200 步，非常适合训练基础操作技能。相比之下，Tien Kung 和 AgileX 机器人通常表现出超过 500 步的较长轨迹，这使得它们更适合训练长时程任务和复杂技能组合。

任务类别

根据自然语言描述，并考虑物体大小、使用场景和操作技能等因素，我们将数据集任务分为六大类：1) 机构化操作（机构操作）。2) 协调操作（协调操作）。3) 基础操作（基础操作）。4) 多物体交互（物体交互）。5) 精密操作（精密操作）。6) 场景理解（场景理解）。除了基础操作外，数据集还包括大量复杂任务，为训练通用机器人策略提供丰富的数据支持。

物品种类多样性

该数据集包含 96 种不同的物体类别。在厨房场景中，它包括草莓、鸡蛋、香蕉和梨等常见食物，以及烤箱和面包机等复杂的可调节电器。在家庭环境中，数据集既包含网球等刚性物体，也包含玩具等可变形物体。办公和工业场景包括需要精确控制的微小物体，如电池和齿轮。这种多样的物体范围增强了数据集的复杂性，并支持训练适用于各种环境的通用操作策略。

数据描述

构建高质量的机器人训练数据集对于开发具有强大泛化能力的端到端具身人工智能模型至关重要。一个理想的数据集应涵盖多样化的场景、任务类型和机器人形态，使模型能够适应不同的环境并可靠地执行各种任务。我们的团队构建了一个大规模、真实世界的机器人学习数据集，记录了在复杂环境中执行长期任务过程中的交互数据，支持具有通用操作能力的模型训练。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

```
@inproceedings{wu2025robomind,
        title={Robomind: Benchmark on multi-embodiment intelligence normative data for robot manipulation},
        author={Wu, Kun and Hou, Chengkai and Liu, Jiaming and Che, Zhengping and Ju, Xiaozhu and Yang, Zhuqin and Li, Meng and Zhao, Yinuo and Xu, Zhiyuan and Yang, Guang and others},
        booktitle={Robotics: Science and Systems (RSS) 2025},
        year={2025},
        publisher={Robotics: Science and Systems Foundation},
        url={https://www.roboticsproceedings.org/rss21/p152.pdf}
}
```

DROID:DistributedRobotInteractionDataset

创建大型、多样化、高质量的机器人操作数据集是实现更强大和鲁棒的机器人操作策略的重要基石。然而，创建此类数据集具有挑战性：在不同环境中收集机器人操作数据面临着后勤和安全问题，并且需要大量硬件和人力投入。因此，即使是目前最通用的机器人操作策略，也主要是在少数具有有限场景和任务多样性的环境中收集的数据上进行训练的。

在这项工作中，我们介绍了 DROID（分布式机器人交互数据集），一个多样化的机器人操作数据集，包含 76k 个演示轨迹或 350 小时的交互数据，由 50 名数据收集者在 12 个月内从北美、亚洲和欧洲的 564 个场景和 86 项任务中收集。我们证明了使用 DROID 进行训练可以带来性能更高、鲁棒性更强和泛化能力更好的策略。我们开源了完整的数据集、策略训练的代码以及一个详细的指南，用于重现我们的机器人硬件设置。

使用说明文档：https://droid-dataset.github.io/droid/

代码：https://github.com/droid-dataset/droid

如果您使用该数据集，请查看并遵守发布方声明的开源协议 droid-dataset.github.io。

```
@article{khazatsky2024droid,
    title   = {DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset},
    author  = {Alexander Khazatsky and Karl Pertsch and Suraj Nair and Ashwin Balakrishna and Sudeep Dasari and Siddharth Karamcheti and Soroush Nasiriany and Mohan Kumar Srirama and Lawrence Yunliang Chen and Kirsty Ellis and Peter David Fagan and Joey Hejna and Masha Itkina and Marion Lepert and Yecheng Jason Ma and Patrick Tree Miller and Jimmy Wu and Suneel Belkhale and Shivin Dass and Huy Ha and Arhan Jain and Abraham Lee and Youngwoon Lee and Marius Memmel and Sungjae Park and Ilija Radosavovic and Kaiyuan Wang and Albert Zhan and Kevin Black and Cheng Chi and Kyle Beltran Hatch and Shan Lin and Jingpei Lu and Jean Mercat and Abdul Rehman and Pannag R Sanketi and Archit Sharma and Cody Simpson and Quan Vuong and Homer Rich Walke and Blake Wulfe and Ted Xiao and Jonathan Heewon Yang and Arefeh Yavary and Tony Z. Zhao and Christopher Agia and Rohan Baijal and Mateo Guaman Castro and Daphne Chen and Qiuyu Chen and Trinity Chung and Jaimyn Drake and Ethan Paul Foster and Jensen Gao and Vitor Guizilini and David Antonio Herrera and Minho Heo and Kyle Hsu and Jiaheng Hu and Muhammad Zubair Irshad and Donovon Jackson and Charlotte Le and Yunshuang Li and Kevin Lin and Roy Lin and Zehan Ma and Abhiram Maddukuri and Suvir Mirchandani and Daniel Morton and Tony Nguyen and Abigail O'Neill and Rosario Scalise and Derick Seale and Victor Son and Stephen Tian and Emi Tran and Andrew E. Wang and Yilin Wu and Annie Xie and Jingyun Yang and Patrick Yin and Yunchu Zhang and Osbert Bastani and Glen Berseth and Jeannette Bohg and Ken Goldberg and Abhinav Gupta and Abhishek Gupta and Dinesh Jayaraman and Joseph J Lim and Jitendra Malik and Roberto Martín-Martín and Subramanian Ramamoorthy and Dorsa Sadigh and Shuran Song and Jiajun Wu and Michael C. Yip and Yuke Zhu and Thomas Kollar and Sergey Levine and Chelsea Finn},
    year    = {2024},
}
```

RoboCasa

RoboCasa 是一个大规模的模拟框架，用于训练通用能力的机器人执行日常任务。它具有逼真且多样化的人本环境，特别关注厨房场景。我们借助生成式 AI 工具（如大型语言模型（LLMs）和文本到图像/3D 生成模型）创建这些环境。我们提供超过 2,500 个 3D 资产，涵盖 150 多个物体类别以及数十种可交互的家具和电器。作为首次发布的一部分，我们包含了一套 100 个任务，代表了广泛多样的日常活动。除了模拟任务外，还提供高质量的人类演示数据集，并利用自动轨迹生成技术，以较低的成本显著扩展训练数据量。

论文：https://robocasa.ai/assets/robocasa_rss24.pdf

代码：https://github.com/robocasa/robocasa

如果您使用该数据集，请查看并遵守发布方声明的开源协议 robocasa。

```
@inproceedings{robocasa2024,
 title={RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots},
 author={Soroush Nasiriany and Abhiram Maddukuri and Lance Zhang and Adeet Parikh and Aaron Lo and Abhishek Joshi and Ajay Mandlekar and Yuke Zhu},
 booktitle={Robotics: Science and Systems},
 year={2024}
}
```

LiveCodeBench

数据集描述 LiveCodeBench 是一个"实时"更新的基准测试，用于全面评估 LLMs 的代码相关能力。 特别地，它评估了 LLMs 在一系列能力上的表现，包括代码生成、自我修复、测试输出预测和代码执行。 这是 LiveCodeBench 的代码生成场景。它还用于通过测试用例反馈评估自我修复。

LiveCodeBench 的问题从竞赛编程网站收集，特别注重保持问题质量、测试用例质量和问题难度多样性。 该场景目前托管了来自 LeetCode、AtCoder 和 Codeforces 的超过 500 个问题。 每个问题实例包括问题描述、输入/输出示例和隐藏的测试用例。 此外，每个问题都标记了其难度级别和发布日期，这允许在不同的时间窗口内衡量模型性能。 目标是为每个问题实例生成正确且高效的解决方案。

初始的代码生成数据集包含了更多的测试用例，导致数据集大小显著增加。这个（精简）版本在尽量保证与原始数据集相似性能的同时，对测试进行了剪枝和采样。未来，livecodebench 将使用这个精简版本进行代码生成评估。

更新日志 由于 LiveCodeBench 是一个持续更新的基准测试，我们提供了不同版本的数据集。具体来说，我们提供以下版本的数据集：

- `release_v1`：初始发布的数据集，包含 2023 年 5 月至 2024 年 3 月期间发布的 400 个问题。
- `release_v2`：更新后的数据集，包含 2023 年 5 月至 2024 年 5 月期间发布的 511 个问题。
- `release_v3`：更新后的数据集，包含 2023 年 5 月至 2024 年 7 月期间发布的 612 个问题。
- `release_v4`：更新后的数据集，包含 2023 年 5 月至 2024 年 9 月期间发布的 713 个问题。
- `release_v5`：更新后的数据集，包含 2023 年 5 月至 2025 年 1 月期间发布的 880 个问题。

你可以使用 `version_tag` 参数来加载所需版本的数据集。此外，你还可以使用版本标签如 `v1`、`v2`、`v1_v3`、`v4_v5` 来获取特定版本中发布的问题。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope。

Math500

该数据集包含了 OpenAI 在其《让我们逐步验证》论文中创建的 MATH 基准测试中的 500 个问题子集。

请参阅 GitHub 仓库获取源文件：https://github.com/openai/prm800k/tree/main?tab=readme-ov-file#math-splits。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope

AIME2025

此数据集包含来自2025年美国数学邀请赛（AIME）I卷和II卷的问题。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope。

C-Eval

C-Eval是一个针对基础模型的综合性中文评估数据集。它由13948道多选题组成，涉及52个不同学科和四个难度等级。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

TruthfulQA

TruthfulQA 是一个用于评估语言模型在生成回答时是否真实可靠的基准数据集。该数据集包含817个问题，涵盖健康、法律、金融和政治等38个类别。这些问题设计巧妙，一些问题可能因为人们的

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope

hellaswag

HellaSwag 数据集是由 Allen Institute for AI 开发的用于评估自然语言生成模型（特别是常识推理能力）的高级基准。它通过上下文填空任务要求模型在多个迷惑性选项中选择最适合的结束句，挑战模型对语境和逻辑关系的理解。该数据集以其上下文的复杂性和高质量的生成候选答案而著称，广泛用于测试和提升自然语言处理模型的性能。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

trivia_qa

TriviaQA 是一个阅读理解数据集，包含超过65万的"问题-答案-证据"三元组。该数据集包括由谜题爱好者创作的9.5万个问题-答案对，以及为每个问题独立收集的高质量证据文档（平均每个问题有六个证据文档），提供了回答问题的可靠远程监督。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

gsm8k

GSM8K（Grade School Math 8K）是一个包含8,500个高质量且语言多样的小学数学应用题的数据集。该数据集旨在支持对基础数学问题进行多步推理的问答任务。这些问题通常需要2到8步才能解决，主要涉及使用基本算术运算（加、减、乘、除）进行一系列初等计算，以得出最终答案。一个聪明的中学生应当能够解决每一个问题：这些问题涉及的概念不超出初等代数的水平，并且绝大部分问题都无需显式定义变量即可解决。解答以自然语言提供，而非纯数学表达，以期为大型语言模型的内在思维过程提供见解。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

cmmlu

CMMLU是一个综合性的中文评估基准，专门用于评估语言模型在中文语境下的知识和推理能力。CMMLU涵盖了从基础学科到高级专业水平的67个主题。它包括：需要计算和推理的自然科学，需要知识的人文科学和社会科学,以及需要生活常识的中国驾驶规则等。此外，CMMLU中的许多任务具有中国特定的答案，可能在其他地区或语言中并不普遍适用。因此是一个完全中国化的中文测试基准。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

DuReader_robust-QG

概述： DuReader数据集属于下游question generation 问题生成类任务，可以用于训练问题生成模型，用于下游的FAQ挖掘等场景。

数据集描述： 本数据集包括训练集（14520）验证集（1417）测试集（1962）。其中，每一条数据有两个属性，分别是输入句子text1和输出句子text2。其中输入句子是答案和相关的段落，输出句子为相应的问题。

范例： {"text1":"15个[SEP]迈克尔.乔丹在NBA打了15个赛季。他在84年进入nba，期间在1993年10月6日第一次退役改打棒球，95年3月18日重新回归","text2":"乔丹打了多少个赛季"}

Clone with HTTP

- http://www.modelscope.cn/datasets/modelscope/DuReader_robust-QG.git

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope

mini_imagenet100

数据集描述 数据集简介 Mini ImageNet数据集是ImageNet大规模视觉识别挑战(ILSVRC)的子数据集，它从ImageNet1K中随机选择100个类别，其中训练集每个类别包含600张带注释的图像，共计60000张，验证集每个类别包含100张带注释的图像，共计10000张。整个数据集共6.4GB，可代替完整的ImageNet数据集，用于快速模型验证、性能评估、小数据集训练等。

数据集支持的任务 可代替完整的ImageNet数据集，用于快速模型验证、性能评估、小数据集训练等。

数据集的格式和结构 数据格式 数据集包括训练集train和验证集val，train和val文件夹之下按文件夹进行分类，共有100个子文件夹，同类别标签的图片在同一个文件夹下，图片格式为JPEG。同时包含与标注文件中label id相对应的类名文件classname.txt。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

cats_and_dogs

数据集简介 本数据集是简单的猫狗二分类数据集，共2个类别，其中训练集包含275张带注释的图像，验证集包含70张带注释的图像。整个数据集共10.3MB，可用于快速模型验证、性能评估、小数据集训练等。

数据集支持的任务 可用于快速模型验证、性能评估、小数据集训练等。

数据集的格式和结构 数据格式 数据集包括训练集train和验证集val，train和val文件夹之下按文件夹进行分类，共有2个子文件夹，同类别标签的图片在同一个文件夹下，图片格式为JPG。同时包含与标注文件中label id相对应的类名文件classname.txt。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

COCO2017

COCO2017数据集，官方地址。

数据集简介 COCO2017包含训练集118287张，验证集5000张，80类。此外，该仓库还提供了一个训练子集（25504张）。

数据集支持的任务 实例分割。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

speech_asr_aishell1_subset

数据集描述 “Aishell-1学术数据集的中文语音识别模型”的Aishell-1数据子集。

数据集简介 “Aishell-1学术数据集的中文语音识别模型”的Aishell-1数据子集。

希尔贝壳中文普通话开源语音数据库AISHELL-ASR0009-OS1录音时长178小时，是希尔贝壳中文普通话语音数据库AISHELL-ASR0009的一部分。AISHELL-ASR0009录音文本涉及智能家居、无人驾驶、工业生产等11个领域。录制过程在安静室内环境中， 同时使用3种不同设备： 高保真麦克风（44.1kHz，16-bit）；Android系统手机（16kHz，16-bit）；iOS系统手机（16kHz，16-bit）。高保真麦克风录制的音频降采样为16kHz，用于制作AISHELL-ASR0009-OS1。400名来自中国不同口音区域的发言人参与录制。经过专业语音校对人员转写标注，并通过严格质量检验，此数据库文本正确率在95%以上。分为训练集、开发集、测试集。（支持学术研究，未经允许禁止商用。）

数据集支持的任务 支持“Aishell-1学术数据集的中文语音识别模型”模型的训练和测试任务

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope

mmlu

MMLU是一项大规模多任务测试，由来自各个知识分支的多项选择题组成。测试涵盖人文、社会科学、自然科学和其他对某些人来说很重要的领域的科目。它涵盖了 57 项任务，包括初等数学、美国历史、计算机科学、法律等。为了在这项测试中获得高精度，模型必须具备广泛的世界知识和解决问题的能力。

如果您使用该数据集，请查看并遵守发布方声明的开源协议，查看详情 ModelScope。

couplet_generation

概述 中文对联生成数据集，适用于训练对联生成模型。

数据集描述 该数据集包含训练集（676460条）与测试集（1673条）。每条数据具备一个特征，即输入句子`text1`，由对联构成。

示例

```
{"text1": "晚风摇树树还挺 晨露润花花更红"}
```

通过HTTP克隆

- http://www.modelscope.cn/datasets/lcl193798/couplet_generation.git

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope.

ChartQA

ChartQA

ChartQA: 一个结合视觉与逻辑推理的图表问题回答基准测试。

论文链接：ChartQA

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope

```
@inproceedings{masry-etal-2022-chartqa,
  title = "{C}hart{QA}: A Benchmark for Question Answering about Charts with Visual and Logical Reasoning",
  author = "Masry, Ahmed  and
    Long, Do  and
    Tan, Jia Qing  and
    Joty, Shafiq  and
    Hoque, Enamul",
  booktitle = "Findings of the Association for Computational Linguistics: ACL 2022",
  month = may,
  year = "2022",
  address = "Dublin, Ireland",
  publisher = "Association for Computational Linguistics",
  url = "https://aclanthology.org/2022.findings-acl.177",
  doi = "10.18653/v1/2022.findings-acl.177",
  pages = "2263--2279",
}
```

AIME2024

此数据集包含来自2024年AIME I和AIME II测试的30道题目。原始来源是AI-MO/aimo-validation-aime ，该来源包含了从2022年至2024年AIME的90道题目。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope。

Physical AI Smart Spaces

用于多摄像头追踪和 2D/3D 物体检测的全面且带注释的数据集。该数据集由 Omniverse 合成生成。

该数据集包含来自近 1,500 个摄像头的超过 250 小时的视频，这些摄像头来自仓库、医院、零售店等室内场景。该数据集采用时间同步技术，使用特征表示法跨多个摄像头追踪人体，无需个人数据。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
 @InProceedings{Tang25AICity25,
author = {Zheng Tang and Shuo Wang and David C. Anastasiu and Ming-Ching Chang and Anuj Sharma and Quan Kong and Norimasa Kobori and Munkhjargal Gochoo and Ganzorig Batnasan and Munkh-Erdene Otgonbold and Fady Alnajjar and Jun-Wei Hsieh and Tomasz Kornuta and Xiaolong Li and Yilin Zhao and Han Zhang and Subhashree Radhakrishnan and Arihant Jain and Ratnesh Kumar and Vidya N. Murali and Yuxing Wang and Sameer Satish Pusegaonkar and Yizhou Wang and Sujit Biswas and Xunlei Wu and Zhedong Zheng and Pranamesh Chakraborty and Rama Chellappa},
title = {The 9th {AI} {C}ity {C}hallenge},
booktitle = {Proc. ICCV Workshops},
address = {Honolulu, HA, USA},
year = {2025}
}
@inproceedings{Wang24AICity24,
author = {Shuo Wang and David C. Anastasiu and Zheng Tang and Ming-Ching Chang and Yue Yao and Liang Zheng and Mohammed Shaiqur Rahman and Meenakshi S. Arya and Anuj Sharma and Pranamesh Chakraborty and Sanjita Prajapati and Quan Kong and Norimasa Kobori and Munkhjargal Gochoo and Munkh-Erdene Otgonbold and Ganzorig Batnasan and Fady Alnajjar and Ping-Yang Chen and Jun-Wei Hsieh and Xunlei Wu and Sameer Satish Pusegaonkar and Yizhou Wang and Sujit Biswas and Rama Chellappa},
title = {The 8th {AI} {C}ity {C}hallenge},
booktitle = {Proc. CVPR Workshops},
pages = {7261--7272},
address = {Seattle, WA, USA},
year = {2024}
}
@misc{Wang24BEVSUSHI,
author = {Yizhou Wang and Tim Meinhardt and Orcun Cetintas and Cheng-Yen Yang and Sameer Satish Pusegaonkar and Benjamin Missaoui and Sujit Biswas and Zheng Tang and Laura Leal-Taix{\'e}},
title = {{BEV-SUSHI}: {M}ulti-target multi-camera {3D} detection and tracking in bird's-eye view},
note = {arXiv:2412.00692},
year = {2024}
}
```

PhysicalAI-Robotics-GR00T-GR1

GR1-100 包含 92 个视频，从第三人称视角展示 Fourier GR1-T2 机器人在实验室中执行各种任务。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-Robotics-GR00T-Teleop-G1

PhysicalAI-Robotics-GR00T-Teleop-G1 数据集包含 1000 条基于 Unitree G1 的真实机器人遥操作轨迹数据，采用上肢控制。机器人根据语言提示选择正确的水果并将其放置在盘子上。共使用 4 种水果：苹果、梨、杨桃、葡萄。机器人配备默认的 RealSense 摄像头和一对 Unitree G1 三指机械手。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-Robotics-GR00T-Teleop-Sim

PhysicalAI-Robotics-GR00T-Teleop-GR1 数据集包含 1000 条远程操作轨迹，这些轨迹是使用具有上肢控制功能的 GR1 机器人进行模拟的。该模拟设置模拟桌面操作任务，并使用虚拟摄像头进行 RGB 观测。该机器人配备了模拟傅里叶手。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-DigitalCousin-Assets

GR1 模拟桌面任务资产由一系列数字资产组成，包括 3D 网格、纹理和对象元数据，旨在用于涉及 GR1 机器人的模拟桌面操作环境。这些资产作为视觉和物理组件（例如杯子、瓶子、碗、容器）填充虚拟场景，从而实现逼真的交互和机器人策略的基准测试。

虽然这些资产不是演示或传感器数据的数据集，但它们对于构建和运行模拟 GR1 桌面任务环境至关重要。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

Cosmos-Drive-Dreams

Cosmos-Drive-Dreams数据集包含来自 RDS-HQ 数据集的**5,843 个**10 秒视频片段的标签，以及Cosmos-Drive-Dreams 根据这些标签生成的**81,802 个**合成视频样本。合成生成的视频长 121 帧，捕捉了各种具有挑战性的场景，例如雨天、雪天、雾天等，这些场景在现实世界的驾驶数据集中可能并不容易获得。该数据集仅供商业/非商业自动驾驶汽车使用。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
@misc{nvidia2025cosmosdrivedreams,
 title     = {Cosmos-Drive-Dreams: Scalable Synthetic Driving Data Generation with World Foundation Models},
 author = {Ren, Xuanchi and Lu, Yifan and Cao, Tianshi and Gao, Ruiyuan and
        Huang, Shengyu and Sabour, Amirmojtaba and Shen, Tianchang and
        Pfaff, Tobias and Wu, Jay Zhangjie and Chen, Runjian and
        Kim, Seung Wook and Gao, Jun and Leal-Taixe, Laura and
        Chen, Mike and Fidler, Sanja and Ling, Huan}
 year      = {2025},
 url       = {https://arxiv.org/abs/2506.09042}
}
```

PhysicalAI-GR00T-Tuned-Tasks

该数据集是 Isaac Lab 生成的多模态轨迹集合。它支持工业环境下的人形机器人 (GR1) 桌面操控任务。每个数据集条目都提供了训练和评估通用机器人策略（例如倒坚果或按颜色对管道进行分类）所需的完整上下文（状态、视觉、语言、动作）。

该数据集非常适合行为克隆、策略学习和通用机器人操作研究。它已用于 GR00T N1 模型的后期训练。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
[1] @inproceedings{mandlekar2023mimicgen，title={MimicGen：一种使用人类演示进行可扩展机器人学习的数据生成系统}，author={Mandlekar, Ajay 和 Nasiriany, Soroush 和 Wen, Bowen 和 Akinola, Iretiayo 和 Narang, Yashraj 和 Fan, Linxi 和 Zhu, Yuke 和 Fox, Dieter}，booktitle={第七届机器人学习年会}，year={2023} }
```

PhysicalAI-Robotics-Manipulation-Augmented

这是一个完全注释的合成数据集，包含 1,000 个示例，演示了单个 Franka Panda 机械臂在 Isaac Lab 中执行固定顺序的三块立方体堆叠任务。机器人始终按照以下顺序堆叠立方体：蓝色（底部）→红色（中间）→绿色（顶部）。

数据集是使用以下管道生成的：

- 收集了 10 个堆叠任务的人类远程操作演示。
- 使用 Isaac Lab 的**Mimic**工具 [1] 在 Isaac Sim 中模拟了 1,000 条高质量轨迹。
- 应用**Cosmos Transfer1**模型 [2]，通过照片级真实感域自适应增强桌面相机的 RGB 视觉效果。

每个演示都包括同步的多模式数据：

- 来自桌面和腕戴式摄像机的 RGB 视频。
- 来自桌面相机的深度、分割和表面法线图。
- 完整的低级机器人和物体状态（关节、末端执行器、夹持器、立方体姿势）。
- 机器人执行的动作序列。

该数据集非常适合行为克隆、策略学习和通用机器人操作研究。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
[1] @inproceedings{mandlekar2023mimicgen,
   title={MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations},
   author={Mandlekar, Ajay and Nasiriany, Soroush and Wen, Bowen and Akinola, Iretiayo and Narang, Yashraj and Fan, Linxi and Zhu, Yuke and Fox, Dieter},
   booktitle={7th Annual Conference on Robot Learning},
   year={2023}
   }
[2] @misc{nvidia2025cosmostransfer1conditionalworldgeneration,
      title = {Cosmos-Transfer1: Conditional World Generation with Adaptive Multimodal Control},
      author = {NVIDIA and Abu Alhaija, Hassan and Alvarez, Jose and Bala, Maciej and Cai, Tiffany and Cao, Tianshi and Cha, Liz and Chen, Joshua and Chen, Mike and Ferroni, Francesco and Fidler, Sanja and Fox, Dieter and Ge, Yunhao and Gu, Jinwei and Hassani, Ali and Isaev, Michael and Jannaty, Pooya and Lan, Shiyi and Lasser, Tobias and Ling, Huan and Liu, Ming-Yu and Liu, Xian and Lu, Yifan and Luo, Alice and Ma, Qianli and Mao, Hanzi and Ramos, Fabio and Ren, Xuanchi and Shen, Tianchang and Tang, Shitao and Wang, Ting-Chun and Wu, Jay and Xu, Jiashu and Xu, Stella and Xie, Kevin and Ye, Yuchong and Yang, Xiaodong and Zeng, Xiaohui and Zeng, Yu},
      journal = {arXiv preprint arXiv:2503.14492},
      year = {2025},
      url = {https://arxiv.org/abs/2503.14492}
   }
```

PhysicalAI-Robotics-GR00T-Eval

EVAL-175拥有在实验室中执行各种任务之前，从机器人的角度拍摄的 123 张初始帧图片。主要用于机器人技术的训练图像数据集。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-Robotics-GR00T-X-Embodiment-Sim

PhysicalAI-Robotics-GR00T-X-Embodiment-Sim是NVIDIA提供的用于 GR00T N1 后期训练的数据集。每个数据集都是来自不同机器人实施例和任务的轨迹的集合。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-SimReady-Warehouse-01

PhysicalAI-SimReady-Warehouse-01是NVIDIA推出的面向物理AI开发者的商业级数据集，专门针对仓库环境设计。该数据集包含22个项目的丰富内容，经过最新更新，为开发者提供了高质量的仿真就绪数据资源，适用于物理AI在仓储场景中的训练与测试需求。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-Robotics-Manipulation-Objects

PhysicalAI-Robotics-Manipulation-Objects 是一个自动生成机器人动作的数据集，用于在厨房环境中执行诸如拾取和放置物体等操作。该数据集是在 IsaacSim 中生成的，利用推理算法和基于优化的运动规划来自动找到任务的解决方案 。该数据集包含一个由 Kinova Gen3 机械臂构建的双手机械手。环境是厨房场景，其中的家具和电器都是程序生成的 。此数据集仅用于研究和开发。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
[1] @inproceedings{garrett2020pddlstream,
 title={Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning},
 author={Garrett, Caelan Reed and Lozano-P{\'e}rez, Tom{\'a}s and Kaelbling, Leslie Pack},
 booktitle={Proceedings of the international conference on automated planning and scheduling},
 volume={30},
 pages={440--448},
 year={2020}
}
[2] @article{Eppner2024,
  title = {scene_synthesizer: A Python Library for Procedural Scene Generation in Robot Manipulation},
  author = {Clemens Eppner and Adithyavairavan Murali and Caelan Garrett and Rowland O'Flaherty and Tucker Hermans and Wei Yang and Dieter Fox},
  journal = {Journal of Open Source Software}
  publisher = {The Open Journal},
  year = {2024},
  Note = {\url{https://scene-synthesizer.github.io/}}
}
[3] @inproceedings{curobo_icra23,
    author={Sundaralingam, Balakumar and Hari, Siva Kumar Sastry and
        Fishman, Adam and Garrett, Caelan and Van Wyk, Karl and Blukis, Valts and
        Millane, Alexander and Oleynikova, Helen and Handa, Ankur and
        Ramos, Fabio and Ratliff, Nathan and Fox, Dieter},
    booktitle={2023 IEEE International Conference on Robotics and Automation (ICRA)},
    title={CuRobo: Parallelized Collision-Free Robot Motion Generation},
    year={2023},
    volume={},
    number={},
    pages={8112-8119},
    doi={10.1109/ICRA48891.2023.10160765}
}
```

PhysicalAI-Robotics-Manipulation-SingeArm

PhysicalAI-Robotics-Manipulation-SingeArm 是一组自动生成动作的数据集，用于 Franka Panda 机器人执行诸如堆叠积木、打开橱柜和抽屉等操作。该数据集是在 IsaacSim 中生成的，利用任务和运动规划算法自动找到任务的解决方案 。环境是桌面场景，其中对象布局和资产纹理由程序生成 。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace


```
[1] @inproceedings{garrett2020pddlstream,
  title={Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning},
  author={Garrett, Caelan Reed and Lozano-P{\'e}rez, Tom{\'a}s and Kaelbling, Leslie Pack},
  booktitle={Proceedings of the international conference on automated planning and scheduling},
  volume={30},
  pages={440--448},
  year={2020}
}
[2] @article{Eppner2024,
  title = {scene_synthesizer: A Python Library for Procedural Scene Generation in Robot Manipulation},
  author = {Clemens Eppner and Adithyavairavan Murali and Caelan Garrett and Rowland O'Flaherty and Tucker Hermans and Wei Yang and Dieter Fox},
  journal = {Journal of Open Source Software}
  publisher = {The Open Journal},
  year = {2024},
  Note = {\url{https://scene-synthesizer.github.io/}}
}
[3] @inproceedings{curobo_icra23,
  author={Sundaralingam, Balakumar and Hari, Siva Kumar Sastry and
      Fishman, Adam and Garrett, Caelan and Van Wyk, Karl and Blukis, Valts and
      Millane, Alexander and Oleynikova, Helen and Handa, Ankur and
      Ramos, Fabio and Ratliff, Nathan and Fox, Dieter},
  booktitle={2023 IEEE International Conference on Robotics and Automation (ICRA)},
  title={CuRobo: Parallelized Collision-Free Robot Motion Generation},
  year={2023},
  volume={},
  number={},
  pages={8112-8119},
  doi={10.1109/ICRA48891.2023.10160765}
}
```


PhysicalAI-Robotics-GraspGen

GraspGen 是用于多种机器人实施例和抓取器的大规模模拟抓取数据集。

发布了超过 5700 万个抓取数据，这些数据来自Objaverse XL(LVIS) 数据集中 8515 个物体的子集。这些抓取数据分别针对三款抓取器：Franka Panda、Robotiq-2f-140 工业抓取器以及单接触式吸盘抓取器（半径 30 毫米）。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-Robotics-Manipulation-Kitchen

PhysicalAI-Robotics-Manipulation-Kitchen 是一个自动生成机器人动作的数据集，这些机器人执行诸如打开和关闭橱柜、抽屉、洗碗机和冰箱等操作。该数据集是在 IsaacSim 中生成的，利用推理算法和基于优化的运动规划来自动找到任务的解决方案 [1, 3]。该数据集包含一个由 Kinova Gen3 机械臂构建的双手机械手。环境是厨房场景，其中的家具和电器都是程序生成的 [2]。该数据集可供商业使用。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
[1] @inproceedings{garrett2020pddlstream,
  title={Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning},
  author={Garrett, Caelan Reed and Lozano-P{\'e}rez, Tom{\'a}s and Kaelbling, Leslie Pack},
  booktitle={Proceedings of the international conference on automated planning and scheduling},
  volume={30},
  pages={440--448},
  year={2020}
}
[2] @article{Eppner2024,
  title = {scene_synthesizer: A Python Library for Procedural Scene Generation in Robot Manipulation},
  author = {Clemens Eppner and Adithyavairavan Murali and Caelan Garrett and Rowland O'Flaherty and Tucker Hermans and Wei Yang and Dieter Fox},
  journal = {Journal of Open Source Software}
  publisher = {The Open Journal},
  year = {2024},
  Note = {\url{https://scene-synthesizer.github.io/}}
}
[3] @inproceedings{curobo_icra23,
  author={Sundaralingam, Balakumar and Hari, Siva Kumar Sastry and
    Fishman, Adam and Garrett, Caelan and Van Wyk, Karl and Blukis, Valts and
    Millane, Alexander and Oleynikova, Helen and Handa, Ankur and
    Ramos, Fabio and Ratliff, Nathan and Fox, Dieter},
  booktitle={2023 IEEE International Conference on Robotics and Automation (ICRA)},
  title={CuRobo: Parallelized Collision-Free Robot Motion Generation},
  year={2023},
  volume={},
  number={},
  pages={8112-8119},
  doi={10.1109/ICRA48891.2023.10160765}
}
```

GalaxeaOpenWorld

简介

Galaxea开放世界数据集是一个在真实人类生活与工作环境中记录的大规模、多样化机器人行为集合。所有演示数据均采用统一的机器人实体进行采集，并配有精确的子任务级语言标注，以同时支持训练与评估。基于此数据集，提出G0双系统框架：通过视觉语言模型（VLM）进行多模态规划，并与视觉-语言-动作模型（VLA）协同实现细粒度执行。G0采用三阶段课程训练：跨实体预训练、单实体预训练和任务特定后训练。通过涵盖桌面操作、少样本学习和长程移动操作的综合基准测试，方法展现出显著有效性。特别值得注意的是，单实体预训练阶段与Galaxea开放世界数据集的结合对实现强劲性能具有关键作用。本工作聚焦机器人视觉-语言-动作（VLA）模型的落地痛点——大规模高质量开放世界机器人数据稀缺与模型泛化能力不足，提出两大核心成果：

- Galaxea Open-World Dataset：首个在真实人类生活/工作场景中采集的大规模机器人行为数据集，解决现有数据集场景单一、本体不一致、标注粗糙的问题；
- G0双系统框架：耦合“慢思考”的视觉-语言模型（G0-VLM）与“快执行”的VLA模型（G0-VLA），通过三阶段训练策略实现复杂任务的规划与精准执行，在桌面操作、少样本学习、长程移动操作等基准测试中表现优异。

主要功能

500+小时的真实世界移动操作数据

所有数据均使用统一的机器人形态收集以确保一致性。

细粒度的子任务语言标注。

涵盖住宅、厨房、零售和办公环境。

数据集以 RLDS 和 LeRobot 格式提供。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

```
@article{galaxea2025,
 title={Galaxea G0: Open-World Dataset and Dual-System VLA Model},
 author={Galaxea Team},
 journal={arXiv preprint arXiv:2509.00576},
 year={2025}
}
```

RealOmni

10Kh RealOmni-Open DataSet该数据集累计拥有超10000小时数据、100万+片段，完整记录复杂家务和清洁的全流程，采用GenDAS Gripper完成数据采集,是行业内最大的开源具身智能数据集。

相比其他数据集的优势

1. **充足的数据量 & 强泛化性**

每项技能都有充足数据支撑，数据采集自3000+家庭、近10000个不同细粒度目标，避免简单重复，确保鲁棒的泛化能力。

2. **真实场景 & 聚焦核心技能**

数据源自真实家庭的自然操作，避免损害质量的技能碎片化问题，聚焦10个核心家庭场景、30项核心技能。

3. **双机械臂 & 长时长任务**

完整记录复杂家务和清洁的全流程，采用GenDAS Gripper完成数据采集。

4. **多模态 & 高质量数据**

包含大视场原始图像、轨迹、标注和关节运动数据；轨迹重建精度与质量达到行业领先水平。

数据集统计 |属性|数值| |-|-| |片段时长中位数|210.0 秒| |存储大小|95 TB| |格式|mcap| |分辨率|1600*1296| |帧率|30 fps| |相机类型|大视场鱼眼相机| |IMU|支持（6轴）| |触觉阵列空间|支持| |阵列空间分辨率|1 mm| |设备|Gen DAS Gripper|

第一阶段内容

> 我们已上传第一阶段数据（仅占总量一小部分），剩余技能的更新将尽快完成。 第一阶段覆盖4大类场景任务下的12项技能，总时长950小时、片段数39761个、存储大小3.45TB。

任务类别	技能名称
衣物折叠与拉链操作	fold_and_store_clothes（折叠收纳衣物）
	zip_clothes（拉合衣物拉链）
烹饪与厨房清洁	clean_container（清洁容器）
	unscrew_bottle_cap_and_pour（拧瓶盖并倾倒）
	clean_bowl（清洁碗具）
杂物整理	desktop_object_sorting（桌面物品整理）
	fold_towel（折叠毛巾）
	fold_and_store_shopping_bag（折叠收纳购物袋）
	drawer_to_take_items（抽屉取物）
	drawer_to_place_items（抽屉放物）
鞋子处理	lace_up_shoes_with_both_hands（双手系鞋带）
	organize_scattered_shoes（整理散落鞋子）

我们会在主流社交平台同步更新进度。除数据外，还提供格式转换、使用指导等配套支持，相关工具链接：https://github.com/genrobot-ai/das-datakit

联系我们 使用过程中如有任何问题、建议，或希望新增数据采集的场景/技能，欢迎联系我们，共同建设"数字化人类所有技能"的项目：

- 邮箱：opendata@genrobot.ai
- 微信公众号、小红书：简智机器人
- X（原Twitter）：https://x.com/GenrobotAI
- 领英（LinkedIn）：https://www.linkedin.com/company/108767412/admin/dashboard/

数据集结构 mcap文件存储在文件目录结构的最终叶子文件夹中。注：每个mcap文件代表一份任务数据。

数据格式 双机械臂任务中：robot0、robot1 分别代表左、右抓手，每个抓手包含以下主题（Topics）：

```
 /robot0/sensor/camera0/compressed   # 鱼眼相机图像数据 — H.264格式压缩编码
/robot0/sensor/camera0/camera_info  # 鱼眼相机内外参
/robot0/sensor/imu                  # 惯性测量单元（IMU）数据
/robot0/sensor/magnetic_encoder     # 磁编码器数据：抓手开合距离
/robot0/vio/eef_pose                # 轨迹数据
```

所有主题通过Protobuf序列化进行持久化存储。

/robot0/sensor/camera0/compressed 协议定义

```
 // 压缩图像
message CompressedImage {
 // 图像时间戳
 google.protobuf.Timestamp timestamp = 1;

 // 帧ID
 string frame_id = 4;

 // 压缩图像数据（H264视频流）
 bytes data = 2;

 // 图像格式
 // 支持值：`webp`, `jpeg`, `png`, `h264`
 string format = 3;

 // 通用头部（包含时间戳）
 Header header = 8;
}

message Header {
  string module_name = 1;       // 模块名
  uint32 sequence_num = 2;      // 序列号
  uint64 timestamp = 3;         // 时间戳
  string topic_name = 4;        // 主题名
  double expect_hz = 5;         // 期望帧率
  repeated Input inputs = 6;    // 输入列表
}
```

RealSource World

RealSource World 是一个大规模的真实世界机器人操作数据集，使用 RS-02 双臂人形机器人采集。该数据集包含在真实环境中执行的各种长距离操作任务，并对每个操作的原子技能和质量评估进行了详细的标注。

主要特点

- **超过 1400 万**帧真实世界的双臂操作演示。
- **超过11,428**集，涵盖**35 项**不同的操控任务。
- **71维本体**感觉状态空间，包括关节位置、速度、加速度、力、扭矩和末端执行器姿态。
- **多摄像头**视觉观察（头部摄像头、左手摄像头、右手摄像头），分辨率为 720×1280，帧率为 30 FPS。
- 对每一集进行**精细的标注，包括原子技能分割和质量评估。**
- **场景多样，**包括厨房、会议室、便利店和家庭环境。
- **双臂协调**任务，展现复杂的双手操作技能。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

```
@misc{realsourceworld,
 title={RealSource World: A Large-Scale Real-World Dual-Arm Manipulation Dataset},
 author={RealSource},
 howpublished={\url{https://huggingface.co/datasets/RealSourceData/RealSource-World}},
 year={2025}
}
```

RoboMIND2.0

RoboMIND2.0 已发布！此版本在 RoboMIND 的基础上进行了重大升级。

我们聚焦于社区高度关注的**双臂操作**和**移动操作**场景中的复杂数据。RoboMIND2.0 引入了许多实用特性，以进一步推动机器人学习与具身智能研究的发展。

主要升级

- RoboMIND2.0 扩展至 **31 万条轨迹**，总计超过 **1,000 小时**的数据。
- RoboMIND2.0 包含 **1.2 万条触觉增强序列**。
- RoboMIND2.0 收集了 **2 万条移动操作轨迹**。
- RoboMIND2.0 覆盖 **6 种主流机器人本体**。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 ModelScope

```
@misc{hou2025robomind20multimodalbimanual,
      title={RoboMIND 2.0: A Multimodal, Bimanual Mobile Manipulation Dataset for Generalizable Embodied Intelligence},
      author={Chengkai Hou and Kun Wu and Jiaming Liu and Zhengping Che and Di Wu and Fei Liao and Guangrun Li and Jingyang He and Qiuxuan Feng and Zhao Jin and Chenyang Gu and Zhuoyang Liu and Nuowei Han and Xiangju Mi and Yaoxu Lv and Yankai Fu and Gaole Dai and Langzhe Gu and Tao Li and Yuheng Zhang and Yixue Zhang and Xinhua Wang and Shichao Fan and Meng Li and Zhen Zhao and Ning Liu and Zhiyuan Xu and Pei Ren and Junjie Ji and Haonan Liu and Kuan Cheng and Shanghang Zhang and Jian Tang},
      year={2025},
      eprint={2512.24653},
      archivePrefix={arXiv},
      url={https://arxiv.org/abs/2512.24653},
}
```

Gen-HumanEgo



1,800+ 小时第一人称人类示范数据，提供时间同步的结构化监督信号，面向具身智能与机器人学习。

Gen-HumanEgo 收录了真实世界中的第一人称人类示范，涵盖不同任务、采集者、环境以及同一活动的不同执行方式。所有数据均使用统一的六相机 DAS-Ego 设备采集。GenRobot 的 **Data Foundation Model（DFM，数据基础模型）** 对录制数据进行处理，生成面向人体运动、空间理解和任务理解的互补训练信号。

数据集包含：

- **第一人称 RGB：** 同步的多视角视频。
- **3D 手部重建：** 3D 手部关键点、MANO 参数和完整手部几何信息。
- **空间理解：** 大视场角 Ego-Depth 深度数据。
- **语义理解：** 包含时间边界的视频级、任务级和子任务级层级语义标注。

数据集概览 |属性|数值| |-|-| |总时长|**1,847.7 小时**| |Episode 数量|**44,632**| |唯一任务数量|**10,257**| |领域|家庭、商业、工业、农业| |原生 RGB 视角|**6 路同步相机**| |分辨率|**1600 × 1300**| |帧率|**30 FPS**| |文件格式|MCAP| |模态|手部21个关键点，完整手部几何模型，深度图，任务和子任务标注|

仓库目录结构 目录按照数据集的场景／技能层级组织。最后两层分别是由文件名前两个字符组成的目录，以及对应 Episode 的 MCAP 文件。

```
Gen-HumanEgo/
├── README.md
├── egov4_urdf.zip
├── assets/
│   └── ...
└── data/
    ├── industry/
    │   └── logistics/
    │       └── sorting_and_packing/
    │           └── daily_work/
    │               └── 00/
    │                   └── 005c469cf7424c4c87054569b2234ebc.mcap
    ├── domestic_services/
    │   └── living_room/
    │       └── clothing_organization/
    │           └── iron_clothes/
    │               └── c0/
    │                   └── c096609eca54434e9fa93b0fff55ce3d.mcap
    └── business/
        └── restaurant/
            └── bakery/
                └── daily_work/
                    ├── a9/
                    │   └── a91e57586c33437292d763fb8fd3777c.mcap
                    └── 10/
                        └── 101207375b8e41fea31bacd94db480e6.mcap
```

License:cc-by-sa-4.0

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace

PhysicalAI-Human-1M-100K

PhysicalAI-Human-1M-100K 概述

PhysicalAI-Human-1M-100K 是 NVIDIA 发布的第一视角人类操作数据集，是约 100 万小时 Human-1M 数据集的开源子集，包含约 10 万小时、3,405,573 条 episode，涵盖 406,815 个任务簇、81,557 个场景和 9 类环境。

数据集集成了第一视角视频、稠密语言标注以及头部和手部运动数据。视频包括单目和双目头戴视角，部分 episode 配有手腕相机；运动数据包括头戴相机 6-DoF 位姿和每只手 21 个关键点。

Human-1M 已用于训练 NVIDIA 的 Cosmos 和 GR00T 模型，可用于商业和非商业用途。

Episode 结构

每条 episode 由熟练操作者完成一项目标明确的实际工作，通常在 5–10 分钟内完成，视频不低于 30 FPS / 480p。标注分为两级：T1 用 1–2 句话概述任务目标，T2 将 episode 切分为 1–10 秒的连续动作片段，适合训练任务理解和细粒度动作执行能力。

任务类别

任务覆盖备餐、上货、打包、零件组装、设备维修等具有经济价值的日常工作，例如缝纫、维修摩托车刹车、包饺子、采摘番茄、固定电路板并接线等，环境包括工业车间、商用厨房、物流场地、农业和家庭等。

场景多样性

数据采用真实场景采集和工作室采集两种方式：前者在家庭和工作场所中进行，覆盖自然的环境与操作差异；后者在受控环境中进行，便于针对特定技能重复生产数据。

数据描述

数据经过标注、任务、多样性和 3D 跟踪四道质量关卡，结合人工审核与定量检查。数据以 LeRobot v3.0 格式发布，视频、语言标注和运动数据通过帧时间戳对齐，可直接用于机器人学习训练。

如果您使用该数据集，请查看并遵守发布方声明的开源协议 HuggingFace。

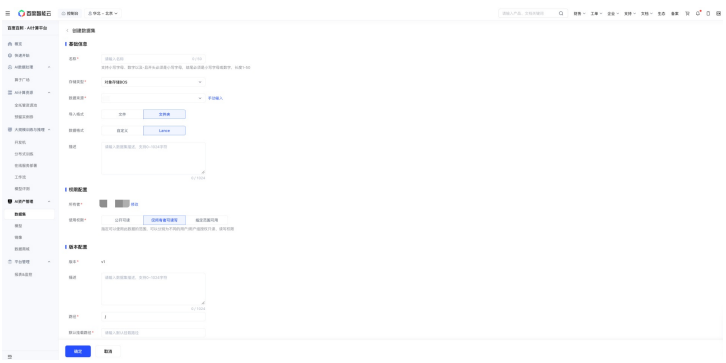
```
@misc{nvidia2026human1m,
        title={Human-1M: Scaling Physical AI through Human Experience},
        author={{NVIDIA}},
        howpublished={Hugging Face Datasets, nvidia/PhysicalAI-Human-1M-100K},
        year={2026},
        publisher={Hugging Face},
        url={https://huggingface.co/datasets/nvidia/PhysicalAI-Human-1M-100K}
}
```

自定义 Lance 格式数据集

在进行模型开发、训练、推理之前，一般需要准备好指定的数据集。自定义数据集支持将对象存储（BOS）中的 Lance 格式目录注册为数据集，提供数据集详情、版本列表、文件管理、数据详情、SQL 查询、转储任务等管理能力，支持图片、音频、视频等多模态二进制列的在线预览。

创建数据集

在自定义数据集页签下单击【创建数据集】。

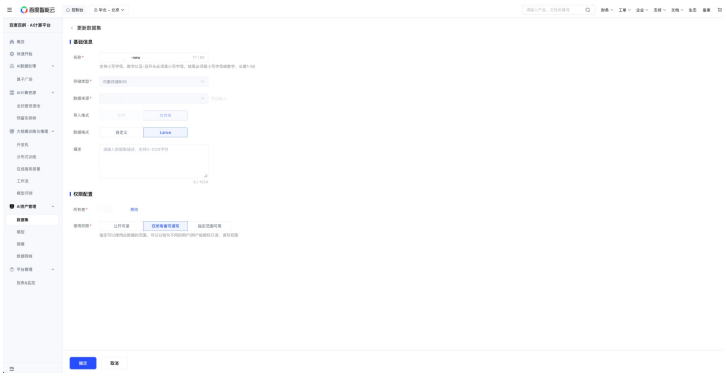


字段	说明
名称	必填，数据集的名称。支持小写字母、数字以及 -，且开头必须是小写字母，结尾必须是小写字母或数字，长度 1~50
存储类型	必选，Lance 数据集选择对象存储BOS
数据来源	必选，选择 Lance 数据集所在的 BOS 存储桶；桶不在列表中时可单击【手动输入】
导入格式	必选，Lance 数据集选择文件夹。文件夹导入的存储路径可被挂载在容器中，常用于训练、推理或开发机的数据卷
数据格式	必选，Lance 数据集选择 Lance。选择自定义则按通用文件数据集处理，不解析表结构
描述	选填，对数据集的描述信息，支持 0~1024 字符
所有者	必选，对数据集有管理权限（编辑、删除等），创建时可修改
使用权限	必选，有三种数据权限可选：公开可读、仅所有者可读写、指定范围可用。
版本	必选，系统自动生成，首个版本为 v1
版本描述	选填，对版本的描述，支持 0~1024 字符
路径	必选，数据集在存储中对应的路径，如 `/lance-xxxx.lance`
默认挂载路径	必选，数据集挂载到负载上时的挂载路径，在挂载时可以修改

填写完成后单击【确定】进行提交。

编辑数据集

在数据集列表中找到目标数据集，在操作列选择编辑，进入【更新数据集】页面。



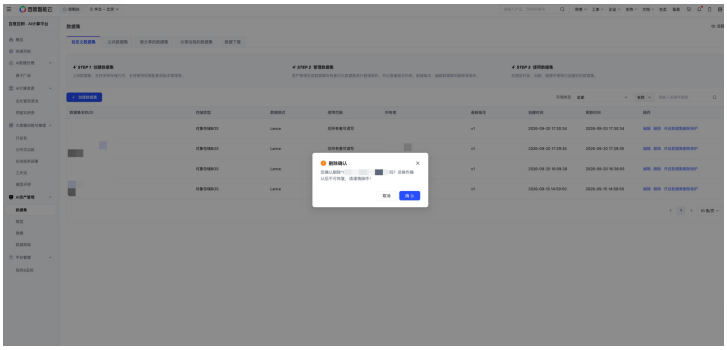
可编辑项：名称、描述、所有者、使用权限。

字段	说明
名称	可修改，命名规则同创建
存储类型 / 数据来源 / 导入格式 / 数据格式	创建后锁定，不可修改；如需指向新的存储路径，请重建数据集
描述	可修改，支持 0~1024 字符
所有者	可修改
使用权限	可修改

修改完成后单击【确定】生效。

删除数据集

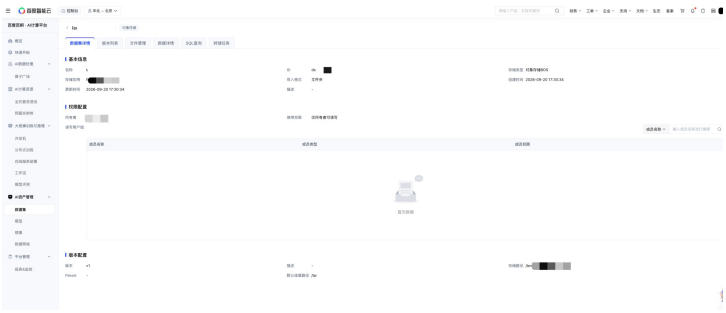
在数据集列表中找到目标数据集，在操作列选择删除，按提示确认即可。



> 说明：删除数据集时会删除数据集下的所有版本信息，但**存储中的数据不会被删除**，BOS 路径下的 Lance 文件予以保留。如需彻底清理，请前往 BOS 控制台手动删除对应目录。

数据集详情

进入数据集后默认展示【数据集详情】页签，包含基本信息、权限配置、版本配置三部分。



分组	字段	说明
基本信息	名称 / ID	数据集名称与系统分配的唯一 ID，API 调用时使用 ID
基本信息	存储实例	数据集所在的 BOS 存储桶
基本信息	存储类型	对象存储BOS
基本信息	导入格式	文件夹
基本信息	创建时间 / 更新时间	数据集的创建与最近更新时间
基本信息	描述	数据集描述，未填写显示 -
权限配置	所有者	数据集所有者
权限配置	使用权限	公开可读 / 仅所有者可读写 / 指定范围可用
版本配置	版本	当前版本号，如 v1
版本配置	存储路径	版本对应的 BOS 路径
版本配置	默认挂载路径	挂载到负载时的默认路径
版本配置	描述 / Fileset	版本描述与文件集信息，未填写显示 -

版本列表

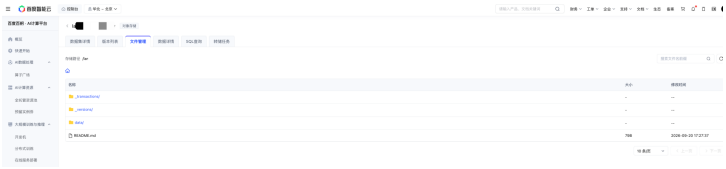
【版本列表】页签展示该数据集的全部版本，包含版本号、存储路径、默认挂载路径、创建时间，以及【导入数据】操作入口。



列	说明
版本号	版本标识，如 v1，单击可查看版本详情
存储路径	该版本对应的 BOS 路径
默认挂载路径	该版本挂载到负载时的默认路径
创建时间	版本创建时间
操作-导入数据	将其他数据集的数据导入

文件管理

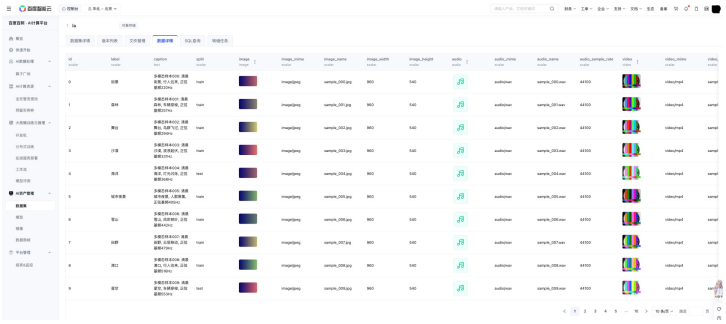
【文件管理】页签以浏览器方式展示 BOS 路径内的文件与目录，支持按文件名搜索、分页浏览。



内容	说明
存储路径	当前浏览的 BOS 路径
_transactions/	Lance 写入事务记录目录
_versions/	Lance 版本清单目录，含 manifest 与最新版本指针文件
data/	Lance 数据文件目录（*.lance）
README.md	说明文档，单击可在线查看
名称 / 大小 / 修改时间	文件与目录的基本属性；目录的大小与修改时间显示为 - / --

数据详情

【数据详情】页签以表格形式展示 Lance 表内容，支持分页浏览、多模态列在线预览。



表格与分页

- 表头第一行为列名，第二行为该列的模态标注（image / audio / video 等）。
- 文本、数值列直接展示单元格内容。
- 右下角支持翻页与每页条数切换（如 10 条/页）。

各模态列的实际渲染行为

模态	列表页表现
image	前端可直接展示；点击放大再取原图。缩略图生成失败时显示图片占位图标；图片源文件超过 32MB 时不返回缩略图
video	内联首帧封面（字段与格式同图片）。视频源文件超过 32MB 时不生成首帧封面预览图
audio	仅显示音频图标占位，不读取内容；点击播放

列模态与预览

二进制列能否预览，取决于该列是否带有模态标注。单击列头右侧的 ⋮ 菜单，选择【修改模态】，可将列标注为图像、音频、视频、文本，或【重置】恢复未标注状态。控制台修改模态会写回字段元数据，即时生效。

推荐在写入时即声明（见「写入规范」一节），注册后打开即预览，无需手动标注。

> 说明：若二进制列未声明模态且未手动标注，单元格显示为占位图标，这是未配置的表现，并非数据损坏。可随时通过【修改模态】补标，即时生效。

写入规范：模态元数据声明

建表 schema 时给每个字段写 metadata，平台在注册/扫描时自动识别模态。模态取值支持 image / audio / video / text 四种。

```
import pyarrow as pa

schema = pa.schema([
    # 图片列：large_binary + blob 编码 + 模态声明
    pa.field("image", pa.large_binary(), metadata={
        "bos-dataset-field:modality": "image",
        "bos-dataset-field:mimetype": "image/jpeg",  # RFC 类型，照实填写
        "bos-dataset-field:format": "jpg",           # 封装格式，照实填写
        "lance-encoding:blob": "true"}),             # 大对象用 blob 编码
    # 音频列
    pa.field("audio", pa.large_binary(), metadata={
        "bos-dataset-field:modality": "audio",
        "bos-dataset-field:mimetype": "audio/wav",
        "bos-dataset-field:format": "wav",
        "lance-encoding:blob": "true"}),
    # 视频列
    pa.field("video", pa.large_binary(), metadata={
        "bos-dataset-field:modality": "video",
        "bos-dataset-field:mimetype": "video/mp4",
        "bos-dataset-field:format": "mp4",
        "lance-encoding:blob": "true"}),
    # 文本列
    pa.field("caption", pa.string(),
             metadata={"bos-dataset-field:modality": "text"}),
    # 普通元数据列可不声明；未知键会被忽略，不影响读写
    pa.field("label", pa.string()),
])

table = pa.Table.from_pylist([...], schema=schema)  # 数据按 schema 顺序放进
lance.write_dataset(table, uri, storage_options=so)
```

配置说明：

键	说明
bos-dataset-field:modality	多媒体预览核心属性，支持 image / audio / video / text；仅 BINARY（及 large_binary）类型字段配置预览有效。**用 bos- 前缀**
bos-dataset-field:mimetype	RFC media type（如 image/jpeg）；未声明 modality 时可由它推导
bos-dataset-field:format	媒体封装格式，照实填写，如 jpg、wav、mp3、mp4
lance-encoding:blob	值为 true 时大对象采用 blob 编码存储，大小可零成本获取且支持按需随机读

SQL 查询

【SQL 查询】页签提供在线 SQL 编辑器，可直接对数据集执行只读查询，用于数据探查与抽样验证。



1. 在编辑器中编写 SQL，表名固定为 `dataset`，编辑器默认填入示例：

```
SELECT * FROM dataset LIMIT 10;
```

2. 常用查询示例：

```
-- 按标签统计行数
SELECT label, COUNT(*) AS cnt FROM dataset GROUP BY label;
-- 查看训练/测试切分比例
SELECT split, COUNT(*) AS cnt FROM dataset GROUP BY split;
```

3. 单击【执行查询】运行，下方展示查询结果。

> 说明：SQL 查询仅支持只读单表（SELECT），表名必须是固定关键字 `dataset`，写成其它表名、多语句、JOIN 或写操作会被拒绝；二进制列在结果中以占位形式展示，如需查看媒体内容请使用【数据详情】预览。

常见问题

Q1：二进制列显示为占位图标，是数据损坏吗？

不是。这是模态未配置的表现，并非数据损坏。通过列头 ⋮ 菜单【修改模态】补标（图像/音频/视频），即时生效；或在写入时按「写入规范」声明元数据，注册后自动预览。

Q2：SQL 查询报错表不存在？

SQL 中的表名必须是固定关键字 `dataset`，代表当前数据集。

模型管理

模型创建及管理

权限管理

权限概述

概述

注册百度智能云账号后，系统自动创建的超级管理员用户即为账号主用户，默认拥有云账户的全部权限和管理全部云资源的权限。在多人共享资源和协作管理场景下，管理员可以创建子用户，实现不同子用户使用不同的用户名密码，并拥有不同的云资源访问权限，更多信息请参见 百度智能云账户身份体系。

默认情况下，子用户没有使用 百舸 的任何权限，因此需要管理员授权子用户拥有相应权限。

前提条件

- 已使用主用户（超级管理员）登录百度智能云控制台。
- 已创建需要授权的 IAM 子用户或用户组。
- 已准备可授权的 百舸资源池，并明确授权作用范围（资源池或资源队列）。

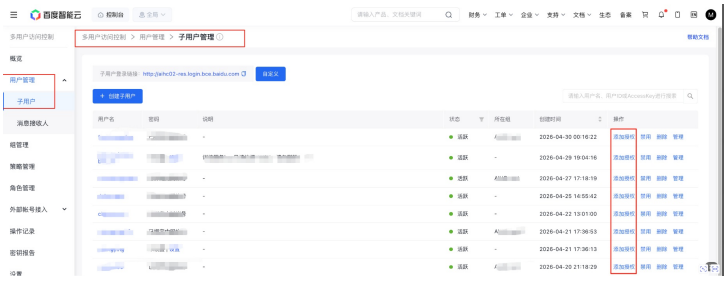
百舸 授权体系

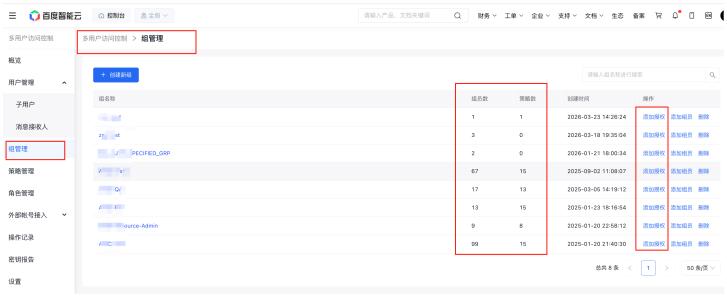
百舸 权限管理包括百度智能云多用户访问控制（**IAM**）策略授权和 百舸内部的角色访问控制（**RBAC**）授权，支持从资源池和资源队列级别对子用户或用户组进行细粒度授权。百舸 授权必须完成两个阶段的授权，才能正常使用百舸平台：

- **第一阶段**：基于 **IAM** 策略的授权，管理员对 IAM 子用户或用户组授予 百舸平台的管理权限（例如创建/删除资源池、购买/释放节点，创建/删除队列，提交工作负载，AI资产管理等），授权将作用于 百舸 整个产品，也是用户使用百舸平台的前提，更多信息请参见 配置 IAM 预置权限策略。
- **第二阶段**：百舸内部 **RBAC** 的授权，管理员对 IAM 子用户授予 百舸资源池的运维管理权限（节点基础运维、全局配置管理、创建管理资源队列等），或者授予资源池内某个队列的管理权限（节点基础运维、队列用户管理、队列配置等）或提交工作负载的权限（使用资源的权限）。

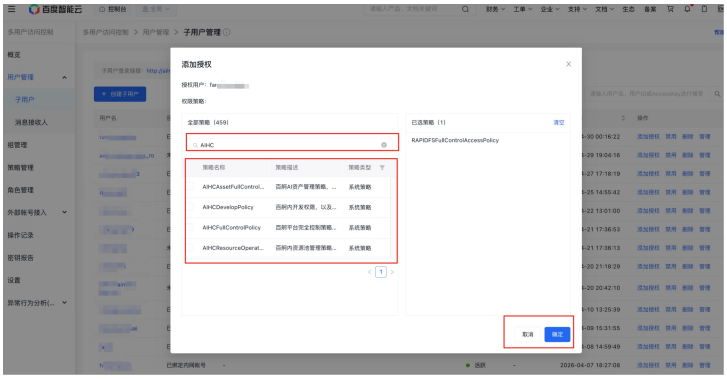
IAM授权说明

- **操作步骤**
  - **按子用户授权**：登陆控制台->多用户访问控制（IAM）->用户管理->子用户。
  - **按用户组授权时**：登陆控制台->多用户访问控制（IAM）->组管理。
- **步骤 1：进入 IAM 并选择目标授权对象**
  - 在 **【子用户】** 或 **【组管理】** 列表中选择目标授权对象，并进入详情页。





- **步骤 2：授予 百舸的相关权限策略**
  - 针对待授权的某个子用户或用户组，点击 **【添加授权】**。在 **【添加授权】** 弹窗中检索并选择 百舸（AIHC） 相关策略，确认策略进入已选策略列表。





- **步骤 3：保存授权并确认生效**
  - 在 **【添加授权】** 弹框添加完权限策略后，点击 **【确定】** 保存。保存后在对象的 **【权限信息】** 列表中确认已存在百舸的相关策略，即表示 IAM 授权已生效。
  - 如需取消该用户的百舸权限，可以在子用户的权限信息中直接 **【删除】** 即可。



百舸的 RBAC 授权（资源池/资源队列级别）

- **操作步骤**
  - 登陆百舸平台->进入资源池详情->成员管理->配置资源池管理员。
  - 登陆百舸平台->进入队列详情->成员管理->配置队列管理员 & 队列内算法开发成员。

资源池管理员授权

- 步骤 1：选择目标资源池权限管理
  - 在 百舸控制台进入 **【全托管资源池】** > **【更多操作/成员管理】**，打开资源池的 RBAC 授权管理页面。



- 步骤 2：在成员管理页面，点击【+添加成员】，弹出添加成员对话框。



- 步骤 3：选择用户或者用户组，添加到本资源池作为资源池管理员。



- 步骤4: 选择完用户或者用户组，点击【确认】，添加成功。如需将某个用户移出资源池管理员，请直接在成员管理页面点击【删除】即可。

队列成员授权

- 步骤 1：选择目标资源队列的权限管理
  - 在 百舸控制台进入 **【全托管资源池/资源队列】** > **【更多操作/成员管理】**，打开资源队列的 RBAC 授权管理页面。
  - 或者 百舸控制台进入 **【全托管资源池/资源队列】** > **【队列详情】** > **【成员管理】**，打开资源池的 RBAC 授权管理页面。



- 步骤 2：在成员管理页面，点击【+添加成员】，弹出添加成员对话框。



- 步骤 3：选择用户或者用户组，选择成员权限【队列管理员】或者【队列内算法开发成员】。



- 步骤4: 点击【确认】，添加成功。如需将某个用户移出队列成员，请直接在成员管理页面点击【删除】即可。

配置IAM预置权限策略

百舸 基于 IAM 策略授权，指的是百度智能云账号的管理员（或拥有系统管理员权限的子用户）通过多用户访问控制（IAM），将预定义的 百舸 系统策略授予子用户/用户组，让其拥有百舸的相关权限。

本文将介绍 百舸 基于 IAM 策略授权的流程及使用方法。

百舸预置策略

您可以使用以下预设策略为子用户授予相关权限：

策略名称	策略描述	权限范围
AIHCFullControlPolicy	百舸平台完全控制策略，该策略为平台最高权限，推荐平台管理者使用。 备注：AIHCFullControlPolicy必须同时关联FCOrderCreatePolicy和FCOrderAccessPolicy权限，才能完成在百舸购买节点/释放节点/百舸云盘扩缩容等操作。	创建资源池 删除资源池 添加节点（购买节点） 释放节点（退订节点） 节点续费/计费变更 云盘管理（扩缩容） 资源池全局配置 资源池监控/工作负载/资源报表/变更记录等查看与管理 关联绑定相关依赖产品（Cprom、PFS、子网等） 成员管理（添加/移出资源池管理员） 封锁资源池节点 同时拥有其他策略的所有权限范围
AIHCResourceOperatorPolicy	资源池管理策略，支持资源池的全局配置管理、创建分配资源队列等（不支持创建/删除资源池和节点，不支持产生费用或者订单的操作） 备注：必须在资源池成员管理中同步授权资源池管理员的RBAC权限。	资源池全局配置 查看资源池监控/工作负载/资源报表/变更记录等详情信息 关联绑定相关依赖产品（Cprom、PFS、子网等） 封锁资源池节点 创建队列 删除队列 编辑队列（调度策略、队列状态等） 管理队列（添加节点/移出节点/转让节点/封锁节点） 资源池成员管理（添加/移出资源池管理员） 队列成员管理（添加/移出队列管理员/队列算法开发成员）
AIHCDevelopPolicy	开发者策略，支持在指定的队列进行算法开发（管理开发机、分布式训练以及推理服务等）。 备注：必须在百舸资源队列的成员管理中同步授权队列管理员或队列算法开发人员的RBAC权限。	查看队列详情/队列资源视图/队列节点列表/工作负载/队列成员/队列监控 **使用队列的资源，**在指定的队列进行算法开发（管理开发机、分布式训练以及推理服务等）
AIHCAssetFullControPolicy	百舸AI资产管理策略，支持创建管理数据集/模型等 备注：无需在百舸内同步授权RBAC的权限。	创建数据集/管理数据集 注册新模型/管理注册模型 管理操作数据下载

> **温馨提示**
>
> - 百舸 的关联云产品/服务较多，为了让您的授权过程更加简单高效，百舸 IAM 系统策略配置包含了常用关联云产品/服务的操作权限。
> - 您只需为子用户授权一次 百舸 IAM 系统策略，即可让其拥有 百舸相关资源和关联云产品/服务的只读/管理权限，而无需理解多种云产品策略并进行多次授权。

操作步骤

直接给子用户授权

您可以直接为子用户关联策略以获取策略包含的权限。

1. 进入多用户访问控制控制台,选择左侧导航栏中的**用户管理>子用户**进入“子用户管理”列表页面。
2. 选择目标子用户单击**用户名**进入用户详情页面。
3. 单击“授权”操作，在“编辑权限”弹出框中勾选所需策略。
4. 单击“确定”即可。

通过用户组给子用户授权

您可以将子用户添加至用户组，该子用户将自动获取该用户组所关联策略的权限。如需解除随组关联策略，仅需将子用户移出相应用户组即可。

1. 进入多用户访问控制控制台,选择左侧导航栏中的**组管理**进入“组管理”列表页面。
2. 选择目标用户组单击**编辑**进入编辑组页面。
3. 在“编辑组”页面勾选所需策略以及添加对应子用户。
4. 单击“确认”即可。

配置IAM自定义权限策略

百度智能云账号管理员（或拥有系统管理员权限的子用户）除了可以直接为子用户或用户组授予百舸预置策略外，还可以基于百舸预置策略创建自定义策略，实现更细粒度的权限控制。

本文将介绍如何通过 IAM 自定义策略，按需限制子用户对关联存储资源（如对象存储 BOS、文件存储 CFS）的访问范围。

应用场景

为简化授权流程，百舸预置策略默认包含了常用关联云产品（如 BOS、CFS）的列表和读取权限，适用于快速授权场景。

如果您希望子用户在使用百舸进行算法开发（如管理开发机、分布式训练、推理服务等）时，仅能查看和访问指定的存储实例（例如仅允许访问某个指定的 BOS 存储桶，而不能查看主账号下的全部存储桶列表），可以通过创建 IAM 自定义策略来实现精细化权限管理。

配置思路

以开发者策略 `AIHCDevelopPolicy` 为例，限制子用户仅能访问指定 BOS 存储桶 `bucket01` 的配置思路如下：

1. **创建自定义策略**：以 `AIHCDevelopPolicy` 预置策略为模板创建自定义策略，保留百舸平台功能权限，并将策略中 BOS 的资源范围（`resource`）置空。
2. **授予自定义策略**：在 IAM 中为目标子用户（或用户组）授予该自定义策略（替换原有的 `AIHCDevelopPolicy` 预置策略）。
3. **配置存储实例权限**：在 BOS 控制台中，单独为该子用户授予目标存储桶（如 `bucket01`）的访问权限。

前提条件

- 已使用百度智能云主账号或具备 IAM 管理权限的子用户登录控制台。
- 已创建需要授权的 IAM 子用户。
- 已准备需要向子用户开放的目标存储资源（本文以 BOS 存储桶 `bucket01` 为例）。

操作步骤

步骤一：基于预置策略创建自定义策略

1. 进入多用户访问控制（IAM）控制台，在左侧导航栏选择**策略管理**。
2. 单击**创建策略**，选择**按策略语法创建**。
3. 填写策略基本信息（如策略名称设置为 `AIHCDevelopPolicyUserDefined`），并选择百舸预置策略 `AIHCDevelopPolicy` 作为策略模板。
4. 在策略语法配置中，找到 `bce:bos` 服务对应的权限配置项，将其 `resource` 字段的值由 `["*"]` 修改为空数组 `[]`，其他权限配置保持不变：

```
{
 "service": "bce:bos",
 "region": "*",
 "resource": [],
 "effect": "Allow",
 "permission": [
   "READ",
   "LIST",
   "ListBuckets"
 ],
 "eid": "bos_list_and_read_access"
}
```

> **说明**：将 `resource` 设置为 `[]` 后，该策略不再默认授予子用户查看所有 BOS 存储桶的权限，具体可访问的存储桶将由后续在 BOS 控制台中的授权决定。

5. 单击**确定**，完成自定义策略创建。

步骤二：为子用户授予自定义策略

您可以直接为子用户授权，也可以通过用户组进行授权：

- **直接给子用户授权**

1. 在 IAM 控制台左侧导航栏选择**用户管理 > 子用户**。
2. 找到目标子用户，单击**授权**（或进入用户详情页单击**添加授权**）。
3. 在弹窗中搜索并勾选步骤一创建的自定义策略 `AIHCDevelopPolicyUserDefined`。
4. 单击**确定**完成授权。

- **通过用户组给子用户授权**
  1. 在 IAM 控制台左侧导航栏选择**组管理**。
  2. 选择目标用户组，单击**编辑**，勾选自定义策略 `AIHCDevelopPolicyUserDefined` 并添加对应子用户。
  3. 单击**确认**完成授权。

> **注意**：若该子用户或所属用户组此前已绑定 `AIHCDevelopPolicy` 或其他包含 BOS 全局读取权限（如 `resource: ["*"]`）的策略，请将其移除，否则子用户仍会继承全局列表权限。

步骤三：在 BOS 控制台授予指定存储桶权限

1. 进入对象存储 BOS 控制台。
2. 在存储桶列表中单击目标存储桶（如 `bucket01`），进入存储桶详情页。
3. 在访问权限相关配置页面中，为目标子用户添加该存储桶的访问权限（根据业务需求授予读取、列表或读写权限）。
4. 保存配置使授权生效。

完成以上配置并结合百舸 RBAC 授权后，该子用户登录百舸控制台时，即可正常使用被授权队列的算法开发功能，且在选择 BOS 存储时仅对已授权的 `bucket01` 具备访问权限。

注意事项

- **配合百舸 RBAC 权限使用**：IAM 自定义策略仅完成第一阶段的云产品访问控制，子用户仍需在百舸控制台的资源池或队列成员管理中完成 RBAC 角色授权，才能在对应资源池或队列中执行操作。
- **IAM 权限叠加规则**：当子用户同时关联多个 IAM 策略时，权限取并集。如需收敛存储访问范围，请确保子用户未同时绑定包含存储全局访问权限（`resource: ["*"]`）的其他策略。
- **CFS 权限精细化控制**：如需对文件存储 CFS 进行同等粒度的权限控制，可参照上述方式修改自定义策略中的 `bce:cfs` 资源配置，并按需为子用户授予指定 CFS 实例的访问权限。

配置预置RBAC权限策略

百舸提供了对接资源池和资源队列 RBAC的授权模式，便于对子用户进行细粒度的访问权限控制。本文介绍如何为子用户配置RBAC权限，实现对百舸资源池和资源队列的权限控制。

声明

百舸AI计算平台对RBAC 权限管理声明如下：

- 禁止未完成RBAC权限授予的子用户访问资源池资源，请及时联系主账号完成RBAC授权，以免带来生产上的不便。
  - 除AIHCAssetFullControPolicy权限策略外，被授予该权限的用户可以查看平台的公共资产（如公共数据集、预置镜像等），用户自定义资产（自定义数据集、用户上传模型等）受具体授予权限管控。
- 子用户将只拥有被指定授予的资源池或者资源队列的访问权限。

授权说明

- 只有如下账号/用户/角色，才能配置百舸的 RBAC 权限：
  - 百度智能云主账号。
  - 具有 AIHCFullControlPolicy 权限的 IAM 用户。
  - 某具体资源池的管理员对资源池下所属队列拥有配置RBAC 权限的能力。

前提条件

- IAM 用户已完成 百舸侧的相关权限配置。

权限说明

百舸内置了三种级别的RBAC权限，如下所示：

角色名称	角色说明	依赖成员具备的IAM系统策略	权限范围
资源池管理员	运维管理资源池和队列的权限（不能创建/删除资源池和节点）	AIHCResourceOperatorPolicy	资源池全局配置 查看资源池监控/工作负载/资源报表/变更记录等详情信息 关联绑定相关依赖产品（Cprom、PFS、子网等） 封锁资源池节点 创建队列 删除队列 编辑队列（调度策略、队列状态等） 管理队列（添加节点/移出节点/转让节点/封锁节点） 资源池成员管理（添加/移出资源池管理员） 队列成员管理（添加/移出队列管理员/队列算法开发成员）
队列管理员	具备队列的管理权限以及添加队列成员（不支持创建/删除队列，不支持修改队列的配额）	AIHCDevelopPolicy	查看队列详情/队列资源视图/队列节点列表/工作负载/队列成员/队列监控 **使用队列的资源，**在指定的队列进行算法开发（管理开发机、分布式训练以及推理服务等）
队列内算法开发成员	具备在本队列内开发的权限，支持在队列内创建开发机、提交训练任务和部署推理服务	AIHCDevelopPolicy	**使用队列的资源，**在指定的队列进行算法开发（管理开发机、分布式训练以及推理服务等）

> 需要同时配置IAM系统策略和百舸内部角色，权限才能生效。如需要设置用户A为资源池1的管理员，则首先需要在IAM系统中授予用户A `AIHCResourceOperatorPolicy` 系统策略，然后在资源池1中为用户A分配资源池管理员的角色.

配置IAM用户的RBAC授权

- **操作步骤**
  - 登陆百舸平台->进入资源池详情->成员管理->配置资源池管理员。
  - 登陆百舸平台->进入队列详情->成员管理->配置队列管理员 & 队列内算法开发成员。

具体操作流程详情见文档

预留实例券

概述

预留实例券介绍

预留实例券（Reserved Instance Voucher，简称RI）是一种能够抵扣**按量付费**节点账单的抵扣权益，且支持多种购买周期。对比包年包月预付费，预留实例券与按量付费节点这种组合模式灵活性更好。**购买预留实例券意味着您承诺了对应的购买时长，预留实例券在购买后无法退订。**

如果有长期稳定的AI训练或推理业务，或者对资源供应的确定性有极高要求，开通预留实例券是优化成本和保障交付的最佳选择。

> 预留实例券到期后券将自动删除，但**节点不会自动释放**。券到期后，节点会转为按量付费原价计费，**请记得及时手动释放节点**，避免产生不必要的费用。

核心优势

预留实例券结合了“包年包月”的低成本与“按量付费”的灵活性，具备以下核心优势：

- **成本优化**：相比纯按量付费模式，预留实例券提供大幅度的价格折扣。
- **资源保障**：AZ级（可用区级）预留实例券在购买后，百舸平台会为您在指定可用区锁定预留相应的计算资源。即使在资源紧张时期，也能保障您随时创建并启动实例。
- **资金灵活**：支持全预付、半预付、0预付三种付费模式。全预付支持一次性按最低总成本支付，半预付支持预先支付50%的费用，剩余费用将于有效期内按每小时支付，0预付支持按每小时支付，有效缓解一次性大额支付带来的现金流压力。
- **账单解耦**：实例券与具体机器解耦，购买后自动匹配并抵扣符合条件的按量付费实例账单，管理更灵活。

适用范围

适用范围	概念	优势
可用区级	实例券生效后将自动匹配并抵扣相同 AZ、实例规格以及 EHC 集群（仅适用 EHC 类型实例）的按量付费实例账单	支持资源预留，购买预留实例券后，百度云将保证您账号下在对应的 AZ 中至少能有与 RI 相同的实例规格 * 台数的按量付费资源。

付费方式

RI 按付费方式可分为：全预付、半预付、0预付。

付费类型	定义	适用场景
全预付	购买券时支付购买周期内对应算力的总额。总价最低。	预算充足、资源需求稳定且使用周期明确，希望通过一次性支付获得最低总成本。
半预付	购买券时支付购买周期内对应算力 50% 的费用，其他费用将于有效期内每小时扣除（类似后付费）。总价居中。	具备一定预算、资源需求稳定，希望降低初期资金投入，同时兼顾长期用量成本与现金流。
0预付	购买时无需支付费用，费用按小时扣除（承诺消费）。	不希望初期投入资金，但仍需长期资源保障的场景。

购买限制

由于预留实例券与节点套餐存在对应的抵扣关系，且预留实例券购买后不支持退订。为降低用户购买后无法合理使用的风险，平台对预留实例券的购买权限进行限制。

- GPU套餐下预留实例券的不同付费方式需要加白后，才可购买。如您在控制台无法选择付费方式，请联系商务对GPU套餐开白后再下单。
- 平台仅支持具备AIHCFullControlPolicy策略的账户创建预留实例券。配置方式参见：配置IAM预置权限策略。

> 注册百度智能云账号后，系统自动创建的超级管理员用户即为账号主用户，默认拥有云账户的全部权限和管理全部云资源的权限。

抵扣机制

预留实例券购买后，会自动匹配您账号下符合条件的按量付费（后付费）实例进行账单抵扣。

1. 匹配规则 在有效期内，AZ级预留实例券将每小时自动检查并匹配满足以下条件的按量付费实例：

- **地域与可用区相同**：例如均为 华北-北京，可用区F。
- **节点规格相同**：例如均为 `aihc.lgn5.c128m2048.8a100.8re`。
- **EHC集群匹配（仅针对RoCE网络机型）**：实例券指定的EHC集群需与节点一致。

2. 抵扣示例 假设您持有1张 华北-北京 可用区F 的 `8*A100` 预留实例券：

- **完全匹配**：若您该可用区下正好有1台同规格的按量付费实例运行，则该实例产生的计算费用将被100%抵扣（具体抵扣金额视券面算力而定）。
- **无实例运行**：若您当前没有运行对应的按量付费实例，实例券仍会继续计费，但此时系统为您保留了一台机器的额度，您可以随时启动新实例而无需担心库存不足。
- **多台实例**：若您运行了2台同规格实例，预留实例券将抵扣其中1台的费用，第2台照常按量付费。

生命周期管理

- **生效时间**：支持“立即生效”或“指定未来时间生效（24小时内）”。
- **有效期**：支持1个月至5年不等的购买周期。
- **到期处理**：
  - **即将到期**：到期前7天、3天、1天系统会发送通知。
  - **已到期**：到期后实例券停止抵扣，且不再保障资源预留。建议开启自动续费功能以避免折扣中断或资源被释放。
- **欠费停服**：若您的账号欠费，预留实例券将进入欠费受限状态，停止抵扣账单，但实例券本身可能会持续出账，请保持账户余额充足。

查看账单与明细

您可以在百舸AI计算平台AIHC控制台或财务中心查看预留实例券的抵扣详情。

预留实例券创建

功能说明

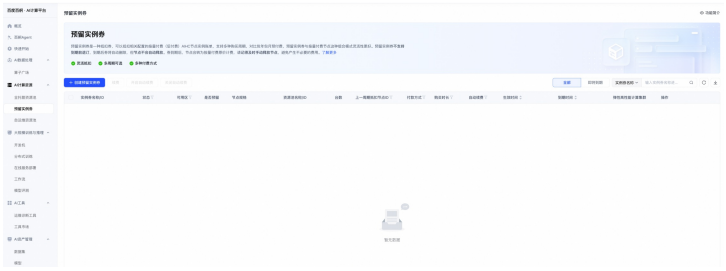
本文介绍如何在百舸计算平台购买预留实例券。

前提条件

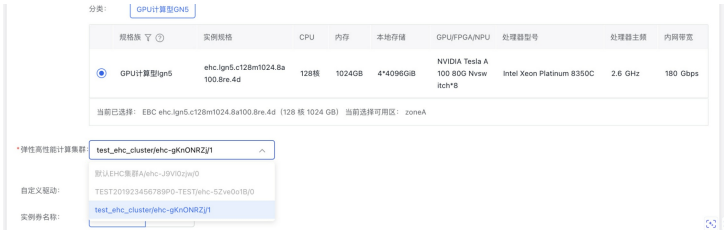
1. 创建预留实例券之前必须完成实名认证。请根据自身情况，进行“企业认证”或者“个人认证”，具体请参考个人认证和企业认证。
2. 目前预留实例券只支持 **全预付、半预付、 0预付** 的付费类型。
3. 创建预留实例券时，需要您的账户余额大于或等于100元整。

操作步骤

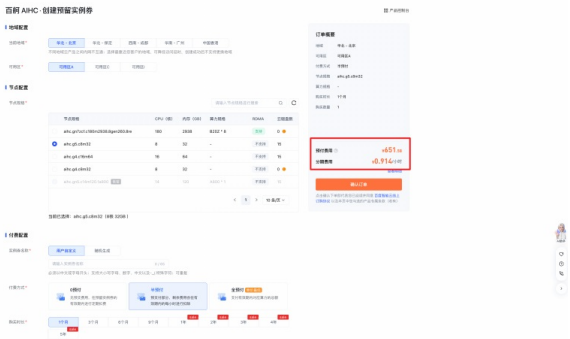
1. 登录百舸AI计算平台AIHC控制台，左侧栏选择 **预留实例券** ，进入预留实例券列表页。
2. 单击 **创建预留实例券**。



3. 选择**地域、可用区**，预留实例券与要抵扣的按量付费-后付费实例在同一个可用区下才可抵扣实例账单。
4. 选择**实例配置**，与需要抵扣的按量付费实例规格一致。
5. 选择**弹性高性能计算集群**，与需要抵扣的后付费实例一致，实例券仅支持抵扣同EHC集群的实例。



6. 设置**实例券名称**，可以根据需要选择用户自定义或者随机生成。
7. 设置**实例数量**，当前1个券支持抵扣1个实例。如您需要抵扣多个百舸实例，请填写对应的券数量。
8. 选择**付款类型**，当前支持全预付、半预付、0预付。
9. 选择**购买时长**，支持1个月，3个月，6个月，9个月，1年，2年，3年，4年，5年。
10. 设置**预留实例券数量**，缺省值为1，如您需要抵扣多个实例，请填写对应的券数量。
11. 设置是否开启**自动续费**，实例券到期后是否开启自动续费快到期的券，默认关闭。
12. 设置自动续费的**续费周期**，当前支持按照购买时长的周期执行自动续费。
13. 设置**生效时间**，默认是立即生效，您也可自定义生效时间，最长可设置24小时，且只能设置为15min为间隔的整点（如：10:00、10:15、10:30、10:45、11:00）。
14. 确认**费用价格**，查看控制台显示的价格是否符合预期，预付费用代表券生效前预先支付的费用，分期费用即在券的有效期内每小时支付的费用，月预估表示折算到每个月需要支付的分期费用。



15. 点击**下一步**，在订单确认页勾选同意《百度智能云线上订购协议》，并点击**确认订单**即可。

最终结果

在预留实例券列表页可看到已创建成功的预留实例券，并可抵扣相同实例规格的按量付费实例账单。



查看抵扣明细

可在抵扣明细页面中查看预留实例券抵扣实例的抵扣明细。

入口1：在预留实例券列表页，选择要查询的实例券，在操作中点击“查看抵扣明细”。

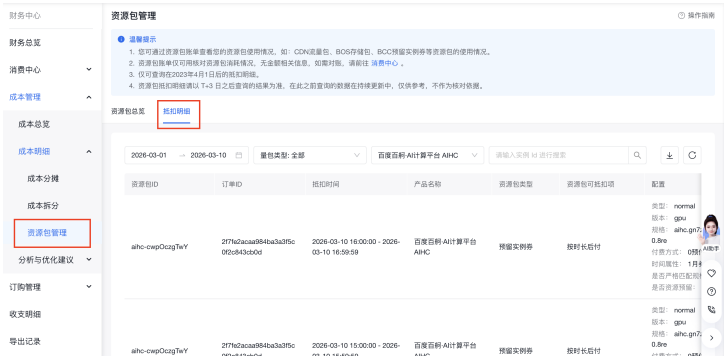


入口2：

1. 在控制台右上角点击**财务**，进入财务中心。



2. 在财务中心页面中的左侧栏点击**资源包管理**，在头部标签页中点击**抵扣明细**。

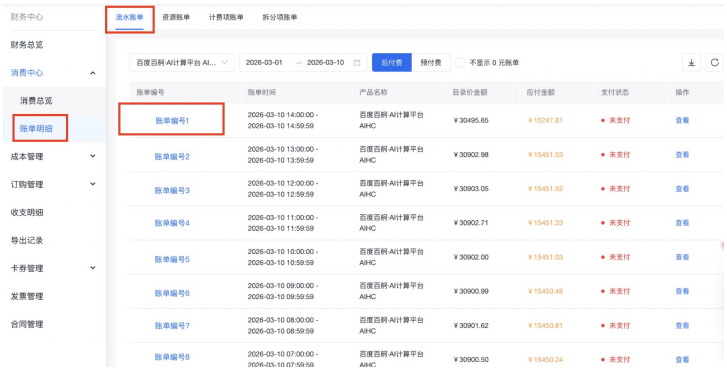


查看预留实例券流水账单

1. 进入**流水账单**。



2. 点击近期的账单编号，进入账单详情。



> **注**：预留实例券和按量付费付费均为15min扣费，每小时出账。

预留实例券续费

功能说明

本文介绍如何对已创建的预留实例券做续费操作。执行手动续费会直接为该预留实例券延长到期时间。

前提条件

实例券处于生效中状态。

操作步骤

手动续费

1. 登录百舸AI计算平台AIHC控制台，左侧栏选择**预留实例券**标签页，进入预留实例券列表页。

2. 在预留实例券列表中找到要续费的实例券，选中对应的复选框，单击顶部菜单栏左上角处的**续费**。



3. 进入订单确认页，完成下单。

4. 下单后，可在实例券列表查看“到期时间”，确认续费后的新到期时间。



开启自动续费

1. 登录百舸AI计算平台AIHC控制台，左侧栏选择**预留实例券**标签页，进入预留实例券列表页。

2. 在预留实例券列表中找到要开启自动续费的实例券，选中对应的复选框，单击顶部菜单栏左上角处的**开启自动续费**或操作列的**开启自动续费**。



3. 在**开启自动续费**弹窗中，续费周期默认**按月**，续费时长默认与券购买时长保持一致。



4. 点击确定后，实例券的自动续费状态变更为**已开启**。



关闭自动续费

1. 登录百舸AI计算平台AIHC控制台，左侧栏选择**预留实例券**标签页，进入预留实例券列表页。

2. 在预留实例券列表中找到要关闭自动续费的实例券，选中对应的复选框，单击顶部菜单栏左上角处的**关闭自动续费**或操作列的**关闭自动续费**。



3. 在**关闭自动续费**弹窗中，点击确定后，实例券的自动续费状态变更为**已关闭**。





模型评测

概述

模型评测用于在统一的评测任务中，对模型能力、性能与结果质量进行量化验证和分析。通过标准化的数据集、评测配置和结果展示，帮助用户高效完成模型选型、版本验收和效果迭代。

核心功能

- 创建评测：选择待评测模型、评测集与评测配置，发起一次可追溯的评测任务。
- 评测详情：查看任务状态、指标结果、样本明细和运行信息，定位模型表现差异。
- 对比评测：将多次评测结果并列比较，辅助模型、版本和参数配置的决策。

适用场景

- 模型上线前的效果验收与基线校验。
- 多模型或多版本的能力、性能对比与选型。
- 具身模型 task、执行参数等迭代后的效果回归。
- 按统一口径沉淀团队评测结果，形成可复用的评测资产。

权限说明

使用模型评测前，需要具备目标项目或工作空间的访问权限，以及相关模型、数据集和计算资源的使用权限。无权限时，请联系项目管理员开通。

典型使用方式

1. 准备待评测模型。
2. 创建评测，配置评测指标、评测规模与运行参数。
3. 在评测详情中查看结果与样本分析，确认模型表现。
4. 发起或查看对比评测，比较多个模型、版本或配置，并据此完成选型或迭代。

创建评测

前提条件

- 拥有可用于评测的资源队列操作权限，且队列拥有符合要求的显卡支持。
- 模型：您可以试用百舸预置好的开源模型验证评测效果，也可以使用您存在百度云存储上的权重进行验证。
- 目前仅支持部分 vla 及 benchmark 评测服务，后续会逐渐拓展不同模型支持及自定义 benchmark 评测功能。

操作步骤

创建评测任务入口 在百舸控制台左边栏 - 【大规模训练与推理】标签下，您可以看到模型评测按钮，点击【创建模型评测任务】后即可新建评测任务

基础配置 首先，您可以配置任务基本信息并选择您需要使用的资源池，在此您需要填入或配置：

- 任务名称
- 可见范围
- 资源池类型
- 队列
- 优先级
- 录制评测视频
- 并根据您选择的资源池&队列可以选择您的存储配置
  - 注意：如果您首次使用该功能，系统会为您创建一个 bos 存储桶来持久存储您的评测结果。



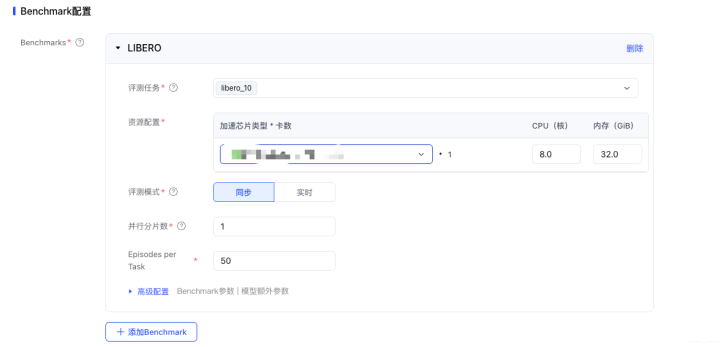
配置模型推理服务 然后您可以配置您的模型推理服务，您可以在此选择：

- 模型来源：
  - 预置公共模型：这里是百舸公共的预置模型权重
  - 自定义模型：您可以通过配置权重路径来让评测服务读取您的自定义模型（基于已适配基座模型微调权重）
- Chunk Size
- Batch Size
- 推理服务资源配置
- 并行配置



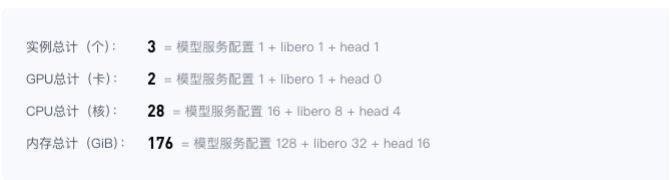
配置 Benchmark 您可以在Benchmark 配置中选择并配置您的评测集：

- 选择您的 benchmark
- 选择评测套件/task
- 为 benchmark 服务器选择配置
- 录制评测视频（您可以在不同 benchmark 下单独配置是否录制视频）
- 评测模式
- 并行分片数
- 每任务评测次数
- 以及针对不同 benchmark 的高级配置



确认资源 上述配置选择完毕后，在提交前您可以二次确认本次任务的总资源占用数

资源合计



提交评测 待点击页面下方提交按钮即可开始评测任务

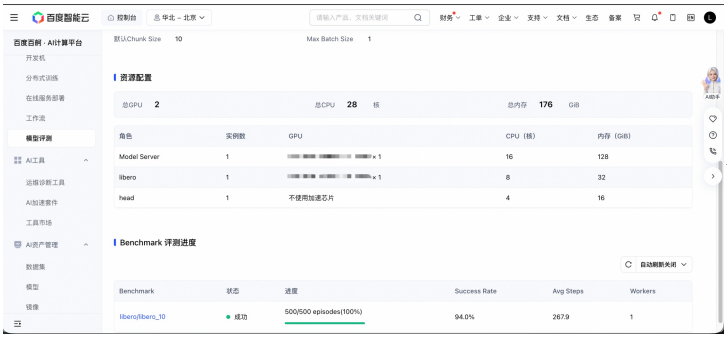
评测详情 & 结果对比

为方便算法工程师查看并对比不同模型在不同 benchmark 下的效果，模型评测功能支持基础的结果查看及跨任务比对

查看评测任务

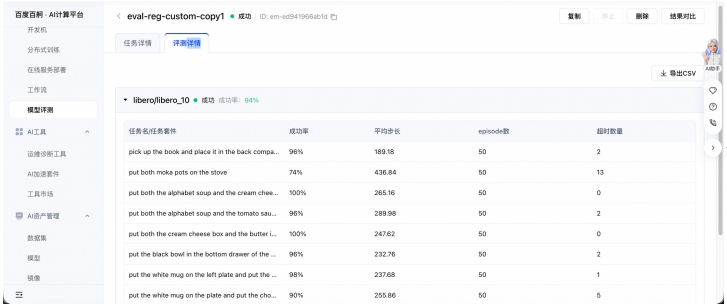
评测任务概览 在评测任务列表中，您可以通过点击任务名进入评测任务详情，在此您可以查看本次评测任务的：

- 配置信息
- 用时
- 评测任务进度（未完成状态）

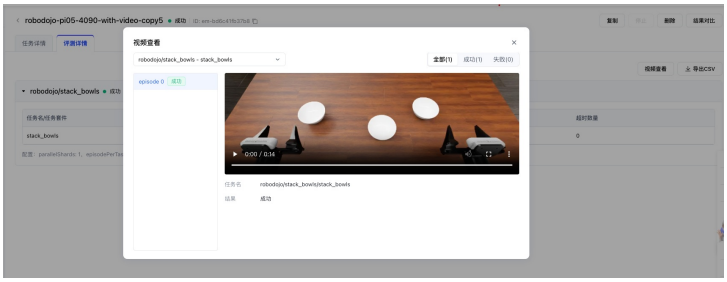


评测任务详情 您也可以通过点击详情页签查看具体任务成功率，目前每个任务包含：

- 任务名
- 成功率
- 平均步长
- episode数
- 超时数量



结果视频预览 如果您在创建评测任务时开启了视频保存，那么在评测详情中您可以点击视频查看查看/下载评测视频：

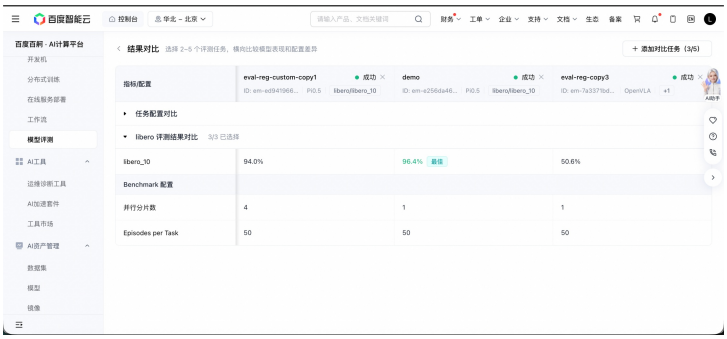


对比评测结果

对比入口 您可以通过以下两种方式进入评测对比：

- 任务详情中右上角的结果对比
- 在任务列表页多选（五个以下）任务，并点击对比

对比详情 您可以在对比中查看不同模型在不同 benchmark 下的运行效果



创建自定义推理镜像的评测任务

简介

具身评测支持您在百舸平台上使用自定义镜像参与模型评测。平台负责 Benchmark 仿真环境的调度、分片执行、结果合并与结果展示，您只需提供符合平台接口约定的推理镜像或推理服务，即可复用平台已有的评测集、任务配置、进度与结果分析能力。

与「预置公共模型」「自定义模型权重」相比，自定义镜像方式更加灵活，不要求模型框架与平台预置框架完全一致，只需要满足接口规则即可，适合已有自研推理实现、需要频繁迭代模型的场景。

功能介绍

镜像作为推理服务端运行

自定义镜像启动后在评测任务中承担模型推理服务端角色。平台在评测运行期间作为客户端，向镜像内的服务发起握手、下发观测、获取动作，并完成一个或多个 Episode 的评测。

镜像需满足的接口约定包括监听地址与端口、就绪探活、消息编码格式、握手与观测/动作消息结构、以及超时要求，详见具身评测自定义推理镜像接口说明。

Agent 辅助适配

平台提供配套 Agent Skill，可在本地协助您按接口约定检查与调整自定义镜像，降低首次接入成本。

计费说明

自定义推理镜像方式下，计费对象为您在评测任务中实际申请的平台资源：

- 调度节点资源
- 推理服务节点资源
- Benchmark 仿真运行资源
- 关联的存储挂载

权限说明

- 需要具备百舸模型评测功能的使用权限，以及目标资源池、队列与存储挂载的使用权限。
- 使用自定义镜像时，需具备对应镜像仓库的拉取权限。
- 评测视频、过程记录等产物写入指定存储路径，访问权限继承该存储挂载的权限设置。

典型使用方式

1. **准备推理镜像**：创建符合要求的推理镜像并准备好对应启动命令。
2. **新建评测任务**：在模型评测中新建任务，模型来源选择「自定义推理镜像」，填写启动命令并配置相应环境变量。



3. **配置 Benchmark**：按适配器限定的兼容范围添加 Benchmark 与 Task，配置评测模式、并行分片数、Episodes per Task 与运行资源；按需开启评测视频录制。
4. **提交并查看结果**：提交任务后，在详情页查看进度、监控、日志、推理调用错误摘要与评测结果；开启视频录制时可在详情页预览 Episode 视频与推理过程记录。

具身评测自定义推理镜像接口说明

镜像启动后作为模型推理服务端运行：应满足以下条件，您可以下载 Skill 通过本地 Agent 帮您调整镜像。

接入前提

镜像请务必保证以下条件：

- 镜像必须能加进 Ray 集群：ray[default]==2.49.2 + Python 3.13.12。
- 推理服务访问方式：提供websocket以及http两种访问方式，websocket服务用于收发msgpack帧，http服务用于框架的配置以及健康检查等控制请求。
- 服务要满足评测协议：同一个端口上既要 GET /config 返 200，又要能握手 WebSocket 收发 msgpack 二进制帧。不满足就是探活超时 900 秒后任务失败。
- 端口由平台分配，用占位符 {port} / {address}（= 0.0.0.0:<端口>）接收，监听 0.0.0.0，HTTP 与 WebSocket 同端口。
- 就绪判据：`GET /config` 返回 200（平台探活依据，权重加载完才返）。

消息格式

所有消息经 msgpack 编码，统一信封（四字段均必填）：

```
{"type": "...", "payload": dict, "seq": int, "timestamp": float}
```

`seq` 由客户端分配，服务端应答回显对应请求的 `seq`。

`payload` 里的数组和图像用约定字典表达，可嵌套，解码需递归：

```
{"__ndarray__": True, "data": b"...", "dtype": "<f4", "shape": [14]}   # dtype 用 numpy dtype.str
{"__image__": True, "format": "png", "data": b"...", "shape": [480,640,3]}   # 服务端须能解 raw/jpeg/png
```

输入

```
// HELLO 请求
{"protocol_version": 1, "harness_version": "0.5.0", "benchmark": "robodojo"}

// EPISODE_START：收到后重建会话、step 归零、重置所有 per-episode 状态
{"task": {"name": "stack_blocks", "suite": "robodojo", "episode_idx": 0}, "mode": "live"}

// OBSERVATION：唯一的高频帧
{"images": {"cam_head": "<ndarray(480,640,3)uint8>", "...": "..."},
 "task_description": "stack blocks",
 "state": "<ndarray(14,)float32>"}
```

相机名各评测集不同，建议您能够通过参数化配置；state 是否下发取决于 HELLO 应答里的声明。

输出

```
 // HELLO 应答：protocol_version 恒为 1；observation_params 声明需要哪些观测；
//           action_spec/observation_spec 供交叉校验（只告警不阻断）
{"protocol_version": 1, "model_server": "MyPolicyServer",
 "observation_params": {"send_state": true},
 "action_spec": {"actions": {"dims": 14, "format": "..."}},
 "observation_spec": {"images": {"format": "rgb_hwc_uint8"}, "state": {"format": "joint_positions"}}}

// ACTION
{"actions": "<ndarray(action_dim,)float32>"}

// ERROR：带内错误，不断连，服务端发完继续处理下一帧
{"error": "observation_failed: ValueError: ..."}
```

ACTION 硬约束：① 回显对应 OBSERVATION 的 seq；② 必须单步动作（1-D (action_dim,)）；③ 动作分块是服务端内部行为，缓存后逐帧返回一行，不上线缆。

时序与超时

```
HELLO 握手 → 每个 episode(EPISODE_START → N×(OBSERVATION → ACTION) → EPISODE_END) → 断开
```

HELLO 往返 30s，单帧 OBSERVATION → ACTION 30s（warmup 须在监听前做完，否则首帧超时）。

湖管理

湖管理快速入门

概述

湖管理提供面向多模态数据湖的元数据管理与权限管理能力，将 BOS 中的 Lance 数据组织为数据目录（Catalog）、数据库（Database）、数据表（Table）三级结构。控制台提供表结构、版本、存储优化、数据血缘与授权的统一入口。

湖管理包含两个功能：

页面	作用
元数据	创建和维护数据目录、数据库、数据表，查看表结构、版本、血缘并执行存储优化
权限管理	按角色为成员授予数据资产管理权限

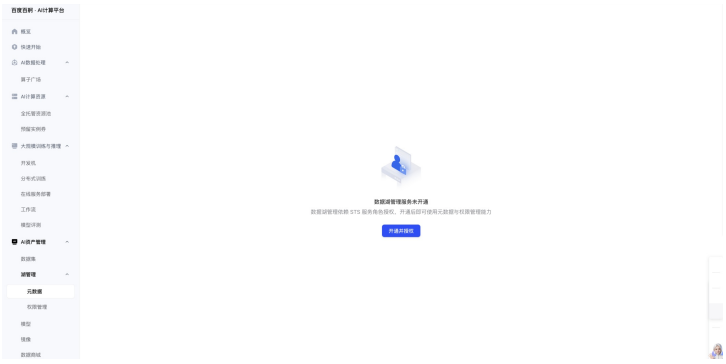
前提条件

在开始之前，请确认：

1. 已开通百度智能云账号，并具备湖管理的访问权限。
2. 若从 BOS 导入建表，数据已经以 Lance 格式存放在对象存储 BOS 中，且知道对应的存储路径（例如 bos://your-bucket/lake/）。

开通数据湖管理服务

首次进入湖管理时，服务处于未开通状态，页面会提示"数据湖管理服务未开通"。数据湖管理依赖 STS 服务角色授权，开通前无法使用元数据与权限管理能力。



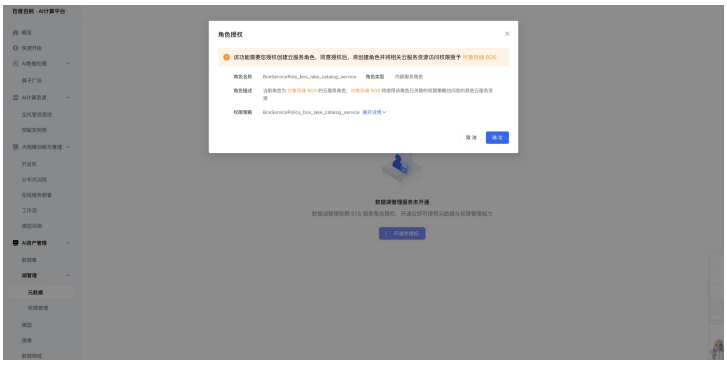
操作步骤

1. 登录百舸控制台。
2. 在左侧导航栏选择 **AI资产管理 > 湖管理 > 元数据**。
3. 在页面中央单击【开通并授权】。
4. 系统弹出【角色授权】窗口，确认授权信息后单击【确定】。
5. 验证通过后，页面自动进入元数据管理界面。

角色授权

开通时需要创建一个云服务角色，用于访问您的 BOS 资源。授权窗口会展示以下信息：

字段	说明
角色名称	BceServiceRole_bos_lake_catalog_service
角色类型	内部服务角色
角色描述	将使用该角色已关联的权限策略访问您的其他云服务资源
权限策略	BceServicePolicy_bos_lake_catalog_service，单击【展开详情】可查看策略包含的具体权限项



数据结构

湖管理采用三级元数据结构：

层级	含义	与存储的关系
数据目录 Catalog	数据湖的顶层逻辑，通常对应一个业务或一个数据域	可绑定一个 BOS 存储根路径，例如 bos://your-bucket/lake/；创建时允许留空
数据库 Database	目录下的业务分类，用于归类数据表	通常对应目录根路径下的一个子路径，例如 bos://your-bucket/lake/embodied_data/；路径可在创建时指定
数据表 Table	具体的 Lance 数据集，供查询与训练使用	对应一个以 .lance 结尾的目录

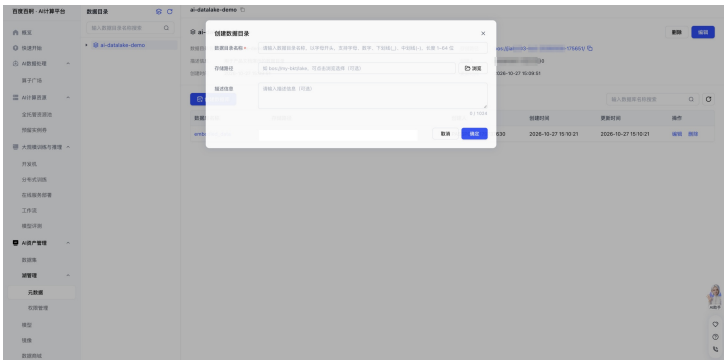
元数据为空时，页面提示“当前还没有创建数据目录（Catalog），先创建一个开始使用数据湖”，右侧展示【数据湖使用流程】，共五步：

1. **创建数据目录 Catalog**：数据湖顶层逻辑，用于组织数据库与数据表。
2. **创建数据库 Database**：在目录下按业务划分库，用于归类数据表。
3. **创建数据表 Table**：从百舸数据集或 BOS 导入 Lance 数据建表，Schema 扫描回填。
4. **管理数据表**：查看版本与 Schema 对比，配置存储优化，版本管理，数据血缘。
5. **配置权限**：在权限管理页按角色（管理员 / 运维 / 只读）授权成员访问。



创建数据目录

使用湖管理前，需先创建数据目录。创建后在左侧【数据目录】树中可见，选中后右侧展示目录详情与库列表。



操作步骤

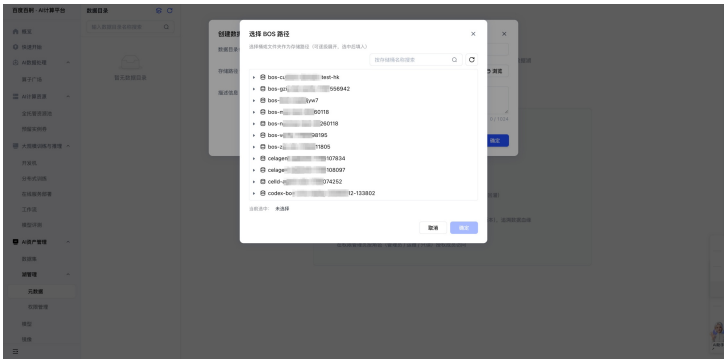
1. 在左侧【数据目录】面板右上角单击创建图标。
2. 填写【数据目录名称】。
3. 按需填写或选择【存储路径】。
4. 按需填写【描述信息】。
5. 单击【确定】，页面提示“创建数据目录成功”，并自动选中新建的目录。

配置如下：

字段	说明
数据目录名称	必填，以字母开头，支持字母、数字、下划线（_）、中划线（-），长度 1～64 位。输入后系统会实时校验并提示“当前名称可用”
存储路径	选填，目录对应的 BOS 根路径，格式为 bos://<桶名>/<前缀>。可手动输入，也可单击【浏览】从桶列表中选择；可留空
描述信息	选填，最长 1024 个字符，建议填写目录的用途与归属业务

选择 BOS 路径

单击存储路径输入框右侧的【浏览】，会弹出【选择 BOS 路径】窗口。窗口顶部支持按存储桶名称搜索，列表中逐个展开桶即可选择下级目录；选中后底部会显示【当前选中】的完整路径。





选中后单击【确定】，路径自动回填到创建窗口。

> 说明：目录本身只是一层元数据，创建目录不会在 BOS 中生成或搬移任何数据。

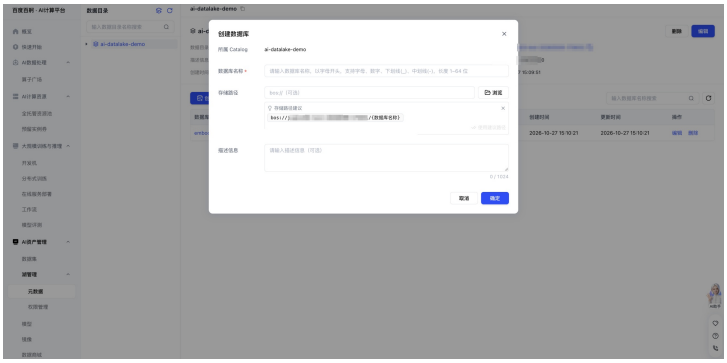
创建数据库

进入某个数据目录后，右侧展示目录详情（名称、存储路径、描述、创建人、创建/更新时间）以及该目录下的数据库列表。新建目录时列表为空，提示“暂无库”。



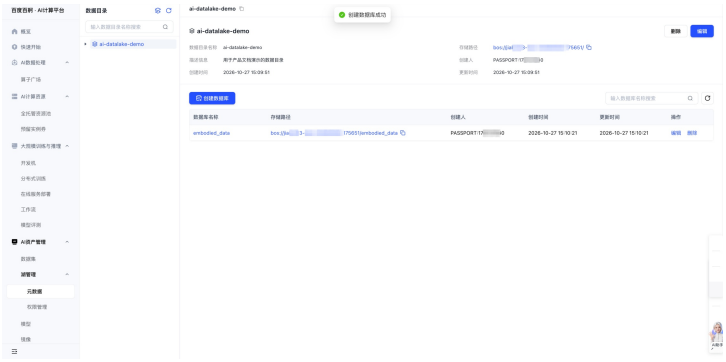
操作步骤

1. 在目录详情页单击【创建数据库】。
2. 【所属 Catalog】由当前目录自动带出，不可修改。
3. 填写【数据库名称】。
4. 设置【存储路径】：该字段为选填项。输入名称后，系统会在【存储路径建议】中给出 `bos://<目录路径>/<数据库名称>`，单击【使用建议路径】可一键填入；也可以留空或手动指定其他路径。
5. 按需填写【描述信息】。
6. 单击【确定】，页面提示“创建数据库成功”，列表中出现新建的库。



数据库名称的命名规则与数据目录一致：以字母开头，支持字母、数字、下划线、中划线，长度 1～64 位。存储路径为选填项，支持手动输入或通过【浏览】选择，建议放在所属数据目录的存储路径之下。

创建完成后，数据库以表格形式展示，列包括：数据库名称、存储路径、创建人、创建时间、更新时间、操作（编辑 / 删除）。表格上方支持按数据库名称搜索与手动刷新。



后续操作

完成数据目录和数据库创建后，可以开始建表和授权：

- 创建与管理数据表
- 为子用户、用户组或 IAM 角色授权

注意事项

1. **路径归属**：数据库与数据表的存储路径建议统一放在所属 Catalog 的存储路径之下，便于按目录整体管理；也可以不绑定根路径，直接在库或表上指定路径。
2. **目录与数据解耦**：Catalog 与 Database 都是元数据记录，创建、编辑、删除元数据不会直接删除 BOS 中的数据。删除前请确认影响范围。
3. **命名约束**：名称以字母开头，仅支持字母、数字、下划线与中划线，长度 1～64 位；名称在同一父级下唯一。
4. **服务开通状态**：开通状态与账号绑定。若页面再次出现"数据湖管理服务未开通"，说明当前账号尚未完成该区域的授权，按"开通数据湖管理服务"一节重新开通即可。

湖管理权限管理

概述

权限管理用于控制成员对数据资产的可见性与操作范围。授权以数据目录、数据库、数据表为资源范围，以子用户、用户组、IAM 角色为被授权对象，通过管理员、运维、只读三档角色区分操作边界。授权后，被授权对象仅能查看和操作已授权的资源及其下级资源。

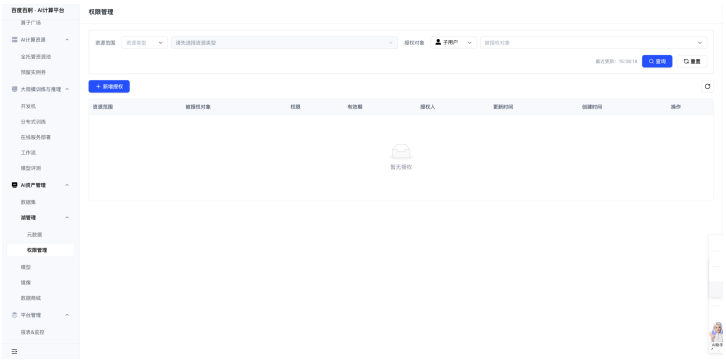
> 说明：湖管理权限控制的是元数据侧的可见性与管理操作。被授权对象访问 BOS 上的实际数据时，仍需要具备对应 BOS 路径的访问权限。

前提条件

已完成服务开通，并已创建数据目录。

权限管理页面

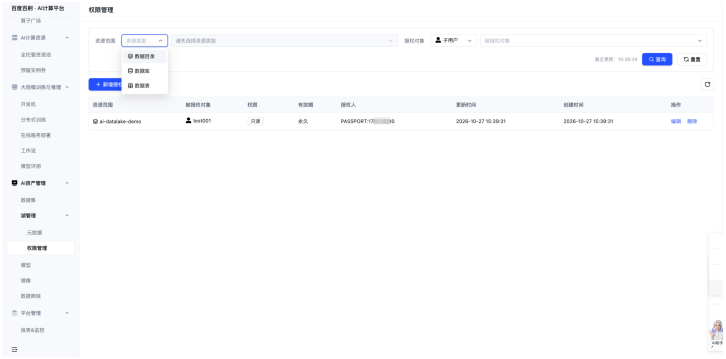
在左侧导航栏选择 **AI资产管理 > 湖管理 > 权限管理**，页面分为上方的查询区和下方的授权列表。



查询区

筛选项	说明
资源范围	先选择资源类型（数据目录 / 数据库 / 数据表），再选择具体的资源名称
授权对象	选择对象类型（子用户 / 用户组 / IAM 角色）
被授权对象	选择具体的子用户、用户组或 IAM 角色

资源类型的下拉选项与授权时的资源层级一致：



单击【查询】按条件过滤列表，单击【重置】清空条件。查询区右侧显示列表的“最近更新”时间，单击列表右上角的刷新图标可重新加载。

授权列表

列表字段包括：

字段	说明
资源范围	被授权的资源名称，带资源类型图标
被授权对象	子用户、用户组或 IAM 角色
权限	管理员 / 运维 / 只读
有效期	授权的有效期限，当前固定为“永久”，不支持设置到期时间
授权人	执行授权操作的身份标识
更新时间 / 创建时间	授权记录的创建与最近修改时间
操作	编辑、删除

没有授权记录时，列表显示“暂无授权”空状态。

新增授权

打开新增授权

单击列表左上角的【新增授权】，从右侧打开【新增授权】抽屉。



选择资源范围

在抽屉顶部的【资源范围】中选择资源层级：数据目录、数据库、数据表。

- **数据目录**：授权作用于该目录及其下所有数据库与数据表。
- **数据库**：授权作用于该数据库及其下所有数据表。
- **数据表**：授权仅作用于单张数据表。

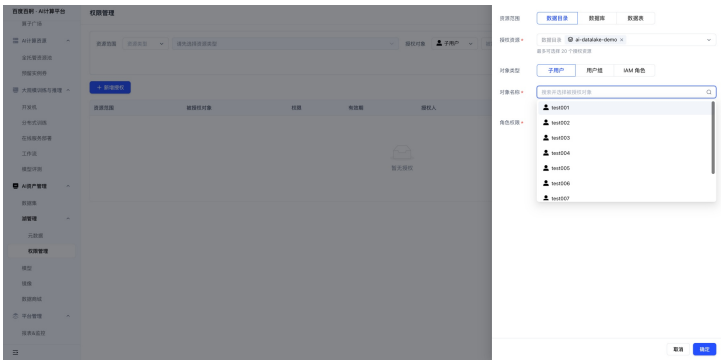
选定层级后，在【授权资源】下拉中搜索并选择具体资源，已选资源以标签形式展示，单击标签上的关闭按钮可移除。单个授权最多可选择 20 个授权资源。

选择被授权对象

【对象类型】支持三种：

对象类型	说明
子用户	百度智能云账号下的单个子用户
用户组	子用户组，组内成员同时获得授权
IAM 角色	通过角色进行跨账号或服务间授权

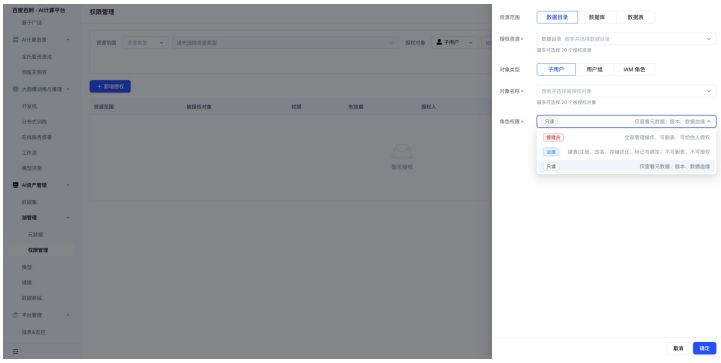
在【对象名称】中搜索并选择被授权对象，单个授权最多可选择 20 个被授权对象。



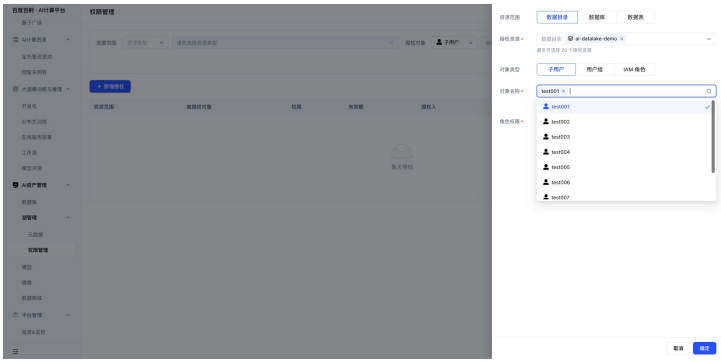
选择角色权限

【角色权限】为必填项，选择后字段右侧会显示该角色的权限说明，下方【角色权限说明】可查看三档角色的完整定义。三档角色的权限范围如下：

角色	权限范围
管理员	全部管理操作，可删表、可给他人授权
运维	建表/注册、改名、存储优化、标记与绑定；不可删表、不可授权
只读	仅查看元数据、版本、数据血缘

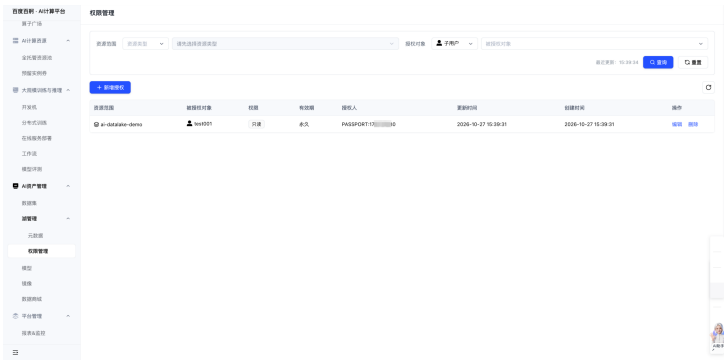


完成资源范围、被授权对象和角色权限选择后，配置示例如下：



提交授权

确认资源范围、被授权对象和角色权限后，单击【确定】。授权成功后页面提示“授权成功”，新增记录出现在授权列表中。

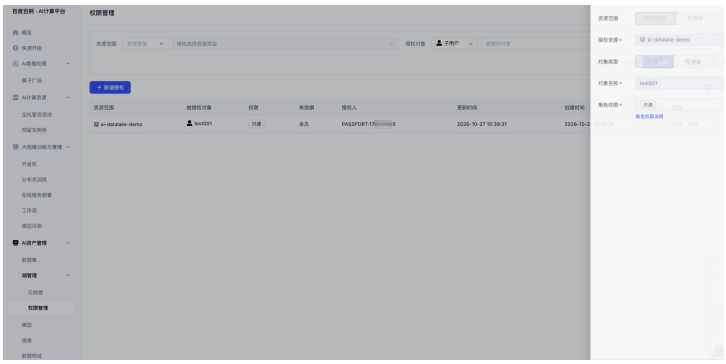


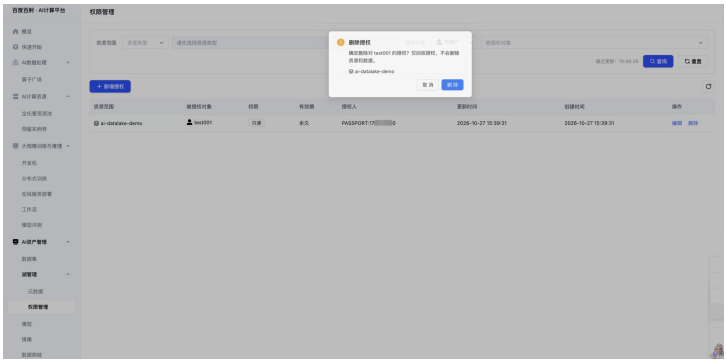
例如，将数据目录 `ai-datalake-demo` 以【只读】角色授权给子用户 `test001`，有效期为永久。授权后，列表中展示资源范围、被授权对象、权限、有效期、授权人及创建时间等完整记录。

编辑与删除授权

在授权列表的【操作】列：

- **编辑**：打开【编辑授权】抽屉，只能调整角色权限。资源范围、授权资源、对象类型、对象名称均为只读，如需变更资源或被授权对象，请删除后重新授权。
- **删除**：撤销授权。确认弹窗提示“确定删除对 <对象> 的授权？仅回收授权，不会删除资源和数据。”





权限模型与验证

资源层级与继承

权限以数据目录、数据库、数据表为资源范围。授权范围覆盖所选资源及其下级资源，对授权后新建的数据库和数据表同样生效：在数据目录上授权后，该目录下的数据库和数据表同时获得对应权限；在数据库上授权同理。

角色权限关系

三档角色为包含关系：管理员包含运维的全部能力，运维包含只读的全部能力。具体权限范围参见"选择角色权限"一节。

验证授权结果

1. 使用被授权对象登录百舸控制台。
2. 进入 **AI资产管理 > 湖管理**，确认目标资源可见，未授权资源不可见。
3. 按所授角色尝试对应操作（查看、存储优化、删除或授权）；如与预期不符，检查资源范围、被授权对象和角色权限是否配置正确。

注意事项

1. **最小权限原则**：日常查看元数据、版本与血缘时建议授予【只读】；需要执行存储优化、改名等操作时授予【运维】；删除数据表和转授权仅管理员可执行。
2. **授权与 BOS 权限分离**：湖管理授权控制元数据侧的可见性与管理操作，不授予 BOS 数据读写权限；访问 BOS 数据还需在 BOS 侧配置对应的桶或路径权限。
3. **删除授权不可恢复**：删除授权后如需恢复，需要重新发起一次授权操作，并重新选择资源范围与角色。
4. **授权人标识**：列表中的授权人显示为账号标识（如 `PASSPORT:<账号 ID>`），可用于审计授权来源。

常见问题

Q1：授权后，被授权对象看不到资源怎么办？

检查资源范围、被授权对象和角色权限是否选择正确，并确认授权记录已出现在授权列表中。湖管理授权只控制元数据侧的可见性与管理操作，访问 BOS 数据还需要在 BOS 侧配置对应路径权限。

Q2：编辑授权时为什么不能修改资源范围或被授权对象？

当前仅支持调整角色权限。如需变更资源或被授权对象，请删除原授权后重新新增。

Q3：删除授权会删除数据吗？

不会。删除授权仅回收授权关系，不删除数据目录、数据库、数据表及 BOS 中的数据。

Q4：一次可以授权多少个资源或对象？

单个授权最多选择 20 个授权资源和 20 个被授权对象；更多资源可分批授权，或在上层数据目录、数据库上授权。

表管理

概述

数据表（Table）是湖管理中最细粒度的管理单元，对应 BOS 上一个以 `.lance` 结尾的 Lance 数据目录。完成建表后，可在控制台查看表结构、版本差异、存储优化任务与数据血缘，无需跳转 BOS。

湖管理当前支持三种建表方式：

创建方式	说明	适用场景
从数据集导入	选择一个数据格式为 Lance 的百舸数据集	数据已经通过百舸数据集管理
从 BOS 导入	直接填写 Lance 表所在的 BOS 目录	数据已经在 BOS 上以 Lance 格式存放
新建空表	指定一个尚未写入数据的 BOS 路径，先登记元数据	需要先建表、后续再由训练或写入任务写入数据

从 BOS 导入时，目标目录需为合法的 Lance 数据集：目录名以 `.lance` 结尾，且包含 Lance 的 `_versions`、`data` 等元数据文件。从数据集导入时，仅需选择已关联有效 Lance 文件的数据集，页面提示"请确保数据集已关联有效的 Lance 文件"。新建空表则要求路径下没有数据，首次写入后由后台自动识别字段。

前提条件

已完成服务开通，并已创建数据目录与数据库。

进入数据库

在数据目录详情页单击数据库名称，进入库内页面。页面顶部展示 目录 / 库 面包屑，下方依次为：

- 库信息区：数据库名称、存储路径、描述信息、创建人、创建时间、更新时间；页面右上角提供【删除】【编辑】按钮。
- 数据表区：左上角为【创建数据表】入口，右上角为表名搜索与刷新按钮。
- 数据表列表：列为数据表名称、创建方式、存储路径、格式、当前版本、行数、创建人、创建时间、更新时间、操作。

数据表列表为空时显示"暂无数据表"。单击列表左上角的【创建数据表】，开始建表。



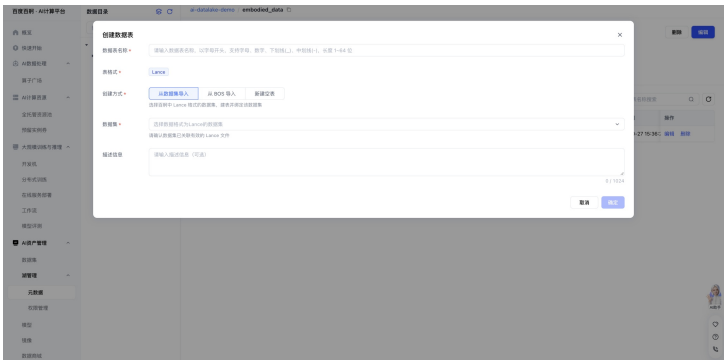
创建数据表

在创建数据表弹窗中完成以下配置。

选择创建方式

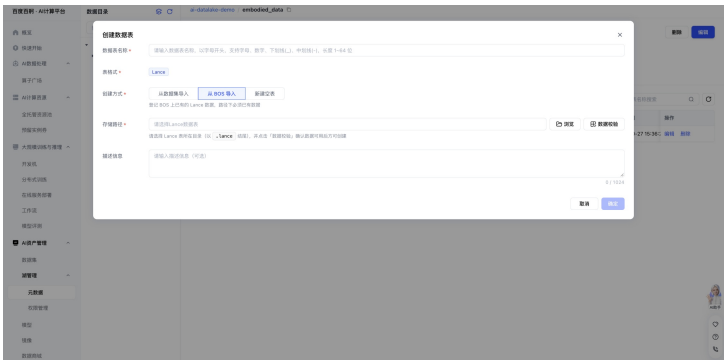
创建数据表弹窗包含以下字段：数据表名称、表格式、创建方式、数据集或存储路径、描述信息。其中【表格式】当前固定为 Lance，【创建方式】提供三个选项：

- **从数据集导入**：选择百舸中 Lance 格式的数据集，建表并绑定该数据集。
- **从 BOS 导入**：登记 BOS 上已有的 Lance 数据，路径下必须已有数据。
- **新建空表**：为还没写入的数据预先建表，路径下必须为空，首次写入后自动识别字段。



切换到【从 BOS 导入】后，弹窗会增加【存储路径】和【数据校验】按钮，并给出提示：

> 请选择 Lance 表所在目录（以 .lance 结尾），并单击【数据校验】确认数据可用后方可创建



切换到【新建空表】后，需要填写一个尚未写入数据的新路径，页面给出【存储路径建议】（bos://<数据库路径>/<数据表名称>.lance）。校验规则为确认路径下没有数据：

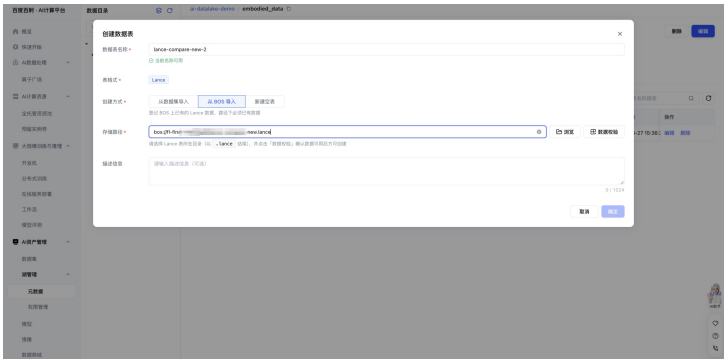
> 请填写一个还没有数据的新路径（以 .lance 结尾），并单击【数据校验】确认路径下没有数据后方可创建



填写参数

字段	说明
数据表名称	必填，以字母开头，支持字母、数字、下划线（_）、中划线（-），长度 1～64 位。输入后实时提示“当前名称可用”
表格式	必填，固定为 Lance
创建方式	必填，从数据集导入 / 从 BOS 导入 / 新建空表
存储路径（BOS 导入、新建空表）	必填，Lance 表所在目录，形如 `bos://<桶名>/<路径>/<表名>.lance`。从 BOS 导入时路径下必须已有数据；新建空表时路径下必须为空
存储路径建议（新建空表）	系统根据数据库路径和表名生成 `bos://<数据库路径>/<表名>.lance`，单击【使用建议路径】可填入
数据集（数据集导入）	必填，仅列出数据格式为 Lance 的数据集；下拉中展示数据集名称、ID、所有者与创建时间
描述信息	选填，最长 1024 个字符

存储路径可以直接输入，也可以单击【浏览】在桶树中逐级选择。从 BOS 导入或新建空表时，【确定】按钮在数据校验完成前保持不可用。

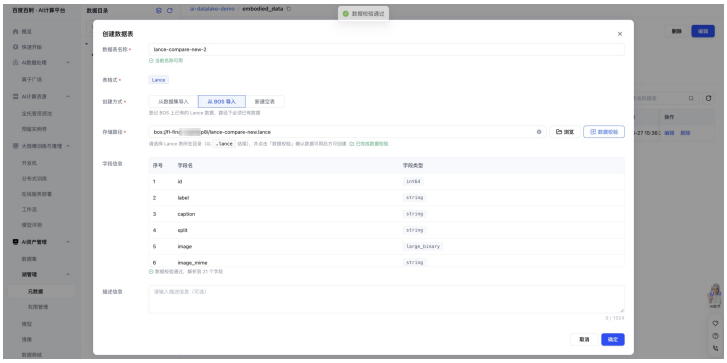


数据校验

三种创建方式的校验方式不同：

- **从数据集导入**：选择数据集后自动解析字段，无需手动校验。解析成功时显示“成功解析 N 个字段”，并在【字段信息】中列出字段。
- **从 BOS 导入**：单击【数据校验】读取目录元数据并解析 Schema。校验通过后，路径下方显示“已完成数据校验”，弹窗内展示【字段信息】表，底部提示“数据校验通过，解析到 N 个字段”。
- **新建空表**：单击【数据校验】确认路径下没有数据。校验通过后，路径下方显示“路径为空，可新建空表，首次写入数据后自动识别字段”，不展示字段信息。

校验通过后，【确定】按钮变为可用。



从 BOS 导入时，字段信息表包含序号、字段名、字段类型三列；从数据集导入时还会展示 Metadata 列（如 `bos-dataset-field:modality`、`lance-encoding:blob`）。字段较多时可在弹窗内滚动查看。

校验不通过时，按页面提示处理：

页面提示	处理方式
路径需以 .lance 结尾	补全目录名，确保以 `.lance` 结尾
存储路径不能以“/”结尾	去掉路径末尾的斜杠
该路径下还没有数据，无法从 BOS 导入。如需先建表、等作业首次写入，请改用「新建空表」	该路径尚无数据；如需先建表、后续由作业写入，单击提示条右侧的按钮切换到新建空表方式
服务读取超时或失败	稍后重试；持续失败时检查该路径是否可被数据湖管理服务角色访问

完成创建

单击【确定】后，列表中出现新表，列包括：数据表名称、创建方式、存储路径、格式、当前版本、行数、创建人、创建时间、更新时间、操作（编辑 / 删除）。



从 BOS 导入或从数据集导入的新表，在后台扫描回填完成前，【当前版本】显示“待写入”、行数为 0；刷新页面即可看到实际版本号与行数。新建空表在首次写入前没有字段信息，首次写入后由后台自动识别并回填 Schema；版本号以页面显示为准。

数据表详情

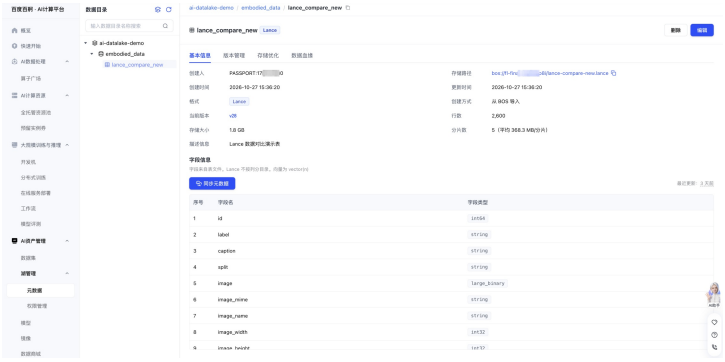
单击表名进入详情页。页面顶部为面包屑 目录 / 库 / 表，右上角提供【删除】【编辑】，下方分为四个页签：基本信息、版本管理、存储优化、数据血缘。

基本信息

基本信息区展示：

字段	示例值
创建人	PASSPORT:***
存储路径	bos://<bucket>/<path>/lance-compare-new.lance
创建时间 / 更新时间	2026-10-27 15:36:20
格式	Lance
创建方式	从 BOS 导入
当前版本	v28
行数	2,600
存储大小	1.8 GB
分片数	5（平均 368.3 MB/分片）
描述信息	建表时填写的描述

> 说明：当前版本、分片数与存储大小会随写入和存储优化持续变化，上表数值及版本相关截图仅为示例，实际以控制台显示为准。



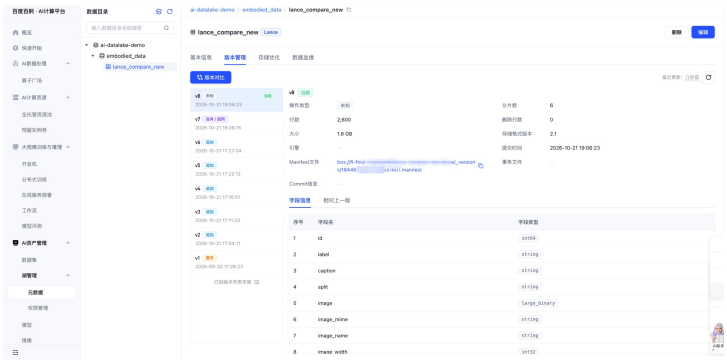
【字段信息】区说明：字段来自表文件，Lance 不按列分目录，向量列以 vector(n) 表示。区域右上角显示最近更新时间，区域左侧提供【同步元数据】按钮，用于在表结构变化后重新拉取字段。字段表按序号列出字段名与字段类型，例如 fixed_size_list<item: float>[768] 表示 768 维向量列。

版本管理

版本管理页左侧是该表的版本时间线，每个版本标注版本号、操作类型与提交时间，当前版本带“当前”标记。示例数据中可见的操作类型包括：覆写、追加、合并 / 加列、未知。

右侧展示选中版本的详情：

指标	说明
操作类型	该版本的写入类型
分片数	该版本的数据分片（fragment）数量
行数 / 删除行数	该版本的总行数与删除行数
大小	该版本数据大小
存储格式版本	Lance 存储格式版本，例如 2.1
引擎	写入该版本的引擎，未上报时显示 --
提交时间	版本提交时间
Manifest 文件	该版本对应的 manifest 路径
事务文件 / Commit 信息	未上报时显示 -

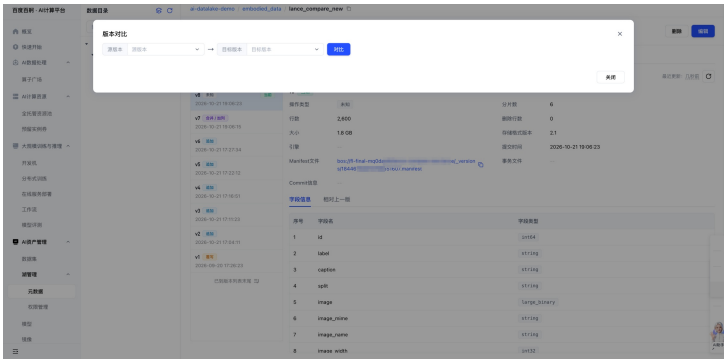


字段信息区域支持在【字段信息】与【相对上一版】之间切换，用于查看当前版本的完整 Schema 或相对上一版本的变化。两个版本 Schema 一致时，页面提示“相对 vN 无 Schema 变化”（N 为上一版本号）。

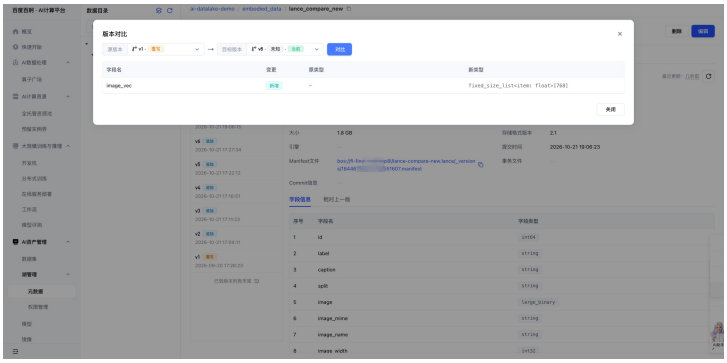


版本对比

单击【版本对比】，在弹窗中选择【源版本】和【目标版本】后单击【对比】。



对比结果按“字段名 / 变更 / 原类型 / 新类型”展示差异。例如 v1 → v8 的对比结果显示 `image_vec` 为“新增”，原类型为 `-`，新类型为 `fixed_size_list<item: float>[768]`；若两个版本 Schema 完全一致，则提示“两个版本 Schema 一致”。



存储优化

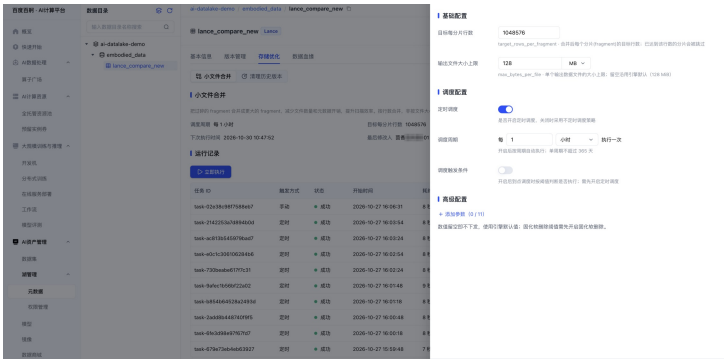
存储优化页包含【小文件合并】与【清理历史版本】两个子页签。每个子功能提供功能说明、右上角的启用开关（开启后显示【配置】入口）以及下方的【运行记录】列表。运行记录列为：任务 ID、触发方式、状态、开始时间、耗时、说明、操作；开关未开启时，【立即执行】按钮不可用。

小文件合并

将碎片化的 fragment 合并为更大的 fragment，减少文件数量与元数据开销，提升扫描效率。合并按行数进行，不按文件大小。

首次开启时，单击右上角开关会打开【开启小文件合并】配置抽屉，单击【确定】后开关置为开启；已开启后，通过【配置】按钮打开的是【编辑小文件合并配置】，两者字段一致：

分组	字段	说明
基础配置	**目标每分片行数**	target_rows_per_fragment，合并后每个分片的目标行数，已达到该行数的分片会被跳过，默认 1048576
基础配置	**输出文件大小上限**	max_bytes_per_file，单个输出数据文件的大小上限，默认 128 MB，留空沿用引擎默认
调度配置	**定时调度**	关闭时采用不定时调度策略，调度周期显示为“不定时”，仅能手动执行
调度配置	**调度周期**	开启定时调度后按“每 N 时/天执行一次”的周期自动执行，单周期不超过 365 天
调度配置	**调度触发条件**	到点调度时按阈值判断是否执行，需先开启定时调度
高级配置	**添加参数**	最多可添加 11 个参数，数值留空即不下发，使用引擎默认值；固化软删除阈值需先开启固化软删除



开启后，页面展示调度周期、下次执行时间（未开启定时调度时不显示）、目标每分片行数、输出文件大小上限与最后修改人，【配置】入口常驻，【立即执行】变为可用。



单击【立即执行】提交任务，页面提示“已提交执行任务”，运行记录新增一条触发方式为“手动”的记录。任务完成后，状态由执行中变为“成功”，并显示耗时与说明（例如“将 5 个分片合并为 5 个”）。任务执行期间再次单击【立即执行】，提示“已有执行任务进行中（后台检测刷新），无需重复触发”。

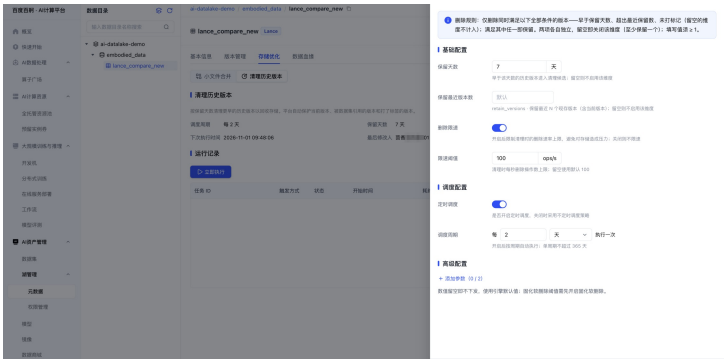
清理历史版本

按保留天数清理更早的历史版本以回收存储。平台自动保护当前版本、被数据集引用的版本和打了标签的版本。

删除规则为：仅删除同时满足“早于保留天数、超出最近保留数、未打标记”的版本；满足其中任一条件即保留。保留天数与保留最近版本数两项各自独立，留空即关闭该维度，且至少保留一个维度；填写值需大于等于 1。

已开启后，单击【配置】，打开【编辑清理历史版本配置】抽屉，配置如下：

分组	字段	说明
基础配置	**保留天数**	早于该天数的历史版本进入清理候选，留空则不启用该维度
基础配置	**保留最近版本数**	retain_versions，最近保留 N 个现存版本（含当前版本），留空则不启用该维度
基础配置	**删除限速**	开启后限制清理时的删除速率上限，关闭则不限制
基础配置	**限速阈值**	每秒删除操作数上限，留空使用默认 100 ops/s
调度配置	**定时调度 / 调度周期**	与【小文件合并】一致，可设为不定时或按周期自动执行
高级配置	**添加参数**	最多可添加 2 个参数，数值留空即不下发，使用引擎默认值



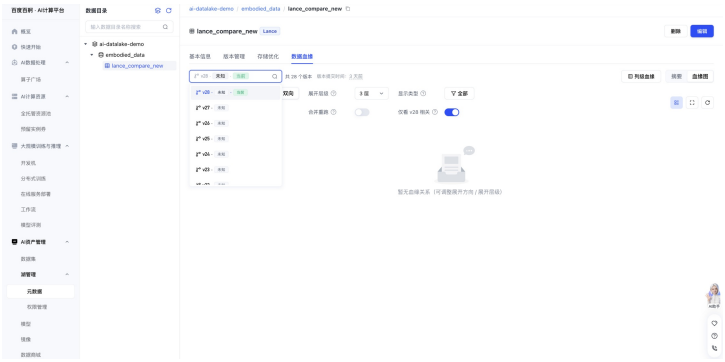
页面展示调度周期、保留天数、删除限速、下次执行时间（未开启定时调度时不显示）与最后修改人；运行记录包含任务 ID、触发方式、状态、开始时间、耗时、说明与操作。



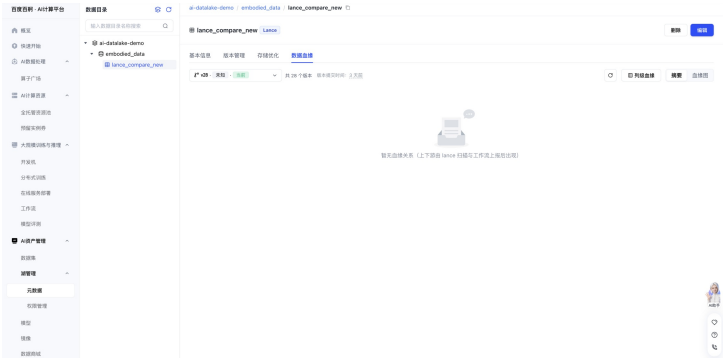
> 说明：小文件合并与清理历史版本**会实际改写或删除 BOS 上的数据文件**，建议在业务低峰期执行，并在执行前确认没有任务正在读取待清理的版本。

数据血缘

数据血缘页顶部提供版本选择器，作用于页内全部视图。下拉按版本倒序列出全部版本，每项标注操作类型，当前版本带“当前”标记。选择器右侧显示“共 N 个版本”与所选版本的提交时间；切换版本后，页面按所选版本重新加载视图。



视图支持【列级血缘】【摘要】【血缘图】三种展示方式，视图切换区旁提供刷新按钮。



切换到【血缘图】后，页面提供以下筛选：

- 展开方向：上游 / 下游 / 双向；
- 展开层级：如 3 层；
- 显示类型：全部；
- 执行节点：聚合 / 展开；
- 开关：【合并重跑】、【仅看 某版本 相关】。

其中【仅看 某版本 相关】的版本号跟随顶部版本选择器变化，开启后仅保留与所选版本相关的节点。视图右侧支持列表视图、全屏与刷新。



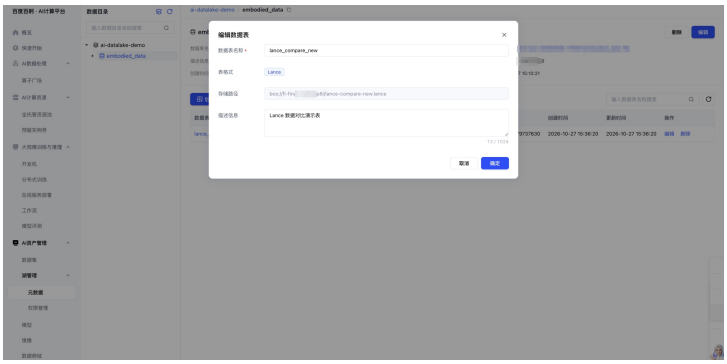
> 说明：血缘图按版本查看，控制台不支持两个版本并排对比。需要比较版本间的 Schema 变化时，请使用【版本管理】中的【版本对比】。

当上下游关系尚未上报时，摘要视图提示“暂无血缘关系（上下游由 lance 扫描与工作流上报后出现）”。血缘图视图提示“暂无血缘关系（可调整展开方向/展开层级）”。

编辑与删除

表列表的操作列与表详情右上角均提供【编辑】与【删除】：

- **编辑**：打开【编辑数据表】弹窗，可修改数据表名称与描述信息；表格式与存储路径只读，不能通过编辑修改。
- **删除**：从湖管理中移除该表的元数据记录。



> 说明：删除表会移除该表的元数据记录，依赖该表元数据的下游配置可能受影响，操作前请确认影响范围。**BOS 上的 Lance 数据文件不会因为删除元数据记录而被清理**；如需回收存储空间，请前往 BOS 控制台删除对应目录。

使用限制

项目	限制
数据表名称	以字母开头，支持字母、数字、下划线（_）、中划线（-），长度 1～64 位；名称在所属数据库内唯一
描述信息	最长 1024 个字符
表格式	当前固定为 Lance
从 BOS 导入	路径以 .lance 结尾，路径下必须已有数据，且通过【数据校验】
新建空表	路径以 .lance 结尾，路径下必须没有数据，且通过【数据校验】
从数据集导入	仅支持已关联有效 Lance 文件的数据集
小文件合并高级参数	最多 11 个
清理历史版本高级参数	最多 2 个
定时调度周期	单周期不超过 365 天
清理历史版本保留策略	保留天数与保留最近版本数至少启用一个维度，填写值需大于等于 1

注意事项

1. **格式限制**：当前表格式仅支持 Lance，其他格式（如 Iceberg）暂不支持。
2. **路径校验**：从 BOS 导入或新建空表时，建表前必须通过【数据校验】；校验会实际访问目标路径，大表的首次校验耗时较长。从数据集导入时，选择数据集后由系统自动解析字段。
3. **版本号来源**：版本信息来自 Lance 自身的版本目录（_versions），版本管理页只做展示与对比，不修改版本数据；在控制台外写入的新版本，刷新页面后即可看到。
4. **存储优化影响数据**：小文件合并与清理历史版本会改写数据文件，执行前请确认上下游任务状态。
5. **血缘依赖上报**：血缘关系依赖 Lance 扫描与工作流上报，未接入上报时血缘图为空。

API参考

API功能更新记录

发布时间	接口/特性	更新介绍
2025-05-21	百舸API v2版本全新发布	采用RPC风格，首批发布训练任务相关接口

概述

> 当前你查看的是最新的百舸v2版本API，v1版本即将废弃，推荐使用当前版本。如果已经在业务中集成v1版本，建议尽快完成迁移。

欢迎使用百度智能云百舸AI计算平台，您可以使用本文档介绍的API对百舸服务进行灵活的操作。

如果您是初次调用百度智能云产品的API，可以观看 API 入门视频指南，快速掌握调用API的能力。

分布式训练相关接口

接口类型	接口描述
创建训练任务	POST ?action=CreateJob&resourcePoolId=xxxx
查询训练任务列表	POST ?action=DescribeJobs&resourcePoolId=xxxx
查询训练任务详情	GET ?action=DescribeJob&resourcePoolId=xxxx
删除训练任务	POST ?action=DeleteJob&resourcePoolId=xxxx
更新训练任务	POST ?action=ModifyJob&&resourcePoolId=xxxx
查询训练任务事件	GET ?action=DescribeJobEvents&resourcePoolId=xxxx
查询训练任务日志	GET ?action=DescribeJobLogs&resourcePoolId=xxxx
查询训练任务Pod事件	GET ?action=DescribePodEvents&resourcePoolId=xxxx
停止训练任务	POST ?action=StopJob&resourcePoolId=xxxx
查询训练任务监控	GET ?action=DescribeJobMetrics&resourcePoolId=xxxx
查询训练任务所在节点列表	GET ?action=DescribeJobNodes&resourcePoolId=xxxx
获取训练任务WebTerminal地址	GET ?action=DescribeJobWebterminal&resourcePoolId=xxxx

在线服务部署相关接口

接口类型	接口描述
创建服务	POST ?action=CreateService
拉取服务列表	GET ?action=DescribeServices&pageNumber=pageNumber&pageSize=pageSize&orderBy=orderBy&order=order
获取服务状态	GET ?action=DescribeServiceStatus&serviceId=serviceId
查询服务详情	GET ?action=DescribeService&serviceId=serviceId
升级服务	POST ?action=ModifyService&serviceId=serviceId&description=description
服务扩缩容	POST ?action=ModifyServiceReplicas&instanceCount=instanceCount
配置公网访问	POST ?action=ModifyServiceNetConfig&publicAccess=publicAccess&eip=eip
拉取服务变更记录	GET ?action=DescribeServiceChangelogs&serviceId=serviceId
查询服务变更详情	GET ?action=DescribeServiceChangelog&changeId=changeId
删除服务	POST ?action=DeleteService
拉取服务pod列表	GET ?action=DescribeServicePods&serviceId=serviceId
摘除pod流量	POST ?action=DisableServicePod&serviceId=serviceId&instanceId=instanceId&block=true
删除pod并重建	POST ?action=DeleteServicePod&serviceId=serviceId&instanceId=instanceId
获取实例组列表	GET ?action=DescribeServicePodGroups&serviceId=serviceId

通用说明

API认证机制

所有API的安全认证一律采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串。对于每个HTTP请求，使用下面所描述的算法生成一个认证字符串。提交认证放在Authorization头域里。服务端根据生成算法验证认证字符串的正确性。认证字符串的格式为bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}。

- version是正整数，目前取值为1。
- timestamp 是生成签名时的 UTC 时间。
- expirationPeriodInSeconds表示签名有效期限。
- signedHeaders 是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如 host;x-bce-date。列表按照字典序排列。（本 API 签名仅使用 host 和 x-bce-date 两个 header）
- signature是256位签名的十六进制表示，由64个小写字母组成。其生成算法见下。

当百度智能云接收到用户的请求后，系统将使用相同的 SK 和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行对比。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。鉴权认证机制的详细内容请参见鉴权认证机制。

鉴权示例

python实现签名参考脚本

> 以下是通用的接口调用签名方法，调用接口时需要根据接口实际需要的headers进行签名

```
 import datetime
import hmac, hashlib
import time
import urllib.parse
import requests


def get_headers(url, method: str, ak, sk):
    domain = urllib.parse.urlparse(url).netloc
    uri = urllib.parse.urlparse(url).path
    gcloud_params = {}
    if url.find('?') > 0:
        tmp = url.split('?')[1]
        for item in tmp.split('&'):
            if item.find('=') > 0:
                gcloud_params[item.split('=')[0]] = item.split('=')[1]
            else:
                gcloud_params[item] = ""
        gcloud_uri = url.split('?')[0]

    headers = {
        "x-bce-date": get_canonical_time(),
        "Content-Type": "application/json",
        "Host": domain,
        "X-API-VERSION": "v2"
```

```
        'x-API-VERSION': 'v2'
    } #
    bce_request = {
        'uri': uri,  # f"http://{domain}:8793/api/cce/service/v2/cluster/cce-lksgpvx5"
        'params': gcloud_params,
        'method': method.upper(),
        'headers': headers
    }
    # print(bce_request)
    auth = gen_authorization(bce_request, ak, sk)
    headers['Authorization'] = get_utf8_value(auth)
    return headers


def get_canonical_time(timestamp=0):
    """
    Get cannonical time.

    :type timestamp: int
    :param timestamp: None
    ========================
    :return:
        **string of canonical_time**
    """
    if timestamp == 0:
        utctime = datetime.datetime.utcnow()
    else:
        utctime = datetime.datetime.utcfromtimestamp(timestamp)
    return "%04d-%02d-%02dT%02d:%02d:%02dZ" % (
        utctime.year, utctime.month, utctime.day,
        utctime.hour, utctime.minute, utctime.second)


def get_canonical_time(timestamp=0):
    """
    Get cannonical time.

    :type timestamp: int
    :param timestamp: None
    ========================
    :return:
        **string of canonical_time**
    """
    if timestamp == 0:
        utctime = datetime.datetime.utcnow()
    else:
        utctime = datetime.datetime.utcfromtimestamp(timestamp)
    return "%04d-%02d-%02dT%02d:%02d:%02dZ" % (
        utctime.year, utctime.month, utctime.day,
        utctime.hour, utctime.minute, utctime.second)


def gen_authorization(request, ak, sk, timestamp=None, expire_period=1800):
    """
    generate authorization string
    if not specify timestamp, then use current time;
    """
    signedheaders = []
    if "headers" in request:
        signedheaders = list(key.lower() for key in request["headers"].keys() if key != '')
        signedheaders.sort()
    authorization = build_authorization(ak, signedheaders, expire_period, timestamp)
    signingkey = _calc_signingkey(authorization, sk)
    signature = _calc_signature(signingkey, request, signedheaders)
    authorization["signature"] = signature
    return serialize_authorization(authorization)


def serialize_authorization(auth):
    """
    serialize Authorization object to authorization string
    """
    val = "/".join((auth['version'], auth['access'], auth['timestamp'], auth['period'],
                    ";".join(auth['signedheaders']), auth['signature']))
    return get_utf8_value(val)


def build_authorization(accesskey, signedheaders, period=1800, timestamp=None):
    """
    build Authorization object
    """
    auth = {}
    auth['version'] = "bce-auth-v1"
    auth['access'] = accesskey
    if not timestamp:
        auth['timestamp'] = get_canonical_time()
    else:
        auth['timestamp'] = timestamp
    auth['period'] = str(period)
    auth['signedheaders'] = signedheaders
    return auth


def _calc_signingkey(auth, sk):
    """ Get a a signing key """
    string_to_sign = "/".join((auth['version'], auth['access'],
                               auth['timestamp'], auth['period']))

    signingkey = hmac.new(bytes(sk, 'utf-8'), bytes(string_to_sign, 'utf-8'),
                          hashlib.sha256).hexdigest()
```

```
    return signingkey


def get_utf8_value(value):
    """
    Get the UTF8-encoded version of a value.
    """
    if isinstance(value, bytes):
        return value.decode('utf-8')
    return value.encode('utf-8')


def normalized_uri(uri):
    """
    Construct a normalized(except slash '/') uri
    eg. /json-api/v1/example/ ==> /json-api/v1/example/
    """
    return urllib.parse.quote(get_utf8_value(uri), safe='-_.~/')


def normalized(msg):
    """
    Construct a normalized uri
    """
    return urllib.parse.quote(get_utf8_value(msg), safe='-_.~')


def canonical_qs(params):
    """
    Construct a sorted, correctly encoded query string
    """
    keys = list(params)
    keys.sort()
    pairs = []
    for key in keys:
        if key == "authorization":
            continue
        val = normalized(params[key])
        pairs.append(urllib.parse.quote(key, safe='') + '=' + val)
    qs = '&'.join(pairs)
    return qs


def canonical_header_str(headers, signedheaders=None):
    """
    calculate canonicalized header string
    """
    headers_norm_lower = dict()
    for (k, v) in headers.items():
        key_norm_lower = normalized(k.lower())
        value_norm_lower = normalized(v.strip())
        headers_norm_lower[key_norm_lower] = value_norm_lower
    keys = list(headers_norm_lower)
    keys.sort()
    if "host" not in keys:
        raise ValueError
    header_list = []
    default_signed = ("host", "content-length", "content-type", "content-md5")
    if signedheaders:
        for key in signedheaders:
            key = normalized(key.lower())
            if key not in keys:
                raise ValueError
            if headers_norm_lower[key]:
                header_list.append(key + ":" + headers_norm_lower[key])
    else:
        for key in keys:
            if key.startswith("x-bce-") or key in default_signed:
                header_list.append(key + ":" + headers_norm_lower[key])
    return '\n'.join(header_list)


def _calc_signature(key, request, signedheaders):
    """Generate BCE signature string."""
    # Create canonical request
    params = {}
    headers = {}
    # print request
    if "params" in request:
        params = request['params']
    if "headers" in request:
        headers = request['headers']
    cr = "\n".join((request['method'].upper(),
                    normalized_uri(request['uri']),
                    canonical_qs(params),
                    canonical_header_str(headers, signedheaders)))
    signature = hmac.new(bytes(key, 'utf-8'), bytes(cr, 'utf-8'), hashlib.sha256).hexdigest()
    return signature


def send_request(url, method, ak, sk, headers={}, data=None, json=None):
    """
    封装发送请求
    """
    headers.update(get_headers(url, method, ak, sk))
    request_func = getattr(requests, method)
    if data:
        res = request_func(url, headers=headers, data=data)
    elif json:
        res = request_func(url, headers=headers, json=json)
```

```
    else:
        res = request_func(url, headers=headers)
    return res.json()


if __name__ == '__main__':
    ak = "xxxx"
    sk = "xxx"
    # 方式1
    jobcreate_url="http://aihc.bj.baidubce.com/?action=DescribeJobs&resourcePoolId=aihc-bhcmp"
    print('-------方式1--------')
    body = {
        "queue":"xxx"
    }
    print(send_request(jobcreate_url, "post", ak, sk, json=body))
    # 方式2
    print('-------方式2--------')
    get_headers(jobcreate_url, "post", ak, sk)
    res = requests.post(jobcreate_url, headers=get_headers(jobcreate_url, "post", ak, sk), json=body).json()
    print(res)
```

日期与时间规定

日期时间表示的地方一律采用UTC时间，遵循ISO 8601，并做以下约束：

1. 表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日
2. 表示时间一律采用hh:mm:ss方式，并在最后加一个大写字母Z表示UTC时间。例如23:00:10Z表示UTC时间23点0分10秒。
3. 凡涉及日期和时间合并表示时，在两者中间加大写字母T，例如2014-06-01T23:00:10Z表示UTC时间2014年6月1日23点0分10秒。

多语言请求示例

百度云已提供多语言基础SDK实现签名认证，您可以基于百度云SDK自行实现百舸接口请求，以下是几种常用语言的客户端实现示例

Python

依赖先安装百度云基础SDK `pip install bce-python-sdk`

```
 import copy
import json
from baidubce import bce_base_client
from baidubce.exception import BceHttpClientError, BceServerError
from baidubce.auth import bce_v1_signer
from baidubce.http import handler, bce_http_client, http_methods
from baidubce.utils import required, Expando
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)


def parse_json(http_response, response):
    """
    解析HTTP响应中的JSON数据并更新response对象。

    :param http_response: HTTP响应对象
    :param response: 要更新的响应对象
    :return: 解析是否成功
    """
    body = http_response.read()
    if body:
        try:
            parsed_data = json.loads(body, object_hook=dict_to_python_object)
            response.__dict__.update(parsed_data.__dict__)

            # 移除metadata key（如果存在）
            if 'metadata' in response.__dict__:
                del response.__dict__['metadata']
        except json.JSONDecodeError as e:
            __logger.error('Failed to parse JSON response: %s', e)
            return False
    http_response.close()
    return True


def dict_to_python_object(d):
    """
    将字典转换为Expando对象，用于JSON解析时的对象钩子。

    :param d: 要转换的字典
    :return: Expando对象
    """
    attr = {}
    for k, v in list(d.items()):
        k = str(k)
        attr[k] = v
    return Expando(attr)

def to_dict(obj):
    if isinstance(obj, dict):
        return {k: to_dict(v) for k, v in obj.items()}
    elif hasattr(obj, '__dict__'):
        return {k: to_dict(v) for k, v in obj.__dict__.items()}
    elif isinstance(obj, list):
```

```
        return [to_dict(i) for i in obj]
    else:
        return obj

class AIHCV2Client(bce_base_client.BceBaseClient):
    """
    AIHC V2 base sdk client
    """

    version = b'v2'

    def __init__(self, config=None):
        bce_base_client.BceBaseClient.__init__(self, config)

    def _compute_service_id(self):
        return b'aihc'

    def _merge_config(self, config=None):
        if config is None:
            return self.config
        else:
            new_config = copy.copy(self.config)
            new_config.merge_non_none_values(config)
            return new_config

    def _send_request(self, http_method, path,
                      body=None, headers=None, params=None,
                      config=None, body_parser=None):
        config = self._merge_config(config)
        if body_parser is None:
            body_parser = parse_json

        if headers is None:
            headers = {
                b'version': AIHCV2Client.version,
                b'Content-Type': b'application/json'
            }
        else:
            headers[b'version'] = AIHCV2Client.version
            headers[b'Content-Type'] = b'application/json'

        return bce_http_client.send_request(
            config, bce_v1_signer.sign, [handler.parse_error, body_parser],
            http_method, path, body, headers, params)

    def _send_job_request(self, http_method, path,
                      body=None, headers=None, params=None,
                      config=None, body_parser=None):
        config = self._merge_config(config)
        if body_parser is None:
            body_parser = parse_json

        if headers is None:
            headers = {
                b'X-API-Version': AIHCV2Client.version
            }
        else:
            headers[b'X-API-Version'] = AIHCV2Client.version

        return bce_http_client.send_request(
            config, bce_v1_signer.sign, [handler.parse_error, body_parser],
            http_method, path, body, headers, params)

#### ============任务相关接口============
    @required(pageNumber=int, pageSize=int)
    def DescribeJobs(
        self,
        resourcePoolId,
        queueID=None,
        queue=None,
        status=None,
        keywordType=None,
        keywork=None,
        orderBy=None,
        order=None,
        pageNumber=1,
        pageSize=10
    ):
        """
        查询训练任务列表。

        参考文档：https://cloud.baidu.com/doc/AIHC/s/xmayvctia

        :param resourcePoolId:
            资源池唯一标识符（必填）
        :type resourcePoolId: string

        :param queueID:
            托管资源池需传入该参数，为队列Id（可选，Query参数）
        :type queueID: string

        :param queue:
            训练任务所属队列，通用资源池须填入队列名称，不填时返回所有。托管资源池须填入队列Id（可选，Body参数）
        :type queue: string

        :param status:
            基于状态筛选任务（可选，Body参数）
        :type status: string

        :param keywordType:
```

```
        筛选关键字类型（可选，Body参数）
    :type keywordType: string

    :param keywork:
        关键字值，当前仅支持name/queueName（可选，Body参数）
    :type keywork: string

    :param orderBy:
        排序字段，支持createdAt，finishedAt，默认为createdAt（可选，Body参数）
    :type orderBy: string

    :param order:
        排序方式，可选 [asc, desc]，asc为升序，desc为降序，默认desc（可选，Body参数）
    :type order: string

    :param pageNumber:
        请求分页参数，表示第几页（可选，Body参数）
    :type pageNumber: int

    :param pageSize:
        单页结果数，默认值为10（可选，Body参数）
    :type pageSize: int

    :return:
        返回训练任务列表
    :rtype: baidubce.bce_response.BceResponse
    """
    path = b'/'
    params = {
        'action': 'DescribeJobs',
        'resourcePoolId': resourcePoolId,
    }
    if queueID:
        params['queueID'] = queueID

    body = {}
    if queue is not None:
        body['queue'] = queue
    if status is not None:
        body['status'] = status
    if keywordType is not None:
        body['keywordType'] = keywordType
    if keywork is not None:
        body['keywork'] = keywork
    if orderBy is not None:
        body['orderBy'] = orderBy
    if order is not None:
        body['order'] = order
    if pageNumber is not None:
        body['pageNumber'] = pageNumber
    if pageSize is not None:
        body['pageSize'] = pageSize

    return self._send_job_request(
        http_methods.POST,
        path,
        body=json.dumps(body),
        params=params
    )


@required(datasetId=str)
def ModifyDataset(self, datasetId, name=None, description=None, visibilityScope=None, visibilityUser=None, visibilityGroup=None):
    """
    修改数据集。

    参考文档：https://cloud.baidu.com/doc/AIHC/s/Imc095v8z

    :param datasetId: 数据集ID（必填，Query参数）
    :type datasetId: str
    :param name: 数据集名称（可选，Body参数）
    :type name: str
    :param description: 数据集描述（可选，Body参数）
    :type description: str
    :param visibilityScope: 可见范围（可选，Body参数）
        可选值：ALL_PEOPLE（所有人可见）、ONLY_OWNER（仅所有者可读写）、USER_GROUP（指定范围可用）
    :type visibilityScope: str
    :param visibilityUser: 用户权限列表（可选，Body参数）
        格式：[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]
    :type visibilityUser: list
    :param visibilityGroup: 用户组权限列表（可选，Body参数）
        格式：[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]
    :type visibilityGroup: list
    :return: 修改结果
    :rtype: baidubce.bce_response.BceResponse
    """
    path = b'/'
    params = {
        'action': 'ModifyDataset',
        'datasetId': datasetId,
    }
    body = {}
    if name is not None:
        body['name'] = name
    if description is not None:
        body['description'] = description
    if visibilityScope is not None:
        body['visibilityScope'] = visibilityScope
    if visibilityUser is not None:
        body['visibilityUser'] = visibilityUser
```

```
        if visibilityGroup is not None:
            body['visibilityGroup'] = visibilityGroup
        return self._send_request(
            http_methods.POST,
            path,
            body=json.dumps(body),
            params=params
        )


if __name__ == '__main__':
    HOST = 'https://aihc.bj.baidubce.com'
    AK = 'your-access-key-id'
    SK = 'your-secret-access-key'

    resourcePoolId = "cce-hcuw9ybk"
    datasetId = "ds-CgyRs2EE"
    name = "test-dataset-2xxx"

    config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

    # create a aihc client
    aihc_client = AIHCV2Client(config)

    # 查询任务列表
    try:
        __logger.info('------------------DescribeJobs start--------------------------------')
        response = aihc_client.DescribeJobs(resourcePoolId=resourcePoolId)
        # 将response转换为python对象
        print(json.dumps(to_dict(response), ensure_ascii=False))

        # response对象的全部key
        __logger.info('response.__dict__.keys(): %s', response.__dict__.keys())

    except BceHttpClientError as e:
        if isinstance(e.last_error, BceServerError):
            __logger.error('send request failed. Response %s, code: %s, msg: %s'
                           % (e.last_error.status_code, e.last_error.code, e.last_error.message))
        else:
            __logger.error('send request failed. Unknown exception: %s' % e)

    # 修改数据集
    try:
        __logger.info('------------------ModifyDataset start--------------------------------')

        # 示例1：仅修改数据集名称
        response = aihc_client.ModifyDataset(datasetId=datasetId, name=name)
        print("修改数据集名称结果:", response)
        print(json.dumps(to_dict(response), ensure_ascii=False, indent=2))

    except BceHttpClientError as e:
        if isinstance(e.last_error, BceServerError):
            __logger.error('send request failed. Response %s, code: %s, msg: %s'
                           % (e.last_error.status_code, e.last_error.code, e.last_error.message))
        else:
            __logger.error('send request failed. Unknown exception: %s' % e)
```

服务域名

百舸OpenAPI通过公网域名按地域接入：

Region	Endpoint	Protocol
北京	aihc.bj.baidubce.com	HTTP and HTTPS
广州	aihc.gz.baidubce.com	HTTP and HTTPS
苏州	aihc.su.baidubce.com	HTTP and HTTPS
保定	aihc.bd.baidubce.com	HTTP and HTTPS
武汉	aihc.fwh.baidubce.com	HTTP and HTTPS
阳泉	aihc.yq.baidubce.com	HTTP and HTTPS

说明： API 支持 HTTP 和 HTTPS 两种调用方式。为了提升数据的安全性，建议通过 HTTPS 调用。

请求头和错误返回

请求头和错误返回

请求头

名称	类型	是否必选	参数位置	描述	示例值
version	String	是	header	API接口的版本。 当前文档对应的百舸API版本为v2，调用接口时version需固定使用v2。	v2
action	String	是	params	API操作的名称。必须唯一，格式：操作类型 + 资源名称（单数/复数） 取值参考具体业务的接口文档说明 action 小写，操作名称首字母大写	例如训练任务创建接口，为：action=CreateJob
clientToken	String	否	params	当调用创建接口时如果遇到了请求超时或服务器内部错误，用户可能会尝试重发请求，这时用户通过clientToken参数避免创建出比预期要多的资源，即保证请求的幂等性。详细参考链接 幂等性	clientToken=be31b98c-5e41-4838-9830-9be700de5a20
authorization	String	是	header	用于验证请求合法性，格式为： bce-auth-v2/{accessKeyId}/{date}/{region}/{service}/{signedHeaders}/{signature} 1. accessKeyId： 是Access Key ID，请参看 获取AK/SK 来获取 2. date：签名的UTC日期，格式为yyyymmdd，例如：20150427。 3. region：所请求服务资源所在的区域，小写格式。例如：bj。 4. service：所请求的服务名称，小写格式。例如：bos。 5. signedHeaders：签名算法中涉及到的HTTP头域列表。HTTP头域名字一律要求小写且头域名字之间用分号（;）分隔，如host;range;x-bce-date。列表按照字典序排列。当signedHeaders为空时表示取默认值。 6. signature： 256位签名的十六进制表示，由64个小写字母组成。它由SK(Secret Access Key)和authStringPrefix哈希得到signingKey，再将canonicalRequest以signingKey为key进行哈希摘要生成 具体参考： 鉴权认证	bce-auth-v1/ffab11fa2b5a4f698d159af2c23f556e/2024-06-06T03:00:20Z/1800/host/3fbe830c1e67389523e2ff09f47ec59c4ed723dcea22018569a3aec968cdfbe6
content-Type	String	是	header	指定数据传输的格式（MIME类型）。当前支持： 1. application/json; charset=utf-8 2. multipart/form-data（仅部分接口支持） 3. text/event-stream （仅部分接口支持）	application/json; charset=utf-8
x-bce-date	String	是	header	当服务器收到请求时，需要比较时间戳与当前服务器时间。当两者相差超过30分钟时，返回RequestExpired错误。这样可以在一定程度上防止重放。 表示日期的字符串，符合API规范。具体请参考 日期与时间规范	2015-04-27T08:23:49Z
host	String	是	header	接口域名。	bcc.bj.baidubce.com
x-bce-content-sha256	String	否		表示内容部分的SHA256签名的十六进制字符串。这里内容指HTTP Request Payload Body。即Content部分在被HTTP encode之前的原始数据。	

错误返回

当用户访问 API 出现错误时，会返回给用户相应的错误码和错误信息，便于用户定位问题，并做出适当的处理。请求发生错误时通过 Response Body 返回详细错误信息，遵循如下格式：

参数名	类型	说明
requestId	String	导致该错误的requestId
code	String	字符串，用于表示具体错误类型
message	String	有关该错误的详细说明

公共错误码

错误码	错误描述	HTTP 状态码	说明
AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或者值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error. Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided does not exist in our records.	403 Forbidden	Access Key ID不存在
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	400 Bad Request	Authorization头域格式错误
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body格式错误。例如不符合指定的Encoding等
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配等问题，应定义更加具体的错误码，例如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 Bad Request	JSON格式不合法
InvalidVersion	The API version specified was invalid.	404 Not Found	URI的版本号不合法
OptInRequired	A subscription for the service is required.	403 Forbidden	没有开通对应的服务
PreconditionFailed	The specified If-Match header doesn't match the ETag header.	412 Precondition Failed	详见ETag的使用
RequestExpired	Request has expired. Timestamp date is XXX.	400 Bad Request	请求超时。XXX要改成x-bce-date的值。如果请求中只有Date，则需要将Date转换为本规范指定的格式。
IdempotentParameterMismatch	The request uses the same client token as a previous, but non-identical request.	403 Forbidden	clientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your Secret Access Key and signing method. Consult the service documentation for details.	401 InvalidAuth	Authorization头域中附带的签名和服务端验证不一致
OverRateLimit	Over rate limit.	429 Too Many Requests	请求超过QPS限制，请等待一段时间后重试
InvalidParam	Invalid request param xxx=yyy	400 Bad Request	请求参数非法
Unauthorized	Unauthorized	401 Unauthorized	认证失败
RateLimit	Request limit exceeded	421 Misdirected Request	请求频率超限
ResourceNotFound	Requested resource not found	404 Not Found	请求的资源不存在

全托管资源池相关接口

资源池

创建资源池

描述 创建资源池，目前支持创建托管资源池。

请求结构

```
 POST ?action=CreateResourcePool
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-|-| |spec|CreateSpec|是|创建资源池配置| |nodes|List<NodeSet>|是|创建资源池的节点配置：一次只支持配置一种规格|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |resourcePoolId|String|创建的资源池 ID| |requestId|String|创建请求的requestId|

请求示例

```
{
  "spec": {
    "region": "bj",
    "name": "cpu_resource_pool",
    "type": "dedicatedV2",
    "network": {
      "nodes": {
        "vpcId": "vpc-f0pp0jbzip3c",
        "subnetIds": [
          "sbn-qqk36ianqkte"
        ],
        "securityGroups": [
          // {
          //     "type": "enterprise",
          //     "id": "esg-1h9e1fbj3kdd",
          //     "name": "aihc-qa-test"
          // }
        ]
      },
      "pods": {
        "subnetIds": [
          "sbn-qqk36ianqkte",
          "sbn-eqysif811df6"
        ]
      }
    },
    "bindingStorages": [
      {
        "provider": "pfs",
        "type": "pfsMT",
        "id": "mt-RsEUd2"
      }
    ],
    "bindingMonitors": [
      {
        "provider": "cprom",
        "id": "cprom-hf8sc8j91urp7"
      }
    ]
  },
  "nodes": [
    {
      "addNodeSpec": {
        "machineSpec": "aihc.g5.c2m8",
        "zoneName": "cn-bj-d",
        "resourceChargingOption": {
          "chargingType": "Prepaid",
          "purchaseTime": 1,
          "purchaseTimeUnit": "MONTH",
          "autoRenew": true,
          "autoRenewTime": 1,
          "autoRenewTimeUnit": "MONTH"
        }
        // "ehcClusterId": "ehc-1P05cvhr"
      },
      "count": 1
    }
  ]
}
```

返回示例

```
{
  "requestId": "dcd2714c-7b3b-4f0f-99a3-2a6ca88f8e29",
  "resourcePoolId": "aihc-xesditpvgcog"
}
```

查询资源池详情

描述 查询指定资源池详情。

请求结构

```
GET ?action=DescribeResourcePool&resourcePoolId=xxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|资源池 ID|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数说明|类型|说明| |-|-|-| |region|String|地域| |resourcePoolId|String|资源池 ID| |type|String|资源池类型：填写 dedicatedV2 即可| |name|String|资源池名称| |description|String|资源池描述| |exposedPublic|Boolean|是否暴露公网，默认否| |forbidDelete|Boolean|是否打开删除保护，默认是| |deschedulerEnabled|Boolean|是否打开碎片治理，默认关闭| |unifiedSchedulerEnabled|Boolean|是否打开仅使用百舸调度器，默认关闭| |network|Network|网络配置| |bindingStorages|List<ProviderInfo>|绑定的存储配置，支持延迟绑定/单个/一组| |bindingMonitors|List<ProviderInfo>|绑定监控配置，支持延迟绑定/单个/一组| |phase|String|资源池状态| |nodeNum|Number|节点数| |createdAt|String|创建时间| |updatedAt|String|更新时间|

返回示例

```
{
  "createdBy": "xiongyuzhu",
  "resourcePoolId": "cce-fsjia104",
  "type": "common",
  "name": "xyz-test",
  "region": "bj",
  "configuration": {
    "exposedPublic": true,
    "forbidDelete": false,
    "deschedulerEnabled": false,
    "unifiedSchedulerEnabled": false,
    "datasetPermissionEnabled": false,
    "volumePermissionEnabled": false,
    "imageNoAuthPullEnabled": false,
    "publicNetInferenceServiceEnable": true
  },
  "createdAt": "2025-09-16T11:53:51+08:00",
  "updatedAt": "2025-09-26T17:10:54+08:00",
  "phase": "running",
  "k8sVersion": "1.30.1",
  "runtimeType": "containerd",
  "runtimeVersion": "1.6.36-cce.1",
  "associatedResources": [
    {
      "provider": "cce",
      "id": "cce-fsjia104",
      "region": "bj"
    }
  ],
  "network": {
    "mode": "vpc-eni",
    "master": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "subnetIds": [
        "sbn-y900qb0vpxc8"
      ]
    },
    "nodes": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "securityGroups": [
        {
          "type": "normal",
          "id": "g-6a72yp17dbqx",
          "name": "cce-fsjia104-worker-slh3hik7"
        }
      ]
    },
    "pods": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "subnetIds": [
        "sbn-y900qb0vpxc8"
      ],
      "securityGroups": [
        {
          "type": "normal",
          "id": "g-j983mi8ax2g8",
          "name": "cce-fsjia104-eni-7qcqrk64"
        }
      ]
    },
    "clusterIPCidr": "172.16.0.0/16",
    "loadBalanceService": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "subnetIds": [
        "sbn-y900qb0vpxc8"
      ]
    },
    "maxPodsPerNode": 256
  },
  "nodeNum": 4
}
```


查询资源池列表

描述 查询资源池列表。

请求结构

```
 GET ?action=DescribeResourcePools
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolType|String|是|Query 参数|资源池类型：填写 dedicatedV2 即可| |keywordType|String|否|Query 参数|资源池模糊查询字段，可选 [resourcePoolName, resourcePoolId,] ，默认值为 resourcePoolName| |keyword|String|否|Query 参数|查询关键词，默认值为空字符串| |orderBy|String|否|Query 参数|资源池查询排序字段，可选 [resourcePoolName, resourcePoolId, createdAt] ，默认值为 resourcePoolName| |order|String|否|Query 参数|排序方式，可选 [ASC, DESC], ASC 为升序，DESC 为降序，默认值为 ASC| |pageNumber|Number|否|Query 参数|页码，默认值

为1| |pageSize|Number|否|Query 参数|单页结果数，默认值为10|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |keywordType|String|资源池模糊查询字段| |keyword|String|查询关键词，默认值为空字符串| |orderBy|String|资源池查询排序字段，可选 [resourcePoolName, resourcePoolId, createdAt]，默认值为 resourcePoolName| |order|String|排序方式，可选 [ASC, DESC], ASC 为升序，DESC 为降序，默认值为 ASC| |pageNumber|Number|页码，默认值为1| |pageSize|Number|单页结果数，默认值为10| |totalCount|Number|资源池列表总数| |resourcePools|List<ResourcePoolSpec>|资源池摘要列表|

返回示例


```
{
  "keywordType": "resourcePoolName",
  "orderBy": "createdAt",
  "order": "DESC",
  "pageSize": 1,
  "pageNumber": 1,
  "totalCount": 4,
  "resourcePools": [
    {
      "createdBy": "xiongyuzhu",
      "resourcePoolId": "cce-fsjia104",
      "type": "common",
      "name": "xyz-test",
      "region": "bj",
      "configuration": {
        "exposedPublic": false,
        "forbidDelete": false,
        "deschedulerEnabled": false,
        "unifiedSchedulerEnabled": false,
        "datasetPermissionEnabled": false,
        "volumePermissionEnabled": false,
        "imageNoAuthPullEnabled": false,
        "publicNetInferenceServiceEnable": true
      },
      "createdAt": "2025-09-16T11:53:51+08:00",
      "updatedAt": "2025-09-26T17:10:54+08:00",
      "phase": "running",
      "k8sVersion": "1.30.1",
      "runtimeType": "containerd",
      "runtimeVersion": "1.6.36-cce.1",
      "associatedResources": [
        {
          "provider": "cce",
          "id": "cce-fsjia104",
          "region": "bj"
        }
      ],
      "network": {
        "mode": "vpc-eni",
        "master": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8",
          "subnetIds": [
            "sbn-y900qb0vpxc8"
          ]
        },
        "nodes": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8"
        },
        "pods": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8",
          "subnetIds": [
            "sbn-y900qb0vpxc8"
          ]
        },
        "clusterIPCidr": "172.16.0.0/16",
        "loadBalanceService": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8",
          "subnetIds": [
            "sbn-y900qb0vpxc8"
          ]
        },
        "maxPodsPerNode": 256
      },
      "nodeNum": 4
    }
  ]
}
```


删除资源池

描述 删除资源池，目前支持删除托管资源池。 需要先清除资源池中的队列，释放节点后才能删除资源池。

请求结构

```
 POST ?action=DeleteResourcePool&resourcePoolId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-|-| |resourcePoolId|String|是|资源池 ID|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |resourcePoolId|String|删除的资源池 ID| |requestId|String|删除请求的requestId|

返回示例

```
{
  "requestId": "66f82a87-7c0a-446d-a42b-a8cbf7302525",
  "resourcePoolId": "aihc-dk35w6cni277"
}
```

队列

创建队列

描述 创建队列，目前支持创建托管资源队列。

请求结构

```
 POST ?action=CreateQueue
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-|-| |queueName|String|是|队列名称：首尾只能是小写字母或数字，支持数字、小写字母和-，最长 63 个字符| |description|String|否|队列描述信息：最长 300 个字符| |resourcePoolId|String|是|队列所属资源池ID| |opened|Boolean|否|队列是否打开：默认为 false| |queueingStrategy|String|是|队列调度策略：支持 NativeBestEffort、BestEffortFIFO、StrictFIFO| |requeueTimeout|Number|否|超时重排阈值：1、NativeBestEffort、StrictFIFO 队列策略不支持超时重排，2、超时重排阈值，单位分钟。空或 0 代表关闭超时重排。取值范围 5～6000| |bindingNodes|List<BindingNodeInfo>|否|节点绑定信息：不指定时创建没有节点的空队列|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |queueName|String|创建的队列名称| |queueId|String|创建的队列ID|

请求示例

```
{
  "queueName": "openapi-create-01",
  "queueingStrategy": "BestEffortFIFO",
  "description": "openapi create test",
  "requeueTimeout": 5,
  "bindingNodes": [
    {
      "machineSpec": "aihc.gn5.c16m64.1a10",
      "count": 1,
      "zoneName": "cn-bj-d"
    }
  ],
  "opened": true,
  "resourcePoolId": "aihc-kis89e33o9hu"
}
```

返回示例

```
{
  "queueId": "aihcq-8yoowbfhvmf3",
  "queueName": "openapi-create-01"
}
```

查询队列详情

描述 获取一个队列的详细信息.

请求结构

```
 GET ?action=DescribeQueue&queueId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |queueId|String|是|Query 参数|标识队列的唯一标识符|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |queue|QueueItem|队列详细信息。注意：由于历史原因，返回了两层队列关系，实际只有children[0]中的队列id是对用户有意义的，使用时，队列id请取children[0].queueId |

调用示例

```
GET http://aihc.bj.baidubce.com?action=DescribeQueue&queueId=aihcq-348tjsqlhoxw
```

返回示例

```
{
  "queueId": "aihcq-h0hvmrziswv", // 注：此 id 对用户透明，请不要使用该 id，使用 children[0] 中的 id
  "queueName": "node1",
  "queueType": "Physical",
```

```
  "resourcePoolId": "aihc-ng4vnzx7fwkb",
  "createdAt": "2026-10-14T17:51:57+08:00",
  "updatedAt": "2026-10-15T17:25:10+08:00",
  "parentQueue": "63a9f0ea7bb98050796b649e85481845",
  "children": [
    {
      "queueId": "aihcq-348tjsqlhoxw", // 使用该 id，可以和控制台对应
      "queueName": "nodel",
      "queueType": "Elastic",
      "resourcePoolId": "aihc-ng4vnzx7fwkb",
      "createdAt": "2026-10-14T17:51:57+08:00",
      "updatedAt": "2026-10-15T17:25:10+08:00",
      "parentQueue": "aihcq-h0hvmrziswv3v",
      "opened": true,
      "reclaimable": false,
      "disableOversell": false,
      "queueingStrategy": "NativeBestEffort",
      "requeueTimeout": 0,
      "autoDeschedulerPendingTime": 0,
      "capability": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0",
            "acceleratorType": "NVIDIA GeForce RTX 4090",
            "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
          }
        ]
      },
      "deserved": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0",
            "acceleratorType": "NVIDIA GeForce RTX 4090",
            "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
          }
        ]
      },
      "guarantee": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0",
            "acceleratorType": "NVIDIA GeForce RTX 4090",
            "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
          }
        ]
      },
      "allocated": {
        "milliCPUcores": "0",
        "memoryGi": "0"
      },
      "workloadAvailable": {},
      "remaining": {},
      "maxDeservedForSubqueue": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0",
            "acceleratorType": "NVIDIA GeForce RTX 4090",
            "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
          }
        ]
      },
      "maxGuaranteeForSubqueue": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0",
            "acceleratorType": "NVIDIA GeForce RTX 4090",
            "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
          }
        ]
      },
      "rdmaCapable": false,
      "erdmaCapable": false,
      "enableMig": false,
      "unavailableNodeCount": 0,
      "unavailableAcceleratorNum": 0,
      "unhealthyNodeCount": 0,
      "readyCordonNodeCount": 0
    }
  ],
  "opened": true,
  "reclaimable": false,
  "disableOversell": false,
  "queueingStrategy": "NativeBestEffort",
  "requeueTimeout": 0,
```

```
    "autoDescheduleFailedPendingTime": 0,
    "capability": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "33653301248",
        "acceleratorCardList": [
            {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
        ]
    },
    "deserved": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
            {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
        ]
    },
    "guarantee": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
            {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
        ]
    },
    "allocated": {},
    "maxDeservedForSubqueue": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
            {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
        ]
    },
    "maxGuaranteeForSubqueue": {
        "milliCPUcores": "8000",
        "cpuCores": "8",
        "memoryGi": "32",
        "acceleratorCardList": [
            {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
        ]
    },
    "bindingNodes": [
        {
            "machineSpec": "bcc.g4.c8m32",
            "nodeNameList": [
                "10.0.2.114"
            ],
            "count": 1,
            "acceleratorType": "cpu",
            "zoneName": ""
        }
    ],
    "rdmaCapable": false,
    "erdmaCapable": false,
    "enableMig": false,
    "unavailableNodeCount": 0,
    "unavailableAcceleratorNum": 0,
    "unhealthyNodeCount": 0,
    "readyCordonNodeCount": 0
}
```


查询队列列表

描述 获取指定资源池的队列列表

- 支持指定关键字查询
- 支持分页

请求结构

```
 GET ?action=DescribeQueues&resourcePoolId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-|-| |resourcePoolId|String|是|Query 参数|标识资源池的唯一标识符| |keywordType|String|否|Query 参数|筛选关键字类型，当前仅支持queueName和queueId| |keyword|String|否|Query 参数|关键字值| |pageNumber|Number|否|Query 参数|请求分页参数，表示第几页| |pageSize|Number|否|Query 参数|单页结果数，默认值为10|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |keywordType|String|筛选关键字类型，当前仅支持queueName和queueId| |keyword|String|关键字值| |pageNumber|Number|请求分页参数，表示第几页，默认值为1| |pageSize|Number|单页结果数，默认值为10| |totalCount|Number|队列总数| |queues|List|队列详情列表。注意：由于历史原因，返回了两层队列关系，实际只有children[0]中的队列id是对用户有意义的，使用时，队列id请取children[0].queueId|

调用示例

```
GET http://aihc.bj.baidubce.com?action=DescribeQueues&resourcePoolId=aihc-ng4vnzx7fwkb
```

返回示例


```
{
  "pageSize": 10,
  "pageNumber": 1,
  "keywordType": "queueName",
  "totalCount": 5,
  "queues": [
    {
      "queueId": "aihcq-h0hvmrziswv3v", // 注：此 id 对用户透明，请不要使用该 id，使用 children[0] 中的 id
      "queueName": "nodel",
      "queueType": "Physical",
      "resourcePoolId": "aihc-ng4vnzx7fwkb",
      "createdAt": "2026-10-14T17:51:57+08:00",
      "updatedAt": "2026-10-15T17:25:10+08:00",
      "parentQueue": "63a9f0ea7bb98050796b649e85481845",
      "children": [
        {
          "queueId": "aihcq-348tjsqlhoxw", // 使用该 id，可以和控制台对应
          "queueName": "nodel",
          "queueType": "Elastic",
          "resourcePoolId": "aihc-ng4vnzx7fwkb",
          "createdAt": "2026-10-14T17:51:57+08:00",
          "updatedAt": "2026-10-15T17:25:10+08:00",
          "parentQueue": "aihcq-h0hvmrziswv3v",
          "opened": true,
          "reclaimable": false,
          "disableOversell": false,
          "queueingStrategy": "NativeBestEffort",
          "requeueTimeout": 0,
          "autoDeschedulerPendingTime": 0,
          "capability": {
            "milliCPUcores": "8000",
            "cpuCores": "8",
            "memoryGi": "32",
            "acceleratorCardList": [
              {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
              }
            ]
          },
          "deserved": {
            "milliCPUcores": "8000",
            "cpuCores": "8",
            "memoryGi": "32",
            "acceleratorCardList": [
              {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
              }
            ]
          },
          "guarantee": {
            "milliCPUcores": "8000",
            "cpuCores": "8",
            "memoryGi": "32",
            "acceleratorCardList": [
              {
                "acceleratorCount": "0",
                "acceleratorType": "NVIDIA GeForce RTX 4090",
                "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
              }
            ]
          },
          "allocated": {
            "milliCPUcores": "0",
            "memoryGi": "0"
          },
          "workloadAvailable": {},
          "remaining": {},
          "rdmaCapable": false,
          "erdmaCapable": false,
          "enableMig": false,
          "unavailableNodeCount": 0,
          "unavailableAcceleratorNum": 0,
          "unhealthyNodeCount": 0,
          "readyCordonNodeCount": 0
        }
      ],
      "opened": true,
      "reclaimable": false,
      "disableOversell": false,
```

```
        "queueingStrategy": "NativeBestEffort",
        "requeueTimeout": 0,
        "autoDeschedulerPendingTime": 0,
        "capability": {
          "milliCPUcores": "8000",
          "cpuCores": "8",
          "memoryGi": "32",
          "acceleratorCardList": [
            {
              "acceleratorCount": "0",
              "acceleratorType": "NVIDIA GeForce RTX 4090",
              "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
          ]
        },
        "deserved": {
          "milliCPUcores": "8000",
          "cpuCores": "8",
          "memoryGi": "32",
          "acceleratorCardList": [
            {
              "acceleratorCount": "0",
              "acceleratorType": "NVIDIA GeForce RTX 4090",
              "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
          ]
        },
        "guarantee": {
          "milliCPUcores": "8000",
          "cpuCores": "8",
          "memoryGi": "32",
          "acceleratorCardList": [
            {
              "acceleratorCount": "0",
              "acceleratorType": "NVIDIA GeForce RTX 4090",
              "acceleratorDescription": "baidu.com/rtx_4090_cgpu"
            }
          ]
        },
        "allocated": {
          "milliCPUcores": "0",
          "cpuCores": "0",
          "memoryGi": "0"
        },
        "workloadAvailable": {
          "milliCPUcores": "5975",
          "cpuCores": "5",
          "memoryGi": "24"
        },
        "remaining": {
          "milliCPUcores": "5975",
          "cpuCores": "5",
          "memoryGi": "24"
        },
        "extenderResources": [
          {
            "resourceType": "cds",
            "capacity": 5,
            "allocated": 0,
            "idle": 5,
            "allocationRate": 0,
            "utilizationRate": 0
          }
        ],
        "rdmaCapable": false,
        "erdmaCapable": false,
        "enableMig": false,
        "unavailableNodeCount": 0,
        "unavailableAcceleratorNum": 0,
        "unhealthyNodeCount": 0,
        "readyCordonNodeCount": 0
      },
      {
        ...
      }
    ]
}
```


删除队列

描述 删除队列，目前支持删除托管资源队列。

请求结构

```
 POST ?action=DeleteQueue&queueId=xxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-| |queueId|String|是|队列ID|

返回头域 除公共头域外，无其他特殊头域。

返回参数 无

返回示例 无

获取队列内节点列表信息

接口描述

查询指定队列下的节点列表，返回节点的详细信息包括基本信息、资源统计等。支持按实例ID筛选。

请求语法

```
 GET ?action=ListQueueNodes&queueId=xxxxx&instanceIds=xxxxx
Host: aihc.bj.baidubce.com
Authorization: authorization string
Content-Type: application/json
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	说明
queueId	String	是	队列ID
instanceIds	String	否	实例ID列表，多个ID用逗号分隔，用于筛选指定实例

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	说明
count	Integer	总数量
result	List<NodeDetail>	节点详情列表

NodeDetail 节点详情

参数名称	类型	说明
createdAt	String	创建时间
updatedAt	String	更新时间
expireTime	String	过期时间
instanceId	String	实例ID
instanceName	String	实例名称
flavorName	String	规格名称
chargeType	String	计费方式，可选值：prepay（预付费）、postpay（后付费）
resourcePoolId	String	资源池ID
resourceQueueId	String	资源队列ID
regionId	String	地域ID
instancePhase	String	实例阶段，可选值：creating、running、deleting 等
isRdmaCard	Integer	是否RDMA卡，0：否，1：是
eriNum	Integer	ERI数量
enableMig	Boolean	是否启用MIG
gpuDescriptor	String	GPU描述
gpuCardType	String	GPU卡类型
nodeStatus	String	节点状态，可选值：creating、ready、ready_cordon、not_ready、deleting 等
k8sNodeName	String	K8s节点名称
ehcClusterId	String	弹性高性能计算集群ID
zoneName	String	可用区
scaleDownDisabled	Boolean	是否禁止缩容
cpuCoresCapacity	Integer	CPU核数容量
cpuCoresAllocated	Integer	CPU核数已分配
cpuCoresRemaining	Integer	CPU核数剩余
cpuCoresWorkloadAvailable	Integer	CPU核数工作负载可用
cpuCoresWorkloadAllocated	Integer	CPU核数工作负载已分配
memoryGiCapacity	Integer	内存容量(GiB)
memoryGiAllocated	Integer	内存已分配(GiB)
memoryGiRemaining	Integer	内存剩余(GiB)
memoryGiWorkloadAvailable	Integer	内存工作负载可用(GiB)
memoryGiWorkloadAllocated	Integer	内存工作负载已分配(GiB)
gpuNumCapacity	Integer	GPU卡数容量
gpuNumRemaining	Float	GPU卡数剩余
gpuNumAllocated	Float	GPU卡数已分配
cpuCoresBestEffortAllocated	Integer	闲时任务CPU核数已分配
memoryGiBestEffortAllocated	Integer	闲时任务内存已分配(GiB)
gpuBestEffortAllocated	Float	闲时任务GPU已分配
taskPodList	List<TaskPod>	任务Pod列表
jobList	List<JobInfo>	任务列表
extenderResources	List<ResourceStatistic>	扩展资源统计
migResource	MigResource	MIG资源信息

TaskPod 任务Pod

参数名称	类型	说明
podName	String	Pod名称
namespace	String	命名空间
jobUid	String	Job UID

返回示例


```
{
 "count": 2,
 "result": [
  {
   "createdAt": "2024-01-15T10:30:00Z",
   "updatedAt": "2024-01-15T10:30:00Z",
   "expireTime": "2025-01-15T10:30:00Z",
   "instanceId": "ins-1",
   "instanceName": "gpu-node-01",
   "flavorName": "bcc.gr1.c8m64.2p",
   "chargeType": "prepay",
   "resourcePoolId": "pool-xyz",
   "resourceQueueId": "queue-123",
   "regionId": "bj",
   "instancePhase": "running",
   "isRdmaCard": 1,
   "eriNum": 0,
   "enableMig": false,
   "gpuDescriptor": "A100",
   "gpuCardType": "A100-80G",
   "nodeStatus": "ready",
   "k8sNodeName": "192.168.1.10",
```

```
      "k8sNodeName": "192.168.1.10",
      "ehcClusterId": "cluster-001",
      "zoneName": "cn-bj-d",
      "scaleDownDisabled": false,
      "cpuCoresCapacity": 64,
      "cpuCoresAllocated": 32,
      "cpuCoresRemaining": 32,
      "cpuCoresWorkloadAvailable": 60,
      "cpuCoresWorkloadAllocated": 4,
      "memoryGiCapacity": 512,
      "memoryGiAllocated": 256,
      "memoryGiRemaining": 256,
      "memoryGiWorkloadAvailable": 500,
      "memoryGiWorkloadAllocated": 12,
      "gpuNumCapacity": 8,
      "gpuNumRemaining": 4,
      "gpuNumAllocated": 4,
      "cpuCoresBestEffortAllocated": 0,
      "memoryGiBestEffortAllocated": 0,
      "gpuBestEffortAllocated": 0,
      "taskPodList": [],
      "extenderResources": [
        {
          "resourceType": "cds",
          "capacity": 5,
          "allocated": 0,
          "idle": 5,
          "allocationRate": 0,
          "utilizationRate": 0
        },
        {
          "resourceType": "pfsL2",
          "capacity": 1,
          "allocated": 0,
          "idle": 1,
          "allocationRate": 0,
          "utilizationRate": 0
        }
      ]
    },
    {
      "createdAt": "2024-01-16T08:20:00Z",
      "updatedAt": "2024-01-16T08:20:00Z",
      "expireTime": "2025-01-16T08:20:00Z",
      "instanceId": "ins-2",
      "instanceName": "gpu-node-02",
      "flavorName": "bcc.gr1.c8m64.2p",
      "chargeType": "postpay",
      "resourcePoolId": "pool-xyz",
      "resourceQueueId": "queue-123",
      "regionId": "bj",
      "instancePhase": "running",
      "isRdmaCard": 0,
      "eriNum": 0,
      "enableMig": false,
      "gpuDescriptor": "A100",
      "gpuCardType": "A100-80G",
      "nodeStatus": "ready",
      "k8sNodeName": "192.168.1.11",
      "ehcClusterId": "cluster-001",
      "zoneName": "cn-bj-d",
      "scaleDownDisabled": false,
      "cpuCoresCapacity": 64,
      "cpuCoresAllocated": 16,
      "cpuCoresRemaining": 48,
      "cpuCoresWorkloadAvailable": 60,
      "cpuCoresWorkloadAllocated": 4,
      "memoryGiCapacity": 512,
      "memoryGiAllocated": 128,
      "memoryGiRemaining": 384,
      "memoryGiWorkloadAvailable": 500,
      "memoryGiWorkloadAllocated": 12,
      "gpuNumCapacity": 8,
      "gpuNumRemaining": 6,
      "gpuNumAllocated": 2,
      "cpuCoresBestEffortAllocated": 0,
      "memoryGiBestEffortAllocated": 0,
      "gpuBestEffortAllocated": 0,
      "taskPodList": [],
      "extenderResources": [
        {
          "resourceType": "cds",
          "capacity": 5,
          "allocated": 0,
          "idle": 5,
          "allocationRate": 0,
          "utilizationRate": 0
        },
        {
          "resourceType": "pfsL2",
          "capacity": 1,
          "allocated": 0,
          "idle": 1,
          "allocationRate": 0,
          "utilizationRate": 0
        }
      ]
    }
  ]
}
```

节点

创建节点

描述 创建节点，目前支持创建托管资源节点。

请求结构

```
 POST ?action=CreateNode
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-| |nodeSet|List< NodeSet>|是|创建的节点规格，只支持创建一种规格|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |result|List|创建的节点列表ID| |requestId|String|请求ID| |code|Number|返回码| |message|String|返回信息|

请求示例

```
{
 "nodeSet": [
  {
    "addNodeSpec": {
      "zoneName": "cn-bj-d",
      "machineSpec": "aihc.g5.c2m8",
      "resourceChargingOption": {
        "chargingType": "Prepaid", // Postpaid
        "purchaseTime": 1,
        "purchaseTimeUnit": "MONTH",
        "autoRenew": true,
        "autoRenewTime": 1,
        "autoRenewTimeUnit": "MONTH"
      },
      "networkInfo": {
        "subnetIds": [
          "sbn-0j2yiz8syt8p"
        ],
        "subnetCidr": "10.5.0.0/16"
      }
      // "ehcClusterId": ""
    },
    "count": 2
  }
 ]
}
```

返回示例

```
 {
   "code": 200,
   "message": "success",
   "requestId": "4ec3201b-bbb1-473d-8dd1-b3c264991d7c",
   "result": [
      "aihcn-zvwobkc8yume",
      "aihcn-ajh92tu5zqlj"
   ]
}
```

查询节点列表

描述 查询队列节点列表，目前支持查询托管资源节点。

请求结构

```
 GET ?action=DescribeNodes&resourcePoolId=xxx&pageSize=10&pageNo=2
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-| |resourcePoolId|String|是|资源池名称| |instanceId|String|否|实例ID| |orderBy|String|否|排序字段：支持CreatedAt| |order|String|否|排序方式，升序：asc，降序：desc| |pageSize|Number|否|分页大小：默认为 10| |pageNo|Number|否|页号：默认为 1|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |nodes|List< Node>|节点列表| |requestId|String|请求ID|

返回示例

```
{
  "requestId": "4af0dc96-779a-4cdc-834d-ff72e79be2e1",
  "nodes": [
    {
      "nodeSpec": {
        "region": "bj",
        "zone": "cn-bj-d",
        "flavorName": "aihc.g5.c4m16",
        "resourcePoolId": "aihc-0j5gqcnciwle",
        "accountId": "0e068cb945c74bc790b83d05e6d8b7e9",
        "instanceName": "instance-zp5h4af9",
        "nodeName": "10.5.26.184",
        "instanceType": "bcc",
        "instanceSpec": "bcc.g4.c4m16",
        "instanceId": "aihcn-ygk8qg7v9uv0",
        "resourceChargingOption": {
          "chargingType": "Postpaid"
        }
      },
      "nodeStatus": {
        "statusPhase": "ready",
        "createdAt": "2025-12-12T04:37:08+08:00",
        "billResourceId": "68775a3c-2f76-4deb-8dc1-57f4de75c3e4",
        "nodeIntervention": {
          "cordon": false
        },
        "resourceQueueId": "aihcq-64gs11k0kc2z",
        "nodeResource": {
          "capacity": {
            "milliCPUcores": 4000,
            "cpuCores": 4,
            "memoryBytes": 17179869184,
            "memoryGi": 16,
            "gpuNum": 0
          }
        }
      }
    }
  ]
}
```

删除节点

描述 删除节点，目前支持删除托管资源节点。

请求结构

```
 POST ?action=DeleteNode&resourcePoolId=xxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-|-| |resourcePoolId|String|否|资源池ID| |instanceIds|List< String>|是|实例ID列表|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |code|Number|返回状态码| |message|String|返回信息| |requestId|String|请求ID|

请求示例

```
{
  "instanceIds": [
    "aihcn-9idn0r5m1puz"
  ]
}
```

返回示例

```
{
  "code": 200,
  "message": "success",
  "requestId": "3c4f5a3e-1aaa-4b1a-8dda-6f913d8246c7",
}
```

自运维资源池相关接口(随产品下线中)

资源池

查询自运维资源池详情

描述 查询指定资源池详情。

请求结构

```
 GET ?action=DescribeResourcePool&resourcePoolId=xxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|资源池 ID|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数说明|类型|说明| |-|-|-| |region|String|地域| |resourcePoolId|String|资源池 ID| |type|String|资源池类型：填写 common 即可| |name|String|资源池名称| |description|String|资源池描述| |exposedPublic|Boolean|是否暴露公网，默认否| |forbidDelete|Boolean|是否打开删除保护，默认是| |deschedulerEnabled|Boolean|是否打开碎片治理，默认关闭| |unifiedSchedulerEnabled|Boolean|是否打开仅使用百舸调度器，默认关闭| |network|Network|网络配置| |bindingStorages|List<ProviderInfo>|绑定的存储配置，支持延迟绑定/单个/一组| |bindingMonitors|List<ProviderInfo>|绑定监控配置，支持延迟绑定/单个/一组| |phase|String|资源池状态| |nodeNum|Number|节点数| |createdAt|String|创建时间| |updatedAt|String|更新时间|

返回示例


```
{
  "createdBy": "xiongyuzhu",
  "resourcePoolId": "cce-fsjia104",
  "type": "common",
  "name": "xyz-test",
  "region": "bj",
  "configuration": {
    "exposedPublic": true,
    "forbidDelete": false,
    "deschedulerEnabled": false,
    "unifiedSchedulerEnabled": false,
    "datasetPermissionEnabled": false,
    "volumePermissionEnabled": false,
    "imageNoAuthPullEnabled": false,
    "publicNetInferenceServiceEnable": true
  },
  "createdAt": "2025-09-16T11:53:51+08:00",
  "updatedAt": "2025-09-26T17:10:54+08:00",
  "phase": "running",
  "k8sVersion": "1.30.1",
  "runtimeType": "containerd",
  "runtimeVersion": "1.6.36-cce.1",
  "associatedResources": [
    {
      "provider": "cce",
      "id": "cce-fsjia104",
      "region": "bj"
    }
  ],
  "network": {
    "mode": "vpc-eni",
    "master": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "subnetIds": [
        "sbn-y900qb0vpxc8"
      ]
    },
    "nodes": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "securityGroups": [
        {
          "type": "normal",
          "id": "g-6a72yp17dbqx",
          "name": "cce-fsjia104-worker-slh3hik7"
        }
      ]
    },
    "pods": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "subnetIds": [
        "sbn-y900qb0vpxc8"
      ],
      "securityGroups": [
        {
          "type": "normal",
          "id": "g-j983mi8ax2g8",
          "name": "cce-fsjia104-eni-7qcqrk64"
        }
      ]
    },
    "clusterIPCidr": "172.16.0.0/16",
    "loadBalanceService": {
      "region": "bj",
      "vpcId": "vpc-d2mc0h3xuvuf",
      "vpcCidr": "10.0.0.0/8",
      "subnetIds": [
        "sbn-y900qb0vpxc8"
      ]
    },
    "maxPodsPerNode": 256
  },
  "nodeNum": 4
}
```


查询自运维资源池列表

描述 查询资源池列表。

请求结构

```
 GET ?action=DescribeResourcePools
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolType|String|是|Query 参数|资源池类型：填写 common 即可| |keywordType|String|否|Query 参数|资源池模糊查询字段，可选 [resourcePoolName, resourcePoolId,] ，默认值为 resourcePoolName| |keyword|String|否|Query 参数|查询关键词，默认值为空字符串| |orderBy|String|否|Query 参数|资源池查询排序字段，可选 [resourcePoolName, resourcePoolId, createdAt] ，默认值为 resourcePoolName| |order|String|否|Query 参数|排序方式，可选 [ASC, DESC], ASC 为升序，DESC 为降序，默认值为 ASC| |pageNumber|Number|否|Query 参数|页码，默认值为1| |pageSize|Number|否|Query 参数|单页结果数，默认值为10|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |keywordType|String|资源池模糊查询字段| |keyword|String|查询关键词，默认值为空字符串| |orderBy|String|资源池查询排序字段，可选 [resourcePoolName, resourcePoolId, createdAt] ，默认值为 resourcePoolName| |order|String|排序方式，可选 [ASC, DESC], ASC 为升序，DESC 为降序，默认值为 ASC| |pageNumber|Number|页码，默认值为1| |pageSize|Number|单页结果数，默认值为10| |totalCount|Number|资源池列表总数| |resourcePools|List<ResourcePoolSpec>|资源池摘要列表|

返回示例


```
{
  "keywordType": "resourcePoolName",
  "orderBy": "createdAt",
  "order": "DESC",
  "pageSize": 1,
  "pageNumber": 1,
  "totalCount": 4,
  "resourcePools": [
    {
      "createdBy": "xiongyuzhu",
      "resourcePoolId": "cce-fsjia104",
      "type": "common",
      "name": "xyz-test",
      "region": "bj",
      "configuration": {
        "exposedPublic": false,
        "forbidDelete": false,
        "deschedulerEnabled": false,
        "unifiedSchedulerEnabled": false,
        "datasetPermissionEnabled": false,
        "volumePermissionEnabled": false,
        "imageNoAuthPullEnabled": false,
        "publicNetInferenceServiceEnable": true
      },
      "createdAt": "2025-09-16T11:53:51+08:00",
      "updatedAt": "2025-09-26T17:10:54+08:00",
      "phase": "running",
      "k8sVersion": "1.30.1",
      "runtimeType": "containerd",
      "runtimeVersion": "1.6.36-cce.1",
      "associatedResources": [
        {
          "provider": "cce",
          "id": "cce-fsjia104",
          "region": "bj"
        }
      ],
      "network": {
        "mode": "vpc-eni",
        "master": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8",
          "subnetIds": [
            "sbn-y900qb0vpxc8"
          ]
        },
        "nodes": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8"
        },
        "pods": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8",
          "subnetIds": [
            "sbn-y900qb0vpxc8"
          ]
        },
        "clusterIPCidr": "172.16.0.0/16",
        "loadBalanceService": {
          "region": "bj",
          "vpcId": "vpc-d2mc0h3xuvuf",
          "vpcCidr": "10.0.0.0/8",
          "subnetIds": [
            "sbn-y900qb0vpxc8"
          ]
        },
        "maxPodsPerNode": 256
      },
      "nodeNum": 4
    }
  ]
}
```

队列

查询自运维队列详情

描述 获取一个队列的详细信息.

请求结构

```
 GET ?action=DescribeQueue&queueId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |queueId|String|是|Query 参数|标识队列的唯一标识符|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |queue|QueueItem|队列详细信息|

返回示例


```
{
  "queueId": "queue-h4rzqodsqgfv",
  "queueName": "feold-backnew-gpu-phy-l1",
  "queueType": "Physical",
  "resourcePoolId": "cce-so2uum7z",
  "createdAt": "2025-09-09T15:46:18+08:00",
  "updatedAt": "2025-09-09T19:42:57+08:00",
  "parentQueue": "63a9f0ea7bb98050796b649e85481845",
  "children": [
    {
      "queueId": "queue-9zyv0dcqw5jh",
      "queueName": "old-gpu-l2q-3",
      "queueType": "Regular",
      "resourcePoolId": "cce-so2uum7z",
      "createdAt": "2025-09-09T15:49:39+08:00",
      "updatedAt": "2025-09-09T19:42:57+08:00",
      "parentQueue": "feold-backnew-gpu-phy-l1",
      "opened": true,
      "reclaimable": false,
      "disableOversell": false,
      "queueingStrategy": "BestEffortFIFO",
      "enableVGPU": true,
      "capability": {
        "acceleratorCardList": [
          {
            "acceleratorCount": "0.5",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "deserved": {
        "acceleratorCardList": [
          {
            "acceleratorCount": "0.5",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "allocated": {
        "milliCPUcores": "0",
        "cpuCores": "0",
        "memoryGi": "0",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0.5",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "runningJobs": 1
    },
    {
      "queueId": "queue-rp92t5b8izfy",
      "queueName": "old-gpu-l2q-2",
      "queueType": "Regular",
      "resourcePoolId": "cce-so2uum7z",
      "createdAt": "2025-09-09T15:48:14+08:00",
      "updatedAt": "2025-09-09T19:42:57+08:00",
      "parentQueue": "feold-backnew-gpu-phy-l1",
      "opened": true,
      "reclaimable": false,
      "disableOversell": false,
      "queueingStrategy": "BestEffortFIFO",
      "enableVGPU": true,
      "capability": {
        "acceleratorCardList": [
          {
            "acceleratorCount": "2.5",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
```

```
        },
        "deserved": {
          "acceleratorCardList": [
            {
              "acceleratorCount": "2.5",
              "acceleratorType": "NVIDIA L20",
              "acceleratorDescription": "baidu.com/l20_cgpu"
            }
          ]
        },
        "allocated": {
          "milliCPUcores": "0",
          "cpuCores": "0",
          "memoryGi": "0",
          "acceleratorCardList": [
            {
              "acceleratorCount": "2.5",
              "acceleratorType": "NVIDIA L20",
              "acceleratorDescription": "baidu.com/l20_cgpu"
            }
          ]
        },
        "runningJobs": 2
      },
      {
        "queueId": "queue-t1n9trr0nd3n",
        "queueName": "old-gpu-l2q",
        "queueType": "Regular",
        "resourcePoolId": "cce-so2uum7z",
        "createdAt": "2025-09-09T15:47:28+08:00",
        "updatedAt": "2025-09-09T19:42:57+08:00",
        "parentQueue": "feold-backnew-gpu-phy-l1",
        "opened": true,
        "reclaimable": false,
        "disableOversell": false,
        "queueingStrategy": "BestEffortFIFO",
        "enableVGPU": true,
        "capability": {
          "acceleratorCardList": [
            {
              "acceleratorCount": "1",
              "acceleratorType": "NVIDIA L20",
              "acceleratorDescription": "baidu.com/l20_cgpu"
            }
          ]
        },
        "deserved": {
          "acceleratorCardList": [
            {
              "acceleratorCount": "1",
              "acceleratorType": "NVIDIA L20",
              "acceleratorDescription": "baidu.com/l20_cgpu"
            }
          ]
        },
        "allocated": {
          "milliCPUcores": "0",
          "cpuCores": "0",
          "memoryGi": "0",
          "acceleratorCardList": [
            {
              "acceleratorCount": "0.8",
              "acceleratorType": "NVIDIA L20",
              "acceleratorDescription": "baidu.com/l20_cgpu"
            }
          ]
        },
        "runningJobs": 1
      }
    ],
    "opened": true,
    "reclaimable": false,
    "disableOversell": false,
    "queueingStrategy": "StrictFIFO",
    "enableVGPU": true,
    "capability": {
      "milliCPUcores": "292000",
      "cpuCores": "292",
      "memoryGi": "3608148426752",
      "acceleratorCardList": [
        {
          "acceleratorCount": "4",
          "acceleratorType": "NVIDIA L20",
          "acceleratorDescription": "baidu.com/l20_cgpu"
        }
      ]
    },
    "deserved": {
      "milliCPUcores": "292000",
      "cpuCores": "292",
      "memoryGi": "3361",
      "acceleratorCardList": [
        {
          "acceleratorCount": "4",
          "acceleratorType": "NVIDIA L20",
          "acceleratorDescription": "baidu.com/l20_cgpu"
        }
      ]
    },
```

```
    "guarantee": {
      "milliCPUcores": "292000",
      "cpuCores": "292",
      "memoryGi": "3361",
      "acceleratorCardList": [
        {
          "acceleratorCount": "4",
          "acceleratorType": "NVIDIA L20",
          "acceleratorDescription": "baidu.com/l20_cgpu"
        }
      ]
    },
    "allocated": {
      "milliCPUcores": "0",
      "cpuCores": "0",
      "memoryGi": "0",
      "acceleratorCardList": [
        {
          "acceleratorCount": "3.8",
          "acceleratorType": "NVIDIA L20",
          "acceleratorDescription": "baidu.com/l20_cgpu"
        }
      ]
    },
    "maxDeservedForSubqueue": {
      "milliCPUcores": "292000000",
      "cpuCores": "292000",
      "memoryGi": "3361",
      "acceleratorCardList": [
        {
          "acceleratorCount": "0",
          "acceleratorType": "NVIDIA L20",
          "acceleratorDescription": "baidu.com/l20_cgpu"
        }
      ]
    },
    "maxGuaranteeForSubqueue": {
      "milliCPUcores": "292000000",
      "cpuCores": "292000",
      "memoryGi": "3361",
      "acceleratorCardList": [
        {
          "acceleratorCount": "4",
          "acceleratorType": "NVIDIA L20",
          "acceleratorDescription": "baidu.com/l20_cgpu"
        }
      ]
    },
    "bindingNodes": [
      {
        "machineSpec": "llama2_13B_train/7k",
        "nodeNameList": [
          "10.0.1.72"
        ],
        "count": 1,
        "acceleratorType": "NVIDIA H20Z"
      },
      {
        "machineSpec": "bcc.gn5i.c112m476.4l20",
        "nodeNameList": [
          "10.0.5.30"
        ],
        "count": 1,
        "acceleratorType": "NVIDIA L20"
      }
    ]
}
```


查询自运维队列列表

描述 获取指定资源池的队列列表

- 支持指定关键字查询
- 支持分页

请求结构

```
 GET ?action=DescribeQueues&resourcePoolId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|标识资源池的唯一标识符| |keywordType|String|否|Query 参数|筛选关键字类型，当前仅支持queueName和queueId| |keyword|String|否|Query 参数|关键字值| |pageNumber|Number|否|Query 参数|请求分页参数，表示第几页| |pageSize|Number|否|Query 参数|单页结果数，默认值为10|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |keywordType|String|筛选关键字类型，当前仅支持queueName和queueId| |keyword|String|关键字值| |pageNumber|Number|请求分页参数，表示第几页，默认值为1| |pageSize|Number|单页结果数，默认值为10| |totalCount|Number|队列总数| |queues|List|队列详情列表|

返回示例


```
{
  "pageSize": 10,
  "pageNumber": 1
```

```
  "pageNumber": 1,
  "keyword": "a",
  "keywordType": "queueName",
  "totalCount": 6,
  "queues": [
    {
      "queueId": "queue-cd1i93jyadbq",
      "queueName": "63a9f0ea7bb98050796b649e85481845",
      "queueType": "Elastic",
      "resourcePoolId": "cce-so2uum7z",
      "createdAt": "2025-08-26T17:13:19+08:00",
      "updatedAt": "2025-09-09T19:26:59+08:00",
      "parentQueue": "63a9f0ea7bb98050796b649e85481845",
      "opened": true,
      "reclaimable": true,
      "disableOversell": false,
      "queueingStrategy": "BestEffortFIFO",
      "capability": {
        "milliCPUcores": "856",
        "memoryGi": "9449599258624",
        "acceleratorCardList": [
          {
            "acceleratorCount": "8",
            "acceleratorType": "KUNLUNXIN-P800",
            "acceleratorDescription": "kunlunxin.com/xpu"
          },
          {
            "acceleratorCount": "1",
            "acceleratorType": "NVIDIA GeForce RTX 3090",
            "acceleratorDescription": "baidu.com/rtx_3090_cgpu"
          },
          {
            "acceleratorCount": "8",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          },
          {
            "acceleratorCount": "16",
            "acceleratorType": "NVIDIA H20Z",
            "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
          }
        ]
      },
      "deserved": {},
      "guarantee": {},
      "allocated": {
        "milliCPUcores": "0",
        "memoryGi": "0"
      },
      "workloadAvailable": {
        "milliCPUcores": "643590",
        "cpuCores": "643",
        "memoryGi": "5922",
        "acceleratorCardList": [
          {
            "acceleratorCount": "8",
            "acceleratorType": "KUNLUNXIN-P800",
            "acceleratorDescription": "kunlunxin.com/xpu"
          },
          {
            "acceleratorCount": "1",
            "acceleratorType": "NVIDIA GeForce RTX 3090",
            "acceleratorDescription": "baidu.com/rtx_3090_cgpu"
          },
          {
            "acceleratorCount": "8",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          },
          {
            "acceleratorCount": "8",
            "acceleratorType": "NVIDIA H20Z",
            "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
          }
        ]
      },
      "remaining": {
        "milliCPUcores": "635590",
        "cpuCores": "635",
        "memoryGi": "5898"
      }
    },
    {
      "queueId": "queue-6gjnkt3dmxiy",
      "queueName": "default",
      "queueType": "Elastic",
      "resourcePoolId": "cce-so2uum7z",
      "createdAt": "2025-08-26T17:13:19+08:00",
      "updatedAt": "2025-09-09T19:26:59+08:00",
      "parentQueue": "63a9f0ea7bb98050796b649e85481845",
      "opened": true,
      "reclaimable": true,
      "disableOversell": false,
      "queueingStrategy": "BestEffortFIFO",
      "capability": {},
      "deserved": {},
      "guarantee": {},
      "allocated": {
        "milliCPUcores": "4000",
        "cpuCores": "4",
```

```
        "memoryGi": "16",
        "acceleratorCardList": [
          {
            "acceleratorCount": "1",
            "acceleratorType": "NVIDIA H20Z",
            "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
          }
        ]
      }
    },
    {
      "queueId": "queue-h4rzqodsqgfv",
      "queueName": "feold-backnew-gpu-phy-l1",
      "queueType": "Physical",
      "resourcePoolId": "cce-so2uum7z",
      "createdAt": "2025-09-09T15:46:18+08:00",
      "updatedAt": "2025-09-09T19:26:59+08:00",
      "parentQueue": "63a9f0ea7bb98050796b649e85481845",
      "opened": true,
      "reclaimable": false,
      "disableOversell": false,
      "queueingStrategy": "StrictFIFO",
      "enableVGPU": true,
      "capability": {
        "milliCPUcores": "292000",
        "cpuCores": "292",
        "memoryGi": "3361",
        "acceleratorCardList": [
          {
            "acceleratorCount": "4",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "deserved": {
        "milliCPUcores": "292000",
        "cpuCores": "292",
        "memoryGi": "3361",
        "acceleratorCardList": [
          {
            "acceleratorCount": "4",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "guarantee": {
        "milliCPUcores": "292000",
        "cpuCores": "292",
        "memoryGi": "3361",
        "acceleratorCardList": [
          {
            "acceleratorCount": "4",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "allocated": {
        "milliCPUcores": "0",
        "cpuCores": "0",
        "memoryGi": "0",
        "acceleratorCardList": [
          {
            "acceleratorCount": "3.8",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "workloadAvailable": {
        "milliCPUcores": "108615",
        "cpuCores": "108",
        "memoryGi": "451",
        "acceleratorCardList": [
          {
            "acceleratorCount": "0",
            "acceleratorType": "NVIDIA H20Z",
            "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
          },
          {
            "acceleratorCount": "4",
            "acceleratorType": "NVIDIA L20",
            "acceleratorDescription": "baidu.com/l20_cgpu"
          }
        ]
      },
      "remaining": {
        "milliCPUcores": "108615",
        "cpuCores": "108",
        "memoryGi": "451"
      }
    },
    {
      "queueId": "queue-9dgwufhc8lwu",
      "queueName": "feold-backnew-elas",
      "queueType": "Elastic",
      "resourcePoolId": "cce-so2uum7z",
      "createdAt": "2025-09-09T11:36:58+08:00",
```

```
    "updatedAt": "2025-09-09T19:26:59+08:00",
    "parentQueue": "63a9f0ea7bb98050796b649e85481845",
    "opened": true,
    "reclaimable": true,
    "disableOversell": false,
    "queueingStrategy": "BestEffortFIFO",
    "capability": {
      "acceleratorCardList": [
        {
          "acceleratorCount": "4",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    },
    "deserved": {
      "acceleratorCardList": [
        {
          "acceleratorCount": "3",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    },
    "guarantee": {
      "acceleratorCardList": [
        {
          "acceleratorCount": "2",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    },
    "allocated": {
      "milliCPUcores": "0",
      "cpuCores": "0",
      "memoryGi": "0",
      "acceleratorCardList": [
        {
          "acceleratorCount": "4",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    }
  },
  {
    "queueId": "queue-zn55rhdirv1r",
    "queueName": "feold-backnew-nor",
    "queueType": "Regular",
    "resourcePoolId": "cce-so2uum7z",
    "createdAt": "2025-09-09T11:36:36+08:00",
    "updatedAt": "2025-09-09T19:26:59+08:00",
    "parentQueue": "63a9f0ea7bb98050796b649e85481845",
    "opened": true,
    "reclaimable": false,
    "disableOversell": false,
    "queueingStrategy": "BestEffortFIFO",
    "capability": {
      "acceleratorCardList": [
        {
          "acceleratorCount": "3",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    },
    "deserved": {
      "acceleratorCardList": [
        {
          "acceleratorCount": "3",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    },
    "guarantee": {},
    "allocated": {
      "milliCPUcores": "0",
      "cpuCores": "0",
      "memoryGi": "0",
      "acceleratorCardList": [
        {
          "acceleratorCount": "2",
          "acceleratorType": "NVIDIA H20Z",
          "acceleratorDescription": "baidu.com/h20z_141g_cgpu"
        }
      ]
    }
  },
  {
    "queueId": "queue-oji7qisf0szs",
    "queueName": "feold-backnew-phy",
    "queueType": "Physical",
    "resourcePoolId": "cce-so2uum7z",
    "createdAt": "2025-09-09T11:36:02+08:00",
    "updatedAt": "2025-09-09T19:26:59+08:00",
    "parentQueue": "63a9f0ea7bb98050796b649e85481845",
    "opened": false,
    "reclaimable": false
```

```
            "disableOversell": false,
            "queueingStrategy": "StrictFIFO",
            "capability": {
                "milliCPUcores": "208000",
                "cpuCores": "208",
                "memoryGi": "2016",
                "acceleratorCardList": [
                    {
                        "acceleratorCount": "8",
                        "acceleratorType": "KUNLUNXIN-P800",
                        "acceleratorDescription": "kunlunxin.com/xpu"
                    }
                ]
            },
            "deserved": {
                "acceleratorCardList": [
                    {
                        "acceleratorCount": "8",
                        "acceleratorType": "KUNLUNXIN-P800",
                        "acceleratorDescription": "kunlunxin.com/xpu"
                    }
                ]
            },
            "guarantee": {
                "acceleratorCardList": [
                    {
                        "acceleratorCount": "8",
                        "acceleratorType": "KUNLUNXIN-P800",
                        "acceleratorDescription": "kunlunxin.com/xpu"
                    }
                ]
            },
            "allocated": {
                "milliCPUcores": "0",
                "cpuCores": "0",
                "memoryGi": "0"
            },
            "workloadAvailable": {
                "milliCPUcores": "202040",
                "cpuCores": "202",
                "memoryGi": "1966",
                "acceleratorCardList": [
                    {
                        "acceleratorCount": "8",
                        "acceleratorType": "KUNLUNXIN-P800",
                        "acceleratorDescription": "kunlunxin.com/xpu"
                    }
                ]
            },
            "remaining": {
                "milliCPUcores": "202040",
                "cpuCores": "202",
                "memoryGi": "1966"
            }
        }
    ]
}
```


节点

查询自运维节点列表

描述 查询队列节点列表，目前支持查询托管资源节点。

请求结构

```
 GET ?action=DescribeNodes&resourcePoolId=xxx&pageSize=10&pageNo=2
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|说明| |-|-|-| |resourcePoolId|String|是|资源池名称| |instanceId|String|否|实例ID| |orderBy|String|否|排序字段：支持CreatedAt| |order|String|否|排序方式，升序：asc，降序：desc| |pageSize|Number|否|分页大小：默认为 10| |pageNo|Number|否|页号：默认为 1|

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |nodes|List<Node>|节点列表| |requestId|String|请求ID|

返回示例

```
{
  "requestId": "4af0dc96-779a-4cdc-834d-ff72e79be2e1",
  "nodes": [
    {
      "nodeSpec": {
        "region": "bj",
        "zone": "cn-bj-d",
        "flavorName": "aihc.g5.c4m16",
        "resourcePoolId": "aihc-0j5gqcnciwle",
        "accountId": "0e068cb945c74bc790b83d05e6d8b7e9",
        "instanceName": "instance-zp5h4af9",
        "nodeName": "10.5.26.184",
        "instanceType": "bcc",
        "instanceSpec": "bcc.g4.c4m16",
        "instanceId": "aihcn-ygk8qg7v9uv0",
        "resourceChargingOption": {
          "chargingType": "Postpaid"
        }
      },
      "nodeStatus": {
        "statusPhase": "ready",
        "createdAt": "2025-12-12T04:37:08+08:00",
        "billResourceId": "68775a3c-2f76-4deb-8dc1-57f4de75c3e4",
        "nodeIntervention": {
          "cordon": false
        },
        "resourceQueueId": "aihcq-64gs11k0kc2z",
        "nodeResource": {
          "capacity": {
            "milliCPUcores": 4000,
            "cpuCores": 4,
            "memoryBytes": 17179869184,
            "memoryGi": 16,
            "gpuNum": 0
          }
        }
      }
    }
  ]
}
```


分布式训练相关接口

创建训练任务

描述 创建一个训练任务到集群中运行。可以指定数据源配置、启动命令以及任务运行的每个节点的计算资源配置等信息。

请求结构

```
 POST ?action=CreateJob&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |name|String|是|Body参数|名称| |jobType|String|否|Body参数|分布式框架类型，支持PyTorchJob，TFJob，MPIJob，RayJob。默认值：PyTorchJob| |jobSpec|JobSpec /Map[string, JobSpec]|是|Body参数|训练任务配置，格式由 jobType 决定：
① 若 jobType=PyTorchJob：传递参数为单个JobSpec 对象；
② 若 jobType=TFJob：传递Map结构（Key 可选值：Chief/Worker/PS/Evaluator，Value 为对应角色的 JobSpec 配置）;③ 若 jobType=RayJob：传递Map结构（Key必须包含Head，Value 为对应角色的 JobSpec 配置）| |command|String|是|Body参数|启动命令| |labels|List<Label>|否|Body参数|训练任务标签，默认包含： 1. aijob.cce.baidubce.com/create-from-aihcp-api: "true" 2. aijob.cce.baidubce.com/ai-user-id: {userId} 3. aijob.cce.baidubce.com/ai-user-name: {userName}| |priority|String|否|Body参数|调度优先级，支持高（high）、中（normal）、低（low），默认值：normal| |datasources|List<Datasource>|否|Body参数|数据源配置，支持以下类型：
• **pfs**：并行文件存储，填 name（pfsId）和 mountPath
• **hostpath**：宿主机路径，填 sourcePath（宿主机路径）和 mountPath
• **bos**：对象存储，填 sourcePath（桶名）和 mountPath
• **cfs**：CFS 文件存储，填 sourcePath、mountPath，options 中须填 cfsInstanceId 和 cfsMountPoint
• **rapidfs**：高速并行文件存储。
• **dataset**：平台数据集，填 id（数据集 ID）和 mountPath，options 中可填 datasetVersion
• **public_dataset**：公开数据集，填 id 和 mountPath，服务端强制只读。详见数据源挂载示例| |enableBccl|Boolean|否|Body参数|是否开启BCCL自动注入，默认值为关闭。当前开启条件: 1.实例数大于等于 2 2.每个实例占整机 8 卡 3.任务开启 RDMA 4.卡型号为A800/HPAS| |faultTolerance|Boolean|否|Body参数|是否开启容错， 默认值为 关闭，目前PyTorchJob支持容错| | faultToleranceArgs | String | 否 | Body | 容错配置字符串，由若干 --flag=value 用空格拼接而成。

约束条件：
• 与 faultToleranceConfig 互斥，二者只能二选一；
• 仅支持 PyTorchJob。

支持的 flag：
• --enable-replace（bool）：开启 replace 容错（弹性 Agent 替换故障 worker）。
• --enable-hang-detection（bool）：开启 hang（卡死）检测；置为 true 后，下列三个参数方可生效。
• --hang-detection-log-timeout-minutes（int）：所有 worker 在该时长内无日志输出即判定卡死。
• --hang-detection-startup-toleration-minutes（int）：任务启动后的免检窗口分钟数，规避初始化/数据加载阶段误判。
• --hang-detection-stack-timeout-minutes（int）：日志仍在输出但进程堆栈在该时长内无变化时，判定为「假活」卡死。
• --max-num-of-unconditional-retry（int，≤ 3）：终态失败时无条件重试次数上限。
• --custom-log-patterns（string，可重复）：自定义容错日志 pattern，worker 日志命中任一即触发容错，最终以 JSON 数组写入 annotation。

示例：
"--enable-replace=true --enable-hang-detection=true --hang-detection-log-timeout-minutes=7 --hang-detection-startup-toleration-minutes=15 --hang-detection-stack-timeout-minutes=3 --max-num-of-unconditional-

retry=2 --custom-log-patterns=timeout1 --custom-log-patterns=timeout2" | |tensorboardConfig|TensorboardConfig|否|Body参数|tensorboard相关配置，RayJob暂不支持tensorboard功能| |alertConfig|AlertConfig|否|Body参数|告警相关配置，PyTorchJob支持所有告警，TFJob支持任务状态相关告警，MPIJob暂不支持告警，RayJob支持任务状态相关告警| |retentionPeriod|String|否|Body参数|终态任务（成功、失败、已停止的任务）的保留时间，参数格式参考：1m、1h、1d，分别代表1分钟、1小时、1天，RayJob暂不支持任务保留时间| |advancedSettings|AdvancedSettings|否|Body参数|RayJob 高级配置（可选），用于控制运行时环境、submitter 重试策略和 Head 节点的容错配置（可开启 BLB，实现同 VPC 内 Head 节点访问入口固定不变）| |visibleScope|Integer |否 | Body参数|训练任务可见范围，参数为1代表队列内可见，为0代表创建人可见，默认队列内可见| |portConfigs|PortConfigs|否|Body参数|自定义监控监听端口配置，当前仅支持 usage=Metrics，且仅允许配置 1 组端口。instanceId、agentId 由服务端自动查询集群绑定的监控实例并填充，无需传入（传入也会被忽略）。要求集群/队列已绑定监控实例，bhcmp 集群不支持该配置| |idleTask|Boolean|否|Body参数|是否提交闲时任务，借用其他队列的闲置资源运行，资源可能被回收，默认 false。仅全托管资源池支持，且队列需开启闲时提交、cce-aibox 版本需 ≥ 1.6.39。开启后任务优先级固定为 lowest（无需也不可再指定 priority）| |preInitCommand|String|否|Body参数|ray前置执行命令，仅ray框架可用| **数据源挂载示例**

pfs（并行文件存储）

```
{
  "type": "pfs",
  "name": "pfs-xxx",
  "mountPath": "/mnt/cluster"
}
```

pfsl3（并行文件存储）

```
{
  "type": "pfsl3",
  "options": {
    "pfsL3InstanceId": "pfs-bjiTxxSqQ",
    "pfsL3MountPoint": "pfs-bjiTxxSqQ.lb-xxxzie99.pfs.bj.baidubce.com",
    "readOnly": false
  },
  "sourcePath": "/",
  "mountPath": "/mnt/pfs"
}
```

hostpath（本地盘）

```
{
  "type": "hostpath",
  "name": "hostpath-1",
  "sourcePath": "/a",
  "mountPath": "/b",
  "options": {
    "readOnly": true
  }
}
```

bos（对象存储）

sourcePath 填写 BOS 桶名及子目录，不需要填写 name。

```
{
  "type": "bos",
  "sourcePath": "xxx-test-bos-bucket/data",
  "mountPath": "/test",
  "options": {
    "readOnly": false
  }
}
```

cfs（CFS 文件存储）

options 中的 cfsInstanceId 和 cfsMountPoint 为必填项。

```
{
  "type": "cfs",
  "name": "",
  "mountPath": "/data",
  "sourcePath": "/data",
  "options": {
    "cfsInstanceId": "cfs-xxx",
    "cfsMountPoint": "cfs-xxx.lb-xxx.cfs.xx.baidubce.com"
  }
}
```

dataset（平台数据集）

id 填写数据集 ID，options.datasetVersion 不填则取最新版本。

```
{
  "type": "dataset",
  "id": "ds-85uR2lP",
  "sourcePath": "/testv2",
  "mountPath": "/ddd",
  "options": {
    "datasetVersion": "2"
  }
}
```

public_dataset（公开数据集）

id 必填，mountPath 可选（不填则用数据集默认挂载路径），服务端强制只读。

```
{
 "type": "public_dataset",
 "id": "ds-xxxxxx",
 "mountPath": "/opt/aihc/models",
 "options": {
  "datasetVersion": "v1"
 }
}
```

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求ID| |jobId|String|任务id| |jobName|String|任务名称|

请求示例

```
{
  "name": "api-0513-2",
  "queue": "default",
  "jobType": "PyTorchJob",
  "command": "sleep 1d",
  "jobSpec": {
     "replicas": 1,
     "image": "registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release",
     "resources": [
     ],
     "envs": [
        {
           "name": "NCCL_DEBUG",
           "value": "DEBUG"
        },
        {
           "name": "NCCL_IB_DISABLE",
           "value": "0"
        }
     ],
     "enableRDMA": true
  },
  "labels": [
  ],
  "datasources": [
     {
        "type": "pfs",
        "name": "pfs-pxE6jz",
        "mountPath": "/mnt/cluster"
     }
  ]
}
```

返回示例

```
{
  "requestId": "679dfe4a-7df0-4030-be01-ae4f87eeb458",
  "jobId": "job-uVuy2Onhe4FS",
  "jobName": "api-0513-2"
}
```

查询训练任务列表

描述 获取指定资源池的任务列表

- 支持指定队列查询
- 支持按时间正序、倒序排序
- 支持分页

请求结构

```
 POST ?action=DescribeJobs&resourcePoolId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queue|String|否|Body 参数|训练任务所属队列，通用资源池须填入队列名称，不填时返回所有。托管资源池须填入队列Id| |status|String|否|Body 参数|基于状态筛选任务| |name|String|否|Body 参数|按任务名称进行模糊搜索。独立过滤字段，推荐替代 keyword 使用| |username|String|否|Body 参数|按创建人（用户名）进行模糊搜索| |jobIDs|List<String>|否|Body 参数|按任务 ID 列表精确筛选。最多传入 500 个任务 ID，元素不可为空/空白、不可重复；不传或为空数组表示不按任务 ID 筛选| |keywordType|String|否|Body 参数|筛选关键字类型，当前仅支持name/queueName。**后续废弃**，建议改用 name/username/jobIDs 等独立字段| |keyword|String|否|Body 参数|关键字值。**后续废弃**，建议改用 name/username/jobIDs 等独立字段| |orderBy|String|否|Body 参数|排序字段，支持createdAt，finishedAt，默认为createdAt| |order|String|否|Body 参数|排序方式，可选 [asc, desc]，asc 为升序，desc 为降序，默认值为 desc| |pageNumber|Number|否|Body 参数|请求分页参数，表示第几页| |pageSize|Number|否|Body 参数|单页结果数，默认值为10|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求ID，用于标识每个请求的唯一性
totalCount	Number	返回任务总数
jobs	List<JobItem>	成功请求时的返回结果

请求示例

```
{
  "pageNumber":1,
  "pageSize":3
}
```

返回示例


```
{
  "requestId": "9fa12e3c-c62d-4570-b866-bfc5069f4cf3",
  "totalCount": 3,
  "jobs": [
    {
      "jobId": "job-oeqpsJhwqq3k",
      "userId": "eca97e148cb74e9683d7b7240829d1ff",
      "name": "test-training-job-0513",
      "status": "Scheduled",
      "createdAt": "2025-05-13T02:18:51Z",
      "finishedAt": "",
      "jobType": "pytorch",
      "resourcePoolId": "cce-cm1jjxrq",
      "queueId": "default",
      "jobSpec": {
        "image": "registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release",
        "imageConfig": {
          "username": "",
          "password": ""
        },
        "replicas": 2,
        "resources": [],
        "envs": [
          {
            "name": "CUDA_DEVICE_MAX_CONNECTIONS",
            "value": "1"
          },
          {
            "name": "NCCL_DEBUG",
            "value": "INFO"
          },
          {
            "name": "AIHC_JOB_NAME",
            "value": "test-training-job-0513"
          },
          {
            "name": "AIHC_TENSORBOARD_LOG_PATH",
            "value": ""
          }
        ],
        "enableRDMA": false,
        "hostNetwork": false
      },
      "command": "sleep 1d",
      "labels": [
        {
          "key": "aijob.cce.baidubce.com/ai-user-id",
          "value": "eca97e148cb74e9683d7b7240829d1ff"
        },
        {
          "key": "aijob.cce.baidubce.com/ai-user-name",
          "value": "root"
        },
        {
          "key": "aijob.cce.baidubce.com/create-from-aihcp-api",
          "value": "true"
        },
        {
          "key": "aijob.cce.baidubce.com/openapi-jobid",
          "value": "job-oeqpsJhwqq3k"
        }
      ],
      "priority": "normal",
      "dataSources": [],
      "enableBccl": false,
      "enableBcclErrorReason": "",
      "enableFaultTolerant": false,
      "faultTolerantArgs": "--enable-hang-detection=false --max-num-of-unconditional-retry=0"
    },
    {
      "jobId": "pytorchjob-4ebe6520-fb61-40a4-8001-e36ab6838a00",
      "userId": "eca97e148cb74e9683d7b7240829d1ff",
      "name": "mqm-bos-openapi2",
      "status": "Succeeded",
      "createdAt": "2025-05-12T12:41:15Z",
      "finishedAt": "2025-05-12T12:58:04Z",
      "jobType": "pytorch",
      "resourcePoolId": "cce-cm1jjxrq",
      "queueId": "default",
      "jobSpec": {
        "image": "registry.baidubce.com/inf-qa/nginx:latest",
        "imageConfig": {
```

```
            "username": "",
            "password": ""
        },
        "replicas": 1,
        "resources": [],
        "envs": [
            {
                "name": "NCCL_DEBUG",
                "value": "INFO"
            },
            {
                "name": "env1",
                "value": "value1"
            },
            {
                "name": "AIHC_JOB_NAME",
                "value": "mqm-bos-openapi2"
            },
            {
                "name": "AIHC_TENSORBOARD_LOG_PATH",
                "value": ""
            },
            {
                "name": "CUDA_DEVICE_MAX_CONNECTIONS",
                "value": "1"
            }
        ],
        "enableRDMA": false,
        "hostNetwork": false
    },
    "command": "sleep 1000",
    "labels": [
        {
            "key": "aijob.cce.baidubce.com/ai-user-id",
            "value": "eca97e148cb74e9683d7b7240829d1ff"
        },
        {
            "key": "aijob.cce.baidubce.com/ai-user-name",
            "value": "root"
        },
        {
            "key": "aijob.cce.baidubce.com/create-from-aihcp-api",
            "value": "true"
        },
        {
            "key": "aijob.cce.baidubce.com/openapi-jobid",
            "value": "pytorchjob-4ebe6520-fb61-40a4-8001-e36ab6838a00"
        },
        {
            "key": "label1",
            "value": "value1"
        },
        {
            "key": "label2",
            "value": "value2"
        }
    ],
    "priority": "high",
    "dataSources": [],
    "enableBccl": false,
    "enableBcclErrorReason": "",
    "enableFaultTolerant": false,
    "faultTolerantArgs": "--enable-hang-detection=false --max-num-of-unconditional-retry=0"
},
{
    "jobId": "pytorchjob-d1409096-04eb-4e68-97dd-898179ab9320",
    "userId": "819eb20a0402403282ae8f84cde4024d",
    "name": "mqm-t",
    "status": "Succeeded",
    "createdAt": "2025-05-12T12:37:54Z",
    "finishedAt": "2025-05-12T12:54:47Z",
    "jobType": "pytorch",
    "resourcePoolId": "cce-cm1jjxrq",
    "queueId": "default",
    "jobSpec": {
        "image": "registry.baidubce.com/inf-qa/nginx:latest",
        "imageConfig": {
            "username": "",
            "password": ""
        },
        "replicas": 1,
        "resources": [
            {
                "name": "sharedMemory",
                "quantity": 10
            }
        ],
        "envs": [
            {
                "name": "AIHC_JOB_NAME",
                "value": "mqm-t"
            },
            {
                "name": "AIHC_TENSORBOARD_LOG_PATH",
                "value": "/output/training_logs"
            },
            {
                "name": "NCCL_IB_DISABLE",
                "value": "1"
            }
```

```
            ],
            "enableRDMA": false,
            "hostNetwork": false
        },
        "command": "sleep 1000",
        "labels": [
            {
                "key": "aijob.cce.baidubce.com/ai-user-id",
                "value": "819eb20a0402403282ae8f84cde4024d"
            },
            {
                "key": "aijob.cce.baidubce.com/ai-user-name",
                "value": "maqiming"
            },
            {
                "key": "aijob.cce.baidubce.com/create-from-aihcp",
                "value": "true"
            },
            {
                "key": "aijob.cce.baidubce.com/openapi-jobid",
                "value": "pytorchjob-d1409096-04eb-4e68-97dd-898179ab9320"
            }
        ],
        "priority": "normal",
        "dataSources": [
            {
                "type": "emptydir",
                "mountPath": "/dev/shm",
                "name": "devshm",
                "options": {
                    "sizeLimit": 10,
                    "medium": "Memory",
                    "readOnly": false
                }
            },
            {
                "type": "pfsl1",
                "sourcePath": "/",
                "mountPath": "/ds",
                "name": "pfs-7xWeAt",
                "options": {
                    "sizeLimit": 0,
                    "medium": "",
                    "readOnly": false
                }
            }
        ],
        "enableBccl": false,
        "enableBcclErrorReason": "",
        "enableFaultTolerant": false,
        "faultTolerantArgs": "--enable-hang-detection=false --max-num-of-unconditional-retry=0"
    }
  ]
}
```


查询训练任务详情

描述 获取一个训练任务的详细信息。

请求结构

```
 POST ?action=DescribeJob&resourcePoolId=xxxx&queueID=xxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID| |needDetail|Bool|否|Body 参数|是否需要详细信息，值为true时将返回Pod及历史Pod列表|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-| |requestId|String|请求ID| |jobid|String|任务id| |userId|String|用户id| |name|String|任务名称| |status|String|任务状态说明
1.Created：排队中，任务已创建，等待调度阶段（允许修改优先级），实例未创建或者所有实例状态为排队中
2.Starting：启动中，Pod 调度完成，实例的状态均为启动中或者包含部分运行中的状态
3.Running：运行中，所有的 Pod 为运行中的状态
4.Stopping：停止中，用户操作停止任务，正在停止任务，正在将实例置为失败或者删除实例
5.Stopped：已停止，任务已经停止，实例状态为失败或者实例已经被删除
6.Failed：失败，任务失败退出，所有实例状态为失败
7.Succeeded：成功，任务成功结束，所有实例状态为成功
8.Abnormal：异常，存在 >=1 的实例状态为失败
9.Restarting：重启中，任务容错或者被抢占，且所有实例状态为排队中
| |jobType|String|任务类型，如：pytorchjob、mpijob等| |resourcePoolId|String|任务所在资源池Id| |queue|String|任务所在资源池队列| |jobSpec|JobSpec|任务配置| |createdAt|String|任务创建时间| |finishedAt|String|任务结束时间| |datasources|List<DataSource>|任务的数据源配置| |labels|List<Label>|任务标签| |priority|String|任务优先级| |enableBccl|Boolean|任务是否开启了BCCL注入| |enableBcclStatus|String|BCCL注入状态,包括: success: 注入成功 failed: 注入失败 unknown: 还未执行注入| |enableBcclErrorReason|String|BCCL注入失败原因| |enableFaultTolerance|Boolean|是否开启容错| |faultToleranceArgs|String|容错配置参数| |gpuCount|Integer|GPU卡数| |nodeCount|Integer|训练任务实例数| |gpuUtilizationPercent|Number|GPU利用率（百分比）| |gpuMemoryUtilizationPercent|Number|GPU内存利用率百分比（百分比）| |runTimeNanoseconds|Integer|训练任务耗时（纳秒）| |visibleScope|Integer |训练任务可见范围，参数为1代表队列内可见，为0代表创建人可见| |pods|List<Pod>|任务Pod列表，在详情接口needDetail参数为False和查询训练任务列表接口中不返回该字段| |historyPods|List<Pod>|历史Pod列表，在详情接口needDetail参数为False和查询训练任务列表接口中不返回该字段| |jobTimeLine|List<JobTimeLine>|任务时间线详情信息，查询训练任务列表接口中不返回该字段| |portConfigs|PortConfigs|自定义监控监听端口配置，创建时未设置则不返回该字段| |idleTask|Boolean|是否为闲时任务，闲时资源可能被回收| |preInitCommand|String|ray前置执行命令，仅ray框架返回| |advancedSettings|AdvancedSettings|RayJob 高级配置，包括运行时环境、submitter 重试策略、Head 节点的容错配置和 BLB 负载均衡| **请求示例**

```
{
  "jobId": "job-s2ea9wyjvh8b",
  "needDetail": true
}
```

返回示例

```
{
  "requestId": "9d95fafe-50cf-41cf-825e-68942dff5e42",
  "jobId": "job-s2ea9wyjvh8b",
  "userId": "3cbc032c16be46aaabe53cf3cf823653",
  "name": "ccc-copy3",
  "status": "Running",
  "createdAt": "2026-03-17T03:35:09Z",
  "finishedAt": "",
  "jobType": "pytorch",
  "resourcePoolId": "cce-k8q7qdl9",
  "queue": "test",
  "jobSpec": {
    "image": "registry.baidubce.com/inf-qa/nginx:latest",
    "imageConfig": {
      "username": "",
      "password": ""
    },
    "replicas": 2,
    "resources": [
      {
        "name": "sharedMemory",
        "quantity": 32
      }
    ],
    "envs": [
      {
        "name": "AIHC_JOB_NAME",
        "value": "ccc-copy3"
      },
      {
        "name": "LOG_COLLECTION",
        "value": "true"
      }
    ],
    "enableRDMA": false,
    "hostNetwork": false
  },
  "command": "sleep 1d",
  "labels": [
    {
      "key": "aijob.cce.baidubce.com/openapi-jobid",
      "value": "job-s2ea9wyjvh8b"
    },
    {
      "key": "aihc.baidubce.com/userid",
      "value": "3cbc032c16be46aaabe53cf3cf823653"
    },
    {
      "key": "aihc.baidubce.com/username",
      "value": "liuqidi"
    },
    {
      "key": "aihc.baidubce.com/workload-type",
      "value": "train"
    },
    {
      "key": "aijob.cce.baidubce.com/ai-user-id",
      "value": "3cbc032c16be46aaabe53cf3cf823653"
    },
    {
      "key": "aijob.cce.baidubce.com/ai-user-name",
      "value": "liuqidi"
    },
    {
      "key": "aijob.cce.baidubce.com/create-from-aihcp",
      "value": "true"
    },
    {
      "key": "aijob.cce.baidubce.com/log-collection",
      "value": "v2"
    }
  ],
  "annotations": {
    "aijob.cce.baidubce.com/aigw-id": "aigw-rehszagj"
  },
  "priority": "normal",
  "dataSources": [
    {
      "type": "emptydir",
      "mountPath": "/dev/shm",
      "name": "devshm",
      "options": {
        "sizeLimit": 32,
        "medium": "Memory",
        "readOnly": false,
        "cfsInstanceId": "",
        "cfsMountPoint": ""
      }
    }
  ],
  "enableBccl": false,
```

```
  "enableBcclErrorReason": "",
  "enableFaultTolerance": false,
  "faultToleranceArgs": "--enable-replace=true --enable-hang-detection=true --hang-detection-log-timeout-minutes=1 --hang-detection-startup-toleration-minutes=10 --hang-detection-stack-timeout-minutes=0 --max-num-of-unconditional-retry=2 --custom-log-patterns=connect timeout",
  "gpuCount": 0,
  "nodeCount": 0,
  "advancedSettings": {
    "submitterBackoffLimit": 1,
    "redisInstanceId": "scs-bj-xxxxxxxx",
    "redisUsername": "default",
    "enableBlb": true,
    "blbInstanceId": "lb-xxxxxx"
  },
  "pods": [
    {
      "PodIP": "10.0.0.214",
      "nodeName": "10.0.0.49",
      "creationTimestamp": "2026-03-17T04:04:59Z",
      "uid": "51f0ba20-b677-46e0-968d-ce85a4b84282",
      "name": "job-s2ea9wyjvh8b-worker-0",
      "podPhase": "Running",
      "status": "Running",
      "replicaType": "worker",
      "restartCount": 2,
      "envs": [
        {
          "name": "AIHC_JOB_NAME",
          "value": "ccc-copy3"
        },
        {
          "name": "LOG_COLLECTION",
          "value": "true"
        },
        {
          "name": "AIHC_JOB_ID",
          "value": "job-s2ea9wyjvh8b"
        },
        {
          "name": "AIHC_TENSORBOARD_LOG_PATH",
          "value": ""
        },
        {
          "name": "NVIDIA_VISIBLE_DEVICES",
          "value": ""
        },
        {
          "name": "BAIGE_ENTRYPOINT_DIR",
          "value": "/baige"
        },
        {
          "name": "BCCL_BUS_BW_CALCULATE_MODE",
          "value": "Agg"
        },
        {
          "name": "BCCL_PROFILING_FILE",
          "value": "/var/bccl/logs/busbw.cal.%h.%p"
        },
        {
          "name": "BCCL_ERROR_FILE",
          "value": "/var/bccl/logs/err.%h.%p.log"
        },
        {
          "name": "BCCL_TRACE_HANG_ENABLE",
          "value": "1"
        },
        {
          "name": "BCCL_HANG_DETECT_INTERVAL",
          "value": "30"
        },
        {
          "name": "BCCL_UNIX_SOCKET_PATH",
          "value": "/var/bccl/sockets"
        },
        {
          "name": "BCCL_TRACE_SLOW_ENABLE",
          "value": "1"
        },
        {
          "name": "ENABLE_FAULT_TOLERANCE",
          "value": "1"
        },
        {
          "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
          "value": "1"
        },
        {
          "name": "JOB_RESOURCE_TYPE",
          "value": ""
        },
        {
          "name": "FT_AGENT_SOCKET_PATH",
          "value": "/fault-tolerance/ftagent.sock"
        },
        {
          "name": "PYTHONPATH",
          "value": "/fault-tolerance/elastic-agent/python/site-packages"
        },
        {
          "name": "NODE_NAME",
          "value": ""
```

```
        },
        {
          "name": "JOB_UID",
          "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
        },
        {
          "name": "JOB_NAME",
          "value": "job-s2ea9wyjvh8b"
        },
        {
          "name": "POD_NAME",
          "value": ""
        },
        {
          "name": "CLUSTER_ID",
          "value": "cce-k8q7qdl9"
        },
        {
          "name": "REGION",
          "value": "bj"
        },
        {
          "name": "PYTHONUNBUFFERED",
          "value": "0"
        },
        {
          "name": "MASTER_PORT",
          "value": "23456"
        },
        {
          "name": "MASTER_ADDR",
          "value": "job-s2ea9wyjvh8b-master-0"
        },
        {
          "name": "WORLD_SIZE",
          "value": "2"
        },
        {
          "name": "RANK",
          "value": "1"
        },
        {
          "name": "PET_RDZV_ENDPOINT",
          "value": "job-s2ea9wyjvh8b-master-0:23456"
        },
        {
          "name": "PET_RDZV_BACKEND",
          "value": "c10d"
        },
        {
          "name": "PET_NNODES",
          "value": "2:2"
        },
        {
          "name": "PET_MAX_RESTARTS",
          "value": "10"
        },
        {
          "name": "PET_LOCAL_ADDR",
          "value": "job-s2ea9wyjvh8b-worker-0"
        }
      ],
      "finishedAt": "",
      "reason": ""
    },
    {
      "PodIP": "10.0.0.220",
      "nodeName": "10.0.0.49",
      "creationTimestamp": "2026-03-17T04:04:58Z",
      "uid": "5cb38e70-769e-4c70-ab1a-d9a879a15efa",
      "name": "job-s2ea9wyjvh8b-master-0",
      "podPhase": "Running",
      "status": "Running",
      "replicaType": "master",
      "restartCount": 2,
      "envs": [
        {
          "name": "LOG_COLLECTION",
          "value": "true"
        },
        {
          "name": "AIHC_JOB_ID",
          "value": "job-s2ea9wyjvh8b"
        },
        {
          "name": "AIHC_TENSORBOARD_LOG_PATH",
          "value": ""
        },
        {
          "name": "NVIDIA_VISIBLE_DEVICES",
          "value": ""
        },
        {
          "name": "AIHC_JOB_NAME",
          "value": "ccc-copy3"
        },
        {
          "name": "BAIGE_ENTRYPOINT_DIR",
          "value": "/baige"
        }
```

```
        },
        {
          "name": "BCCL_BUS_BW_CALCULATE_MODE",
          "value": "Agg"
        },
        {
          "name": "BCCL_PROFILING_FILE",
          "value": "/var/bccl/logs/busbw.cal.%h.%p"
        },
        {
          "name": "BCCL_ERROR_FILE",
          "value": "/var/bccl/logs/err.%h.%p.log"
        },
        {
          "name": "BCCL_TRACE_HANG_ENABLE",
          "value": "1"
        },
        {
          "name": "BCCL_HANG_DETECT_INTERVAL",
          "value": "30"
        },
        {
          "name": "BCCL_UNIX_SOCKET_PATH",
          "value": "/var/bccl/sockets"
        },
        {
          "name": "BCCL_TRACE_SLOW_ENABLE",
          "value": "1"
        },
        {
          "name": "ENABLE_FAULT_TOLERANCE",
          "value": "1"
        },
        {
          "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
          "value": "1"
        },
        {
          "name": "JOB_RESOURCE_TYPE",
          "value": ""
        },
        {
          "name": "FT_AGENT_SOCKET_PATH",
          "value": "/fault-tolerance/ftagent.sock"
        },
        {
          "name": "PYTHONPATH",
          "value": "/fault-tolerance/elastic-agent/python/site-packages"
        },
        {
          "name": "NODE_NAME",
          "value": ""
        },
        {
          "name": "JOB_UID",
          "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
        },
        {
          "name": "JOB_NAME",
          "value": "job-s2ea9wyjvh8b"
        },
        {
          "name": "POD_NAME",
          "value": ""
        },
        {
          "name": "CLUSTER_ID",
          "value": "cce-k8q7qdl9"
        },
        {
          "name": "REGION",
          "value": "bj"
        },
        {
          "name": "PYTHONUNBUFFERED",
          "value": "0"
        },
        {
          "name": "MASTER_PORT",
          "value": "23456"
        },
        {
          "name": "MASTER_ADDR",
          "value": "job-s2ea9wyjvh8b-master-0"
        },
        {
          "name": "WORLD_SIZE",
          "value": "2"
        },
        {
          "name": "RANK",
          "value": "0"
        },
        {
          "name": "PET_RDZV_ENDPOINT",
          "value": "job-s2ea9wyjvh8b-master-0:23456"
        },
        {
          "name": "PET_RDZV_BACKEND",
          "value": "c10d"
```

```
            },
            {
              "name": "PET_NNODES",
              "value": "2:2"
            },
            {
              "name": "PET_MAX_RESTARTS",
              "value": "10"
            },
            {
              "name": "PET_LOCAL_ADDR",
              "value": "job-s2ea9wyjvh8b-master-0"
            }
          ],
          "finishedAt": "",
          "reason": ""
        }
      ],
      "historyPods": [
        {
          "PodIP": "10.0.0.218",
          "nodeName": "10.0.0.49",
          "creationTimestamp": "2026-03-17T03:49:58Z",
          "uid": "73489812-cb30-4cd4-9914-6a3ccdc6f5ba",
          "name": "job-s2ea9wyjvh8b-worker-0",
          "podPhase": "Failed",
          "status": "Stopped",
          "replicaType": "worker",
          "restartCount": 1,
          "envs": [
            {
              "name": "AIHC_JOB_NAME",
              "value": "ccc-copy3"
            },
            {
              "name": "LOG_COLLECTION",
              "value": "true"
            },
            {
              "name": "AIHC_JOB_ID",
              "value": "job-s2ea9wyjvh8b"
            },
            {
              "name": "AIHC_TENSORBOARD_LOG_PATH",
              "value": ""
            },
            {
              "name": "NVIDIA_VISIBLE_DEVICES",
              "value": ""
            },
            {
              "name": "BAIGE_ENTRYPOINT_DIR",
              "value": "/baige"
            },
            {
              "name": "BCCL_BUS_BW_CALCULATE_MODE",
              "value": "Agg"
            },
            {
              "name": "BCCL_PROFILING_FILE",
              "value": "/var/bccl/logs/busbw.cal.%h.%p"
            },
            {
              "name": "BCCL_ERROR_FILE",
              "value": "/var/bccl/logs/err.%h.%p.log"
            },
            {
              "name": "BCCL_TRACE_HANG_ENABLE",
              "value": "1"
            },
            {
              "name": "BCCL_HANG_DETECT_INTERVAL",
              "value": "30"
            },
            {
              "name": "BCCL_UNIX_SOCKET_PATH",
              "value": "/var/bccl/sockets"
            },
            {
              "name": "BCCL_TRACE_SLOW_ENABLE",
              "value": "1"
            },
            {
              "name": "ENABLE_FAULT_TOLERANCE",
              "value": "1"
            },
            {
              "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
              "value": "1"
            },
            {
              "name": "JOB_RESOURCE_TYPE",
              "value": ""
            },
            {
              "name": "FT_AGENT_SOCKET_PATH",
              "value": "/fault-tolerance/ftagent.sock"
            },
            {
              "name": "PYTHONPATH",
```

```
            "value": "/fault-tolerance/elastic-agent/python/site-packages"
          },
          {
            "name": "NODE_NAME",
            "value": ""
          },
          {
            "name": "JOB_UID",
            "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
          },
          {
            "name": "JOB_NAME",
            "value": "job-s2ea9wyjvh8b"
          },
          {
            "name": "POD_NAME",
            "value": ""
          },
          {
            "name": "CLUSTER_ID",
            "value": "cce-k8q7qdl9"
          },
          {
            "name": "REGION",
            "value": "bj"
          },
          {
            "name": "PYTHONUNBUFFERED",
            "value": "0"
          },
          {
            "name": "MASTER_PORT",
            "value": "23456"
          },
          {
            "name": "MASTER_ADDR",
            "value": "job-s2ea9wyjvh8b-master-0"
          },
          {
            "name": "WORLD_SIZE",
            "value": "2"
          },
          {
            "name": "RANK",
            "value": "1"
          },
          {
            "name": "PET_RDZV_ENDPOINT",
            "value": "job-s2ea9wyjvh8b-master-0:23456"
          },
          {
            "name": "PET_RDZV_BACKEND",
            "value": "c10d"
          },
          {
            "name": "PET_NNODES",
            "value": "2:2"
          },
          {
            "name": "PET_MAX_RESTARTS",
            "value": "10"
          },
          {
            "name": "PET_LOCAL_ADDR",
            "value": "job-s2ea9wyjvh8b-worker-0"
          }
        ],
        "finishedAt": "2026-03-17T04:04:56Z",
        "reason": ""
      },
      {
        "PodIP": "10.0.0.219",
        "nodeName": "10.0.0.49",
        "creationTimestamp": "2026-03-17T03:35:12Z",
        "uid": "789bd4eb-dfb8-4151-b97d-991907b45a67",
        "name": "job-s2ea9wyjvh8b-worker-0",
        "podPhase": "Failed",
        "status": "Stopped",
        "replicaType": "worker",
        "restartCount": 0,
        "envs": [
          {
            "name": "AIHC_JOB_NAME",
            "value": "ccc-copy3"
          },
          {
            "name": "LOG_COLLECTION",
            "value": "true"
          },
          {
            "name": "AIHC_JOB_ID",
            "value": "job-s2ea9wyjvh8b"
          },
          {
            "name": "AIHC_TENSORBOARD_LOG_PATH",
            "value": ""
          },
          {
            "name": "NVIDIA_VISIBLE_DEVICES",
            "value": ""
```

```
    },
    {
      "name": "BAIGE_ENTRYPOINT_DIR",
      "value": "/baige"
    },
    {
      "name": "BCCL_BUS_BW_CALCULATE_MODE",
      "value": "Agg"
    },
    {
      "name": "BCCL_PROFILING_FILE",
      "value": "/var/bccl/logs/busbw.cal.%h.%p"
    },
    {
      "name": "BCCL_ERROR_FILE",
      "value": "/var/bccl/logs/err.%h.%p.log"
    },
    {
      "name": "BCCL_TRACE_HANG_ENABLE",
      "value": "1"
    },
    {
      "name": "BCCL_HANG_DETECT_INTERVAL",
      "value": "30"
    },
    {
      "name": "BCCL_UNIX_SOCKET_PATH",
      "value": "/var/bccl/sockets"
    },
    {
      "name": "BCCL_TRACE_SLOW_ENABLE",
      "value": "1"
    },
    {
      "name": "ENABLE_FAULT_TOLERANCE",
      "value": "1"
    },
    {
      "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
      "value": "1"
    },
    {
      "name": "JOB_RESOURCE_TYPE",
      "value": ""
    },
    {
      "name": "FT_AGENT_SOCKET_PATH",
      "value": "/fault-tolerance/ftagent.sock"
    },
    {
      "name": "PYTHONPATH",
      "value": "/fault-tolerance/elastic-agent/python/site-packages"
    },
    {
      "name": "NODE_NAME",
      "value": ""
    },
    {
      "name": "JOB_UID",
      "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
    },
    {
      "name": "JOB_NAME",
      "value": "job-s2ea9wyjvh8b"
    },
    {
      "name": "POD_NAME",
      "value": ""
    },
    {
      "name": "CLUSTER_ID",
      "value": "cce-k8q7qdl9"
    },
    {
      "name": "REGION",
      "value": "bj"
    },
    {
      "name": "PYTHONUNBUFFERED",
      "value": "0"
    },
    {
      "name": "MASTER_PORT",
      "value": "23456"
    },
    {
      "name": "MASTER_ADDR",
      "value": "job-s2ea9wyjvh8b-master-0"
    },
    {
      "name": "WORLD_SIZE",
      "value": "2"
    },
    {
      "name": "RANK",
      "value": "1"
    },
    {
      "name": "PET_RDZV_ENDPOINT",
      "value": "job-s2ea9wyjvh8b-master-0:23456"
```

```
            "value": "job-s2ea9wyjvh8b-master-0:23456"
          },
          {
            "name": "PET_RDZV_BACKEND",
            "value": "c10d"
          },
          {
            "name": "PET_NNODES",
            "value": "2:2"
          },
          {
            "name": "PET_MAX_RESTARTS",
            "value": "10"
          },
          {
            "name": "PET_LOCAL_ADDR",
            "value": "job-s2ea9wyjvh8b-worker-0"
          }
        ],
        "finishedAt": "2026-03-17T03:49:56Z",
        "reason": ""
      },
      {
        "PodIP": "10.0.0.219",
        "nodeName": "10.0.0.49",
        "creationTimestamp": "2026-03-17T03:49:58Z",
        "uid": "a3c5f722-f8e3-405e-a11b-6e779879ee2e",
        "name": "job-s2ea9wyjvh8b-master-0",
        "podPhase": "Failed",
        "status": "Stopped",
        "replicaType": "master",
        "restartCount": 1,
        "envs": [
          {
            "name": "LOG_COLLECTION",
            "value": "true"
          },
          {
            "name": "AIHC_JOB_ID",
            "value": "job-s2ea9wyjvh8b"
          },
          {
            "name": "AIHC_TENSORBOARD_LOG_PATH",
            "value": ""
          },
          {
            "name": "NVIDIA_VISIBLE_DEVICES",
            "value": ""
          },
          {
            "name": "AIHC_JOB_NAME",
            "value": "ccc-copy3"
          },
          {
            "name": "BAIGE_ENTRYPOINT_DIR",
            "value": "/baige"
          },
          {
            "name": "BCCL_BUS_BW_CALCULATE_MODE",
            "value": "Agg"
          },
          {
            "name": "BCCL_PROFILING_FILE",
            "value": "/var/bccl/logs/busbw.cal.%h.%p"
          },
          {
            "name": "BCCL_ERROR_FILE",
            "value": "/var/bccl/logs/err.%h.%p.log"
          },
          {
            "name": "BCCL_TRACE_HANG_ENABLE",
            "value": "1"
          },
          {
            "name": "BCCL_HANG_DETECT_INTERVAL",
            "value": "30"
          },
          {
            "name": "BCCL_UNIX_SOCKET_PATH",
            "value": "/var/bccl/sockets"
          },
          {
            "name": "BCCL_TRACE_SLOW_ENABLE",
            "value": "1"
          },
          {
            "name": "ENABLE_FAULT_TOLERANCE",
            "value": "1"
          },
          {
            "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
            "value": "1"
          },
          {
            "name": "JOB_RESOURCE_TYPE",
            "value": ""
          },
          {
            "name": "FT_AGENT_SOCKET_PATH",
            "value": "/fault-tolerance/ftagent.sock"
```

```
            },
            {
              "name": "PYTHONPATH",
              "value": "/fault-tolerance/elastic-agent/python/site-packages"
            },
            {
              "name": "NODE_NAME",
              "value": ""
            },
            {
              "name": "JOB_UID",
              "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
            },
            {
              "name": "JOB_NAME",
              "value": "job-s2ea9wyjvh8b"
            },
            {
              "name": "POD_NAME",
              "value": ""
            },
            {
              "name": "CLUSTER_ID",
              "value": "cce-k8q7qdl9"
            },
            {
              "name": "REGION",
              "value": "bj"
            },
            {
              "name": "PYTHONUNBUFFERED",
              "value": "0"
            },
            {
              "name": "MASTER_PORT",
              "value": "23456"
            },
            {
              "name": "MASTER_ADDR",
              "value": "job-s2ea9wyjvh8b-master-0"
            },
            {
              "name": "WORLD_SIZE",
              "value": "2"
            },
            {
              "name": "RANK",
              "value": "0"
            },
            {
              "name": "PET_RDZV_ENDPOINT",
              "value": "job-s2ea9wyjvh8b-master-0:23456"
            },
            {
              "name": "PET_RDZV_BACKEND",
              "value": "c10d"
            },
            {
              "name": "PET_NNODES",
              "value": "2:2"
            },
            {
              "name": "PET_MAX_RESTARTS",
              "value": "10"
            },
            {
              "name": "PET_LOCAL_ADDR",
              "value": "job-s2ea9wyjvh8b-master-0"
            }
        ],
        "finishedAt": "2026-03-17T04:04:55Z",
        "reason": ""
    },
    {
        "PodIP": "10.0.0.216",
        "nodeName": "10.0.0.49",
        "creationTimestamp": "2026-03-17T03:35:12Z",
        "uid": "f66f7118-6cad-46c6-af17-6d45afb8a3c2",
        "name": "job-s2ea9wyjvh8b-master-0",
        "podPhase": "Failed",
        "status": "Stopped",
        "replicaType": "master",
        "restartCount": 0,
        "envs": [
            {
              "name": "LOG_COLLECTION",
              "value": "true"
            },
            {
              "name": "AIHC_JOB_ID",
              "value": "job-s2ea9wyjvh8b"
            },
            {
              "name": "AIHC_TENSORBOARD_LOG_PATH",
              "value": ""
            },
            {
              "name": "NVIDIA_VISIBLE_DEVICES",
              "value": ""
            },
```

```
        {
            "name": "AIHC_JOB_NAME",
            "value": "ccc-copy3"
        },
        {
            "name": "BAIGE_ENTRYPOINT_DIR",
            "value": "/baige"
        },
        {
            "name": "BCCL_BUS_BW_CALCULATE_MODE",
            "value": "Agg"
        },
        {
            "name": "BCCL_PROFILING_FILE",
            "value": "/var/bccl/logs/busbw.cal.%h.%p"
        },
        {
            "name": "BCCL_ERROR_FILE",
            "value": "/var/bccl/logs/err.%h.%p.log"
        },
        {
            "name": "BCCL_TRACE_HANG_ENABLE",
            "value": "1"
        },
        {
            "name": "BCCL_HANG_DETECT_INTERVAL",
            "value": "30"
        },
        {
            "name": "BCCL_UNIX_SOCKET_PATH",
            "value": "/var/bccl/sockets"
        },
        {
            "name": "BCCL_TRACE_SLOW_ENABLE",
            "value": "1"
        },
        {
            "name": "ENABLE_FAULT_TOLERANCE",
            "value": "1"
        },
        {
            "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
            "value": "1"
        },
        {
            "name": "JOB_RESOURCE_TYPE",
            "value": ""
        },
        {
            "name": "FT_AGENT_SOCKET_PATH",
            "value": "/fault-tolerance/ftagent.sock"
        },
        {
            "name": "PYTHONPATH",
            "value": "/fault-tolerance/elastic-agent/python/site-packages"
        },
        {
            "name": "NODE_NAME",
            "value": ""
        },
        {
            "name": "JOB_UID",
            "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
        },
        {
            "name": "JOB_NAME",
            "value": "job-s2ea9wyjvh8b"
        },
        {
            "name": "POD_NAME",
            "value": ""
        },
        {
            "name": "CLUSTER_ID",
            "value": "cce-k8q7qdl9"
        },
        {
            "name": "REGION",
            "value": "bj"
        },
        {
            "name": "PYTHONUNBUFFERED",
            "value": "0"
        },
        {
            "name": "MASTER_PORT",
            "value": "23456"
        },
        {
            "name": "MASTER_ADDR",
            "value": "job-s2ea9wyjvh8b-master-0"
        },
        {
            "name": "WORLD_SIZE",
            "value": "2"
        },
        {
            "name": "RANK",
            "value": "0"
        }
```

```
        },
        {
          "name": "PET_RDZV_ENDPOINT",
          "value": "job-s2ea9wyjvh8b-master-0:23456"
        },
        {
          "name": "PET_RDZV_BACKEND",
          "value": "c10d"
        },
        {
          "name": "PET_NNODES",
          "value": "2:2"
        },
        {
          "name": "PET_MAX_RESTARTS",
          "value": "10"
        },
        {
          "name": "PET_LOCAL_ADDR",
          "value": "job-s2ea9wyjvh8b-master-0"
        }
      ],
      "finishedAt": "2026-03-17T03:49:56Z",
      "reason": ""
    },
    {
      "PodIP": "10.0.0.218",
      "nodeName": "10.0.0.49",
      "creationTimestamp": "2026-03-17T03:49:58Z",
      "uid": "73489812-cb30-4cd4-9914-6a3ccdc6f5ba",
      "name": "job-s2ea9wyjvh8b-worker-0",
      "podPhase": "Failed",
      "status": "Stopped",
      "replicaType": "worker",
      "restartCount": 1,
      "envs": [
        {
          "name": "AIHC_JOB_NAME",
          "value": "ccc-copy3"
        },
        {
          "name": "LOG_COLLECTION",
          "value": "true"
        },
        {
          "name": "AIHC_JOB_ID",
          "value": "job-s2ea9wyjvh8b"
        },
        {
          "name": "AIHC_TENSORBOARD_LOG_PATH",
          "value": ""
        },
        {
          "name": "NVIDIA_VISIBLE_DEVICES",
          "value": ""
        },
        {
          "name": "BAIGE_ENTRYPOINT_DIR",
          "value": "/baige"
        },
        {
          "name": "BCCL_BUS_BW_CALCULATE_MODE",
          "value": "Agg"
        },
        {
          "name": "BCCL_PROFILING_FILE",
          "value": "/var/bccl/logs/busbw.cal.%h.%p"
        },
        {
          "name": "BCCL_ERROR_FILE",
          "value": "/var/bccl/logs/err.%h.%p.log"
        },
        {
          "name": "BCCL_TRACE_HANG_ENABLE",
          "value": "1"
        },
        {
          "name": "BCCL_HANG_DETECT_INTERVAL",
          "value": "30"
        },
        {
          "name": "BCCL_UNIX_SOCKET_PATH",
          "value": "/var/bccl/sockets"
        },
        {
          "name": "BCCL_TRACE_SLOW_ENABLE",
          "value": "1"
        },
        {
          "name": "ENABLE_FAULT_TOLERANCE",
          "value": "1"
        },
        {
          "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
          "value": "1"
        },
        {
          "name": "JOB_RESOURCE_TYPE",
          "value": ""
        },
```

```
            {
              "name": "FT_AGENT_SOCKET_PATH",
              "value": "/fault-tolerance/ftagent.sock"
            },
            {
              "name": "PYTHONPATH",
              "value": "/fault-tolerance/elastic-agent/python/site-packages"
            },
            {
              "name": "NODE_NAME",
              "value": ""
            },
            {
              "name": "JOB_UID",
              "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
            },
            {
              "name": "JOB_NAME",
              "value": "job-s2ea9wyjvh8b"
            },
            {
              "name": "POD_NAME",
              "value": ""
            },
            {
              "name": "CLUSTER_ID",
              "value": "cce-k8q7qdl9"
            },
            {
              "name": "REGION",
              "value": "bj"
            },
            {
              "name": "PYTHONUNBUFFERED",
              "value": "0"
            },
            {
              "name": "MASTER_PORT",
              "value": "23456"
            },
            {
              "name": "MASTER_ADDR",
              "value": "job-s2ea9wyjvh8b-master-0"
            },
            {
              "name": "WORLD_SIZE",
              "value": "2"
            },
            {
              "name": "RANK",
              "value": "1"
            },
            {
              "name": "PET_RDZV_ENDPOINT",
              "value": "job-s2ea9wyjvh8b-master-0:23456"
            },
            {
              "name": "PET_RDZV_BACKEND",
              "value": "c10d"
            },
            {
              "name": "PET_NNODES",
              "value": "2:2"
            },
            {
              "name": "PET_MAX_RESTARTS",
              "value": "10"
            },
            {
              "name": "PET_LOCAL_ADDR",
              "value": "job-s2ea9wyjvh8b-worker-0"
            }
        ],
        "finishedAt": "2026-03-17T04:04:56Z",
        "reason": ""
    },
    {
        "PodIP": "10.0.0.219",
        "nodeName": "10.0.0.49",
        "creationTimestamp": "2026-03-17T03:35:12Z",
        "uid": "789bd4eb-dfb8-4151-b97d-991907b45a67",
        "name": "job-s2ea9wyjvh8b-worker-0",
        "podPhase": "Failed",
        "status": "Stopped",
        "replicaType": "worker",
        "restartCount": 0,
        "envs": [
            {
              "name": "AIHC_JOB_NAME",
              "value": "ccc-copy3"
            },
            {
              "name": "LOG_COLLECTION",
              "value": "true"
            },
            {
              "name": "AIHC_JOB_ID",
              "value": "job-s2ea9wyjvh8b"
            },
            {
```

```
            "name": "AIHC_TENSORBOARD_LOG_PATH",
            "value": ""
        },
        {
            "name": "NVIDIA_VISIBLE_DEVICES",
            "value": ""
        },
        {
            "name": "BAIGE_ENTRYPOINT_DIR",
            "value": "/baige"
        },
        {
            "name": "BCCL_BUS_BW_CALCULATE_MODE",
            "value": "Agg"
        },
        {
            "name": "BCCL_PROFILING_FILE",
            "value": "/var/bccl/logs/busbw.cal.%h.%p"
        },
        {
            "name": "BCCL_ERROR_FILE",
            "value": "/var/bccl/logs/err.%h.%p.log"
        },
        {
            "name": "BCCL_TRACE_HANG_ENABLE",
            "value": "1"
        },
        {
            "name": "BCCL_HANG_DETECT_INTERVAL",
            "value": "30"
        },
        {
            "name": "BCCL_UNIX_SOCKET_PATH",
            "value": "/var/bccl/sockets"
        },
        {
            "name": "BCCL_TRACE_SLOW_ENABLE",
            "value": "1"
        },
        {
            "name": "ENABLE_FAULT_TOLERANCE",
            "value": "1"
        },
        {
            "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
            "value": "1"
        },
        {
            "name": "JOB_RESOURCE_TYPE",
            "value": ""
        },
        {
            "name": "FT_AGENT_SOCKET_PATH",
            "value": "/fault-tolerance/ftagent.sock"
        },
        {
            "name": "PYTHONPATH",
            "value": "/fault-tolerance/elastic-agent/python/site-packages"
        },
        {
            "name": "NODE_NAME",
            "value": ""
        },
        {
            "name": "JOB_UID",
            "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
        },
        {
            "name": "JOB_NAME",
            "value": "job-s2ea9wyjvh8b"
        },
        {
            "name": "POD_NAME",
            "value": ""
        },
        {
            "name": "CLUSTER_ID",
            "value": "cce-k8q7qdl9"
        },
        {
            "name": "REGION",
            "value": "bj"
        },
        {
            "name": "PYTHONUNBUFFERED",
            "value": "0"
        },
        {
            "name": "MASTER_PORT",
            "value": "23456"
        },
        {
            "name": "MASTER_ADDR",
            "value": "job-s2ea9wyjvh8b-master-0"
        },
        {
            "name": "WORLD_SIZE",
            "value": "2"
        },
        {
```

```
            {
              "name": "RANK",
              "value": "1"
            },
            {
              "name": "PET_RDZV_ENDPOINT",
              "value": "job-s2ea9wyjvh8b-master-0:23456"
            },
            {
              "name": "PET_RDZV_BACKEND",
              "value": "c10d"
            },
            {
              "name": "PET_NNODES",
              "value": "2:2"
            },
            {
              "name": "PET_MAX_RESTARTS",
              "value": "10"
            },
            {
              "name": "PET_LOCAL_ADDR",
              "value": "job-s2ea9wyjvh8b-worker-0"
            }
          ],
          "finishedAt": "2026-03-17T03:49:56Z",
          "reason": ""
        },
        {
          "PodIP": "10.0.0.219",
          "nodeName": "10.0.0.49",
          "creationTimestamp": "2026-03-17T03:49:58Z",
          "uid": "a3c5f722-f8e3-405e-a11b-6e779879ee2e",
          "name": "job-s2ea9wyjvh8b-master-0",
          "podPhase": "Failed",
          "status": "Stopped",
          "replicaType": "master",
          "restartCount": 1,
          "envs": [
            {
              "name": "LOG_COLLECTION",
              "value": "true"
            },
            {
              "name": "AIHC_JOB_ID",
              "value": "job-s2ea9wyjvh8b"
            },
            {
              "name": "AIHC_TENSORBOARD_LOG_PATH",
              "value": ""
            },
            {
              "name": "NVIDIA_VISIBLE_DEVICES",
              "value": ""
            },
            {
              "name": "AIHC_JOB_NAME",
              "value": "ccc-copy3"
            },
            {
              "name": "BAIGE_ENTRYPOINT_DIR",
              "value": "/baige"
            },
            {
              "name": "BCCL_BUS_BW_CALCULATE_MODE",
              "value": "Agg"
            },
            {
              "name": "BCCL_PROFILING_FILE",
              "value": "/var/bccl/logs/busbw.cal.%h.%p"
            },
            {
              "name": "BCCL_ERROR_FILE",
              "value": "/var/bccl/logs/err.%h.%p.log"
            },
            {
              "name": "BCCL_TRACE_HANG_ENABLE",
              "value": "1"
            },
            {
              "name": "BCCL_HANG_DETECT_INTERVAL",
              "value": "30"
            },
            {
              "name": "BCCL_UNIX_SOCKET_PATH",
              "value": "/var/bccl/sockets"
            },
            {
              "name": "BCCL_TRACE_SLOW_ENABLE",
              "value": "1"
            },
            {
              "name": "ENABLE_FAULT_TOLERANCE",
              "value": "1"
            },
            {
              "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
              "value": "1"
            },
            {
              "name": "JOB_RESOURCE_TYPE",
```

```
            "name": "JOB_RESOURCE_TYPE",
            "value": ""
          },
          {
            "name": "FT_AGENT_SOCKET_PATH",
            "value": "/fault-tolerance/ftagent.sock"
          },
          {
            "name": "PYTHONPATH",
            "value": "/fault-tolerance/elastic-agent/python/site-packages"
          },
          {
            "name": "NODE_NAME",
            "value": ""
          },
          {
            "name": "JOB_UID",
            "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
          },
          {
            "name": "JOB_NAME",
            "value": "job-s2ea9wyjvh8b"
          },
          {
            "name": "POD_NAME",
            "value": ""
          },
          {
            "name": "CLUSTER_ID",
            "value": "cce-k8q7qdl9"
          },
          {
            "name": "REGION",
            "value": "bj"
          },
          {
            "name": "PYTHONUNBUFFERED",
            "value": "0"
          },
          {
            "name": "MASTER_PORT",
            "value": "23456"
          },
          {
            "name": "MASTER_ADDR",
            "value": "job-s2ea9wyjvh8b-master-0"
          },
          {
            "name": "WORLD_SIZE",
            "value": "2"
          },
          {
            "name": "RANK",
            "value": "0"
          },
          {
            "name": "PET_RDZV_ENDPOINT",
            "value": "job-s2ea9wyjvh8b-master-0:23456"
          },
          {
            "name": "PET_RDZV_BACKEND",
            "value": "c10d"
          },
          {
            "name": "PET_NNODES",
            "value": "2:2"
          },
          {
            "name": "PET_MAX_RESTARTS",
            "value": "10"
          },
          {
            "name": "PET_LOCAL_ADDR",
            "value": "job-s2ea9wyjvh8b-master-0"
          }
        ],
        "finishedAt": "2026-03-17T04:04:55Z",
        "reason": ""
      },
      {
        "PodIP": "10.0.0.216",
        "nodeName": "10.0.0.49",
        "creationTimestamp": "2026-03-17T03:35:12Z",
        "uid": "f66f7118-6cad-46c6-af17-6d45afb8a3c2",
        "name": "job-s2ea9wyjvh8b-master-0",
        "podPhase": "Failed",
        "status": "Stopped",
        "replicaType": "master",
        "restartCount": 0,
        "envs": [
          {
            "name": "LOG_COLLECTION",
            "value": "true"
          },
          {
            "name": "AIHC_JOB_ID",
            "value": "job-s2ea9wyjvh8b"
          },
          {
            "name": "AIHC_TENSORBOARD_LOG_PATH",
```

```
            "value": ""
        },
        {
            "name": "NVIDIA_VISIBLE_DEVICES",
            "value": ""
        },
        {
            "name": "AIHC_JOB_NAME",
            "value": "ccc-copy3"
        },
        {
            "name": "BAIGE_ENTRYPOINT_DIR",
            "value": "/baige"
        },
        {
            "name": "BCCL_BUS_BW_CALCULATE_MODE",
            "value": "Agg"
        },
        {
            "name": "BCCL_PROFILING_FILE",
            "value": "/var/bccl/logs/busbw.cal.%h.%p"
        },
        {
            "name": "BCCL_ERROR_FILE",
            "value": "/var/bccl/logs/err.%h.%p.log"
        },
        {
            "name": "BCCL_TRACE_HANG_ENABLE",
            "value": "1"
        },
        {
            "name": "BCCL_HANG_DETECT_INTERVAL",
            "value": "30"
        },
        {
            "name": "BCCL_UNIX_SOCKET_PATH",
            "value": "/var/bccl/sockets"
        },
        {
            "name": "BCCL_TRACE_SLOW_ENABLE",
            "value": "1"
        },
        {
            "name": "ENABLE_FAULT_TOLERANCE",
            "value": "1"
        },
        {
            "name": "TORCH_DISABLE_SHARE_RDZV_TCP_STORE",
            "value": "1"
        },
        {
            "name": "JOB_RESOURCE_TYPE",
            "value": ""
        },
        {
            "name": "FT_AGENT_SOCKET_PATH",
            "value": "/fault-tolerance/ftagent.sock"
        },
        {
            "name": "PYTHONPATH",
            "value": "/fault-tolerance/elastic-agent/python/site-packages"
        },
        {
            "name": "NODE_NAME",
            "value": ""
        },
        {
            "name": "JOB_UID",
            "value": "0e426ffd-fb2b-4a35-a776-ad517ce9e0e0"
        },
        {
            "name": "JOB_NAME",
            "value": "job-s2ea9wyjvh8b"
        },
        {
            "name": "POD_NAME",
            "value": ""
        },
        {
            "name": "CLUSTER_ID",
            "value": "cce-k8q7qdl9"
        },
        {
            "name": "REGION",
            "value": "bj"
        },
        {
            "name": "PYTHONUNBUFFERED",
            "value": "0"
        },
        {
            "name": "MASTER_PORT",
            "value": "23456"
        },
        {
            "name": "MASTER_ADDR",
            "value": "job-s2ea9wyjvh8b-master-0"
        },
        {
            "name": "WORLD_SIZE",
```

```
                "value": "2"
              },
              {
                "name": "RANK",
                "value": "0"
              },
              {
                "name": "PET_RDZV_ENDPOINT",
                "value": "job-s2ea9wyjvh8b-master-0:23456"
              },
              {
                "name": "PET_RDZV_BACKEND",
                "value": "c10d"
              },
              {
                "name": "PET_NNODES",
                "value": "2:2"
              },
              {
                "name": "PET_MAX_RESTARTS",
                "value": "10"
              },
              {
                "name": "PET_LOCAL_ADDR",
                "value": "job-s2ea9wyjvh8b-master-0"
              }
            ],
            "finishedAt": "2026-03-17T03:49:56Z",
            "reason": ""
        }
    ],
    "jobTimeLine": [
        {
            "conditionType": "Created",
            "conditionMessage": "任务创建",
            "time": "2026-03-17T03:35:09Z"
        },
        {
            "conditionType": "Scheduled",
            "conditionMessage": "任务启动",
            "time": "2026-03-17T03:35:13Z"
        },
        {
            "conditionType": "Running",
            "conditionMessage": "任务运行",
            "time": "2026-03-17T03:35:44Z"
        },
        {
            "conditionType": "Abnormal",
            "conditionMessage": "任务异常",
            "time": "2026-03-17T03:49:56Z"
        },
        {
            "conditionType": "Restarting",
            "conditionMessage": "任务重启",
            "time": "2026-03-17T03:49:57Z"
        },
        {
            "conditionType": "Scheduled",
            "conditionMessage": "任务启动",
            "time": "2026-03-17T03:49:59Z"
        },
        {
            "conditionType": "Running",
            "conditionMessage": "任务运行",
            "time": "2026-03-17T03:50:20Z"
        },
        {
            "conditionType": "Abnormal",
            "conditionMessage": "任务异常",
            "time": "2026-03-17T04:04:55Z"
        },
        {
            "conditionType": "Restarting",
            "conditionMessage": "任务重启",
            "time": "2026-03-17T04:04:57Z"
        },
        {
            "conditionType": "Scheduled",
            "conditionMessage": "任务启动",
            "time": "2026-03-17T04:04:59Z"
        },
        {
            "conditionType": "Running",
            "conditionMessage": "任务运行",
            "time": "2026-03-17T04:05:19Z"
        }
    ]
}
```


删除训练任务

描述 删除一个训练任务。

请求结构

```
 POST ?action=DeleteJob&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域外，无其他特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求ID| |jobId|String|任务ID| |jobName|String|任务名称|

请求参数

```
{
  "jobId": "job-u6e9q83ns7om"
}
```

返回示例

```
{
  "requestId": "e2039e63-3e6a-4b3a-8d66-3e3f712da88f",
  "jobId": "job-oeqpsJhwqq3k"
}
```

更新训练任务

描述 更新训练任务相关配置，目前仅支持更新任务优先级。

请求结构

```
 POST /?action=ModifyJob&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID| |priority|String|是|Body 参数|训练任务优先级|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |jobId|String|成功请求时的返回结果，任务id| |requestId|String|请求ID，用于标译每个请求的唯一性|

请求示例

```
{
 "priority": "normal",
 "jobId":"job-yah3hc9k2wd5"
}
```

返回示例

```
{
  "requestId": "2bfb7976-59ea-487d-ba8d-8f1a465fc10d",
  "jobId": "job-yah3hc9k2wd5"
}
```

查询训练任务事件

描述 获取一个任务系统事件。

请求结构

```
 POST ?action=DescribeJobEvents&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID| |startTime|String|否|Body 参数|获取任务事件的起始时间，Unix时间戳格式，默认为任务创建时间| |endTime|String|否|Body 参数|获取任务事件的结束时间，Unix时间戳格式，默认为 now|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求ID，用于标译每个请求的唯一性| |events|List< Event>|事件列表| |total|Number|事件的总数|

请求示例

```
{
  "jobId": "job-xf3iqxrm6pyk"
}
```


返回示例


```
{
  "events": [
    {
      "reason": "SuccessfulCreatePod",
      "message": "Created pod: test-training-job-0513-2-master-0",
      "firstTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "lastTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "count": 1,
      "type": "Normal"
    },
    {
      "reason": "SuccessfulCreateService",
      "message": "Created service: test-training-job-0513-2-worker-0",
      "firstTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "lastTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "count": 1,
      "type": "Normal"
    },
    {
      "reason": "SuccessfulCreateService",
      "message": "Created service: test-training-job-0513-2-master-0",
      "firstTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "lastTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "count": 1,
      "type": "Normal"
    },
    {
      "reason": "SuccessfulCreatePod",
      "message": "Created pod: test-training-job-0513-2-worker-0",
      "firstTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "lastTimestamp": "2025-05-13 02:22:40 +0000 UTC",
      "count": 1,
      "type": "Normal"
    }
  ],
  "requestId": "570c5cf5-1c1f-4a7c-bc11-875bc757541f",
  "total": 4
}
```


查询训练任务日志

描述 获取一个任务中某个pod的日志。

请求结构

```
 POST ?action=DescribeJobLogs&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列id| |jobId|String|是|Body 参数|训练任务ID| |podName|String|是|Body 参数|训练任务节点名称| |keywords|String|否|Body 参数|日志关键字查询条件，用于筛选包含指定关键字的日志| |startTime|String|否|Body 参数|日志的起始时间，unix时间戳；未设置则返回 Pod 从启动以来的所有日志。有效的时间范围为1970年到当前时间| |endTime|String|否|Body 参数|日志的结束时间，unix时间戳；未设置则返回 Pod 从启动以来的所有日志。有效的时间范围为1970年到当前时间| |maxLines|String|否|Body 参数|日志的最大行数；未设置则返回 Pod 从启动以来的所有日志。| |chunkSize|String|否|Body 参数|输出日志按着chunk数进行汇聚，例如将10行日志为1条记录，默认1，表示每一行日志作为1条记录| |marker|String|否|Body 参数|日志查询标识符，配合返回值中的 nextMarker 参数使用。使用方法为：第一次请求不写 marker 参数，获取返回值，如果 nextMarker 字段不为空，则将nextMarker 的值作为 marker 参数传入，获取更多的日志，直到 nextMarker 为空|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求ID| |jobId|String|任务ID| |podName|String|Pod名称| |podUIDs|List<String>|pod id列表，容错场景包含容错pod id| |logs|List<String>|日志条目| |nextMarker|String|下次查询的日志标识符|

请求示例


```
{
  "jobId": "job-4cAchnROTINE",
  "podName":"test-training-job-0513-2-master-0"
}
```


返回示例


```
{
  "requestId": "356e255d-0a63-4352-8124-aaa1c7641547",
  "jobId": "job-4cAchnROTINE",
  "podName": "test-training-job-0513-2-master-0",
  "podUIDs": [
    "xxx"
  ]
  "logs": [
    ""
  ],
  "nextMarker": ""
}
```

查询训练任务Pod事件

描述 获取任务某个Pod的系统事件

请求结构

```
 POST ?action=DescribePodEvents&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID| |podName|String|是|Body 参数|训练任务节点名称| |startTime|String|否|Body 参数|任务pod事件的起始时间，Unix时间戳格式，默认为 Pod 创建时间| |endTime|String|否|Body 参数|任务pod事件的结束时间，Unix时间戳格式，默认为 now|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-| |requestId|String|请求ID| |events|PodEvents|成功请求时的返回结果| |total|Number|事件总数|

请求示例

```
{
  "jobId": "job-xf3iqxrm6pyk",
  "podName": "job-xf3iqxrm6pyk-worker-0"
}
```

返回示例

```
{
  "events": [],
  "requestId": "6fe43c4b-5ff5-4ae3-bca6-011815d63bd9",
  "total": 0
}
```

停止训练任务

描述 停止一个运行中的任务，目前支持PyTorchJob，MPIJob类型的任务。

请求结构

```
 POST ?action=StopJob&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-| |requestId|String|请求ID| |jobId|String|成功请求时的返回结果，任务id|

请求示例

```
{
  "jobId": "job-hokm91je2q44"
}
```

返回示例

```
{
  "requestId": "7bffb9a9-fd5b-4e63-8b4e-15e1eb199b08",
  "jobId": "job-hokm91je2q44"
}
```

查询训练任务监控

描述 查询指定任务的监控数据的指标类型

请求结构

```
 POST ?action=DescribeJobMetrics&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID| |metricType|String|是|Body 参数|查询监控数据的指标类型，取值如下： GpuUsage：GPU 使用率。 GpuMemoryUsage：GPU Memory 使用率。 CpuUsage：CPU 使用率。 MemoryUsage：Memory 使用率。 DiskReadRate：磁盘读取速率，单位为 bytes/s。 DiskWriteRate：磁盘写入速率，单位为 bytes/s。 RDMASendDataRate: rdma 发送数据速度，单位为 bytes/s。 RDMARecvDataRate: rdma 接收数据速度，单位为 bytes/s。 PCIESendDataRate: pcie 发送数据速度，单位为 bytes/s。 PCIERecvDataRate: pcie 接收数

据速度，单位为 bytes/s。 NVLinkSendDataRate: nvlink 发送数据速度，单位为 bytes/s。 NVLinkRecvDataRate: nvlink 接收数据速度，单位为 bytes/s。 GpuTemperature: gpu 温度。单位为摄氏度。 GpuPowerUsage: gpu 功率。单位为瓦w。 GpuPipeTensorUsage: gpu pipe tensor 使用率。 RDMAHealth: rdma 健康状态。 RDMASendErrorRate: rdma 发送端丢包率，单位为 个/s。 RDMARecvErrorRate: rdma 接收端丢包率，单位为 个/s。 RDMASendPacketsRate: rdma 发包率，单位为 个/s。 RDMARecvPacketsRate: rdma 收包率，单位为 个/s。 CpuTime: cpu 使用量。 MemoryAllocation: mem 使用量，单位为 bytes。| |startTime|String|否|Body 参数|可选，Unix时间戳格式，默认为任务启动时间| |endTime|String|否|Body 参数|可选，Unix时间戳格式，运行中的任务默认为当前时间，已停止的任务为任务的停止时间| |timeStep|String|否|Body 参数|返回监控数据的时间间隔，默认值是 5 分钟。| |rateInterval|String|否|Body 参数|指标变化周期频率，默认为5分钟|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |名称|类型|说明| |-|-|-| |requestId|String|请求ID| |jobId|String|任务ID| |metrics|Array of Metric|返回结果|

请求示例

```
{
  "jobId": "job-sgrt5rigvep2",
  "metricType": "MemoryUsage"
}
```

返回示例

```
{
  "requestId": "91cba5a7-378c-4b6e-b37a-49336f03c0f0",
  "jobId": "job-0hy1pdtr3r7y",
  "metrics": [
    {
      "podName": "job-0hy1pdtr3r7y-master-0",
      "metrics": [
        {
          "time": 1758628017,
          "value": "0.005723205764186557"
        }
      ]
    },
    {
      "podName": "job-0hy1pdtr3r7y-worker-0",
      "metrics": [
        {
          "time": 1758628017,
          "value": "0.005319009570690938"
        }
      ]
    }
  ]
}
```

查询训练任务所在节点列表

描述 查询训练任务所在节点的名称

请求结构

```
 POST ?action=DescribeJobNodes&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |名称|类型|说明| |-|-|-| |requestId|String|请求ID| |jobId|String|任务ID| |nodes|List<String>|任务所在的Node列表|

请求示例

```
{
  "jobId": "job-sgrt5rigvep2"
}
```

返回示例

```
{
  "requestId": "99a25b52-f37a-4b75-b86a-1e7d5b7094d6",
  "jobId": "job-sgrt5rigvep2",
  "nodes": [
    "192.168.12.xxx"
  ]
}
```

获取训练任务WebTerminal地址

描述 获取训练任务中指定容器的web Terminal

请求结构

```
 POST ?action=DescribeJobWebterminal&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必须|参数位置|说明| |-|-|-|-| |resourcePoolId|String|是|Query 参数|自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless| |queueID|String|是|Query 参数|训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id| |jobId|String|是|Body 参数|训练任务ID| |podName|String|是|Body 参数|训练任务节点名称| |handshakeTimeoutSecond|String|否|Body 参数|连接超时参数，仅在建立连接时使用，单位秒，默认值30，最小值1| |pingTimeoutSecond|String|否|Body 参数|心跳超时参数，单位秒，默认值900，最小值1，最大值3600|

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域 除公共头域，无其它特殊头域。

返回参数 |名称|类型|说明| |-|-|-| |requestId|String|请求ID| |webTerminalUrl|String|WebTerminal的连接地址|

请求示例

```
{
  "jobId": "job-0hy1pdtr3r7y",
  "podName": "job-0hy1pdtr3r7y-worker-0"
}
```

返回示例

```
{
 "requestId": "2f2d2ab7-2c8e-45fd-a562-8bbded9ef77f",
 "webTerminalUrl": "wss://cce-webshell-
bj.bce.baidu.com/api/v1/containerwebshell/establish/ppsurbh9ro763fmyx1ueaws6une8ek5mg3t3tt268ta8gmayngaf4ziw66fzu10iq1r4clpxuif547ux6p93vt98d0v24e9t5rm29qofxsti41lt1wntydhr7pkx8cyt?handshakeTimeoutSecond=30&pingTimeoutSecond=900"
}
```

批量停止训练任务

批量停止训练任务

接口描述

本接口用于批量停止训练任务，目前支持PyTorchJob，TFJob。

请求结构

```
 POST ?action=BatchStopJobs&resourcePoolId=xxxx&queueID=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必选	参数位置	描述
queueID	String	是	Query参数	训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id
resourcePoolId	String	是	Query参数	自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
jobList	List	是	Body参数	需要停止的训练任务列表，每个元素为任务对象，包含 jobId 字段
jobList[].jobId	String	是	Body参数	训练任务的唯一标识（Job ID）。不能为空，且同一次请求中的 Job ID 不允许重复。

返回状态码 成功返回200，失败见公共错误码和业务错误码

返回头域

除公共头域外，无其它特殊头域。

返回参数

参数名称	类型	描述
requestId	String	请求ID
result	StopBatchResult	批量停止结果对象

请求示例

```
{
  "jobList": [
    {
      "jobId": "job-r4sczcz7t99a"
    },
    {
      "jobId": "job-gkixx7kngksu"
    },
    {
      "jobId": "job-koephhtzwebl"
    }
  ]
}
```


返回示例


```
{
  "requestId": "636832d5-0be4-4be2-b042-38b6c082458c",
  "result": {
    "successList": [
      {
        "jobId": "job-r4sczcz7t99a",
        "success": true
      },
      {
        "jobId": "job-gkixx7kngksu",
        "success": true
      },
      {
        "jobId": "job-koephhtzwebl",
        "success": true
      }
    ],
    "failedList": [],
    "successNum": 3,
    "failedNum": 0,
    "success": true
  }
}
```


错误码

以下为训练服务领域相关的错误码，按资源池、队列、模板、任务、实例、数据集、日志下载分组。

错误码	错误描述	HTTP 状态码	说明
ResourcePoolNotFound	ResourcePool cce-xxxx belongs to account {accountId} not found	404 Not Found	资源池不存在
ResourcePoolNotReady	ResourcePool cce-xxxx not ready	412 Precondition Failed	资源池未就绪
EssentialPluginNotFound	Essential plugin {name} not found in resource pool cce-xxxx	412 Precondition Failed	必备插件未安装
EssentialPluginNotReady	Essential plugin {name} not ready in resource pool cce-xxxx	412 Precondition Failed	必备插件未就绪
EssentialPluginVersionMismatch	必备插件版本不满足要求	412 Precondition Failed	必备插件版本不匹配
QueueNotFound	Queue {queue} not found	404 Not Found	队列不存在
QueueNotSupportTrain	Queue {queue} does not support train workload	400 Bad Request	队列不支持训练任务
AITemplateNotFound	AITemplate {name} not found	404 Not Found	AI 模板不存在
JobNotFound	Job job-xxxx not found	404 Not Found	训练任务不存在
JobAlreadyExists	Job job-xxxx already exists	409 Conflict	训练任务已存在
JobStatusConflict	Job status conflict, current status is {status}	409 Conflict	任务当前状态不允许该操作
PodNotFound	Pod {podName} not found	404 Not Found	实例 Pod 不存在
PodStatusConflict	Pod status conflict	409 Conflict	实例状态冲突（如 Pod 未处于运行中）
ContainerNotFound	Container {name} not found in pod {podName}	404 Not Found	容器不存在
DatasetNotFound	Dataset {datasetId} not found	404 Not Found	数据集不存在
PolicyPathConflict	path {path} Conflict	400 Bad Request	数据源挂载路径冲突
DownloadTaskNotFound	Download task {taskId} not found	404 Not Found	日志下载任务不存在
DownloadTaskStatusConflict	Download task status conflict, current status is {status}	409 Conflict	日志下载任务状态冲突
ResourceTemplateMismatch	Resource template mismatch for queue {queue}, role {role}	400 Bad Request	资源模板与队列不匹配
TensorboardAccessDenied	系统检测到 Tensorboard 日志内容存在风险，已临时禁止访问，请联系工作人员处理	403 Forbidden	Tensorboard 内容风险，访问被禁
AlreadyExists	AlreadyExists	409 Conflict	资源已存在

在线服务部署相关接口

创建服务

接口描述

注意：创建服务挂载BOS存储，需要提前在资源池安装BOSFS，安装请联系技术支持人员。

请求结构

```
 POST ?action=CreateService
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
    "log": {
        "persistent": persistent
    },
    "instanceCount": instanceCount,
    "misc": {
        "podAnnotations": podAnnotations,
        "podLabels": podLabels,
        "gracePeriodSec": gracePeriodSec,
        "fedPodsPerIns": fedPodsPerIns
    },
    "resourcePool": {
        "resourcePoolId": resourcePoolId,
        "resourcePoolName": resourcePoolName,
        "queueName": queueName,
        "resourcePoolType": resourcePoolType
    },
    "deploy": {
        "schedule": {
            "priority": priority
        },
        "canaryStrategy": {
            "maxSurge": maxSurge,
            "maxUnavailable": maxUnavailable
        }
    },
    "name": name,
    "acceleratorType": acceleratorType,
    "workloadType": workloadType,
    "containers": [
        {
            "name": name,
            "cpus": cpus,
            "memory": memory,
            "acceleratorCount": acceleratorCount,
            "command": command,
            "ports": [
                {
                    "name": name,
                    "port": port
                }
            ],
            "envs": envs,
            "image": {
                "imageType": imageType,
                "imageUrl": imageUrl
            },
            "volumeMounts": volumeMounts,
            "startupsProbe": startupsProbe,
            "readinessProbe": readinessProbe,
            "livenessProbe": livenessProbe
        }
    ],
    "access": {
        "publicAccess": publicAccess,
        "networkType": networkType,
        "aiGateway": {
            "enableAuth": enableAuth
        },
        "blbId": "lb-agtst8xa"
    }
}
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|参数类型|是否必须|参数位置|参数说明| |-|-|-|-| |clientToken|String|否|query|clientToken保证请求的幂等性| |serviceConf| ServiceConf|是|body|服务详情|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|标识唯一请求，用于日志排查，在body返回| |serviceId|String|服务ID|

请求示例

```
 POST /?action=CreateService
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
    "log": {
        "persistent": false
    },
    "instanceCount": 1,
    "misc": {
        "podAnnotations": {
            "prometheus.io/scrape": "false"
        },
        "podLabels": {
            "scheduling.volcano.sh/group-min-member": "1"
        },
        "gracePeriodSec": 30,
        "fedPodsPerIns": 0
    },
    "resourcePool": {
```

```
  "resourceConf": {
    "resourcePoolId": "cce-7t7mqjci",
    "resourcePoolName": "aihc-pom",
    "queueName": "default",
    "resourcePoolType": ""
  },
  "deploy": {
    "schedule": {
      "priority": "high"
    },
    "canaryStrategy": {
      "maxSurge": 25,
      "maxUnavailable": 25
    }
  },
  "name": "openapi-test-3",
  "acceleratorType": "",
  "workloadType": "",
  "containers": [
    {
      "name": "custom-container",
      "cpus": 1,
      "memory": 2,
      "acceleratorCount": 0,
      "command": [ // 一定要带/bin/sh -c
        "/bin/sh",
        "-c",
        "sleep inf" # 将"sleep inf"换成自己的启动命令
      ],
      "ports": [
        {
          "name": "HTTP",
          "port": 10088
        }
      ],
      "envs": {},
      "image": {
        "imageType": 0,
        "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
      },
      "volumeMounts": [ #只使用一种则只填写一种即可，与volumns保持一致
        {
          "volumnName": "bos0vc7w305", # bos的volumnName，volumeMounts中挂载的BOS卷名称必须与volumes 中定义的名称保持一致
          "mountPath": "/mnt/aihc_model_dir", #bos的目标路径
          "readOnly": false # 是否只读
        },
        {
          "volumnName": "pfsvl4iutqd", # pfs的volumnName，volumeMounts中挂载的pfs卷名称必须与volumes 中定义的名称保持一致
          "mountPath": "/vllm-works/models", #pfs的目标路径
          "readOnly": false
        },
        {
          "mountPath": "/vllm-workspace/model/qwen", #pfs的目标路径
          "readOnly": false,
          "volumnName": "dataseti3ah7ecl"
        }
      ],
      "startupsProbe": null,
      "readinessProbe": null,
      "livenessProbe": null
    }
  ],
  "storage": {
    "shmSize": 7,
    "volumns": [
      {
        "volumeType": "bos", #bos类型挂载
        "volumnName": "bos0vc7w305", #bos的volumnName
        "pfs": null,
        "hostpath": null,
        "dataset": {},
        "bos": {
          "secret": {
            "name": "",
            "namespace": "aihc-pom"
          },
          "sourcePath": "aihc-models-bj/Qwen/Qwen3-0.6B-FP8" #bos的源路径
        }
      },
      {
        "volumeType": "pfs", #pfs类型挂载
        "volumnName": "pfsvl4iutqd",  #pfs的volumnName
        "pfs": {
          "instanceId": "pfs-DjKrQp", #pfs实例ID，即资源池页面中存储信息项的PFS ID
          "instanceType": "",
          "hostMountPath": "",
          "mountTargetId": [],
          "clusterIP": "",
          "clientID": "pfs-DjKrQp", #pfs实例ID，即资源池页面中存储信息项的PFS ID
          "clusterPort": "",
          "sourcePath": "/mnt/model" #pfs的源路径
        },
        "hostpath": null
      },
      {
        "volumeType": "dataset", // 数据集挂载
        "dataset": {
          "datasetId": "ds-YJ1IUXel", // 数据集ID
          "datasetName": "",
          "pfs": {
```

```
                    "instanceId": "pfs-Eo2rEd", // PFS ID
                    "instanceType": "plusl2", // pfs存储介质版本，plusl2:极速型L2
                    "sourcePath": "/mnt/models" # pfs的源路径
                },
                "source": "private", #私有的，需要填写
                "storageType": "pfs", #存储介质类型pfs
                "versionId": "ds-59hVnEwM" #数据集详情里面 latestVersionId
            },
            "dynamic": false,
            "volumnName": "dataseti3ah7ecl"
        }
    ]
  },
  "access": {
    "publicAccess": false,
    "networkType": "aiGateway",
    "aiGateway": {
      "enableAuth": true,
      "aigwId": "aigw-zvmg45kn",
      "aigwName": "xyz",
      "protocol": "http",
      "strategy": "round-robin",
      "version": "v2"
    },
    "blbId": "lb-agtst8xa" // blb id
  }
}
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 0ed611a9-b3bd-416c-9f7f-a05be5c2091d
Date: Fri, 16 Aug 2024 06:29:48 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
  "requestId": "0ed611a9-b3bd-416c-9f7f-a05be5c2091d",
  "serviceId": "s-rb7bcd92fe75"
}
```

拉取服务列表

接口描述 拉取服务列表。

请求结构

```
 GET ?action=DescribeServices&pageNumber=pageNumber&pageSize=pageSize&orderBy=orderBy&order=order
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|参数位置|是否必须|参数说明| |-|-|-|-| |pageNumber|Integer|Query参数|否|从 1 开始| |pageSize|Integer|Query参数|否|分页大小，默认为10| |orderBy|String|Query参数|否|暂时只支持创建时间createdAt，需要和order同时出现| |order|String|Query参数|否|asc，desc，默认 desc。需要和orderby同时出现|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |services|List|服务列表| |totalCount|Integer|条目数| |requestId|String|唯一标识| |pageNumber|Integer|从 1 开始| |pageSize|Integer|分页大小，默认为10| |orderBy|String|暂时只支持创建时间createAt| |order|String|asc，desc，默认 desc。需要和orderby同时出现|

请求示例

```
 GET ?action=DescribeServices&pageNumber=1&pageSize=10&orderBy=createAt&order=desc
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 7459b092-4c7b-4a0f-9fb5-ba1ad5d33af2
Date:Wed,
23 Apr 2025 12: 58: 45 GMT
Content-Type: application/json:charset=UTF-8
Server: Service
{
  "order": "desc",
  "orderBy": "createdAt",
  "pageNumber": 1,
  "pageSize": 10,
  "requestId": "7459b092-4c7b-4a0f-9fb5-ba1ad5d33af2",
  "services": [
    {
      "createdAt": 1745412284,
      "hpa": {},
      "id": "aihcmdl-rc331ce73f91",
      "name": "auto-qgx--20uoo",
      "networkType": "aiGateway",
      "publicAccess": true,
      "queueName": "default",
      "region": "bj",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourceSpec": {
```

```
      "resourceSpec": {
        "acceleratorCount": 1,
        "acceleratorType": "baidu.com/l20_cgpu",
        "cpus": 12,
        "memory": 35
      },
      "updatedAt": 1745412284
    },
    {
      "createdAt": 1745411842,
      "creator": "openapi",
      "hpa": {},
      "id": "s-r7d45dcffa0a",
      "name": "openapi-test-4",
      "networkType": "aiGateway",
      "queueName": "default",
      "region": "bj",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourceSpec": {
        "cpus": 1,
        "memory": 2
      },
      "updatedAt": 1745411842
    },
    {
      "createdAt": 1745411808,
      "hpa": {},
      "id": "aihcmdl-rfcf77508835",
      "name": "auto-96bgwhb06c",
      "networkType": "aiGateway",
      "publicAccess": true,
      "queueName": "default",
      "region": "bj",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourceSpec": {
        "acceleratorCount": 1,
        "acceleratorType": "baidu.com/l20_cgpu",
        "cpus": 12,
        "memory": 35
      },
      "updatedAt": 1745411808
    },
    {
      "createdAt": 1745411275,
      "hpa": {},
      "id": "aihcmdl-r9809fbec793",
      "name": "auto-v0fwin19gb",
      "networkType": "aiGateway",
      "publicAccess": true,
      "queueName": "default",
      "region": "bj",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourceSpec": {
        "acceleratorCount": 1,
        "acceleratorType": "baidu.com/l20_cgpu",
        "cpus": 16,
        "memory": 64
      },
      "updatedAt": 1745411275
    },
    {
      "createdAt": 1745411131,
      "hpa": {},
      "id": "aihcmdl-r85ff1deb349",
      "name": "auto-qu1u5l9thv",
      "networkType": "aiGateway",
      "publicAccess": true,
      "queueName": "default",
      "region": "bj",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourceSpec": {
        "acceleratorCount": 1,
        "acceleratorType": "baidu.com/l20_cgpu",
        "cpus": 16,
        "memory": 64
      },
      "updatedAt": 1745411131
    },
    {
      "createdAt": 1745410603,
      "creator": "openapi",
      "hpa": {},
      "id": "s-rb7bcd92fe75",
      "name": "openapi-test-3",
      "networkType": "aiGateway",
      "queueName": "default",
      "region": "bj",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourceSpec": {
        "cpus": 1,
        "memory": 2
      },
      "updatedAt": 1745410603
    },
    {
```

```
            "createdAt": 1745410028,
            "hpa": {},
            "id": "aihcmdl-rc8697ce5771",
            "name": "auto-tcb9m19d02",
            "networkType": "aiGateway",
            "publicAccess": true,
            "queueName": "default",
            "region": "bj",
            "resourcePoolId": "cce-7t7mqjci",
            "resourcePoolName": "aihc-pom",
            "resourceSpec": {
                "acceleratorCount": 1,
                "acceleratorType": "baidu.com/l20_cgpu",
                "cpus": 8,
                "memory": 16
            },
            "updatedAt": 1745410028
        },
        {
            "createdAt": 1745409692,
            "creator": "liufangyu03",
            "hpa": {},
            "id": "s-r140a4257acb",
            "name": "xxxx",
            "networkType": "aiGateway",
            "queueName": "default",
            "region": "bj",
            "resourcePoolId": "cce-7t7mqjci",
            "resourcePoolName": "aihc-pom",
            "resourceSpec": {
                "acceleratorCount": 4,
                "acceleratorType": "baidu.com/h800_80g_cgpu",
                "cpus": 120,
                "memory": 1000
            },
            "updatedAt": 1745409889
        },
        {
            "createdAt": 1745409660,
            "hpa": {},
            "id": "s-r026a7a78a6b",
            "name": "infer-serverless-test",
            "networkType": "aiGateway",
            "publicAccess": true,
            "queueName": "aihcq-bedausy6sn5n",
            "region": "bj",
            "resourcePoolId": "aihc-serverless",
            "resourcePoolName": " ",
            "resourcePoolType": "serverless",
            "resourceSpec": {
                "acceleratorCount": 1,
                "acceleratorType": "kunlunxin.com/xpu",
                "cpus": 4,
                "memory": 16
            },
            "updatedAt": 1745409660
        },
        {
            "createdAt": 1745408668,
            "creator": "zhangshipei",
            "hpa": {
                "metricBased": {
                    "scaleIndicators": {}
                }
            },
            "id": "s-r2139a83023f",
            "name": "aaa2",
            "networkType": "aiGateway",
            "publicAccess": true,
            "queueName": "aihcq-hjqikltkep1f",
            "region": "bj",
            "resourcePoolId": "aihc-serverless",
            "resourcePoolName": "aihc-pom",
            "resourcePoolType": "serverless",
            "resourceSpec": {
                "acceleratorCount": 1,
                "acceleratorType": "kunlunxin.com/xpu",
                "cpus": 20,
                "memory": 50
            },
            "updatedAt": 1745412862
        }
    ],
    "totalCount": 16
}
```


获取服务状态

接口描述 获取服务状态。

请求结构

```
 GET ?action=DescribeServiceStatus&serviceId=serviceId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|是|Query参数|单个服务id|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名|参数类型|参数说明| |-|-|-| |service|Map<String, ServiceBriefStat>|服务的状态，map的key对应serviceId，value为应用的状态信息| |requestId|String|唯一标识|

请求示例

```
 GET ?action=DescribeServiceStatus&serviceId=s-r7d45dcffa0a
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 0ca81095-4446-423f-976e-ca38d99e682c
Date:Wed, 23 Apr 2025 13:05:17 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "requestId": "0ca81095-4446-423f-976e-ca38d99e682c",
    "service": {
        "s-r7d45dcffa0a": {
            "availableIns": 1,
            "status": 2,
            "totalIns": 1
        }
    }
}
```

查询服务详情

接口描述 查询服务详情信息。

请求结构

```
 GET ?action=DescribeService&serviceId=serviceId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|必选|Query参数|服务ID|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名|参数类型|参数说明| |-|-|-| |service| ServiceConf|服务的详细配置信息| |status|ServiceStatus|服务网络状态信息| |instances|List<InsInfo>|实例信息| |creator|String|创建人| |createdAt|Integer|创建时间| |updatedAt|Integer|更新时间| |requestId|String|请求Id|

请求示例

```
 GET ?action=DescribeService&serviceId=s-r7d45dcffa0a
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

返回示例 返回server:Service

```
 HTTP/1.1 200 OK
x-bce-request-id: 63df3394-3edd-4977-a5d4-0d0fd021a42d
Date:Wed,
23 Apr 2025 13: 08: 46 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "createdAt": 1745411842,
    "creator": "openapi",
    "instances": [
        {
            "containers": [
                {
                    "container": {
                        "command": [
                            "/bin/sh",
                            "-c",
                            "sleep inf"
                        ],
                        "cpus": 1,
                        "image": {
                            "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
                        },
                        "memory": 2,
                        "name": "custom-container",
                        "ports": [
                            {
                                "name": "HTTP",
                                "port": 10088
                            }
                        ],
```

```
                "volumeMounts": [
                  {
                    "volumnName": "bosv2vu04u433", #bos类型挂载
                    "mountPath": "/mnt/aihc_model_dir",
                    "readOnly": false
                  },
                  {
                    "volumnName": "pfse471xhc4", #pfs类型挂载
                    "mountPath": "/vllm-works/models",
                    "readOnly": false
                  }
                ]
              },
              "status": {
                "containerStatus": "Running",
                "createdAt": 1745411851
              }
            }
          ],
          "instanceId": "s-r7d45dcffa0a-7ddffdd8db-sbrhh",
          "status": {
            "availableContainers": 1,
            "createdAt": 1745411849,
            "nodeIP": "10.0.7.38",
            "podIP": "10.0.50.96",
            "status": "Running",
            "totalContainers": 1
          }
        }
      ],
      "requestId": "63df3394-3edd-4977-a5d4-0d0fd021a42d",
      "service": {
        "access": {
          "aiGateway": {
            "enableAuth": true,
            "aigwId": "aigw-6akegg6n",
            "aigwName": "tmp",
            "protocol": "http",
            "strategy": "round-robin",
            "version": "v2"
          },
          "blbId": "",
          "networkType": "aiGateway"
        },
        "containers": [
          {
            "command": [
              "/bin/sh",
              "-c",
              "sleep inf"
            ],
            "cpus": 1,
            "image": {
              "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
            },
            "memory": 2,
            "name": "custom-container",
            "ports": [
              {
                "name": "HTTP",
                "port": 10088
              }
            ],
            "volumeMounts": [
              {
                "volumnName": "bosv2vu04u433", #bos类型挂载
                "mountPath": "/mnt/aihc_model_dir",
                "readOnly": false
              },
              {
                "volumnName": "pfse471xhc4", #pfs类型挂载
                "mountPath": "/vllm-works/models",
                "readOnly": false
              }
            ]
          }
        ],
        "deploy": {
          "canaryStrategy": {
            "maxSurge": 25,
            "maxUnavailable": 25
          },
          "schedule": {
            "priority": "high"
          }
        },
        "hpa": {},
        "instanceCount": 1,
        "log": {},
        "misc": {
          "gracePeriodSec": 30,
          "podAnnotations": {
            "prometheus.io/scrape": "false"
          },
          "podLabels": {
            "scheduling.volcano.sh/group-min-member": "1"
          }
        },
        "name": "openapi-test-4",
        "resourcePool": {
```

```
            "queueName": "default",
            "resourcePoolId": "cce-7t7mqjci",
            "resourcePoolName": "aihc-pom"
        },
        "storage": { // 存储挂载
            "shmSize": 7,
            "volumns": [
                {
                    "volumeType": "bos", #bos类型挂载
                    "volumnName": "bosv2vu04u433",
                    "pfs": null,
                    "hostpath": {
                        "sourcePath": ""
                    },
                    "dataset": {},
                    "bos": {
                        "secret": {
                            "name": "",
                            "namespace": "aihc-pom"
                        },
                        "sourcePath": "aihc-models-bj/Qwen/Qwen3-0.6B-FP8", #bos的源路径
                    }
                },
                {
                    "volumeType": "pfs", #pfs类型挂载
                    "volumnName": "pfse471xhc4",
                    "pfs": {
                        "instanceId": "pfs-DjKrQp",
                        "instanceType": "",
                        "hostMountPath": "",
                        "mountTargetId": null,
                        "clusterIP": "",
                        "clientID": "pfs-DjKrQp",
                        "clusterPort": "",
                        "sourcePath": "/mnt/model" #pfs的源路径
                    },
                    "hostpath": {
                        "sourcePath": ""
                    },
                    "dataset": {}
                }
            ]
        }
    },
    "status": {
        "accessIPs": {
            "external": "120.48.101.59",
            "internal": "10.0.7.54"
        },
        "accessPorts": [
            {
                "containerPort": 10088,
                "name": "HTTP",
                "servicePort": 10088
            }
        ],
        "aiGateway": {
            "basePath": "/s-r2678fdf80f2/",
            "tokens": {
                "serveless": "Bearer 6a8eb902-c894be1e2f4b-8b7a4475aeaa",
                "servelessId": "cs-VR3mwFBMhYE5"
            }
        },
        "blbShortId": "",
        "briefStat": {
            "availableIns": 1,
            "status": 2,
            "totalIns": 1
        }
    },
    "updatedAt": 1745411842
}
```


升级服务

接口描述 更新服务

请求结构

```
 POST ?action=ModifyService&serviceId=serviceId&description=description
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
    "log": {
        "persistent": persistent
    },
    "instanceCount": instanceCount,
    "misc": {
        "podAnnotations": podAnnotations,
        "podLabels": podLabels,
        "gracePeriodSec": gracePeriodSec,
        "fedPodsPerIns": fedPodsPerIns
    },
    "resourcePool": {
        "resourcePoolId": resourcePoolId,
        "resourcePoolName": resourcePoolName,
        "queueName": queueName,
        "resourcePoolType": resourcePoolType
    },
    "deploy": {
        "schedule": {
            "priority": priority
        },
        "canaryStrategy": {
            "maxSurge": maxSurge,
            "maxUnavailable": maxUnavailable
        }
    },
    "name": name,
    "acceleratorType": acceleratorType,
    "workloadType": workloadType,
    "containers": [
        {
            "name": name,
            "cpus": cpus,
            "memory": memory,
            "acceleratorCount": acceleratorCount,
            "command": command,
            "ports": [
                {
                    "name": name,
                    "port": port
                }
            ],
            "envs": envs,
            "image": {
                "imageType": imageType,
                "imageUrl": imageUrl
            },
            "volumeMounts": volumeMounts,
            "startupsProbe": startupsProbe,
            "readinessProbe": readinessProbe,
            "livenessProbe": livenessProbe
        }
    ],
    "access": {
        "publicAccess": publicAccess,
        "networkType": networkType,
        "aiGateway": {
            "enableAuth": enableAuth
        }
    }
}
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-| |serviceId|String|是|Query参数|query 参数| |description|String|否|Query参数|更新描述| |serviceConf| ServiceConf|是|body参数|与创建参数一致，但不允许修改网络配置|

其中ServceConf中支持修改的参数如下：

参数名称	参数类型	是否必须	参数位置	参数说明
acceleratorType	String	是	Body参数	加速芯片类型
instanceCount	Integer	是	Body参数	部署实例数，取值>=0
storage	StorageConf	否	Body参数	存储卷、共享内存配置
containers	List<ContainerConf>	是	Body参数	服务容器信息，最少需要1个，最多10个，见containers结构
log	LogConf	否	Body参数	日志配置
deploy	DeployConf	否	Body参数	部署配置，默认为最大超量和最大不可用均为25%
misc	Misc	否	Body参数	实例label、annotations配置

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-| |serviceId|String|服务ID| |requestId|String|请求唯一标识|

请求示例

```
 POST ?action=ModifyService&serviceId=s-r7d45dcffa0a&description=two instance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
{
    "access": {
      "aiGateway": {
        "enableAuth": true
      },
      "networkType": "aiGateway",
      "publicAccess": false
    },
    "name": "openapi-test-4",
    "acceleratorType": "",
    "containers": [
      {
        "name": "custom-container",
        "cpus": 1,
        "memory": 2,
        "acceleratorCount": 0,
        "command": ["/bin/sh", "-c", "sleep inf"],
        "ports": [
          {
            "name": "HTTP",
            "port": 10088
          }
        ],
        "envs": {},
        "image": {
          "imageType": 0,
          "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
        },
        "volumeMounts": [],
        "startupsProbe": null,
        "readinessProbe": null,
        "livenessProbe": null
      }
    ],
    "deploy": {
      "canaryStrategy": {
        "maxSurge": 25,
        "maxUnavailable": 25
      },
      "schedule": {
        "priority": "high"
      }
    },
    "instanceCount": 2,
    "log": {
      "persistent": false
    },
    "misc": {
      "fedPodsPerIns": 0,
      "gracePeriodSec": 30,
      "pdMode": null,
      "podAnnotations": {
        "prometheus.io/scrape": "false"
      },
      "podLabels": {
        "scheduling.volcano.sh/group-min-member": "1"
      }
    },
    "resourcePool": {
      "queueName": "default",
      "resourcePoolId": "cce-7t7mqjci",
      "resourcePoolName": "aihc-pom",
      "resourcePoolType": ""
    },
    "workloadType": ""
}
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 7020b540-5710-454a-bc52-0222831c0a6
Date:Wed, 23 Apr 2025 13:41:53 GMT
Content-Type: application/json;charset=UTF-8
Server:Service
{
    "requestId": "7020b540-5710-454a-bc52-0222831c0a6f",
    "serviceId": "s-r7d45dcffa0a"

}
```

服务扩缩容

接口描述 服务扩缩容

请求结构

```
 POST ?action=ModifyServiceReplicas&instanceCount=instanceCount
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|是|Query参数|操作对象| |instanceCount|Integer|是|Query参数|目标实例数量|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识|

请求示例

```
 POST ?action=ModifyServiceReplicas&serviceId=s-r7d45dcffa0a&insCount=3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 93070e5c-f30c-4b7f-b728-1fdc398026c7
Date:Wed, 23 Apr 2025 21:51:45 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "requestId": "93070e5c-f30c-4b7f-b728-1fdc398026c7"
}
```

配置公网访问

接口描述 配置公网访问

请求结构

```
 POST ?action=ModifyServiceNetConfig&publicAccess=publicAccess&eip=eip
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|是|Query参数|服务ID| |publicAccess|Boolean|是|Query参数|是否开启公网，ture开启，false关闭| |eip|String|否|Query参数|未使用云原生网关（networkType=""）开启公网时必填，填入未绑定的EIP IP，如需更换EIP，需要先关闭公网，再选新的EIP。开启云原生网关（networkType=aiGateway）自动分配eip，指定不生效|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识|

请求示例

```
 POST ?action=ModifyServiceNetConfig&serviceId=s-f9c14dcf3da2&publicAccess=true&eip=180.76.177.176
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: ee48ef22-a66d-4063-a011-a1d7d5dec59e
Date:Thu, 24 Apr 2025 11:40:09 GMT
Content-Type: application/json;charset=UTF-8
Server:Service
{
    "requestId": "ee48ef22-a66d-4063-a011-a1d7d5dec59e"
}
```

拉取服务变更记录

接口描述 拉取服务变更记录

请求结构

```
 GET ?action=DescribeServiceChangelogs&serviceId=serviceId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|是|Query参数|服务ID| |changeType|Integer|否|Query参数|默认全部类型，1:创建 2:更新 3:扩缩容 4:流量接入变更| |order|String|否|Query参数|asc，desc，默认 desc。需要和orderBy同时出现| |orderBy|String|否|Query参数|暂时只支持createdAt，需要和order同时出现| |pageNumber|Integer|否|Query参数|默认为1| |pageSize|Integer|否|Query参数|分页大小，默认为10|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识| |changeLogs|List<ServiceChangeRecord>|记录列表| |totalCount|Integer|该服务变更数| |pageNumber|Integer|从 1 开始| |pageSize|Integer|分页大小，默认为10| |orderBy|String|暂时只支持创建时间createdAt| |order|String|asc，desc，默认 desc。需要和orderby同时出现|

请求示例

```
 GET ?action=DescribeServiceChangelogs&serviceId=s-f9c14dcf3da2
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 24ede897-b0e1-4f08-b543-f903e2fe8d02
Date:Thu,
24 Apr 2025 15: 01: 50 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "changeLogs": [
        {
            "changeId": "ch-815bd9505339",
            "changeType": 4,
            "createdAt": 1745466679,
            "creator": "openapi",
            "prev": "ch-387bdbbc44e7"
        },
        {
            "changeId": "ch-387bdbbc44e7",
            "changeType": 4,
            "createdAt": 1745466548,
            "creator": "openapi",
            "prev": "ch-e1a21840dda0"
        },
        {
            "changeId": "ch-e1a21840dda0",
            "changeType": 4,
            "createdAt": 1745466318,
            "creator": "openapi",
            "prev": "ch-6a36f575c761"
        },
        {
            "changeId": "ch-6a36f575c761",
            "changeType": 4,
            "createdAt": 1745466008,
            "creator": "openapi",
            "prev": "ch-fba5613c902e"
        },
        {
            "changeId": "ch-fba5613c902e",
            "changeType": 1,
            "createdAt": 1745465584,
            "creator": "openapi"
        }
    ],
    "order": "",
    "orderBy": "",
    "pageNumber": 1,
    "pageSize": 10,
    "requestId": "24ede897-b0e1-4f08-b543-f903e2fe8d02",
    "totalCount": 5
}
```

查询服务变更详情

接口描述 查询服务变更详情

请求结构

```
 GET ?action=DescribeServiceChangelog&changeId=changeId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |changeId|String|是|Query参数|变更 ID|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名|参数类型|参数说明| |-|-|-| |changeId|String|变更ID| |prev|String|前一条变更记录| |changeType|Integer|变更类型，1:创建 2:更新 3:扩缩容 4:流量接入变更| |description|String|变更简短描述| |creator|String|操作人| |createdAt|Integer|变更创建时间| |requestId|String|请求唯一标识|

请求示例

```
 GET ?action=DescribeServiceChangelog&changeId=ch-815bd9505339
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Fri, 16 Aug 2024 06:29:48 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS
{
    "changeId": "ch-815bd9505339",
    "changeType": 4,
    "createdAt": 1745466679,
    "creator": "openapi",
    "description": "",
    "prev": "ch-387bdbbc44e7",
    "requestId": "626d61a5-bea5-4d41-9363-ff1d8f1e05ff"
}
```

删除服务

接口描述 删除服务

请求结构

```
 POST ?action=DeleteService
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-| |serviceId|String|是|Query参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-| |requestId|String|请求唯一标识|

请求示例

```
 POST ?action=DeleteService&serviceId=s-r35255397dbf
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date: Fri, 16 Aug 2024 06:29:48 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "requestId": "1214cca7-4ad5-451d-9215-71cb844c0a50"
}
```

拉取服务pod列表

接口描述 拉取服务pod列表

请求结构

```
 GET ?action=DescribeServicePods&serviceId=serviceId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-| |serviceId|String|是|Query参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-| |requestId|String|请求唯一标识| |pods|List< InsInfo>|实例列表|

请求示例

```
 GET ?action=DescribeServicePods&serviceId=s-f9c14dcf3da2
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date:Thu,
24 Apr 2025 15: 27: 43 GMT
Content-Type: application/json:charset=UTF-8
Server: Service
{
    "pods": [
        {
            "containers": [
                {
                    "container": {
                        "command": [
                            "/bin/sh",
                            "-c",
                            "sleep inf"
                        ],
                        "cpus": 1,
                        "image": {
                            "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
                        },
                        "memory": 2,
                        "name": "custom-container",
                        "ports": [
                            {
                                "name": "HTTP",
                                "port": 10089
                            }
                        ]
                    },
                    "status": {
                        "containerStatus": "Running",
                        "createdAt": 1745465654
                    }
                }
            ],
            "instanceId": "s-f9c14dcf3da2-595b6bcc54-7jms9-0",
            "status": {
                "availableContainers": 1,
                "createdAt": 1745465590,
                "nodeIP": "10.0.7.38",
                "podIP": "10.0.50.97",
                "status": "Running",
                "totalContainers": 1
            }
        },
        {
            "containers": [
                {
                    "container": {
                        "command": [
                            "/bin/sh",
                            "-c",
                            "sleep inf"
                        ],
                        "cpus": 1,
                        "image": {
                            "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
                        },
                        "memory": 2,
                        "name": "custom-container",
                        "ports": [
                            {
                                "name": "HTTP",
                                "port": 10089
                            }
                        ]
                    },
                    "status": {
                        "containerStatus": "Running",
                        "createdAt": 1745465672
                    }
                }
            ],
            "instanceId": "s-f9c14dcf3da2-595b6bcc54-7jms9-1",
            "status": {
                "availableContainers": 1,
                "createdAt": 1745465590,
                "nodeIP": "10.0.7.38",
                "podIP": "10.0.50.129",
                "status": "Running",
                "totalContainers": 1
            }
        }
    ],
    "requestId": "f639e7a2-e596-4bee-97bb-9e27b265155d"
}
```

摘除Pod流量

接口描述 摘除Pod流量

请求结构

```
 POST ?action=DisableServicePod&serviceId=serviceId&instanceId=instanceId&block=true
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|是|Query参数|服务ID| |instanceId|String|是|Query参数|实例ID，单机情况下是服务下某个pod的id，分布式情况下是指某个实例组的Id，该实例一般包含一个或多个pod| |block|Boolean|是|Query参数|是否封锁流量，不填默认false|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识|

请求示例

```
 POST ?action=DisableServicePod&serviceId=s-f9c14dcf3da2&instanceId=s-f9c14dcf3da2-595b6bcc54-7jms9&block=true
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 623d8716-17a3-4285-96fd-ab41e3c15d93
Date:Thu, 24 Apr 2025 15:45:43 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "requestId": "623d8716-17a3-4285-96fd-ab41e3c15d93"
}
```

删除Pod并重建

接口描述 删除Pod，并触发重建

请求结构

```
 POST ?action=DeleteServicePod&serviceId=serviceId&instanceId=instanceId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |serviceId|String|是|Query参数|服务ID| |instanceId|String|是|Query参数|实例ID,单机情况下是服务下某个pod的id，分布式情况下是指某个实例组的Id，该实例组一般包含一个或多个pod|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识|

请求示例

```
 POST ?action=DeleteServicePod&serviceId=s-f9c14dcf3da2&instanceId=s-f9c14dcf3da2-595b6bcc54-7jms9
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 26221774-5763-4a69-adcf-ce7e56b934ef
Date:Thu, 24 Apr 2025 16:01:56 GMT
Content-Type: application/json;charset=UTF-8
Server: Service
{
    "requestId": "26221774-5763-4a69-adcf-ce7e56b934ef"
}
```

获取实例组列表

接口描述 获取实例组列表

请求结构

```
 GET ?action=DescribeServicePodGroups&serviceId=serviceId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|参数位置|是否必须|参数说明| |-|-|-|-|-| |serviceId|String|Query参数|是|服务id|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识| |resourcePoolId|String|资源池ID| |resourcePoolName|String|资源池名称| |queueName|String|队列| |servicePodGroups|List< InsGroupStatus>|服务实

例组列表|

请求示例

```
GET ?action=DescribeServicePodGroups&serviceId=s-f9c14dcf3da2
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 1214cca7-4ad5-451d-9215-71cb844c0a50
Date:Thu,
24 Apr 2025 15: 37: 44 GMT
Content-Type: application/json;charset=UTF-8
Server:Service
{
    "queueName": "default",
    "requestId": "7edcaa5e-bbd8-46c9-bf3e-7bf77f3e4ecc",
    "resourcePoolId": "cce-7t7mqjci",
    "resourcePoolName": "aihc-pom",
    "servicePodGroups": [
        {
            "availablePods": 2,
            "createdAt": 1745465590,
            "id": "s-f9c14dcf3da2-595b6bcc54-7jms9",
            "masters": [
                {
                    "containers": [
                        {
                            "container": {
                                "command": [
                                    "/bin/sh",
                                    "-c",
                                    "sleep inf"
                                ],
                                "cpus": 1,
                                "image": {
                                    "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
                                },
                                "memory": 2,
                                "name": "custom-container",
                                "ports": [
                                    {
                                        "name": "HTTP",
                                        "port": 10089
                                    }
                                ]
                            },
                            "status": {
                                "containerStatus": "Running",
                                "createdAt": 1745465654
                            }
                        }
                    ],
                    "instanceId": "s-f9c14dcf3da2-595b6bcc54-7jms9-0",
                    "status": {
                        "availableContainers": 1,
                        "createdAt": 1745465590,
                        "nodeIP": "10.0.7.38",
                        "podIP": "10.0.50.97",
                        "status": "Running",
                        "totalContainers": 1
                    }
                }
            ],
            "status": "Running",
            "totalPods": 2,
            "workers": [
                {
                    "containers": [
                        {
                            "container": {
                                "command": [
                                    "/bin/sh",
                                    "-c",
                                    "sleep inf"
                                ],
                                "cpus": 1,
                                "image": {
                                    "imageUrl": "registry.baidubce.com/inference/vllm-openai:v0.8.3"
                                },
                                "memory": 2,
                                "name": "custom-container",
                                "ports": [
                                    {
                                        "name": "HTTP",
                                        "port": 10089
                                    }
                                ]
                            },
                            "status": {
                                "containerStatus": "Running",
                                "createdAt": 1745465672
                            }
                        }
```

```
                },
                "instanceId": "s-f9c14dcf3da2-595b6bcc54-7jms9-1",
                "status": {
                    "availableContainers": 1,
                    "createdAt": 1745465590,
                    "nodeIP": "10.0.7.38",
                    "podIP": "10.0.50.129",
                    "status": "Running",
                    "totalContainers": 1
                }
            }
        ]
    }
]
}
```


错误码

推理服务接口业务错误码

Code	Message	HTTP Status Code	说明
InvalidParameter	Invalid parameter	400 Bad Request	请求参数错误
NotImplemented	Api is not implemented	400 Bad Request	Api未实现
Forbidden	Operation is not allowed	403 Forbidden	访问禁止，没有对应的操作权限
NotFound	Resource could not be found	403 Forbidden	资源获取失败，操作的资源不存在
DuplicatedName	Name already exists	400 Bad Request	名称重复

查询推理服务日志

查询推理服务日志

描述

获取推理服务中实例的日志。

请求结构

```
 POST ?action=DescribeServiceLogs
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
serviceId	String	是	Body 参数	服务ID
instanceId	String	是	Body 参数	实例ID
startTime	String	否	Body 参数	日志的起始时间，RFC3339格式；例如：2026-03-20T01:40:35Z。
endTime	String	否	Body 参数	日志的结束时间，RFC3339格式；例如：2026-03-27T01:40:35Z。。
maxLines	Integer	否	Body 参数	日志的最大行数；未设置和超过100行则返回100行日志。
marker	String	否	Body 参数	日志查询标识符，配合返回值中的 nextMarker 参数使用。使用方法为：第一次请求不写 marker 参数，获取返回值，如果 nextMarker 字段不为空，则将nextMarker 的值作为 marker 参数传入，获取更多的日志，直到 nextMarker 为空

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求ID
logs	List	日志条目
nextMarker	String	下次查询的日志标识符

请求示例

```
{
  "serviceId": "s-re4267be4f9b",
  "instanceId": "s-re4267be4f9b-7cb86bcd7f-wm5f5", #实例列表中实例ID
  "startTime": "2026-02-11T03:42:51Z",
  "endTime": "2026-02-11T09:20:32Z",
  "maxLines": 100
}
```

返回示例

```
{
  "requestId": "62830339-81ef-479e-83b2-e1ec74487622"
  "logs": [
    ""2026-03-23T19:41:57.29046408+08:00 stdout F \u001b[1;36m(APIServer pid=40)\u001b[0;0m INFO:     10.0.34.215:54292 - \"GET /metrics HTTP/1.1\" 200 OK",",
    ...
  ],
  "nextMarker": "CPmI8N_EMxD__-mQ8N_EMxgJIAE"
}
```


获取推理服务WebTerminal地址

获取推理服务WebTerminal地址

描述

获取推理服务中指定实例的web Terminal

请求结构

```
 POST ?action=DescribeServiceWebterminal
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json
X-API-Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
resourcePoolId	String	是	Body 参数	自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
queue	String	是	Body 参数	训练任务所属队列，自运维资源池须填入队列名称，托管资源池须填入队列Id
resourcePoolType	String	是	Body 参数	资源池类型,取值为""、serverless、public，空字符串为自运维资源池，serverless是全托管资源池，public预留字段暂未使用
serviceId	String	是	Body 参数	服务ID
instanceId	String	是	Body 参数	实例ID
handshakeTimeoutSecond	Integer	否	Body 参数	连接超时参数，仅在建立连接时使用，单位秒，默认值30，最小值30，最大值9999
pingTimeoutSecond	Integer	否	Body 参数	心跳超时参数，单位秒，默认值900，最小值30，最大值3600

返回头域

除公共头域，无其它特殊头域。

返回参数

名称	类型	说明
requestId	String	请求ID
WebTerminalUrl	String	WebTerminal的连接地址

请求示例

```
{
   "serviceId": "s-r6531bcd74d9",
   "instanceId": "s-r6531bcd74d9-65c9b6958f-h9gq6",
   "resourcePoolId": "aihc-serverless",
   "queue": "aihcq-21wqq2l2zris",
   "resourcePoolType":"serverless",
   "commandType": "sh"
}
```

返回示例


```
{
   "requestId": "ad719b9d-e8ab-493f-9739-f67009ff65b0",
   "webTerminalUrl": "wss://cce-webshell-bj.bce.baidu.com/api/v1/containerwebshell/establish/hizbhf7z8xdd17ipk71utdbqq3upj4cwv6nngp77rrw8a1gapqcv4esxdg93xrm5x0i3nw6w03pl0igrhil9o9xjt3cp73a9shtpxhgnmbv8k32z2p1tc113i2z544g1?handshakeTimeoutSecond=30\u0026pingTimeoutSecond=900"
}
```


创建多角色服务

接口描述

创建多角色服务（服务组），即由 prefill 和 decode 两个角色组成的 PD 分离推理服务，并由 LLM 调度器对外提供统一访问入口。

> **仅支持 serverless（全托管）资源池。** prefill 和 decode 使用的队列必须在同一个资源池里。 创建是异步的：接口返回只表示请求已受理，部署是否成功要通过「查询详情」或控制台查看各角色状态。

请求结构

```
 POST /?action=CreateGroupService
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization:authorization string
version: v2
{
  "groupName": "string",
  "aiGateway": { ... },
  "roles": [ { ... }, { ... } ],
  "llmScheduler": { ... }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	参数类型	是否必须	参数位置	参数说明
clientToken	String	否	query	保证请求幂等性。相同 clientToken 的重复请求不会重复创建，直接返回已创建的 serviceGroupId
groupName	String	是	body	服务组名称。只能由小写字母、数字和 - 组成，以小写字母开头，以小写字母或数字结尾，长度不超过 32 个字符；同一账号、同一地域下不能重名
aiGateway	AiGatewayConf	是	body	AI 网关配置
roles	Array of ServiceConf	是	body	角色列表。必须是 prefill 和 decode 两个角色
llmScheduler	LlmSchedulerConf	是	body	LLM 调度器配置。

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	唯一标识请求，用于日志排查
serviceGroupId	String	服务组 ID，后续查询、删除、鉴权开关都用它

请求示例

下面的示例中，只有每个角色的 resourcePool.queueName 需要填资源信息。

```
 POST /?action=CreateGroupService&clientToken=3f2b6c1e-0d6a-4d7a-9a52-1c1a6a0f6e21 HTTP/1.1
Host:aihc.bj.baidubce.com
Authorization:authorization string
version: v2

{
  "groupName": "qwen3-32b-pd",
  "aiGateway": {
    "aigwId": "aigw-xxxxxxxx",
    "enableAuth": true,
    "strategy": "round-robin",
    "protocol": "http"
  },
  "roles": [
    {
      "roleName": "prefill",
      "instanceCount": 1,
      "acceleratorType": "baidu.com/a800_80g_cgpu",
      "resourcePool": { "queueName": "aihcq-xxxxxxxxxxx" },
      "storage": {
        "shmSize": 32,
        "volumns": [
          { "volumeType": "bos", "volumnName": "vn-model", "bos": { "sourcePath": "<bucket>/<模型路径>" } }
        ]
      },
      "containers": [
        {
          "name": "custom-container",
          "cpus": 16,
          "memory": 128,
          "acceleratorCount": 1,
          "command": ["/bin/sh", "-c", "<prefill 的启动命令>"],
          "ports": [{ "name": "HTTP", "port": 8000 }],
          "image": { "imageType": 1, "imageUrl": "<镜像地址>" },
          "volumeMounts": [{ "volumnName": "vn-model", "mountPath": "/mnt/model" }],
          "readinessProbe": {
            "initialDelaySeconds": 15, "timeoutSeconds": 15, "periodSeconds": 15,
            "successThreshold": 1, "failureThreshold": 3,
            "handler": { "httpGet": { "path": "/health", "port": 8000 } }
          }
        }
      ],
      "misc": { "enableRDMA": true }
    },
    {
      "roleName": "decode",
      "instanceCount": 1,
      "acceleratorType": "baidu.com/a800_80g_cgpu",
      "resourcePool": { "queueName": "aihcq-xxxxxxxxxxx" },
      "...": "decode 角色的其余配置与 prefill 类似，端口必须与 prefill 一致"
    }
  ],
  "llmScheduler": {
    "enable": true,
    "resourceSpec": { "cpus": 2, "memory": 8 },
    "replicas": 1,
    "inferEngine": "sglang",
    "policy": "pd_split",
    "pdPolicy": { "prefillPolicy": "prefix_aware", "decodePolicy": "random" }
  }
}
```

返回示例

```
 HTTP/1.1 200 OK

{
  "requestId": "0ca81095-4446-423f-976e-ca38d99e682c",
  "serviceGroupId": "g-xxxxxxxxxxx"
}
```

拉取多角色服务列表

接口描述

分页拉取当前账号、当前地域下的多角色服务（服务组）列表。

请求结构

```
 POST /?action=DescribeGroupServices
HTTP/1.1
Host:aihc.bj.baidubce.com
Authorization: authorization string
version: v2
{
  "pageNumber": 1,
  "pageSize": 10,
  "keyword": "string",
  "orderBy": "createdAt",
  "order": "desc"
}
```

这个接口是 POST，参数放在请求体里。 请求体里没填的参数，也可以放在 URL 的查询串里，两处都有时以请求体为准。

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	参数类型	是否必须	参数位置	参数说明
pageNumber	Int	否	body	页码，默认 1，必须大于等于 1
pageSize	Int	否	body	每页条数，默认 10，必须大于等于 1。实测传 100000 也被接受
keyword	String	否	body	按服务组名称模糊过滤
orderBy	String	否	body	排序字段，目前只支持 `createdAt`
order	String	否	body	排序方式，`asc` 或 `desc`。必须和 `orderBy` 同时填写或同时不填

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	唯一标识请求，用于日志排查
totalCount	Int	符合条件的服务组总数
pageNumber	Int	本次请求的页码
pageSize	Int	本次请求的每页条数
serviceGroups	Array of ServiceGroupBrief	服务组列表

请求示例

```
 POST /?action=DescribeGroupServices
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
  "pageNumber": 1,
  "pageSize": 10,
  "keyword": "qwen3",
  "orderBy": "createdAt",
  "order": "desc"
}
```

返回示例

```
 HTTP/1.1 200 OK

{
  "requestId": "174e6078-00d4-4b6f-b90f-63451cbcbe67",
  "totalCount": 1,
  "pageNumber": 1,
  "pageSize": 10,
  "serviceGroups": [
    {
      "id": "g-xxxxxxxxxxx",
      "groupName": "qwen3-32b-pd",
      "resourcePoolId": "aihc-xxxxxxxxxxx",
      "resourcePoolType": "serverless",
      "creator": "liushuai37",
      "createdAt": 1792410249,
      "updatedAt": 1792410249
    }
  ]
}
```

查询多角色服务详情

接口描述

查询单个多角色服务（服务组）的详情，包括服务组配置、各角色的状态概要，以及 LLM 调度器的实例信息。

详情里**不包含**调用信息（访问地址、端口、鉴权 token）。调用信息请用「查询多角色服务访问信息」接口获取。

请求结构

```
 GET /?action=DescribeGroupService&serviceGroupId={serviceGroupId}
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	参数类型	是否必须	参数位置	参数说明
serviceGroupId	String	是	query	服务组 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	唯一标识请求，用于日志排查
serviceGroup	ServiceGroupDetail	服务组详情

请求示例

```
 GET /?action=DescribeGroupService&serviceGroupId=g-xxxxxxxxxxx
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

返回示例

```
 HTTP/1.1 200 OK

{
  "requestId": "...",
  "serviceGroup": {
    "serviceGroupId": "g-xxxxxxxxxxx",
    "groupName": "qwen3-32b-pd",
    "creator": "liushuai37",
    "createdAt": 1792410249,
    "updatedAt": 1792410249,
    "resourcePool": {
      "resourcePoolId": "aihc-xxxxxxxxxxx",
      "resourcePoolType": "serverless",
      "resourcePoolName": "",
      "queueName": ""
    },
    "aiGateway": {
      "aigwId": "aigw-xxxxxxxx",
      "aigwName": "infer-model",
      "enableAuth": true,
      "protocol": "http",
      "strategy": "round-robin",
      "version": "v2"
    },
    "llmScheduler": {
      "enable": true,
      "inferEngine": "sglang",
      "policy": "pd_split",
      "pdPolicy": { "prefillPolicy": "prefix_aware", "decodePolicy": "random" },
      "replicas": 1,
      "resourceSpec": { "cpus": 2, "memory": 8 }
    },
    "roles": [
      {
        "id": "s-xxxxxxxxxxx",
        "roleName": "prefill",
        "resourceSpec": { "cpus": 16, "memory": 128, "acceleratorCount": 1, "acceleratorType": "baidu.com/a800_80g_cgpu" },
        "briefStat": { "status": 2, "availableIns": 1, "totalIns": 1, "reason": "" },
        "createdAt": 1792410249,
        "updatedAt": 1792410607
      },
      { "id": "s-yyyyyyyyyyy", "roleName": "decode", "...": "..." }
    ],
    "llmSchedulerInstances": [ { "instanceId": "...", "status": { "...": "..." }, "containers": [ "..." ] } ],
    "publicModel": { "enable": false, "engine": "", "modelId": "", "scenario": "" }
  }
}
```

删除多角色服务

接口描述

删除多角色服务（服务组），同时删除组内的所有角色服务和 LLM 调度器，并清理网关上为该服务组创建的路由和消费者。

删除的特点：

- 服务组本身是同步删除的：接口返回成功后，再查询列表、访问信息会立即得到「不存在」。各角色服务的资源清理由后台异步完成。
- 同名服务组上有别的操作正在进行时（例如刚创建完、后台还在部署），删除会返回 `503 ResourceBusy`，等几秒后重试即可。
- 已经删除的服务组再次删除，返回 `404 NotFound`。
- 已经关联了流量策略的服务组不允许删除，需要先在控制台解除关联。后端代码会拒绝这种删除，但这个场景没有实测，返回的错误码未验证。

请求结构

```
 POST /?action=DeleteGroupService&serviceGroupId={serviceGroupId}
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	参数类型	是否必须	参数位置	参数说明
serviceGroupId	String	是	query	服务组 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	唯一标识请求，用于日志排查

请求示例

```
 POST /?action=DeleteGroupService&serviceGroupId=g-xxxxxxxxxxx
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

返回示例

```
 HTTP/1.1 200 OK

{
  "requestId": "f78ab14f-c970-4c90-b7ee-5b75fa636608"
}
```

查询多角色服务访问信息

接口描述

查询多角色服务（服务组）的访问信息，包括访问 IP、端口、网关路径前缀、鉴权 token 和鉴权开关状态。调用服务所需的信息都在这个接口里，「查询详情」不返回这些内容。

返回里包含鉴权 token，请妥善保管，不要写进日志或提交到代码库。

请求结构

```
 GET /?action=DescribeGroupServiceAccess&serviceGroupId={serviceGroupId}
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	参数类型	是否必须	参数位置	参数说明
serviceGroupId	String	是	query	服务组 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	唯一标识请求，用于日志排查
enableAuth	Bool	是否开启鉴权
accessIPs	AccessIPConf	访问 IP：external（外网）、internal（内网）
accessPorts	Array of AccessPortConf	访问端口
aiGateway	AIGateWay	网关访问信息

请求示例

```
 GET /?action=DescribeGroupServiceAccess&serviceGroupId=g-xxxxxxxxxxx
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

返回示例

```
 HTTP/1.1 200 OK

{
  "requestId": "ad2b801e-e2c6-4bbe-bafd-f897a38cdc95",
  "enableAuth": true,
  "accessIPs": {
    "external": "x.x.x.x",
    "internal": "10.x.x.x"
  },
  "accessPorts": [
    { "name": "llm-scheduler", "containerPort": 8000, "servicePort": 8000 }
  ],
  "aiGateway": {
    "basePath": "/g-xxxxxxxxxxx/",
    "tokens": {
      "serveless": "Bearer xxxxxxxx-xxxxxxxxxxx-xxxxxxxxxxx",
      "servelessId": "cs-xxxxxxxxxxx"
    }
  },
  "domains": null
}
```

开发机相关接口

创建开发机

接口描述 创建新的开发机

请求结构

```
 POST ?action=CreateDevInstance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string

{
    "name": name,
    "conf": {
        "resourcePool": {
            "resourcePoolType": resourcePoolType,
            "resourcePoolId": resourcePoolId,
            "resourcePoolName": resourcePoolName,
            "queueName": queueName
        },
        "resources": {
            "acceleratorType": acceleratorType,
            "acceleratorCount": acceleratorCount,
            "cpus": cpus,
            "memory": memory,
            "shmSize":shmSize,
        },
        "image": {
            "imageType": imageType,
            "imageUrl": imageUrl,
            "username": username,
            "password": password
        },
        "access": {
            "blbId": blbId,
            "sshEnable": sshEnable,
            "sshRSAPubKey": sshRSAPubKey,
            "portInfo": [{
                "name": name,
                "port": port ,
                "accessPort": accessPort
            }]
        },
        "scheduleConf": {
            "cpuNodeAffinity": cpuNodeAffinity ,
            "priority": priority
        },
        "envs": envs,
        "workspaceDir": workspaceDir,
        "startCmd": startCmd,
        "volumnConfs": [
            {
                "volumnType": volumnType,
                "pfs": {
                    "instanceId": instanceId,
                    "sourcePath": sourcePath
                },
                "dataset": {
                    "datasetId": datasetId,
                    "versionId": versionId,
                    "source": source,
                    "storageType": storageType,
                    "bos": {
                        "srcPath": srcPath,
                        "cacheLimitSize": cacheLimitSize
                    },
                    "pfs": {
                        "clientID": clientID,
                        "clusterPort": clusterPort,
                        "pfsParentDir": pfsParentDir,
                        "pfsPath": pfsPath,
                        "instanceType": instanceType,
                        "region": region,
                        "instanceId": instanceId
```

```
                    "instanceId": instanceId,
                    "clusterIP": clusterIP,
                    "mountTargetId": mountTargetId,
                    "hostMountPath": hostMountPath,
                    "srcPath": srcPath
                },
                "mountPath": mountPath
            },
            "cfs": {
                "instanceId": instanceId,
                "sourcePath": sourcePath,
                "mountPoint": mountPoint
            },
            "cds": {
                "capacity": 100
            },
            "bos": {
                "sourcePath": sourcePath
            }
            "mountPath": mountPath,
            "readOnly": readOnly
        }
    ]
  },
  "visibleScope": {
    "type": type
  },
  "notify": {
    "notifyRuleId": notifyRuleId,
    "isOpen": isOpen
  },
  "isPublicMgmt": isPublicMgmt,
  "creator": creator,
  "creatorId": creatorId
}
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|参数类型|是否必须|参数位置|参数说明| |-|-|-|-| |name|string|是|Body参数|开发机名称，支持大小写字母、数字、中文以及-_ /.特殊字符，且必须以字母开头，字母或数字结尾，长度不超过 256| |conf|DevInstanceConf|是|Body参数|开发机配置信息| |visibleScope|VisibleScope|否|Body参数|开发机可见范围，默认为仅自己可见| |notify|NotifyDetail|否|Body参数|通知配置| |isPublicMgmt|Bool|否|Body参数|是否公开管理，true 为是，默认为否| |creator|String|是|Body参数|开发机所有者（创建者）| |creatorId|String|是|Body参数|开发机所有者用户ID|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstanceId|string|开发机ID|

请求示例

```
 POST ?action=CreateDevInstance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
{
    "name": "dingtest-openapi",
    "conf": {
        "resourcePool": {
            "resourcePoolType": "",
            "resourcePoolId": "cce-i8zab7dh",
            "queueName": "default",
            "resourcePoolName": "bjtest-regress-no-delete-songmingming01"
        },
        "resources": {
            "cpus": 1,
            "acceleratorCount": 0,
            "acceleratorType": "",
            "memory": 2,
            "shmSize": 0
        },
        "image": {
            "imageType": 0,
            "imageUrl": "registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04",
            "username": "",
            "password": ""
        },
        "access": {
            "sshEnable": true,
            "sshRSAPubKey": "xxx",
            "portInfo": [
                {
                    "accessPort": 3278,
                    "name": "dev_ssh",
                    "port": 22
                }
            ],
            "blbId": "lb-3f002c91"
        },
        "scheduleConf": {
            "priority": "high",
            "cpuNodeAffinity": false
        },
        "envs": {
            "ENV1": "ENV1",
            "ENV2": "ENV2"
        },
        "workspaceDir": "/root/workspace",
        "startCmd": "echo 1",
        "volumnConfs": [
            {
                "volumnType": "cds",
                "cds": {
                    "capacity": 100
                },
                "mountPath": "/.rootfs",
                "readOnly": true
            },
            {
                "dataset": {
                        "datasetId": "ds-1rVhsjPs",
                        "defaultMountPath": "",
                        "pfs": {
                                "clientID": "pfs-Eo2rEd",          //数据集详情中 storageInstance
                                "clusterIP": "-",
                                "hostMountPath": "/pfs/pfs-Eo2rEd", //PFS 详情中的 mountTarget 中的 mountPath
                                "instanceId": "pfs-DjKrQp",        //数据集详情中 storageInstance
                                "instanceType": "plusl2",          //PFS 详情中的 instanceType
                                "mountTargetId": [                 //PFS 详情中的 mountTarget 列表中的 mountTargetId
                                    "mt-QhJFfc",
                                    "mt-ITRef1",
                                    "mt-RsEUd2"
                                ],
                                "region": "",
                                "srcPath": "/Train"                //数据集详情中 storagePath
                        },
                        "storageType": "pfs",
                        "versionId": "ds-4IESdTVS"                 //数据集详情中 latestVersionId
                },
                "mountPath": "/mnt/dataset/ubbrtu",
                "readOnly": true,
                "volumnType": "dataset"
            }
        ],
        "startCmd": "xxxx",
    },
    "visibleScope": {
        "type": 1
    },
    "notify": {
        "isOpen": false,
        "notifyRuleId": "notify-3oilatp1b"
    },
    "isPublicMgmt": true,
    "creator": "dingjiachen",
    "creatorId": "1ac3b20c495f47f59ad18d0615805ac7"
}
```

返回示例

```
{
  "devInstanceId": "dv-f458ede5",
  "requestId": "daa778cc-3625-gray-cce0-bcecanarytag"
}
```

变更配置

接口描述 变更开发机配置

请求结构

```
 POST ?action=ModifyDevInstance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
    "name": name,
    "id": id,
    "conf": {
        "resources": {
            "acceleratorType": acceleratorType,
            "acceleratorCount": acceleratorCount,
            "cpu": cpus,
            "memory": memory,
            "shmSize":shmSize
        },
        "access": {
            "blbId": blbId,
            "sshEnable": sshEnable,
            "sshRSAPubKey": sshRSAPubKey,
            "portInfo": [{
                "name": name,
                "port": port,
                "accessPort": accessPort
            }]
        },
        "scheduleConf": {
            "priority": priority,
            "cpuNodeAffinity": cpuNodeAffinity
        },
        "envs": envs,
        "workspaceDir": workspaceDir,
        "startCmd": startCmd,
        "image": {
            "imageType": imageType,
            "imageUrl": imageUrl,
            "username": username,
            "password": password
        },
        "volumnConfs": [
            {
                "volumnType": volumnType,
                "pfs": {
                    "instanceId": instanceId,
                    "sourcePath": sourcePath
                },
                "dataset": {
                    "datasetId": datasetId,
                    "versionId": versionId,
                    "source": source,
                    "storageType": storageType,
                    "bos": {
                        "srcPath": srcPath,
                        "cacheLimitSize": cacheLimitSize
                    },
                    "pfs": {
                        "clientID": clientID,
                        "clusterPort": clusterPort,
                        "pfsParentDir": pfsParentDir,
                        "pfsPath": pfsPath,
                        "instanceType": instanceType,
                        "region": region,
                        "instanceId": instanceId,
                        "clusterIP": clusterIP,
                        "mountTargetId": mountTargetId,
                        "hostMountPath": hostMountPath,
                        "srcPath": srcPath
                    },
                    "mountPath": mountPath
                },
                "cfs": {
                    "instanceId": instanceId,
                    "sourcePath": sourcePath,
                    "mountPoint": mountPoint
                },
                "cds": {
                    "capacity": capacity
                },
                "bos": {
                    "sourcePath": sourcePath
                }
                "mountPath": mountPath,
                "readOnly": readOnly
            }
        ]
    },
    "visibleScope": {
        "type": type
    },
    "notify": {
        "notifyRuleId": notifyRuleId,
        "isOpen": isOpen
    },
    "isPublicMgmt": isPublicMgmt,
    "creator": creator,
    "creatorId": creatorId
}
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |name|String|是|Body参数|开发机名称（需满足正则表达式^[a-z][a-z0-9-]*[a-z0-9]$，长度不高于32 ；不可修改）| |id|String|是|Body参数|开发机实例ID| |conf|DevInstanceConf|是|Body参数|开发机配置信息| |visibleScope|VisibleScope|否|Body参数|开发机可见范围。 | |notify|NotifyDetail|否|Body参数|通知配置| |isPublicMgmt|Bool|否|Body参数|是否公开管理，true为是| |creator|String|是|Body参数|开发机所有者| |creatorId|String|是|Body参数|开发机所有者用户ID|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstanceId|string|开发机ID|

请求示例

```
 POST ?action=ModifyDevInstance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
{
    "name": "dingtest-openapi",
    "id": "dv-64706ff6",
    "conf": {
        "resourcePool": {
            "resourcePoolType": "",
            "resourcePoolId": "cce-i8zab7dh",
            "queueName": "default",
            "resourcePoolName": "bjtest-regress-no-delete-songmingming01"
        },
        "resources": {
            "cpus": 2,
            "acceleratorCount": 0,
            "acceleratorType": "",
            "memory": 2,
            "shmSize": 0
        },
        "image": {
            "imageType": 0,
            "imageUrl": "registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04",
            "username": "",
            "password": ""
        },
        "access": {
            "sshEnable": true,
            "sshRSAPubKey": "xxx",
            "portInfo": [
                {
                    "accessPort": 3278,
                    "name": "dev_ssh",
                    "port": 22
                }
            ],
            "blbId": "lb-3f002c91"
        },
        "scheduleConf": {
            "priority": "high",
            "cpuNodeAffinity": false
        },
        "envs": {
            "ENV1": "ENV1",
            "ENV2": "ENV2"
        },
        "workspaceDir": "/root/workspace",
        "startCmd": "echo 1",
        "volumnConfs": [
            {
                "volumnType": "cds",
                "cds": {
                    "capacity": 100
                },
                "mountPath": "/.rootfs",
                "readOnly": true
            }
        ]
    },
    "visibleScope": {
        "type": 1
    },
    "notify": {
        "isOpen": false,
        "notifyRuleId": "notify-3oilatp1b"
    },
    "isPublicMgmt": true,
    "creator": "dingjiachen",
    "creatorId": "1ac3b20c495f47f59ad18d0615805ac7"
}
```

返回示例

```
 {
    "devInstanceId": "dv-f458ede5",
    "requestId": "daa778cc-3625-gray-cce0-bcecanarytag"
}
```

删除开发机

接口描述 删除开发机。已删除的开发机不可再次删除。

请求结构

```
 POST ?action=DeleteDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |devInstanceId|string|是|Query参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstanceId|string|开发机ID|

请求示例

```
 POST ?action=DeleteDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 {
   "devInstanceId": "dv-f458ede5",
   "requestId": "daa778cc-3625-gray-cce0-bcecanarytag"
}
```

查询开发机详情

接口描述 查询开发机详情。

请求结构

```
 GET ?action=DescribeDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |devInstanceId|String|是|Query参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstance| DevInstanceDetail|开发机详情|

请求示例

```
 GET ?action=DescribeDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
{
    "devInstance": {
        "accountId": "eca97e148cb74e9683d7b7240829d1ff",
        "conf": {
            "access": {
                "blbId": "lb-3f002c91",
                "portInfo": [],
                "sshEnable": true,
                "sshRSAPubKey": "xxx"
            },
            "image": {
                "imageType": 0,
                "imageUrl": "registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04",
                "password": "",
                "username": ""
            },
            "resourcePool": {
                "queueName": "default",
                "resourcePoolId": "cce-i8zab7dh",
                "resourcePoolName": "bjtest-regress-no-delete-songmingming01",
                "resourcePoolType": ""
            },
            "resources": {
                "acceleratorCount": 0,
                "acceleratorType": "",
                "cpus": 1,
                "memory": 2,
                "shmSize": 0
            },
            "scheduleConf": {
                "cpuNodeAffinity": true,
                "priority": "high"
            },
            "updatedAt": 0,
            "startCmd": "",
            "workspaceDir": "$HOME/workspace",
            "envs": {
                "abc": "123",
                "cde": "456"
            },
            "volumnConfs": [
                {
                    "bos": null,
                    "cds": {
                        "capacity": 100
                    },
                    "cfs": null,
                    "dataset": null,
                    "mountPath": "/.rootfs",
                    "pfs": null,
                    "readOnly": true,
                    "volumnType": "cds"
                }
            ]
        },
        "createdAt": 1747876700,
        "creator": "dingjiachen",
        "creatorId": "1ac3b20c495f47f59ad18d0615805ac7",
        "id": "dv-64706ff6",
        "loginInfo": {
            "jupyter": {
                "url": "https://console.bce.baidu.com/api/aihcbox/gray/proxy/v1/webide/dv-64706ff6/bj/jupyter"
            },
            "vscode": {
                "url": "https://console.bce.baidu.com/api/aihcbox/gray/proxy/v1/webide/dv-64706ff6/bj/vscode"
            }
        },
        "name": "dingtest-openapi",
        "notify": {
            "isOpen": false,
            "notifyRuleId": ""
        },
        "isPublicMgmt": false,
        "region": "bj",
        "serviceInstanceInfo": {
            "cdsInstanceId": "v-sgvDsFW1",
            "internalIP": "192.168.0.71",
            "nodeIP": "192.168.15.186",
            "podName": "aibox-r7b1d3715728-7bdd7cdfbd-xlhjd",
            "publicIP": "120.48.54.166",
            "serviceInstanceId": "aibox-r7b1d3715728"
        },
        "status": 2,
        "statusReason": "",
        "updatedAt": 1747906751,
        "version": "v3",
        "visibleScope": {
            "type": 1
        }
    },
    "requestId": "4c515c3f-3e5d-4524-b4db-2551b298fea6"
}
```


查询开发机列表

接口描述 查询开发机列表

请求结构

```
 GET ?action=DescribeDevInstances&pageNumber=1&pageSize=10&resourcePoolId=&queueName=&onlyMyDevs=false&status=&queryKey=devName&queryVal=
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |pageNumber|Number|可选|Query参数|分页（从 1 开始）| |pageSize|Number|可选|Query参数|分页大小（默认10）| |onlyMyDevs|String|可选|Query参数|是否只看自己开发机（value 为 "true" 则只看自己创建的，否则可以看到其他人创建的）| |resourcePoolId|String|可选|Query参数|资源池 ID （如果非空，则筛选该资源池的开发机；如果值为：serverless， 则仅显示托管资源池的开发机）| |queueName|String|可选|Query参数|队列名称（如果非空，则进行过滤）| |status|String|可选|Query参数|开发机状态| |queryKey|String|可选|Query参数|过滤的key，可用的值 devInstanceName 模糊匹配devInstanceId 精准匹配creator 模糊匹配| |queryVal|string|可选|Query参数|过滤的Value，和 valueKey 配套|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |totalCount|int|条目数| |devInstances|List< DevInstanceBriefInfo>|开发机列表|

请求示例

```
 GET ?action=DescribeDevInstances&pageNumber=1&pageSize=10&resourcePoolId=&queueName=&onlyMyDevs=false&status=&queryKey=devName&queryVal=
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例


```
{
    "devInstances": [
        {
            "accountId": "eca97e148cb74e9683d7b7240829d1ff",
            "createdAt": 1747876700,
            "creator": "dingjiachen",
            "creatorId": "1ac3b20c495f47f59ad18d0615805ac7",
            "id": "dv-64706ff6",
            "imageUrl": "registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04",
            "name": "dingtest-openapi",
            "queueName": "default",
            "region": "bj",
            "resourcePoolId": "cce-i8zab7dh",
            "resourcePoolName": "bjtest-regress-no-delete-songmingming01",
            "resources": {
                "acceleratorCount": 0,
                "acceleratorType": "",
                "cpus": 1,
                "memory": 2,
                "shmSize": 0
            },
            "status": 2,
            "statusReason": "",
            "updatedAt": 1747897205,
            "version": "v3"
        },
        {
            "accountId": "eca97e148cb74e9683d7b7240829d1ff",
            "createdAt": 1747739074,
            "creator": "fanjikang",
            "creatorId": "497d2cff4fc34f16ae29c721dcee7254",
            "id": "dv-42a0e34d",
            "imageUrl": "registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04",
            "name": "fjk-test-label",
            "queueName": "default",
            "region": "bj",
            "resourcePoolId": "cce-i8zab7dh",
            "resourcePoolName": "bjtest-regress-no-delete-songmingming01",
            "resources": {
                "acceleratorCount": 0,
                "acceleratorType": "",
                "cpus": 1,
                "memory": 2,
                "shmSize": 0
            },
            "status": 5,
            "statusReason": "",
            "updatedAt": 1747895372,
            "version": "v3"
        }
    ],
    "requestId": "1589ae2f-c4e2-4bc5-8264-1d3061e89af6",
    "totalCount": 2
}
```


查询开发机事件

接口描述 查询开发机事件。

请求结构

```
GET ?
action=DescribeDevInstanceEvents&startTime=startTime&message=message&instanceId=instanceId&clusterUuid=clusterUuid&devInstanceId=devInstanceId&queueName=queueName&tab=tab&pageNumber=pag
e
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否可选|参数说明| |-|-|-| |devInstanceId|String|必选|开发机ID| |startTime|String|必选|开始时间（格式为 YYYY-MM-DDTHH:MM:SS.sssZ；例：2025-05-18T17:12:20.761Z）| |endTime|String|必选|结束时间（格式为 YYYY-MM-DDTHH:MM:SS.sssZ；例：2025-05-13T17:12:20.761Z）| |eventType|String|可选|事件类型（Normal，Warning，不填则为全部）| |message|String|可选|筛选| |pageNumber|Number|可选|分页数（从 1 开始）| |pageSize|Number|可选|分页大小|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求ID，用于标译每个请求的唯一性| |events|List< Event >|事件列表| |totalCount|Number|事件的总数|

请求示例

```
GET ?action=DescribeDevInstanceEvents&startTime=2025-05-09T03:59:26.670Z&endTime=2025-05-16T03:59:26.671Z&message=&instanceId=aibox-r823466e46eb&devInstanceId=dv-
7ff141fa&pageNumber=1&pageSize=10
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
{
    "events": [
        {
            "count": 1,
            "firstTimestamp": 1747878370,
            "lastTimestamp": 1747878370,
            "message": "Started container main",
            "reason": "Started",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878370,
            "lastTimestamp": 1747878370,
            "message": "Successfully pulled image \"registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04\" in 378ms (378ms including waiting). Image size: 474104263 bytes.",
            "reason": "Pulled",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878370,
            "lastTimestamp": 1747878370,
            "message": "Created container main",
            "reason": "Created",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878370,
            "lastTimestamp": 1747878370,
            "message": "Pulling image \"registry.baidubce.com/inference/aibox-ubuntu:v2.0-22.04\"",
            "reason": "Pulling",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878361,
            "lastTimestamp": 1747878361,
            "message": "Started container sidecar",
            "reason": "Started",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878360,
            "lastTimestamp": 1747878360,
            "message": "Successfully pulled image \"registry.baidubce.com/inference/aibox-sidecar:v1.6-cs4.20.0-jpt3.6.5\" in 487ms (487ms including waiting). Image size: 1478994373 bytes.",
            "reason": "Pulled",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878360,
            "lastTimestamp": 1747878360,
            "message": "Created container sidecar",
            "reason": "Created",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878360,
            "lastTimestamp": 1747878360,
            "message": "Pulling image \"registry.baidubce.com/inference/aibox-sidecar:v1.6-cs4.20.0-jpt3.6.5\"",
            "reason": "Pulling",
            "type": "Normal"
        },
        {
            "count": 1,
            "firstTimestamp": 1747878359,
            "lastTimestamp": 1747878359,
            "message": "Successfully assigned aihc-pom/aibox-re4e2857afcc-56446c66bc-hx8lm to 192.168.15.186",
            "reason": "Scheduled",
            "type": "Normal"
        }
    ],
    "requestId": "2af19415-1f64-4fea-86b4-407996ce4573",
    "totalCount": 9
}
```


开启实例

接口描述 开发机开启实例（开机）。当开发机状态为：已停止实例（5）、已失败（18）时可以开启实例，其他状态无法开启实例。

请求结构

```
 POST ?action=StartDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-| |devInstanceId|string|是|Query参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstanceId|string|开发机ID|

请求示例

```
 POST ?action=StartDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 {
    "devInstanceId": "dv-f458ede5",
    "requestId": "daa778cc-3625-gray-cce0-bcecanarytag"
}
```

停止实例

接口描述 开发机停止实例（关机）。当开发机状态为：已关机（5）、停止中（11）、失败（18）、异常（19）、已删除（20）时不允许停止实例，其他状态均可。

请求结构

```
 POST ?action=StopDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |devInstanceId|string|是|Query参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstanceId|string|开发机ID|

请求示例

```
 POST ?action=StopDevInstance&devInstanceId=dv-343423e3
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
 {
    "devInstanceId": "dv-f458ede5",
    "requestId": "daa778cc-3625-gray-cce0-bcecanarytag"
}
```

定时停止实例

接口描述 可以通过此接口开启或者关闭开发机定时关机。只有运行中（status 为 3）的开发机允许定时关机。

请求结构

```
 POST ?action=TimedStopDevInstance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
    "devInstanceId": devInstanceId,
    "delaySec": delaySec,
    "enable": enable
}
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |devInstanceId|String|是|Body参数|操作对象| |delaySec|Number|否|Body参数|定时关机时间（单位秒，当 enable 为 true 时需大于 3600 秒）| |enable|Bool|否|Body参数|开启或关闭定时关机|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |devInstanceId|string|开发机ID|

请求示例

```
 POST ?action=TimedStopDevInstance
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
{
    "devInstanceId": "dv-f458ede5",
    "delaySec": 3600,
    "enable": true
}
```

返回示例

```
 {
    "devInstanceId": "dv-f458ede5",
    "requestId": "daa778cc-3625-gray-cce0-bcecanarytag"
}
```

制作开发机镜像

接口描述 制作开发机镜像，只有开发机状态为运行中（status 为 3）的时候才能制作镜像。

请求结构

```
 POST ?action=CreateDevInstanceImagePackJob
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
{
    "imageName": imageName,
    "imageTag": imageTag,
    "registry": registry,
    "namespace": namespace,
    "username": username,
    "password": password,
    "devInstanceId": devInstanceId
}
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否必须|参数位置|参数说明| |-|-|-|-|-| |imageName|string|是|Body 参数|镜像名称| |imageTag|string|是|Body 参数|镜像 tag| |registry|string|是|Body 参数|镜像仓库| |namespace|string|是|Body 参数|镜像仓库命名空间| |username|string|是|Body 参数|镜像仓库用户名| |password|string|是|Body 参数|镜像仓库密码| |devInstanceId|string|是|Body 参数|操作对象|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|string|请求唯一标识| |imagePackJobId|string|镜像制作任务ID|

请求示例

```
 POST ?action=CreateDevInstanceImagePackJob
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
{
    "imageName": "dind-ubuntu",
    "imageTag": "v1.0-22.04_liujianfei_20250516112654",
    "registry": "ccr-2hpgw1bh-vpc.cnc.bj.baidubce.com",
    "namespace": "kaifaji",
    "username": "dingjiachen",
    "password": "avc",
    "devInstanceId": "dv-7ff141fa"
}
```

返回示例

```
 {
    "imagePackJobId": "tk-8ae583c2",
    "requestId": "209f99df-dbc1-gray-cce0-bcecanarytag",
}
```

查询镜像任务详情

接口描述 查询制作镜像任务详情。

请求结构

```
 GET ?action=DescribeDevInstanceImagePackJob&devInstanceId=devInstanceId&imagePackJobId=imagePackJobId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization:authorization string
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名|参数类型|是否可选|参数说明| |-|-|-| |devInstanceId|String|必选|开发机ID| |imagePackJobId|String|必选|镜像制作任务 ID|

返回头域 除公共头域，无其它特殊头域。

返回参数 |参数名称|类型|说明| |-|-|-| |requestId|String|请求唯一标识| |devInstanceImagePackJob| ImagePackJobDetail|开发机制作镜像任务详情|

请求示例

```
 GET ?action=DescribeDevInstanceImagePackJob&devInstanceId=devInstanceId&imagePackJobId=imagePackJobId
HTTP/1.1
Host:aihc.bj.baidubce.com
version: v2
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2024-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

返回示例

```
{
    "devInstanceImagePackJob": {
        "createdAt": 1747892885,
        "devInstanceId": "dv-42a0e34d",
        "id": "tk-4f0eb964",
        "imageName": "dingtest",
        "imageTag": "dingtest-openapi-2",
        "namespace": "test",
        "registry": "ccr-2ccrtest-vpc.cnc.bj.baidubce.com",
        "status": 30,
        "updatedAt": 1747892901
    },
    "requestId": "05e631e6-fb55-4308-bfe1-e9848b8c95c1"
}
```


错误码

公共错误码

Code	Message	HTTP Status Code	说明
AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或者值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error. Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided does not exist in our records.	403 Forbidden	Access Key ID不存在
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	400 Bad Request	Authorization头域格式错误
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body格式错误。例如不符合指定的Encoding等
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配等问题，应定义更加具体的错误码，例如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 Bad Request	JSON格式不合法
InvalidVersion	The API version specified was invalid.	404 Not Found	URI的版本号不合法
OptInRequired	A subscription for the service is required.	403 Forbidden	没有开通对应的服务
PreconditionFailed	The specified If-Match header doesn't match the ETag header.	412 Precondition Failed	详见ETag的使用
RequestExpired	Request has expired. Timestamp date is XXX.	400 Bad Request	请求超时。XXX要改成x-bce-date的值。如果请求中只有Date，则需要将Date转换为本规范指定的格式。
IdempotentParameterMismatch	The request uses the same client token as a previous, but non-identical request.	403 Forbidden	clientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your Secret Access Key and signing method. Consult the service documentation for details.	400 Bad Request	Authorization头域中附带的签名和服务端验证不一致

开发机错误码

Code	Message	HTTP Status Code	说明
NotFound	Resource could not be found	403 Forbidden	资源查询不到
Forbidden	Operation is not allowed	403 Forbidden	对应操作不允许，或没有权限
InvalidParameter	Invalid parameter	400 Bad Request	参数错误

数据集相关接口

创建数据集

****接口描述**** 创建数据集，同时创建一个初始版本。

****权限说明**** 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

****请求结构****

```
 POST ?action=CreateDataset
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
name	String	是	Body参数	数据集名称支持小写字母、数字和-，必须以小写字母开头，必须以小写字母或数字结尾，长度限制1-50。
storageType	String	是	Body参数	存储类型，可选项：PFS、BOS
storageInstance	String	是	Body参数	存储实例ID，对应PFS、BOS的ID
importFormat	String	是	Body参数	导入格式 FILE：文件 FOLDER：文件夹
description	String	否	Body参数	数据集的描述
owner	String	否	Body参数	所有者，不传递时默认为创建者
visibilityScope	String	是	Body参数	可见范围 ALL_PEOPLE：所有人可见 ONLY_OWNER：仅所有者可读写 USER_GROUP：指定范围可用
visibilityUser	List<PermissionEntry>	否	Body参数	用户权限列表 [{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]
visibilityGroup	List<PermissionEntry>	否	Body参数	用户组权限列表 [{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]
initVersionEntry	DatasetVersionEntry	是	Body参数	初始版本相关信息

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-| |id|String|数据集ID|

请求示例

```
{
  "name": "test1",
  "storageType": "BOS",
  "storageInstance": "bucket1",
  "importFormat": "FOLDER",
  "description": "test dataset",
  "owner": "d1a5cf0143be4de9911342051106f70f",
  "visibilityScope": "USER_GROUP",
  "visibilityUser": [
    {
      "id": "ebcf430f84b046cca9fe1c62e3d739bc",
      "name": "lisi",
      "permission": "r"
    },
    {
      "id": "f410f7b6ee5b48a7a2bd7f3675bc19e0",
      "name": "wangwu",
      "permission": "rw"
    }
  ],
  "initVersionEntry": {
    "description": "dataset first version",
    "storagePath": "/path/to/dir",
    "mountPath": "/mnt/datasets/test1"
  }
}
```

响应示例

```
{
  "requestId": "e643706e-7491-4eba-b8cf-1529fa25653d",
  "id": "ds-ilkcJJZp"
}
```

删除数据集

接口描述 删除数据集，同时删除所有版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
 POST ?action=DeleteDataset&datasetId={datasetId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-|-| |datasetId|String|是|Query参数|数据集ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 无响应参数。

请求示例 无。

响应示例

```
{
  "requestId": "07a79925-6852-4870-a5fe-d9c560af04ca"
}
```

修改数据集

接口描述 修改数据集。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
POST ?action=ModifyDataset&datasetId={datasetId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |datasetId|String|是|Query参数|数据集ID| |name|String|否|RequestBody参数|数据集名称| |description|String|否|RequestBody参数|描述|
|visibilityScope|String|否|RequestBody参数|可见范围
ALL_PEOPLE：所有人可见
ONLY_OWNER：仅所有者可读写
USER_GROUP：指定范围可用| |visibilityUser|List<PermissionEntry>|否|RequestBody参数|用户权限列表
[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]| |visibilityGroup|List<PermissionEntry>|否|RequestBody参数|用户组权限列表
[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]|

响应头域 除公共头域外，无其它特殊头域。

响应参数 无响应参数。

请求示例

```
{
  "name": "my-dataset-1",
  "description": "my dataset",
  "visibilityScope": "ALL_PEOPLE"
}
```

响应示例

```
{
  "requestId": "91113d76-4d4d-43b7-855e-d650a60020fa"
}
```

获取数据集详情

接口描述 获取数据集详情，同时返回最新版本信息。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
GET ?action=DescribeDataset&datasetId={datasetId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |datasetId|String|是|Query参数|数据集ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-| |id|String|数据集ID| |name|String|数据集名称| |storageType|String|存储类型PFS：并行存储PFSBOS：对象存储BOS| |storageInstance|String|存储实例| |importFormat|String|导入格式FILE：文件FOLDER：文件夹| |description|String|描述| |owner|String|所有者| |ownerName|String|所有者名称| |visibilityScope|String|可见范围ALL_PEOPLE：所有人可见ONLY_OWNER：仅所有者可读写
USER_GROUP：指定范围可用| |visibilityUser|List<PermissionEntry>|用户权限列表[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]仅对管理员或Owner返回此属性|
|visibilityGroup|List<PermissionEntry>|用户组权限列表[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]仅对管理员或Owner返回此属性| |permission|String|当前用户拥有的读写权限：r：只读rw：读写| |latestVersionId|String|最新版本ID| |latestVersion|String|最新版本号| |latestVersionEntry|DatasetVersionEntry|最新版本相关信息| |createdAt|String|创建时间| |updatedAt|String|更新时间|

请求示例 无。

响应示例


```
{
  "requestId": "b44d5dc7-d987-495c-90cb-de635f508ba7",
  "id": "ds-ilkcJJZp",
  "name": "my-dataset-1",
  "storageType": "BOS",
  "storageInstance": "bucket1",
  "importFormat": "FOLDER",
  "description": "my dataset",
  "owner": "d1a5cf0143be4de9911342051106f70f",
  "ownerName": "zhangsan",
  "visibilityScope": "ALL_PEOPLE",
  "permission": "rw",
  "latestVersionId": "ds-gUgkU83Z",
  "latestVersion": "1",
  "latestVersionEntry": {
    "requestId": "b44d5dc7-d987-495c-90cb-de635f508ba7",
    "id": "ds-gUgkU83Z",
    "version": "1",
    "description": "dataset first version",
    "storagePath": "/path/to/dir",
    "mountPath": "/mnt/datasets/test1",
    "createUser": "eca97e148cb74e9683d7b7240829d1ff",
    "createUserName": "root",
    "createdAt": "2025-06-11T00:36:08Z",
    "updatedAt": "2025-06-11T01:18:29Z"
  },
  "createdAt": "2025-06-11T00:36:08Z",
  "updatedAt": "2025-06-11T01:18:29Z"
}
```


获取数据集列表

接口描述 获取数据集列表。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
GET ?action=DescribeDatasets
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |keyword|String|否|Query参数|名称关键字| |storageType|String|否|Query参数|存储类型PFS：并行存储PFSBOS：对象存储BOS| |storageInstances|String|否|Query参数|存储实例列表，英文逗号(,)分隔| |importFormat|String|否|Query参数|导入格式FILE：文件FOLDER：文件夹| |pageNumber|Integer|否|Query参数|请求分页参数，表示第几页，默认值为1| |pageSize|Integer|否|Query参数|单页结果数，不传递该参数默认返回全部|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |totalCount|Integer|数据集总数| |datasets|List<Dataset>|数据集列表|

请求示例 无。

响应示例

```
{
  "requestId": "beafc892-e82c-4d81-8a11-ec275a80995b",
  "totalCount": 12,
  "datasets": [
    {
      "id": "ds-0jaV0FAF",
      "name": "develop-dataset-ghco",
      "storageType": "PFS",
      "storageInstance": "pfs-7xWeAt",
      "importFormat": "FOLDER",
      "owner": "3025f29850b24632bafa8ba7b6849195",
      "ownerName": "zhangsan",
      "visibilityScope": "ALL_PEOPLE",
      "permission": "rw",
      "latestVersionId": "ds-9V7U45A5",
      "latestVersion": "1",
      "createdAt": "2025-06-05T16:40:00Z",
      "updatedAt": "2025-06-05T16:40:00Z"
    },
    {
      "id": "ds-TXzhsuDR",
      "name": "develop-dataset-mfee",
      "storageType": "PFS",
      "storageInstance": "pfs-7xWeAt",
      "importFormat": "FOLDER",
      "owner": "3025f29850b24632bafa8ba7b6849195",
      "ownerName": "zhangsan",
      "visibilityScope": "ALL_PEOPLE",
      "permission": "rw",
      "latestVersionId": "ds-33xclZz6",
      "latestVersion": "1",
      "createdAt": "2025-06-05T09:37:09Z",
      "updatedAt": "2025-06-05T09:37:09Z"
    }
  ]
}
```


获取数据集版本详情

****接口描述**** 获取数据集版本详情。

****权限说明**** 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

****请求结构****

```
GET ?action=DescribeDatasetVersion&datasetId={datasetId}&versionId={versionId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

****请求头域**** 除公共头域外，无其它特殊头域。

****请求参数**** |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |datasetId|String|是|Query参数|数据集ID| |versionId|String|是|Query参数|数据集版本ID|

****响应头域**** 除公共头域外，无其它特殊头域。

****响应参数**** |参数名称|类型|描述| |-|-| |id|String|数据集ID| |name|String|数据集名称| |storageType|String|存储类型PFS：并行存储PFSBOS：对象存储BOS| |storageInstance|String|存储实例| |importFormat|String|导入格式FILE：文件FOLDER：文件夹| |description|String|描述| |owner|String|所有者| |ownerName|String|所有者名称| |visibilityScope|String|可见范围ALL_PEOPLE：所有人可见ONLY_OWNER：仅所有者可读写USER_GROUP：指定范围可用| |visibilityUser|List<PermissionEntry>|用户权限列表[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]仅对管理员或Owner返回此属性| |visibilityGroup|List<PermissionEntry>|用户组权限列表[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]仅对管理员或Owner返回此属性| |permission|String|当前用户拥有的读写权限：r：只读rw：读写| |versionEntry|DatasetVersionEntry|指定版本相关信息| |createdAt|String|创建时间| |updatedAt|String|更新时间|

****请求示例**** 无。

****响应示例****

```
{
  "requestId": "d8beb7a8-f800-4554-9966-a764586ba6a0",
  "id": "ds-iIkcJJZp",
  "name": "my-dataset-1",
  "storageType": "BOS",
  "storageInstance": "bucket1",
  "importFormat": "FOLDER",
  "description": "my dataset",
  "owner": "d1a5cf0143be4de9911342051106f70f",
  "ownerName": "zhangsan",
  "visibilityScope": "ALL_PEOPLE",
  "permission": "rw",
  "versionEntry": {
    "requestId": "d8beb7a8-f800-4554-9966-a764586ba6a0",
    "id": "ds-gUgkU83Z",
    "version": "1",
    "description": "dataset first version",
    "storagePath": "/path/to/dir",
    "mountPath": "/mnt/datasets/test1",
    "createUser": "eca97e148cb74e9683d7b7240829d1ff",
    "createUserName": "root",
    "createdAt": "2025-06-11T00:36:08Z",
    "updatedAt": "2025-06-11T01:18:29Z"
  },
  "createdAt": "2025-06-11T00:36:08Z",
  "updatedAt": "2025-06-11T01:18:29Z"
}
```


获取数据集版本列表

接口描述 获取数据集版本列表。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
 GET ?action=DescribeDatasetVersions&datasetId={datasetId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |datasetId|String|是|Query参数|数据集ID| |pageNumber|Integer|否|Query参数|请求分页参数，表示第几页，默认值为1| |pageSize|Integer|否|Query参数|单页结果数，不传递该参数默认返回全部|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |totalCount|Integer|数据集版本总数| |versions|List<DatasetVersionEntry>|数据集版本列表|

请求示例 无。

响应示例


```
{
  "requestId": "f21efb29-8c93-4090-b8fc-82e2d21997c7",
  "totalCount": 2,
  "versions": [
    {
      "id": "ds-1Wi75n2k",
      "version": "2",
      "description": "new version",
      "storagePath": "/path/to/version2",
      "mountPath": "/mnt/datasets/my-dataset-1/v2",
      "createUser": "eca97e148cb74e9683d7b7240829d1ff",
      "createUserName": "root",
      "createdAt": "2025-06-11T01:40:20Z",
      "updatedAt": "2025-06-11T01:40:20Z"
    },
    {
      "id": "ds-gUgkU83Z",
      "version": "1",
      "description": "dataset first version",
      "storagePath": "/path/to/dir",
      "mountPath": "/mnt/datasets/test1",
      "createUser": "eca97e148cb74e9683d7b7240829d1ff",
      "createUserName": "root",
      "createdAt": "2025-06-11T00:36:08Z",
      "updatedAt": "2025-06-11T01:18:29Z"
    }
  ]
}
```


创建数据集版本

接口描述 创建数据集版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
POST ?action=CreateDatasetVersion&datasetId={datasetId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |datasetId|String|是|Query参数|数据集ID| |description|String|否|RequestBody参数|版本描述| |storagePath|String|是|RequestBody参数|存储路径| |mountPath|String|是|RequestBody参数|默认挂载路径|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-| |id|String|数据集版本ID|

请求示例

```
{
  "description": "new version",
  "storagePath": "/path/to/version2",
  "mountPath": "/mnt/datasets/my-dataset-1/v2"
}
```

响应示例

```
{
  "requestId": "2642efe1-12ec-4ad5-9e34-7911e3430041",
  "id": "ds-1Wi75n2k"
}
```

删除数据集版本

接口描述 删除数据集版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
POST ?action=DeleteDatasetVersion&datasetId={datasetId}&versionId={versionId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |datasetId|String|是|Query参数|数据集ID| |versionId|String|是|Query参数|数据集版本ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 无响应参数。

请求示例 无。

响应示例

```
{
  "requestId": "904221cc-6fdf-4a19-a23f-62139e80e3ac"
}
```

模型管理相关接口

创建模型

接口描述 创建模型，同时创建一个初始版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
 POST ?action=CreateModel
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
name	String	是	Body参数	模型名称支持小写字母、数字和-，必须以小写字母开头，必须以小写字母或数字结尾，长度限制1-50。
description	String	否	Body参数	描述
modelFormat	String	是	Body参数	模型格式，HuggingFace、MegatronCore等
owner	String	否	Body参数	所有者，不传递时默认为创建者
visibilityScope	String	否	Body参数	可见范围ONLY_OWNER：仅所有者可读写
initVersionEntry	ModelVersionEntry	是	Body参数	初始版本相关信息

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-| |id|String|模型ID|

请求示例

```
{
  "name": "test1",
  "description": "test model",
  "modelFormat": "HuggingFace",
  "owner": "d1a5cf0143be4de9911342051106f70f",
  "visibilityScope": "ONLY_OWNER",
  "initVersionEntry": {
     "source": "UserUpload",
     "storageBucket": "bucket1",
     "storagePath": "/path/to/dir",
     "modelMetrics": "{\"Results\":[{\"Metrics\":{\"loss\":2.13,\"lr\":0.0005},\"Dataset\":{\"DatasetId\":\"ds-xxx\"}}]}",
     "description": "first version"
  }
}
```

响应示例

```
{
  "requestId": "1159de20-f93d-4673-99fc-1c4cce681625",
  "id": "mo-vnufEepi"
}
```

删除模型

接口描述 删除模型，同时删除所有版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
 POST ?action=DeleteModel&modelId={modelId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-|-| |modelId|String|是|Query参数|模型ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 无响应参数。

请求示例 无。

响应示例

```
{
  "requestId": "56f4c0c9-d455-4253-a1a8-f452dd639644"
}
```

修改模型

接口描述 修改模型。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
POST ?action=ModifyModel&modelId={modelId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |modelId|String|是|Query参数|模型ID| |name|String|是|RequestBody参数|模型名称| |description|String|是|RequestBody参数|描述|

响应头域 除公共头域外，无其它特殊头域。

响应参数 无响应参数。

请求示例

```
{
  "name": "my-model-1",
  "description": "my model"
}
```

响应示例

```
{
  "requestId": "384ccc3f-fb52-4bdc-be37-978a0169bbed",
  "id": "mo-vnufEepi"
}
```

获取模型详情

接口描述 获取模型详情。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
GET ?action=DescribeModel&modelId={modelId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |modelId|String|是|Query参数|模型ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |name|String|模型名称| |id|String|模型ID| |initSource|String|模型创建时的来源UserUpload：用户上传| |modelFormat|String|模型格式| |description|String|描述| |createdAt|String|创建时间| |updatedAt|String|更新时间| |owner|String|所有者| |ownerName|String|所有者名称| |visibilityScope|String|可见范围| |versionEntry|ModelVersionEntry|最新版本的详细信息|

请求示例 无。

响应示例

```
{
  "requestId": "a87246f6-5491-4161-b258-48b0633d744c",
  "name": "my-model-1",
  "id": "mo-vnufEepi",
  "initSource": "UserUpload",
  "modelFormat": "HuggingFace",
  "description": "my model",
  "createdAt": "2025-06-11T02:00:01Z",
  "updatedAt": "2025-06-11T02:02:00Z",
  "owner": "d1a5cf0143be4de9911342051106f70f",
  "ownerName": "zhangsan",
  "visibilityScope": "ONLY_OWNER",
  "versionEntry": {
    "version": "1",
    "id": "mo-QuPbZsfA",
    "source": "UserUpload",
    "taskId": "",
    "storageBucket": "bucket1",
    "storagePath": "/path/to/dir",
    "modelMetrics": "{\"Results\":[{\"Metrics\":{\"loss\":2.13,\"lr\":0.0005},\"Dataset\":{\"DatasetId\":\"ds-xxx\"}}]}",
    "createdAt": "2025-06-11T02:00:01Z",
    "createUser": "eca97e148cb74e9683d7b7240829d1ff",
    "createUserName": "root"
  }
}
```

获取模型列表

接口描述 获取模型列表。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
GET ?action=DescribeModels
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |keyword|String|否|Query参数|模型名称，用于筛选模糊匹配| |pageNumber|Integer|否|Query参数|分页参数，没传默认1| |pageSize|Integer|否|Query参数|分页大小，没传默认返回全部|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |totalCount|Integer|模型总数| |models|List<Model>|模型列表|

请求示例 无。

响应示例


```
{
  "requestId": "448cea7e-e3a3-4638-8e1e-92fb02792694",
  "totalCount": 1,
  "models": [
    {
      "name": "my-model-1",
      "id": "mo-vnufEepi",
      "initSource": "UserUpload",
      "latestVersion": "1",
      "latestVersionId": "mo-QuPbZsfA",
      "modelFormat": "HuggingFace",
      "description": "my model",
      "updatedAt": "2025-06-11T02:02:00Z",
      "createdAt": "2025-06-11T02:00:01Z",
      "owner": "d1a5cf0143be4de9911342051106f70f",
      "ownerName": "zhangsan",
      "visibilityScope": "ONLY_OWNER",
      "tags": {}
    }
  ]
}
```


获取模型版本详情

接口描述 获取模型版本详情。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
GET ?action=DescribeModelVersion&modelId={modelId}&versionId={versionId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |modelId|String|是|Query参数|模型ID| |versionId|String|是|Query参数|模型版本ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |name|String|模型名称| |id|String|模型ID| |initSource|String|模型创建时的来源UserUpload：用户上传| |modelFormat|String|模型格式| |description|String|描述| |createdAt|String|模型创建时间| |updatedAt|String|模型更新时间| |owner|String|所有者| |ownerName|String|所有者名称| |visibilityScope|String|可见范围| |versionEntry|ModelVersionEntry|查询的版本的详细信息|

请求示例 无。

响应示例

```
{
  "requestId": "1b3e3dac-51b7-4114-998e-f6548cb319f4",
  "name": "my-model-1",
  "id": "mo-vnufEepi",
  "initSource": "UserUpload",
  "modelFormat": "HuggingFace",
  "description": "my model",
  "createdAt": "2025-06-11T02:00:01Z",
  "updatedAt": "2025-06-11T02:02:00Z",
  "owner": "d1a5cf0143be4de9911342051106f70f",
  "ownerName": "zhangsan",
  "visibilityScope": "ONLY_OWNER",
  "versionEntry": {
    "version": "1",
    "id": "mo-QuPbZsfA",
    "source": "UserUpload",
    "taskId": "",
    "storageBucket": "bucket1",
    "storagePath": "/path/to/dir",
    "modelMetrics": "{\"Results\":[{\"Metrics\":{\"loss\":2.13,\"lr\":0.0005},\"Dataset\":{\"DatasetId\":\"ds-xxx\"}}]}",
    "description": "first version",
    "createdAt": "2025-06-11T02:00:01Z",
    "createUser": "eca97e148cb74e9683d7b7240829d1ff",
    "createUserName": "root"
  }
}
```


获取模型版本列表

接口描述 获取模型版本列表。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
GET ?action=DescribeModelVersions&modelId={modelId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |modelId|String|是|Query参数|模型ID| |pageNumber|Integer|否|Query参数|分页参数，没传默认1| |pageSize|Integer|否|Query参数|分页大小，没传默认返回全部|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |totalCount|Integer|模型版本总数| |versions|List<ModelVersionEntry>|模型版本列表|

请求示例 无。

响应示例


```
{
  "requestId": "4aca2b1b-9f61-40a2-86ea-2d33aaf96257",
  "totalCount": 2,
  "versions": [
    {
      "version": "2",
      "id": "mo-FJ9ycYDq",
      "source": "UserUpload",
      "taskId": "",
      "storageBucket": "bucket2",
      "storagePath": "/path/to/version2",
      "modelMetrics": "{\"Results\":[{\"Metrics\":{\"loss\":2.13,\"lr\":0.0005},\"Dataset\":{\"DatasetId\":\"ds-xxx\"}}]}",
      "description": "new version",
      "createdAt": "2025-06-11T02:18:14Z",
      "createUser": "eca97e148cb74e9683d7b7240829d1ff",
      "createUserName": "root"
    },
    {
      "version": "1",
      "id": "mo-QuPbZsfA",
      "source": "UserUpload",
      "taskId": "",
      "storageBucket": "bucket1",
      "storagePath": "/path/to/dir",
      "modelMetrics": "{\"Results\":[{\"Metrics\":{\"loss\":2.13,\"lr\":0.0005},\"Dataset\":{\"DatasetId\":\"ds-xxx\"}}]}",
      "description": "first version",
      "createdAt": "2025-06-11T02:00:01Z",
      "createUser": "eca97e148cb74e9683d7b7240829d1ff",
      "createUserName": "root"
    }
  ]
}
```


新建模型版本

接口描述 新建模型版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
POST ?action=CreateModelVersion&modelId={modelId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |modelId|String|是|Query参数|模型ID| |storageBucket|String|是|RequestBody参数|模型存储的BOS桶| |storagePath|String|是|RequestBody参数|BOS桶中的存储路径| |description|String|否|RequestBody参数|描述| |source|String|是|RequestBody参数|该版本模型的来源UserUpload：用户上传| |modelMetrics|String|否|RequestBody参数|模型指标，JSON格式。格式如下，其中 Metrics、Dataset 中的内容用户可自定义填写：{"Results":[{"Metrics":{"loss":2.13,"lr":0.0005},"Dataset":{"DatasetId":"ds-xxx"}}]}|

响应头域 除公共头域外，无其它特殊头域。

响应参数 |参数名称|类型|描述| |-|-|-| |id|String|模型版本ID|

请求示例

```
{
  "storageBucket": "bucket2",
  "storagePath": "/path/to/version2",
  "description": "new version",
  "source": "UserUpload",
  "modelMetrics": "{\"Results\":[{\"Metrics\":{\"loss\":2.13,\"lr\":0.0005},\"Dataset\":{\"DatasetId\":\"ds-xxx\"}}]}"
}
```

响应示例

```
{
  "requestId": "6a85d6f8-444a-49ac-b3c5-b19e7adf6916",
  "id": "mo-FJ9ycYDq"
}
```

删除模型版本

接口描述 删除模型版本。

权限说明 鉴权认证机制的详细内容请参见鉴权认证。

API Explorer 去调试

您可以在 API Explorer 中直接运行该接口，免去您计算签名的困扰。运行成功后，API Explorer 可以自动生成 SDK 代码示例。

请求结构

```
POST ?action=DeleteModelVersion&modelId={modelId}&versionId={versionId}
HTTP/1.1
Host: aihc.bj.baidubce.com
Authorization: authorization string
version: v2
```

请求头域 除公共头域外，无其它特殊头域。

请求参数 |参数名称|类型|是否必需|参数位置|描述| |-|-|-|-| |modelId|String|是|Query参数|模型ID| |versionId|String|是|Query参数|模型版本ID|

响应头域 除公共头域外，无其它特殊头域。

响应参数 无响应参数。

请求示例 无。

响应示例

```
{
  "requestId": "32a89d59-2509-4f65-9742-ae32c6e6ac65"
}
```

工作流相关接口

查询工作流模版详情

描述

查询指定工作流模版（Pipeline）的详细信息。

请求结构

```
GET /?action=DescribePipeline&pipelineId=xxxx
Host: aihc.bj.baidubce.com
Authorization: authorization string
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
pipelineId	String	是	Query 参数	Pipeline ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
pipelineId	String	Pipeline ID
name	String	名称
pipelineType	String	Pipeline 类型
resourcePoolId	String	资源池 ID
createTime	String	创建时间
updateTime	String	更新时间

请求示例

```
GET /?action=DescribePipeline&pipelineId=pipe-4zu7gy
```

返回示例


```
{
  "$schema": "https://10.209.54.223:8789/api/aihc/workflow-service/v1/schemas/Pipeline.json",
  "cceClusterId": "cce-gklhjnq8",
  "concurrencyPolicy": "Allow",
  "createdAt": "2026-08-19T10:17:38Z",
  "cronExpression": "0 0 * * *",
  "description": "示例 Pipeline",
  "disabled": "false",
  "id": 679,
  "manifest":
"dmVyc2lvbjogdjEKa2luZDogUGlwZWxpbmVUZW1wbGF0ZQp0YXNrVGVtcGxhdGVzOgogIC0gbmFtZTogY3VzdG9tLXRhc2stdGVtcGxhdGUKICAgIHR5cGU6IEN1c3RvbVRhc2sKICAgIGlucHV0czoKICAgICAgLSBuYW1lOiBiYW1lOiBiaxd
  "name": "syf-test-openapi-pipeline",
  "pipelineId": "pipe-b3twd8",
  "pipelineType": "cron",
  "region": "bj",
  "requestId": "cb72b146-f674-4add-a86c-dc2d8f4360e8",
  "resourcePoolId": "aihc-g9qm0av79z06",
  "status": "running",
  "updatedAt": "2026-08-24T08:50:23Z",
  "userId": "bea4cf865bc94bc183ecc43a70b4f910",
  "userName": "v_suiyufei",
  "version": 0
}
```


查询工作流运行列表

描述

查询指定工作流模版（Pipeline）下的运行实例（PipelineRun）列表，支持状态、时间范围、分页等过滤条件。

请求结构

```
GET /?action=DescribePipelineRuns&pipelineId=xxxx
Host: aihc.bj.baidubce.com
Authorization: authorization string
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
pipelineId	String	是	Query 参数	Pipeline ID
status	String	否	Query 参数	运行状态过滤，如 running、succeeded、failed
startTime	String	否	Query 参数	运行开始时间下限（ISO 8601 格式）
endTime	String	否	Query 参数	运行开始时间上限（ISO 8601 格式）
pageNumber	Integer	否	Query 参数	页码，从 1 开始，默认 1
pageSize	Integer	否	Query 参数	每页条数，默认 10

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
items	Array	运行实例列表，见下表
total	Integer	总条数
pageNumber	Integer	当前页码
pageSize	Integer	每页条数

items 数组元素字段说明：

字段名	类型	说明
runId	String	运行 ID
pipelineId	String	所属 Pipeline ID
status	String	运行状态
startedAt	String	开始时间
finishedAt	String	完成时间
createTime	String	创建时间

请求示例

```
GET /?action=DescribePipelineRuns&pipelineId=pipe-4zu7gy&pageNumber=1&pageSize=10
```

返回示例


```
{
  "items": [
    {
      "argoPhase": "succeeded",
      "argoWorkflowName": "syf-test-openapi-pipeline-0826-ncxv7",
      "argoWorkflowNamespace": "argo",
      "argoWorkflowUid": "17f4aef6-975d-4f85-88d5-74b8c4e90f9b",
      "createdAt": "2026-08-27T02:24:56Z",
      "failureEvents": "",
      "finishedAt": "2026-08-27T02:26:00Z",
      "id": 8270,
      "inputParameters": "{}",
      "manifest":
"YXBpVmVyc2lvbjogIlIKdmVyc2lvbjogdjEKa2luZDogUGlwZWxpbmVUZW1wbGF0ZQptZXRhZGF0YToge30KaW5wdXRzOgogICAgLSBuYW1lOiBDY2VDbHVzdGVySUQKICAgICAgdmFsdWU6IGNjZS1na2xoam5xOAogICAgLSBu

      "pipelineId": "pipe-15z9n2",
      "retryable": false,
      "runId": "17f4aef6-975d-4f85-88d5-74b8c4e90f9b",
      "runType": "manual",
      "startedAt": "2026-08-27T02:24:55Z",
      "status": "succeeded",
      "updatedAt": "2026-08-27T02:27:35Z"
    }
  ],
  "pageNumber": 1,
  "pageSize": 20,
  "requestId": "bd9e9ecb-4a4b-44b3-892c-529fd87255f0",
  "total": 1
}
```


创建工作流模版

描述

创建一个工作流模版（Pipeline）。可指定名称、Pipeline 定义（manifest）、类型、调度表达式、资源池等信息。支持 normal（普通）与 cron（定时）两种类型。

请求结构

```
 POST /?action=CreatePipeline
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json; charset=utf-8
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
name	String	是	Body参数	Pipeline 名称，1～255 字符
manifest	String	是	Body参数	Pipeline 定义的 Base64 字符串
description	String	否	Body参数	描述
pipelineType	String	否	Body参数	Pipeline 类型，支持 normal、cron，默认值：normal
cronExpression	String	否	Body参数	五段式 Cron 表达式，pipelineType=cron 时必选
disabled	String	否	Body参数	是否禁用，取值 true/false（保持 Console 兼容）
resourcePoolId	String	否	Body参数	资源池 ID
concurrencyPolicy	String	否	Body参数	并发策略，支持 Allow、Forbid、Replace

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
pipelineId	String	创建的 Pipeline ID
name	String	Pipeline 名称
pipelineType	String	Pipeline 类型
status	String	状态，新建初始状态为 init
createdAt	String	创建时间
updatedAt	String	更新时间

请求示例

```
{
    "name": "syf-test-openapi-pipeline-0824-1",
    "manifest": "version: v1\nkind: PipelineTemplate\ntaskTemplates:\n  - name: custom-task-template\n    type: CustomTask\n    inputs:\n      - name: queue_id\n        type: string\n        hint: 队列ID\n      - name: command\n        type: string\n        hint: 执行命令\n      - name: image_url\n        type: string\n        hint: 镜像地址\n      - name: cpu\n        type: int\n        hint: CPU申请配额(核) \n        defaultValue: 1\n      - name: memory\n        type: int\n        hint: 内存申请配额(GB)\n        defaultValue: 2\n    spec:\n      queue: '{{inputs.parameters.queue_id}}'\n      jobType: PyTorchJob\n      command: '{{inputs.parameters.command}}'\n      jobSpec:\n        replicas: 1\n        image: '{{inputs.parameters.image_url}}'\n        resources:\n          - name: cpu\n            quantity: '{{inputs.parameters.cpu}}'\n          - name: memory\n            quantity: '{{inputs.parameters.memory}}'\n    datasources: []\ntasks:\n  - name: first_workflow_task_demo\n    taskTemplateName: custom-task-template\n    inputs:\n      - name: queue_id\n        value: aihcq-a7gfnn99lplg\n      - name: command\n        value: echo \"This is your first workflow task.\"\n      - name: image_url\n        value: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release\n      - name: cpu\n        value: 0.5\n      - name: memory\n        value: 1",
    "description": "示例 Pipeline",
    "pipelineType": "normal",
    "disabled": "false",
    "resourcePoolId": "aihc-g9qm0av79z06",
    "concurrencyPolicy": "Allow"
}
```

返回示例

```
{
    "createdAt": "2026-08-24T15:16:48.852666212+08:00",
    "name": "syf-test-openapi-pipeline-0824-1",
    "pipelineId": "pipe-qttxg5",
    "pipelineType": "normal",
    "requestId": "0837f065-e365-4a30-8ce5-bd73bf9de7bd",
    "status": "init",
    "updatedAt": "2026-08-24T15:16:48.852666259+08:00"
}
```

更新工作流模版

描述

更新一个已有的工作流模版（Pipeline）。Body 中所有字段均为可选，仅透传调用方实际提供的字段（未传字段保持原值不变）。

请求结构

```
POST /?action=UpdatePipeline&pipelineId=xxxx
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json; charset=utf-8
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
pipelineId	String	是	Query参数	待更新的 Pipeline ID
name	String	否	Body参数	Pipeline 名称，1～255 字符
description	String	否	Body参数	描述
manifest	String	否	Body参数	Pipeline 定义的 Base64 字符串
pipelineType	String	否	Body参数	Pipeline 类型，支持 normal、cron
cronExpression	String	否	Body参数	五段式 Cron 表达式
disabled	String	否	Body参数	是否禁用，取值 true/false
resourcePoolId	String	否	Body参数	资源池 ID
concurrencyPolicy	String	否	Body参数	并发策略，支持 Allow、Forbid、Replace

返回头域

除公共头域，无其它特殊头域。

返回参数

更新后的 Pipeline 对象扁平输出，并附加 requestId。

参数名称	类型	说明
requestId	String	请求 ID
pipelineId	String	Pipeline ID
name	String	Pipeline 名称
pipelineType	String	Pipeline 类型
status	String	状态
createdAt	String	创建时间
updatedAt	String	更新时间

> 返回体透传后端返回的全部 Pipeline 字段，上表为主要字段示例。

请求示例


```
{
  "name": "syf-test-openapi-pipeline",
  "manifest": "version: v1\nkind: PipelineTemplate\ntaskTemplates:\n  - name: custom-task-template\n    type: CustomTask\n    inputs:\n      - name: queue_id\n        type: string\n        hint: 队列ID\n      - name: command\n        type: string\n        hint: 执行命令\n      - name: image_url\n        type: string\n        hint: 镜像地址\n      - name: cpu\n        type: int\n        hint: CPU申请配额(核）\n        defaultValue: 1\n      - name: memory\n        type: int\n        hint: 内存申请配额(GB)\n        defaultValue: 2\n    spec:\n      queue: '{{inputs.parameters.queue_id}}'\n      jobType: PyTorchJob\n      command: '{{inputs.parameters.command}}'\n      jobSpec:\n        replicas: 1\n        image: '{{inputs.parameters.image_url}}'\n        resources:\n          - name: cpu\n            quantity: '{{inputs.parameters.cpu}}'\n          - name: memory\n            quantity: '{{inputs.parameters.memory}}'\n      datasources: []\ntasks:\n  - name: first_workflow_task_demo\n    taskTemplateName: custom-task-template\n    inputs:\n      - name: queue_id\n        value: aihcq-a7gfnn99lplg\n      - name: command\n        value: echo \"This is your first workflow task.\"\n      - name: image_url\n        value: registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release\n      - name: cpu\n        value: 0.5\n      - name: memory\n        value: 1",
  "description": "示例 Pipeline",
  "pipelineType": "cron",
  "disabled": "false",
  "cronExpression": "0 0 * * *",
  "resourcePoolId": "aihc-g9qm0av79z06",
  "concurrencyPolicy": "Allow"
}
```


返回示例


```
{
  "$schema": "https://10.209.54.223:8789/api/aihc/workflow-service/v1/schemas/Pipeline.json",
  "cceClusterId": "cce-gklhjnq8",
  "concurrencyPolicy": "Allow",
  "createdAt": "2026-08-19T10:17:38Z",
  "cronExpression": "0 0 * * *",
  "description": "示例 Pipeline",
  "disabled": "false",
  "id": 679,
  "manifest":
"dmVyc2lvbjogdjEKa2luZDogUGlwZWxpbmVUZW1wbGF0ZQp0YXNrVGVtcGxhdGVzOgogIC0gbmFtZTogY3VzdG9tLXRhc2stdGVtcGxhdGUKICAgIHR5cGU6IEN1c3RvbVRhc2sKICAgIGlucHV0czoKICAgICAgLSBuYW1lOiBxdWV1ZV9pZAogICAgICAgIHR5cGU6IHN0cmluZwogICAgICAgIGhpbnQ6IOmYn+WIl0lECiAgICAgIC0gbmFtZTogY29tbWFuZAogICAgICAgIHR5cGU6IHN0cmluZwogICAgICAgIGhpbnQ6IOaJp+ihjOWRveS7pAogICAgICAtIG5hbWU6IGltYWdlX3VybAogICAgICAgIHR5cGU6IHN0cmluZwogICAgICAgIGhpbnQ6IOmVnOWDj+WcsOWdgAogICAgICAtIG5hbWU6IGNwdQogICAgICAgIHR5cGU6IGludAogICAgICAgIGhpbnQ6IENQVeeUs+ivt+mFjemineaguOaVsCkKICAgICAgICBkZWZhdWx0VmFsdWU6IDEKICAgICAgLSBuYW1lOiBtZW1vcnkKICAgICAgICB0eXBlOiBpbnQKICAgICAgICBoaW50OiDlhoXlrZjnlLPor7fphY3pop0oR0IpCiAgICAgICAgZGVmYXVsdFZhbHVlOiAyCiAgICBzcGVjOgogICAgICBxdWV1ZTogJ3t7aW5wdXRzLnBhcmFtZXRlcnMucXVldWVfaWR9fScKICAgICAgam9iVHlwZTogUHlUb3JjaEpvYgogICAgICBjb21tYW5kOiAne3tpbnB1dHMucGFyYW1ldGVycy5jb21tYW5kfX0nCiAgICAgIGpvYlNwZWM6CiAgICAgICAgcmVwbGljYXM6IDEKICAgICAgICBpbWFnZTogJ3t7aW5wdXRzLnBhcmFtZXRlcnMuaW1hZ2VfdXJsfX0nCiAgICAgICAgcmVzb3VyY2VzOgogICAgICAgICAgLSBuYW1lOiBjcHUKICAgICAgICAgICAgcXVhbnRpdHk6ICd7e2lucHV0cy5wYXJhbWV0ZXJzLmNwdX19JwogICAgICAgICAgLSBuYW1lOiBtZW1vcnkKICAgICAgICAgICAgcXVhbnRpdHk6ICd7e2lucHV0cy5wYXJhbWV0ZXJzLm1lbW9yeX19JwogICAgICBkYXRhc291cmNlczogW10KdGFza3M6CiAgLSBuYW1lOiBmaXJzdF93b3JrZmxvd190YXNrX2RlbW8KICAgIHRhc2tUZW1wbGF0ZU5hbWU6IGN1c3RvbS10YXNrLXRlbXBsYXRlCiAgICBpbnB1dHM6CiAgICAgIC0gbmFtZTogcXVldWVfaWQKICAgICAgICB2YWx1ZTogYWloY3EtYTdnZm45OWxwbGcKICAgICAgLSBuYW1lOiBjb21tYW5kCiAgICAgICAgdmFsdWU6IGVjaG8gXCJUaGlzIGlzIHlvdXIgZmlyc3Qgd29ya2Zsb3cgdGFzay5cIgogICAgICAtIG5hbWU6IGltYWdlX3VybAogICAgICAgIHZhbHVlOiByZWdpc3RyeS5iYWlkdWJjZS5jb20vYWloYy1haWFrL2FpYWstbWVnYXRyb246dWJ1bnR1MjAuMDQtY3UxMS44LXRvcmNoMS4xNC4wLXB5MzhfdjEuMi43LjEyX3JlbGVhc2UKICAgICAgLSBuYW1lOiBjcHUKICAgICAgICB2YWx1ZTogMC41CiAgICAgIC0gbmFtZTogbWVtb3J5CiAgICAgICAgdmFsdWU6IDE=",
  "name": "syf-test-openapi-pipeline",
  "pipelineId": "pipe-b3twd8",
  "pipelineType": "cron",
  "region": "bj",
  "requestId": "4b474be9-4859-46b1-992e-cd7186fcfbf0",
  "resourcePoolId": "aihc-g9qm0av79z06",
  "status": "running",
  "updatedAt": "2026-08-24T08:50:23Z",
  "userId": "bea4cf865bc94bc183ecc43a70b4f910",
  "userName": "v_suiyufei",
  "version": 0
}
```


删除工作流运行实例

描述

删除指定的工作流运行实例（PipelineRun）。

请求结构

```
POST /?action=DeletePipelineRun&runId=xxxx
Host: aihc.bj.baidubce.com
Authorization: authorization string
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
runId	String	是	Query 参数	运行 ID

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
runId	String	被删除的运行 ID

请求示例

```
POST /?action=DeletePipelineRun&runId=c9cbe6eb-bf8b-4edb-8172-6a3550d1aec0
```

返回示例

```
{
  "requestId": "f91dd545-f310-4ba8-9177-2b790f61d7ec",
  "runId": "d3b84f40-1e31-434e-a999-aad49c892ef2"
}
```

停止工作流运行

描述

停止指定的工作流运行。支持两种停止方式：优雅停止（等待任务完成）和强制终止（立即 kill Pod）。

请求结构

```
POST /?action=StopPipelineRun&runId=xxxx
Host: aihc.bj.baidubce.com
Authorization: authorization string
ContentType: application/json; charset=utf-8
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
runId	String	是	Query 参数	运行 ID
terminate	Boolean	否	Body 参数	强制终止标志。true 强制终止（kill Pod），false 优雅停止，默认 false
message	String	否	Body 参数	停止原因或备注信息

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
runId	String	运行 ID

请求示例

```
{
  "terminate": false,
  "message" : "暂时停止"
}
```

返回示例

```
{
  "requestId": "dfaf585e-ea49-4753-9337-730d3c1d689f",
  "runId": "907ba59e-d53c-452b-bc25-616f87454516"
}
```

运行工作流

描述

基于已有的工作流模版（Pipeline）提交一次运行实例。对于 cron 类型的 Pipeline，可通过 isCronRunOnce 立即触发一次运行。

请求结构

```
POST /?action=CreatePipelineRun
Host:aihc.bj.baidubce.com
Authorization:authorization string
ContentType: application/json; charset=utf-8
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
pipelineId	String	是	Body参数	Pipeline ID
isCronRunOnce	Boolean	否	Body参数	Cron Pipeline 立即运行一次

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
runId	String	运行 ID
argoWorkflowUid	String	Argo Workflow UID

请求示例

```
{
  "pipelineId" : "pipe-4zu7gy",
  "isCronRunOnce": false
}
```

返回示例

```
{
  "argoWorkflowUid": "c9cbe6eb-bf8b-4edb-8172-6a3550d1aec0",
  "requestId": "fadedc90-0f47-4a51-b842-8c069e69c6ab",
  "runId": "c9cbe6eb-bf8b-4edb-8172-6a3550d1aec0"
}
```

重试工作流运行

描述

重试指定的工作流运行实例（PipelineRun）。支持两种模式：从头全量重跑（full）或从失败节点断点续跑（breakpoint）。

请求结构

```
 POST /?action=RetryPipelineRun&runId=xxxx
Host: aihc.bj.baidubce.com
Authorization: authorization string
ContentType: application/json; charset=utf-8
Version: v2
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
runId	String	是	Query 参数	运行 ID
mode	String	否	Body 参数	重试模式。`full` 从头全量重跑（新建运行）；`breakpoint` 从失败节点断点续跑（沿用 runId）。默认 `full`

返回头域

除公共头域，无其它特殊头域。

返回参数

参数名称	类型	说明
requestId	String	请求 ID
runId	String	运行 ID（breakpoint 模式下与入参相同；full 模式下为新建运行的 ID）
argoWorkflowName	String	Argo Workflow 名称
argoWorkflowUid	String	Argo Workflow UID

请求示例

```
{
  "mode": "full"
}
```

返回示例

```
{
  "argoWorkflowName": "",
  "argoWorkflowUid": "17f4aef6-975d-4f85-88d5-74b8c4e90f9b",
  "requestId": "d25391dd-46fc-4961-b279-534cf92b1187",
  "runId": "17f4aef6-975d-4f85-88d5-74b8c4e90f9b"
}
```

数据结构

JobSpec

创建训练任务配置，被如下接口引用：创建训练任务

参数名称	类型	必选	说明
image	String	是	镜像地址，需要包含tag
imageConfig	ImageConfig	否	镜像仓库访问配置，仅私有镜像时需要配置
replicas	Integer	是	worker副本数
resources	List<Resource>	否	配置资源配额
envs	List<Env>	否	worker环境变量，默认注入:worker环境变量，默认注入: 1. AIHC_JOB_NAME ，值为name字段的值 2. NCCL_IB_DISABLE，开启rdma后默认注入，值为0 3. NCCL_DEBUG，nccl日志的级别，值为INFO
enableRDMA	Boolean	否	对自运维资源池生效，是否开启RDMA，默认false，开启后将自动添加NCCL_IB_DISABLE=0的环境变量，添加rdma/hca资源，并配置10GB共享内存到训练节点的容器中 全托管资源池创建分布式训练任务，若您选择的GPU机型支持RDMA属性，平台会自动为任务开启并配置RDMA网络特性，无需配置此字段 虚拟化GPU队列若需开启RDMA，GPU资源必须是整数卡，小数卡不支持开启RDMA，并且资源池调度组件（CCE AI Job Scheduler）版本需要 ≥1.7.48
hostNetwork	Boolean	否	是否使用宿主机网络，开启后作业worker将使用宿主机网络；全托管资源池仅支持容器网络，使用宿主机网络请联系开发人员，打开后多实例调度到同节点可能导致端口冲突影响调度，请谨慎使用

TensorboardConfig

参数名称	类型	必选	说明
enable	Boolean	否	是否启用 TensorBoard。不传tensorboardConfig 或置 false 时不创建。**RayJob 框架不支持 TensorBoard**。
datasourceType	String	否	enable=true 时必填，TensorBoard 日志源的数据源类型，当前支持pfs、bos。
datasourceName	String	否	enable=true 时必填，数据源实例标识。pfs 填 pfsId；bos 填 bos桶名。
logPath	String	否	日志路径。当datasourceType 为 bos/pfs时必填，作为数据源 SourcePath。
disableAutoAddJobID	Boolean	否	是否禁止自动在 logPath 追加 jobId。默认false（自动追加）。

AlertConfig

参数名称	类型	必选	说明
instanceId	String	是	对应集群监控实例 ID
alertItems	List<String>	是	任务告警对应值如下： jobRunning：任务进入运行状态 jobFT：任务发生容错 nodeFT：任务所在节点出现故障 jobFailed：任务失败退出 jobSucceed：任务成功结束 jobHang：任务挂起（或：任务卡死） 备注：TFJob支持jobRunning、jobFailed、jobSucceed类型告警，PyTorchJob支持类型所有告警
for	String	否	时间间隔：默认是0m
notifyRuleId	String	是	对应通知策略ID（notify-***）
自定义 Webhook 请求体固定为 {"msgtype":"text","text":{"content":"告警内容"}}，content 由系统自动填充告警详情；钉钉、飞书渠道请使用各自官方文档定义的消息结构。			

AdvancedSettings

参数名称	类型	必选	说明
runtimeEnv	String	否	任务运行时环境声明
submitterBackoffLimit	int32	否	RayJob Submitter 实例的最大重试次数，默认值为 0；开启 Head 节点容错（即已配置百度智能云云数据库 Redis 实例）时，该值须大于 0
redisInstanceId	String	否	百度智能云云数据库 Redis 实例 ID，须与任务所使用的资源池处于同一 VPC 网络内
redisUsername	String	否	账号名称，须有读写权限，默认账号为 default
redisPassword	String	否	实例的登录密码
enableBlb	Boolean	否	是否开启 BLB 负载均衡功能，该功能启用的前提是已开启 Head 节点容错能力（即已配置redisInstanceId）。开启后将默认创建按量付费模式的内网应用型 BLB 实例，相关计费规则可参考 BLB计费说明
blbInstanceId	String	否	内网应用型 BLB 实例 ID，仅作为详情返回字段，创建请求时不支持传入

PortConfigs

自定义监控监听端口配置，被如下接口引用：创建训练任务、查询训练任务详情

参数名称	类型	必选	说明
portConfig	List<PortConfig>	是	端口配置列表，OpenAPI 创建训练任务仅允许配置 1 组
instanceId	String	否	Cprom 监控实例 ID。由服务端自动查询集群绑定的监控实例并填充，无需传入（传入也会被忽略）
agentId	String	否	Cprom 监控代理 ID。由服务端自动查询集群绑定的监控实例并填充，无需传入（传入也会被忽略）

PortConfig

单个端口配置，被如下接口引用：创建训练任务、查询训练任务详情

参数名称	类型	必选	说明
containerPort	Integer	是	容器内服务监听端口号，取值范围 1-29999 或 32768-65535；2222、6379、8265、9101、10000、10001、23456、52365 为系统保留端口，不可使用
usage	String	是	端口用途，当前仅支持 Metrics
path	String	否	端口请求路径，须以/ 开头，最长 1024 字符且不能包含空格；usage=Metrics 时默认值为 /metrics

ImageConfig

任务镜像配置，被如下接口引用：创建训练任务

参数名称	类型	必选	说明
username	String	是	私有镜像仓库用户名
password	String	是	私有镜像仓库密码

Resource

任务资源配置，被如下接口引用：创建训练任务

参数名称	类型	必选	说明
name	String	是	资源名称示例，支持设置GPU/CPU/内存以及共享内存资源，按资源类型： • GPU资源：需要根据型号按照百度的资源描述符填入，例如A800为 baidu.com/a800_80g_cgpu，更多型号参考下表。 • 虚拟化GPU资源：在虚拟化队列中指定算力资源时，1000 表示0.1卡，10000 表示1卡，如baidu.com/a800_80g_cgpu_core: 1000表示0.1卡，若申请量小于1卡（数值＜10000）时，最低为0.1卡（对应1000），且只能以0.1卡为步长递增（即1000的整数倍），若申请量大于等于1卡（数值≥10000）时，仅允许申请整卡（即数值必须为10000的整数倍）。 • CPU 资源：填写 cpu • 内存资源：填写 memory • 共享内存资源：填写 sharedMemory
quantity	Number	是	资源量，可传递int或float类型，GPU资源单位为卡数，cpu 资源单位为核，内存资源单位为GB，共享内存资源单位为GB

GPU 芯片和资源名称映射表：

芯片	资源名
A800-SXM4-80GB	baidu.com/a800_80g_cgpu
A100-SXM4-40GB	baidu.com/a100_40g_cgpu
A100-SXM-80GB	baidu.com/a100_80g_cgpu
A10	baidu.com/a10_24g_cgpu
H800	baidu.com/h800_80g_cgpu
Tesla V100-SXM2-16GB	baidu.com/v100_16g_cgpu
Tesla V100-SXM2-32GB	baidu.com/v100_32g_cgpu
L20	baidu.com/l20_cgpu
L40	baidu.com/l40_cgpu
H20	baidu.com/h20_96g_cgpu
H20Z	baidu.com/h20z_141g_cgpu
H20-3e	baidu.com/h20_141g_cgpu

Env

环境变量信息，被如下接口引用：创建训练任务、查询训练任务详情

参数名称	类型	必选	说明
name	String	否	标签名
value	String	否	标签值

Label

训练任务标签信息，被如下接口引用：创建训练任务

参数名称	类型	必选	说明
key	String	是	标签名
value	String	是	标签值

JobItem

单个训练任务详情信息，被如下接口引用：查询训练任务详情、查询训练任务列表

参数名称	类型	说明
jobid	String	任务id
userId	String	用户id
name	String	任务名称
status	String	任务状态说明 1.Created：排队中，任务已创建，等待调度阶段（允许修改优先级），实例未创建或者所有实例状态为排队中 2.Starting：启动中，Pod 调度完成，实例的状态均为启动中或者包含部分运行中的状态 3.Running：运行中，所有的 Pod 为运行中的状态 4.Stopping：停止中，用户操作停止任务，正在停止任务，正在将实例置为失败或者删除实例 5.Stopped：已停止，任务已经停止，实例状态为失败或者实例已经被删除 6.Failed：失败，任务失败退出，所有实例状态为失败 7.Succeeded：成功，任务成功结束，所有实例状态为成功 8.Abnormal：异常，存在 >=1 的实例状态为失败 9.Restarting：重启中，任务容错或者被抢占，且所有实例状态为排队中
jobType	String	任务类型，如：pytorchjob、mpijob等
resourcePoolId	String	任务所在资源池Id
queue	String	任务所在资源池队列
jobSpec	JobSpec	任务配置
createdAt	String	任务创建时间
finishedAt	String	任务结束时间
datasources	List<DataSource>	任务的数据源配置
labels	List<Label>	任务标签
priority	String	任务优先级
enableBccl	Boolean	任务是否开启了BCCL注入
enableBcclStatus	String	BCCL注入状态,包括: success: 注入成功 failed: 注入失败 unknown: 还未执行注入
enableBcclErrorReason	String	BCCL注入失败原因
enableFaultTolerance	Boolean	是否开启容错
faultToleranceArgs	String	容错配置参数
gpuCount	Integer	GPU卡数
nodeCount	Integer	训练任务实例数
gpuUtilizationPercent	Number	GPU利用率（百分比）
gpuMemoryUtilizationPercent	Number	GPU内存利用率百分比（百分比）
runTimeNanoseconds	Integer	训练任务耗时（纳秒）
visibleScope	Integer	训练任务可见范围，参数为1代表队列内可见，为0代表创建人可见
pods	List<Pod>	任务Pod列表，在详情接口needDetail参数为False和查询训练任务列表接口中不返回该字段
historyPods	List<Pod>	历史Pod列表，在详情接口needDetail参数为False和查询训练任务列表接口中不返回该字段
jobTimeLine	List<JobTimeLine>	任务时间线详情信息，查询训练任务列表接口中不返回该字段
idleTask	Boolean	是否为闲时任务

DataSource

训练任务数据源配置，被如下接口引用：创建训练任务

参数名称	类型	必选	说明
type	String	是	数据源类型，枚举值：pfs/pfsl3/hostpath/dataset/bos/cfs/rapidfs/public_dataset；
name	String	是	数据源名称，如果type类型为pfs时，此处必须填写pfs实例id; type类型为bos/pfsl3时，默认为空
sourcePath	String	是	源路径，type类型为pfs时，表示pfs存储中的路径，默认为/; type类型为hostpath时，表示宿主机路径
mountPath	String	是	容器内挂载路径
options	Option	否	数据源参数
id	String	否	当 type=dataset或public_dataset时必填，表示数据集 ID；type为其他类型无需填写。

Option

训练任务详情-数据卷挂载选项信息，被如下接口引用：查询训练任务详情、查询训练任务列表

参数名称	类型	说明
readOnly	Boolean	是否以只读模式挂载到容器中
datasetVersion	String	数据集版本，当type=dataset或public_dataset时，可以填入，默认最新数据集版本
cfsInstanceId	String	CFS 实例 ID，type=cfs 时必填
cfsMountPoint	String	CFS 挂载点域名，type=cfs 时必填，格式如cfs-xxx.lb-xxx.cfs.xx.baidubce.com
pfsL3InstanceId	String	PFSl3 实例 ID，type=pfsl3 时必填
pfsL3MountPoint	String	PFSl3 挂载点域名，type=pfsl3 时必填，格式如pfs-bjiTkxxSqQ.lb-58axxe99.pfs.bj.baidubce.com

JobTimeLine

任务时间线详情信息，用于描述任务生命周期内关键状态节点的变更记录，被如下接口引用：查询训练任务详情

参数名称	类型	说明
conditionType	String	任务状态类型： 1.Created：任务创建 2.Scheduled：任务启动 3.Running：任务运行、 4.Abnormal：任务异常 5.FaultTolerantStart：启动容错 6.Restarting：任务重启 7.Succeeded：任务成功 8.Failed；任务失败 9.ManualTermination：任务停止 任务进入后生命周期结束的状态：Succeeded（任务成功）、Failed（任务失败）、ManualTermination（任务停止）
conditionMessage	String	任务状态描述，由 conditionType 自动映射生成（如 conditionType 为 Created 则显示 “任务创建”，为 Succeeded 则显示 “任务成功”）
time	String	任务进入当前状态的时间（如 2024-05-20T14:30:00+08:00）

Pod

Pod详情信息，被如下接口引用：查询训练任务详情

参数名称	类型	说明
PodIP	String	Pod IP
nodeName	String	任务pod所在节点的名称
creationAt	String	Pod创建时间
uid	String	Pod的id
name	String	任务Pod名称
status	String	Pod状态: 1.Pending: 排队中, Pod已经被创建，但是还没有被度到节点上 2.Starting: 启动中，已调度，镜像拉取、存储挂载、创建容器等过程 3.Running: 运行中，实例进入running状态，执行训练代码 4.Failed: 失败，pod 失败退出 5.Succeed: 成功，pod 成功退出 6.Unknown: 异常, pod状态不可知
replicaType	String	任务Pod的副本类型，pytorch中包含master和worker两种副本类型
restartCount	Number	任务Pod重启次数
envs	List<Env>	Pod环境变量
finishedAt	String	Pod完成时间
reason	String	Pod失败原因

Event

任务事件信息，被如下接口引用：查询训练任务事件

参数说明	类型	说明
reason	String	原因
message	String	详细信息
firstTimestamp	String	首次出现时间
lastTimestamp	String	最后出现时间
count	Number	出现次数
type	Number	事件类型

Metrics

查询任务监控的返回信息，被如下接口引用：查询训练任务监控

参数名称	类型	说明
PodName	String	任务Pod名称
metrics	List<Metric>	监控数据，当前支持查询监控数据的指标类型，取值如下： 1.GpuUsage：GPU使用率 2.GpuMemoryUsage： GPU显存使用率 3.CpuUsage：CPU 使用率 4.MemoryUsage：内存使用率 5.DiskReadRate：磁盘读取速率 6.DiskWriteRate：磁盘写入速率

Metric

查询任务Pod的监控数据，被如下接口引用：查询训练任务监控

参数名称	类型	说明
time	String	时间戳（Unix Timestamp），单位为毫秒
value	Number	监控数据的值。

ServiceConf

参数名称	参数类型	是否必须	参数位置	参数说明
name	String	是	Body参数	服务名称,需满足正则表达式^[a-z][a-z0-9-]*[a-z0-9]$，长度不高于50
acceleratorType	String	是	Body参数	加速芯片类型
workloadType	String	是	Body参数	fed 或 ""，单机用""，分布式推理使用fed
instanceCount	Integer	是	Body参数	部署实例数，取值>=0
resourcePool	ResourcePoolConf	是	Body参数	资源池描述
storage	StorageConf	否	Body参数	存储卷、共享内存配置
containers	Array of ContainerConf	是	Body参数	服务容器信息，最少需要1个，最多10个，见containers结构
access	AccessConf	否	Body参数	服务访问配置信息
log	LogConf	否	Body参数	日志配置
deploy	DeployConf	否	Body参数	部署配置，默认为最大超量和最大不可用均为25%
misc	Misc	否	Body参数	实例label、annotations配置

ContainerConf

创建自定义服务的配置，被如下接口引用：创建服务

参数名	参数类型	是否必须	参数说明
name	String	是	容器名称，需满足正则表达式^a-z0-9?$，长度不高于63
cpus	Integer	是	cpu核数 > 0
memory	Integer	是	内存大小 > 0 ，单位GiB
acceleratorCount	Integer	是	加速卡数量 >= 0
command	List	否	启动命令
runArgs	List	否	启动参数
ports	List	否	[]int结构json串，允许容器选择不暴露或暴露多个端口
envs	Map<String, String>	否	map[string]string结构
image	ImageConf	是	镜像信息，见请求示例
volumeMounts	List	否	存储挂载，见请求示例
readinessProbe	ProbeConf	否	就绪探针信息
startupsProbe	ProbeConf	否	启动探针信息
livenessProbe	ProbeConf	否	存活探针信息

PortConf

创建自定义服务的配置，被如下接口引用

参数名	参数类型	是否必须	参数说明
name	String	是	port名字，仅可选HTTP、GRPC、Metrics、Other，没有区别，名字表明端口用途，注：如果有Metrics端口则后端默认开启Prometheus服务
port	Integer	是	端口号

ImageConf

创建自定义服务的配置，被如下接口引用

参数名	参数类型	是否必须	参数说明
imageType	Integer	是	镜像类型 0：预置镜像 1:ccr 2:其他
imageUrl	String	是	镜像
username	String	否	镜像拉取用户名
password	String	否	镜像拉取密码

VolumnMountConf

参数名	参数类型	是否必须	参数说明
volumnName	String	是	卷名
mountPath	String	是	挂载到容器的地址
readOnly	Boolean	否	默认为false

ProbeConf

参数名	参数类型	是否必须	参数说明
initialDelaySeconds	Integer	是	初始化检查延时，大于等于0
timeoutSeconds	Integer	是	检查超时时长，大于等于1
periodSeconds	Integer	是	轮询检查时间间隔，大于等于1
successThreshold	Integer	是	连接成功次数阈值,超过该值可被认为成功，大于等于1,livenessProbe和startupProbe必须是1
failureThreshold	Integer	是	连接失败次数阈值,超过该值可被认为失败，大于等于1
handler	ProbeHandlerConf	是	健康检查方法

ProbeHandlerConf

参数名	参数类型	是否必须	参数说明
exec	ExecAction	否	通过在容器内执行命令判断，exec httpGet tcpSocketAction有且仅能只有一个
httpGet	HTTPGetAction	否	通过向容器发送HTTP请求判断，exec httpGet tcpSocketAction有且仅能只有一个
tcpSocketAction	TCPSocketAction	否	通过向容器发起TCP连接判断，exec httpGet tcpSocketAction有且仅能只有一个

ExecAction

参数名	参数类型	是否必须	参数说明
command	List	是	在容器中执行指定命令，如果执行成功后退出码为0则健康检查成功

HTTPGetAction

参数名	参数类型	是否必须	参数说明
path	String	是	探针的调用路径
port	Integer	是	探针的调用端口号

TCPSocketAction

参数名	参数类型	是否必须	参数说明
port	Integer	是	探针的调用端口号

AccessConf

参数名	参数类型	是否必须	参数说明
publicAccess	Boolean	否	是否开启公网访问，不填默认false
eip	String	否	开启公网时且networkType为空时必填，选用的EIP IP地址,
aiGateway	AiGatewayConf	否	网关配置，netType是aiGateway时必填
networkType	String	否	网络类型： 空 或者 aiGateway,
blbId	String	否	blb id，netType设置为""时，blb id须设置

LogConf

参数名	参数类型	是否必须	参数说明
persistent	Boolean	否	是否开启日志持久化，开启后将持久化容器标准输出日志

DeployConf

参数名	参数类型	是否必须	参数说明
canaryStrategy	CanaryStrategyConf	否	更新策略，默认maxSurge、maxUnavailable为25%
schedule	ScheduleConf	否	优先级配置，可选：low, normal,high,"" ,默认为normal

CanaryStrategyConf

参数名	参数类型	是否必须	参数说明
maxSurge	Integer	否	滚动更新过程中超出预期的实例数量占预期实例数的百分比，默认为0，和maxUnavailable不能同时为0
maxUnavailable	Integer	否	滚动更新过程中不可用实例的数量占预期实例数的百分比，默认为0，和maxSurge不能同时为0

Misc

参数名	参数类型	是否必须	参数说明
podLabels	Map<String, String>	否	kubernetes pod labels #设置多个时以,分割 "scheduling.volcano.sh/group-min-member": "1" #最小实例数 "aihc.baidu.com/reschedulable": "true" #开启碎片迁移
podAnnotations	Map<String, String>	否	kubernetes pod Annotations
gracePeriodSec	Integer	否	优雅退出时间
fedPodsPerIns	Integer	否	分布式部署模式（workloadType="fed"）下，pod组的实例数，必须大于1
enableRDMA	Boolean	否	是否启用rdma

ResourcePoolConf

参数名	参数类型	是否必须	参数说明
resourcePoolId	String	是	资源池ID，对于托管资源池固定值：aihc-serverless
queueName	String	是	队列名
resourcePoolType	String	是	资源池类型,取值为""、serverless、public，空字符串为自运维资源池，serverless是全托管资源池，public预留字段暂未使用
resourcePoolName	String	是	资源池名称，托管资源池不填

StorageConf

参数名	参数类型	是否必须	参数说明
shmSize	Integer	否	共享内存大小，单位GiB，默认为0
volumns	List<VolumnConf>	否	存储卷配置

VolumnConf

参数名	参数类型	是否必须	参数说明
volumeType	String	是	存储卷的类型，可选"pfs"、"hostpath"、"emptydir"
volumnName	String	是	存储卷的名字，满足正则表达式^[a-z][a-z0-9-]*[a-z0-9]$，长度不高于50。 升级服务时，如果该卷配置发生变化，卷名建议重新生成且不与之前重复
pfs	PFSConfig	否	存储卷类型为pfs时填写

PFSConfig

参数名	参数类型	是否必须	参数说明
sourcePath	String	是	文件在pfs中的路径
InstanceId	String	是	pfs的实例ID

ServiceBriefInfo

参数名	参数类型	参数说明
id	String	服务ID
name	String	服务名称
resourcePoolId	String	资源池ID
resourcePoolName	String	资源池名称
queueName	String	队列名
region	String	资源池所在地域
publicAccess	Boolean	公网访问开关
creator	String	服务创建人
networkType	String	网络类型
createdAt	Integer	创建时间
updatedAt	Integer	更新时间
resourcePoolType	String	资源池类型
workloadType	String	负载类型,fed 或 ""，单机用""，分布式推理使用fed
resourceSpec	ResourceSpec	资源详情

ServiceBriefStat

参数名	参数类型	参数说明
status	Integer	服务状态，1:部署中 2:运行中 3:未运行 4:异常
availableIns	Integer	可用实例数
totalIns	Integer	总实例数
reason	String	异常信息（如果有）

ServiceStatus

参数名	参数类型	参数说明
accessIPs	AccessIPConf	访问IP
accessPorts	List	访问端口
blbShortId	String	BLB实例ID
aiGateway	AiGateway	Gateway认证信息
briefStat	ServiceBriefStat	服务简要信息

AccessIPConf

参数名	参数类型	参数说明
internal	String	内网ip
external	String	外网ip

AccessPortConf

参数名	参数类型	参数说明
name	String	系统生成的端口号名称
containerPort	Integer	容器暴露的端口号
servicePort	Integer	对应的访问端口号

AIGateWay

参数名	参数类型	参数说明
basePath	String	访问地址的基路径，包括/auth/s-r3c228e93b86/ V1版本网关和/s-r3c228e93b86/ V2版本网关两种格式
tokens	Tokens	认证Token信息

Token

参数名	参数类型	参数说明
serveless	String	网关认证Token,新/老网关开启鉴权后需要使用的鉴权信息authorization，是bearer token
servelessId	String	网关的customer id

ServiceChangeRecord

参数名	参数类型	参数说明
changeId	String	变更ID
prev	String	前一条变更记录
changeType	Integer	变更类型，1:创建 2:更新 3:扩缩容 4:流量接入变更
description	String	变更描述
creator	String	操作人
createdAt	Integer	变更创建时间

InsInfo

参数名	参数类型	参数说明
instanceId	String	实例ID
status	InsStauts	实例状态
containers	ContainerInfo	容器信息

InsStauts

参数名	参数类型	参数说明
blocked	Boolean	是否封锁流量
status	String	实例状态 Creating：该实例正在创建中，包括调度、运行中但未就绪等情况 Running：该实例所有容器均就绪 Exception：实例处在异常状态 Terminating：该实例正在终止
createdAt	Integer	实例创建时间
availableContainers	Integer	就绪的容器数量
totalContainers	Integer	总容器数
podIP	String	实例IP
nodeIP	String	实例所在节点IP
reason	String	pod级别的异常信息

ContainerInfo

参数名	参数类型	参数说明
containerId	String	系统生成的容器唯一ID，非容器名
container	ContainerConf	容器信息
status	ContainerStatus	容器状态

ContainerStatus

参数名	参数类型	参数说明
containerStatus	String	容器状态信息，包括Waiting
createdAt	Integer	状态开始时间，Running状态和Terminated状态返回
reason	String	状态原因，容器 Waiting 或 Terminated 时返回

ScheduleConf

参数名	参数类型	是否必须	参数说明
priority	String	否	资源抢占；优先级（"high" "normal" "low"，默认为 high，即开启资源抢占）

InsGroupStatus

参数名	参数类型	参数说明
id	String	实例组ID
masters	List	master 实例列表
workers	List	worker 实例列表
status	String	实例组状态
blocked	Boolean	流量禁用状态
createdAt	Integer	创建时间
availablePods	Integer	可用实例数
totalPods	Integer	实例总数

ResourceSpec

参数名	参数类型	参数说明
cpus	Integer	cpu数
memory	Integer	内存
acceleratorCount	Integer	卡数
acceleratorType	String	加速芯片类型

AiGatewayConf

参数名	参数类型	是否必须	参数说明
enableAuth	Boolean	是	是否开启鉴权
aigwId	String	是	实例Id
aigwName	String	否	网关实例名称
strategy	String	是	负载均衡策略，round-robin：轮训，least-conn：最小连接数、random：随机
version	String	是	网关版本，v2：目前支持的稳定版本

Dataset

参数名	参数类型	是否必须	参数说明
id	String	是	数据集ID。新建数据集时，无需指定ID。
name	String	是	数据集名称
storageType	String	是	存储类型PFS：并行存储PFSBOS：对象存储BOS
storageInstance	String	是	存储实例
importFormat	String	是	导入格式FILE：文件FOLDER：文件夹
description	String	否	描述
owner	String	是	所有者
ownerName	String	是	所有者名称
visibilityScope	String	是	可见范围ALL_PEOPLE：所有人可见ONLY_OWNER：仅所有者可读写USER_GROUP：指定范围可用
visibilityUser	List<PermissionEntry>	否	用户权限列表[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]仅对管理员或Owner返回此属性
visibilityGroup	List<PermissionEntry>	否	用户组权限列表[{"id": "xxx", "name": "xxx", "permission": "r"}, {"id": "yyy", "name": "xxx", "permission": "rw"}]仅对管理员或Owner返回此属性
permission	String	是	当前用户拥有的读写权限：r：只读rw：读写
latestVersionId	String	否	最新版本ID
latestVersion	String	否	最新版本号
createdAt	String	是	创建时间
updatedAt	String	是	更新时间

DatasetVersionEntry

参数名	参数类型	是否必须	参数说明
id	String	否	数据集版本ID。新建版本时，无需指定ID。
version	String	否	版本号。新建版本时，无需指定版本号。
description	String	否	版本描述
storagePath	String	是	存储路径
mountPath	String	是	默认挂载路径
createUser	String	否	创建用户

PermissionEntry

参数名	参数类型	是否必须	参数说明
id	String	是	用户或用户组ID
name	String	是	用户或用户组名
permission	String	是	权限，枚举值：r：只读rw：读写

Model

参数名	参数类型	是否必须	参数说明
id	String	否	模型ID。新建模型时，无需指定ID。
name	String	是	模型名称
initSource	String	是	模型创建时的来源UserUpload：用户上传
latestVersion	String	否	最新版本
latestVersionId	String	否	最新版本ID
modelFormat	String	是	模型格式
description	String	否	描述
updatedAt	String	否	更新时间
createdAt	String	否	创建时间
owner	String	是	所有者
ownerName	String	是	所有者名称
visibilityScope	String	是	可见范围

ModelVersionEntry

参数名	参数类型	是否必须	参数说明
id	String	否	模型版本ID。新建版本时，无需指定ID。
version	String	否	版本号。新建版本时，无需指定版本号。
source	String	是	该版本模型的来源UserUpload：用户上传
storageBucket	String	是	模型存储的BOS桶
storagePath	String	是	BOS桶中的存储路径
modelMetrics	String	否	模型指标，JSON格式。格式如下，其中 Metrics、Dataset 中的内容用户可自定义填写：{"Results":[{"Metrics":{"loss":2.13,"lr":0.0005},"Dataset":{"DatasetId":"ds-xxx"}}]}
description	String	否	版本描述

DevInstanceConf

参数名	参数类型	是否必须	参数说明
resourcePool	ResourcePool	是	资源池信息；不可修改
resources	Resources	是	资源规格信息
image	Image	是	开发机镜像信息；不可修改
access	Access	否	开发机访问信息（如果不开启 ssh 以及 自定义端口，可以不用）
scheduleConf	ScheduleConf	否	开发机调度配置信息
volumnConfs	List	是	开发机存储挂载信息列表（必须挂载一个 cds 类型的存储）
startCmd	String	否	开发机启动命令
workspaceDir	String	否	开发机 WebIDE 工作目录
envs	MapString	否	开发机环境变量

ResourcePool

参数名	参数类型	是否必须	参数说明
resourcePoolType	String	否	资源池类型（托管资源池为 serverless，自运维资源池为空字符串，默认为空，自然运维资源池）
resourcePoolId	String	否	资源池 ID（托管资源池不需要）
resourcePoolName	String	否	资源池名称（托管资源池不需要）
queueName	String	是	队列名称，（自运维资源池可在资源池队列管理中查看有哪些可选队列，托管资源池则为资源队列 ID）

Resources

参数名	参数类型	是否必须	参数说明
acceleratorType	String	否	加速卡类型（如 baidu.com/a800_80g_cgpu，默认为空，不使用加速卡）
acceleratorCount	Number	否	加速卡数量,默认为0即不使用加速卡，创建GPU开发机需大于0）
cpus	Number	是	cpu核数 （需要大于 0，整数，如 1）
memory	Number	是	内存大小 （需要大于 0 ，整数，单位GiB）
shmSize	Number	否	共享内存大小，单位GiB，默认值为0（为0时会申请内存的一半作为共享内存），如需调整共享内存可设置为其他值

Image

参数名	参数类型	是否必须	参数说明
imageType	Number	是	镜像类型（ 预置镜像 1 自定义镜像 2 ）
imageUrl	String	是	镜像地址
username	String	否	镜像仓库用户名
password	String	否	镜像仓库密码

Access

参数名	参数类型	是否必须	参数说明
blbId	String	否	关联的 blb shortID
sshEnable	Boolean	否	是否开启 ssh（默认不开启）
sshRSAPubKey	String	否	ssh 公钥（如果开启 ssh，此项不能为空）
portInfo	List<PortInfo>	否	端口信息（如果开启 ssh，必须有相应的端口信息，name 为 dev_ssh， port 为 22）

PortInfo

参数名	参数类型	是否必须	参数说明
accessPort	Number	是	访问服务的端口
port	Number	是	容器内服务端口
name	String	是	端口名称

ScheduleConf

参数名	参数类型	是否必须	参数说明
priority	String	否	资源抢占；优先级（"high" "normal" "low"，默认为 high，即开启资源抢占）
cpuNodeAffinity	Bool	否	是否优先调度到CPU节点（默认否）

VolumnConf

参数名	参数类型	是否必须	参数说明
volumeType	String	是	存储类型（支持 cds，pfs、dataset、bos、cfs）
mountPath	String	是	目标路径（容器内挂载路径）
readOnly	Bool	否	是否只读，默认否
bos	Bos	否	bos类型存储信息
cds	Cds	否	cds类型存储信息
pfs	Pfs	否	pfs类型存储信息
dataset	Dataset	否	dataset类型存储信息
cfs	Cfs	否	cfs类型存储信息

CDS

参数名	参数类型	是否必须	参数说明
capacity	Number	是	容量（单位 G，需要大于 100；更新时，容量不能小于更新前）

BOS

参数名	参数类型	是否必须	参数说明
sourcePath	String	是	源路径（填写格式为 BucketName/path）
Version	String	否	bos 版本（目前支持版本 v1，v2；默认根据集群组件版本自动选择，v2优先）
CacheLimitSize	String	否	缓存大小限制

PFS

参数名	参数类型	是否必须	参数说明
instanceId	String	是	pfs 实例ID
sourcePath	String	是	源路径

CFS

参数名	参数类型	是否必须	参数说明
instanceId	String	是	CFS实例ID
sourcePath	String	是	源路径
mountPoint	String	是	挂载点

Dataset

参数名	参数类型	是否必须	参数说明
datasetId	String	是	数据集ID
versionId	String	是	数据集版本ID（数据集详情中 latestVersionId）
storageType	String	是	存储类型（当前只允许值为 pfs 和 bos ）
mountPath	String	是	数据集挂载路径
pfs	DatasetPfsConf	否	数据集 pfs 信息
bos	Bos	否	数据集 bos 信息

DatasetPfsConf

参数名	参数类型	是否必须	参数说明
clientID	String	是	pfs 实例 ID（数据集详情中 storageInstance）
instanceType	String	是	pfs 实例类型 （PFS 详情中的 instanceType）
region	String	是	pfs 地域
instanceId	String	是	pfs 实例 ID （数据集详情中 storageInstance）
clusterIP	String	是	pfs 实例 endpoint
mountTargetId	List	是	pfs 挂载点 （PFS 详情中的 mountTarget 列表中的 mountTargetId）
hostMountPath	String	是	pfs 挂载路径 （PFS 详情中的 mountTarget 中的 mountPath）
srcPath	String	是	源路径 （数据集详情中 storagePath）

VisibleScope

参数名	参数类型	是否必须	参数说明
type	Number	否	可见范围类型 （1 公开可见 0 仅自己可见；默认为仅自己可见）

LoginInfo

参数名	参数类型	参数说明
jupyter	Url	jupyter 登录信息
vscode	Url	vscode 登录信息

URL

参数名	参数类型	参数说明
url	String	访问的 url

ServiceInstanceInfo

参数名	参数类型	参数说明
internalIP	String	访问开发机的内网 IP
nodeIP	String	开发机所在集群节点IP
podName	String	开发机实例 pod 名称
serviceInstanceId	String	开发机底层服务实例ID
publicIP	String	访问开发机的公网IP
cdsInstanceId	String	开发机绑定的 cds 实例 ID

TimedStopDevInstanceDetail

参数名	参数类型	参数说明
delaySec	String	定时关机自设定时间后多少秒关机
startTime	String	定时关机的设定时间

ImagePackJobBrief

参数名	参数类型	参数说明
reason	String	镜像制作任务失败原因
id	String	镜像制作任务ID

NotifyDetail

参数名	参数类型	参数说明
isOpen	String	是否开启通知
notifyRuleId	String	通知规则ID

Event

参数说明	类型	说明
reason	String	原因
message	String	详细信息
firstTimestamp	String	首次出现时间
lastTimestamp	String	最后出现时间
count	Number	出现次数
type	Number	事件类型

DevInstanceDetail

参数说明	类型	说明
accountId	String	账户id
id	String	开发机ID
name	String	开发机名称
conf	DevInstanceConf	开发机配置信息
serviceInstanceInfo	ServiceInstanceInfo	开发机服务实例信息
creator	String	开发机所有者
creatorId	String	开发机所有者ID
notify	NotifyDetail	通知信息
region	String	地域
status	Number	开发机状态
statusReason	String	开发机状态原因
version	String	开发机版本
visisbleScope	VisibleScope	可见范围
loginInfo	LoginInfo	登录信息
timedStopDevInstance	TimedStopDevInstanceDetail	开发机定时关机信息
isPublicMgmt	bool	是否公共管理
createdAt	Number	创建时间
updatedAt	Number	更新时间

status	语义
0	创建中
1	排队中
2	部署中
3	运行中
4	实例停止中
5	实例停止
6	实例开启中
7	实例开启
10	镜像制作中
11	删除中
18	失败
19	异常
20	已删除

ImagePackJobDetail

devInstanceId	String	开发机ID
id	String	镜像任务 ID
imageName	String	镜像名称
imageTag	String	镜像tag
namespace	String	镜像仓库命名空间
registry	String	镜像仓库
status	Number	任务状态
updatedAt	Number	开发机配置最近一次变更时间

status	说明
1	镜像任务初始化
2	镜像任务运行中
3	镜像制作中
4	镜像推送中
5	镜像制作任务完成
30	镜像制作任务失败

AcceleratorCard

参数名	参数类型	是否必须	参数说明
acceleratorType	String	是	计算卡类型
acceleratoCount	String	是	计算卡数量
description	String	否	计算卡描述符

ResourceAmount

参数名	参数类型	是否必须	参数说明
milliCPUcores	String	否	cpu 统计，单位为 m
memoryGi	String	否	内存统计，单位为 Gi
acceleratorCardList	List<AcceleratorCard>	否	计算卡信息统计

BindingNodeInfo

参数名	参数类型	是否必须	参数说明
acceleratorType	String	否	计算卡类型
machineSpec	String	否	节点所属套餐
count	Integer	是	物理队列中 machineSpec 节点数量
nodeNameList	List	是	物理队列中 machineSpec 节点列表
storagePath	String	是	BOS桶中的存储路径
modelMetrics	String	否	模型指标，JSON格式。格式如下，其中 Metrics、Dataset 中的内容用户可自定义填写
description	String	否	版本描述

QueueItem

单个队列详情信息，被如下接口引用：查询队列详情、查询队列列表

参数说明	类型	说明
resourceQueueId	String	队列 ID
resourceQueueName	String	队列名称
description	String	描述
createdAt	String	创建时间
updatedAt	String	更新时间
resourceQueueType	String	队列类型（regular， elastic或者physical）
resourcePoolID	String	所属资源池 ID
parentQueue	String	父队列 ID
children	List	子队列列表
opened	Boolean	队列是否开启
reclaimable	Boolean	是否可回收
preemptable	Boolean	是否开启队列内优先级抢占
disableOversell	Boolean	禁止超用
queueingStrategy	String	设置队列调度策略（BestEffortFIFO 或者 StrictFIFO）
enableVGPU	Boolean	是否开启虚拟化
capability	ResourceAmount	队列配额上限
deserved	ResourceAmount	队列配额申请
guarantee	ResourceAmount	队列配额预留
allocated	ResourceAmount	队列分配情况
runningJobs	Integer	队列运行 podgroup 数量
inqueueJobs	Integer	队列入队 podgroup 数量
pendingJobs	Integer	队列等待入队 podgroup 数量
bindingNodes	List	队列节点列表
maxDeservedForSubqueue	ResourceAmount	子队列可分配 deserved 配额
maxGuaranteeForSubqueue	ResourceAmount	子队列可分配 guarantee 配额

ResourcePoolSpec

参数说明	类型	说明
region	String	地域
resourcePoolId	String	资源池 ID
type	String	资源池类型：1、通用资源池：common，2、托管资源池：dedicatedV2
name	String	资源池名称
description	String	资源池描述
exposedPublic	Boolean	是否暴露公网，默认否
forbidDelete	Boolean	是否打开删除保护，默认是
deschedulerEnabled	Boolean	是否打开碎片治理，默认关闭
unifiedSchedulerEnabled	Boolean	是否打开仅使用百舸调度器，默认关闭
associatedResources	List<AssociateResource>	关联的资源列表
network	Network	网络配置
bindingStorages	List<ProviderInfo>	绑定的存储配置，支持延迟绑定/单个/一组
bindingMonitors	List<ProviderInfo>	绑定监控配置，支持延迟绑定/单个/一组
phase	String	资源池状态
nodeNum	Number	节点数
createdAt	String	创建时间
updatedAt	String	更新时间

Network

参数说明	类型	说明
mode	String	网络模式：vpc路由/vpc-eni
master	List<NetworkInfo>	master网络配置
nodes	List<NetworkInfo>	节点网络配置
pods	List<NetworkInfo>	容器网络配置
clusterIPCidr	String	clusterIP网段
loadBalanceService	List<NetworkInfo>	LB Service网络配置
maxPodsPerNode	Number	节点最大 Pod数，vpc路由模式下必填，vpc-eni模式下选填

ProviderInfo

参数说明	类型	说明
provider	String	提供方产品代号，eg CProm/PFS
type	String	产品资源类型，比如pfsl1/pfsl2
id	String	产品资源唯一ID
options	Map<String,String>	额外参数
region	String	地域，默认与资源池地域一致
zone	String	可用区，默认标识不指定可用区

NetworkInfo

参数说明	类型	说明
region	String	地域
vpcId	String	vpc短实例 ID
vpcUuid	String	vpc长实例 ID
vpcCidr	String	vpc子网掩码
subnetIDs	List<String>	子网 ID 列表
subnetCidr	String	子网掩码，适用于vpc路由模式下
securityGroups	List<SecurityGroup>	安全组信息

SecurityGroup

参数说明	类型	说明
type	String	安全组类型，企业/普通
id	String	安全组 ID，开启自动生成后，可选
name	String	安全组名称，开启自动生成后，可选

AssociateResource

参数说明	类型	说明
provider	String	提供方产品代号，eg CCE/BCC
id	String	产品资源短ID
uuid	String	产品资源长ID
region	String	地域，默认与资源池地域一致
zone	String	可用区，默认标识不指定可用区

Configuration

参数说明	类型	说明
forbidDelete	String	是否打开删除保护，默认是
deschedulerEnabled	String	是否打开碎片治理，默认关闭
unifiedSchedulerEnabled	String	是否打开仅使用百舸调度器，默认关闭

ResourcePoolsStatisticWithType

参数说明	类型	说明
type	String	资源池类型
nodesStatistic	NodesStatistic	节点统计信息
acceleratorsStatistic	AcceleratorsStatistic	加速卡统计信息
resourcesStatistic	List<ResourcesStatistic>	资源统计信息
resourcePoolStatistics	List<ResourcePoolStatistics>	各资源池统计信息

NodesStatistic

参数说明	类型	说明
totalCount	Number	节点总数

AcceleratorsStatistic

参数说明	类型	说明
resourceType	String	资源类型
capacity	Number	容量总数
allocated	Number	已分配总数
idle	Number	空闲总数
allocationRate	Number	分配率
utilizationRate	Number	使用率
acceleratorStatistics	Number	按照卡的类型进行统计

AcceleratorStatistics

参数说明	类型	说明
acceleratorType	String	加速芯片类型
resourceType	String	资源类型
capacity	Number	容量总数
allocated	Number	已分配总数
idle	Number	空闲总数
allocationRate	Number	分配率
utilizationRate	Number	使用率

ResourcePoolStatistics

参数说明	类型	说明
resourcePoolID	String	资源池id
type	String	资源池类型
name	String	资源池名称
nodesStatistic	List<NodesStatistic>	节点统计信息
resourcesStatistic	List<String>	资源统计信息
acceleratorsStatistic	List<AcceleratorStatistics>	加速卡统计信息

ResourcesStatistic

参数说明	类型	说明
resourceType	String	资源类型
capacity	Number	容量总数
allocated	Number	已分配总数
idle	Number	空闲总数
allocationRate	Number	分配率
utilizationRate	Number	使用率

StopBatchResult

批量停止训练任务-批量停止

参数名称	类型	说明
successNum	Number	成功任务数量
failedNum	Number	失败任务数量
success	Boolean	整体是否全部成功
successList	List<JobResult>	成功任务列表
failedList	List<JobResult>	失败任务列表（无失败则为空）

JobResult

批量停止训练任务-批量停止

参数名称	类型	说明
jobId	String	训练任务唯一标识id
success	Boolean	任务是否处理成功（成功列表固定为 true，失败列表固定为 false）
errorMsg	String	失败原因描述，失败时返回，成功返回空

CreateSpec

参数名称	类型	是否必须	说明
region	String	是	地域：支持 bj、bd、su、gz、yq、hkg、cd）
name	String	是	资源池名称：支持大小写字母、数字、中文以及-_ /.特殊字符，且必须以字母开头,最长65个字符
description	String	否	资源池描述信息：最长支持300字符）
type	String	是	资源池类型：支持 dedicatedV2（托管资源池）
network	Network	是	资源池网络配置信息：其中 1、nodes 节点子网必填 vpcId，subnetIds（只能配置一个），2、pods 容器子网选填，只需配置 （可以配置多个），会自动添加节点子网
bindingStorages	List<ProviderInfo>	否	资源池绑定存储配置信息：其中 1、只支持配置一个，2、provider 为 pfs，3、type 支持 pfsL2、pfsMT，4、只需要配置 provider、type、id
bindingMonitors	List<ProviderInfo>	否	资源池绑定监控配置信息：其中 1、只支持配置一个，2、provider 为 cprom，3、只需要配置 provider、id

NodeSet

参数名称	类型	是否必须	说明
addNodeSpec	AddNodeSpec	是	节点规格配置
count	Number	是	添加的节点数量：创建个数取值范围 1 ~ 50

AddNodeSpec

参数名称	类型	是否必须	说明
machineSpec	String	是	机器规格配置，eg：ehc.lgn5ne.c128m1024.8a800.8re.4d
zoneName	String	是	机器规则所在可用区，eg：cn-bj-d
resourceChargingOption	ResourceChargingOption	是	资源计费配置
ehcClusterId	String	否	机器所属ehc集群id

ResourceChargingOption

参数名称	类型	是否必须	说明
chargingType	String	是	资源计费类型：1、支持 Prepaid（预付费）、Postpaid（后付费）2、预付费时创建会直接自动付费
purchaseTime	Number	否	预付费购买时长：1、Prepaid 时必填，2、支持 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 24, 36
purchaseTimeUnit	String	否	预付费购买时长单位：1、Prepaid 时必填，2、支持 MONTH
autoRenew	Boolean	否	是否自动续费：1、Prepaid 时必填，默认为 false，2、后付费不支持
autoRenewTime	Number	否	预付费购买时长：1、autoRenew 为 true 时必填，2、支持 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 24, 36
autoRenewTimeUnit	String	否	预付费购买时长单位：1、autoRenew 为 true 时必填，支持 MONTH

BindingNodeInfo

参数名称	类型	是否必须	说明
machineSpec	String	是	机器规格配置，eg：ehc.lgn5ne.c128m1024.8a800.8re.4d
zoneName	String	是	机器规则所在可用区，eg：cn-bj-d
count	Number	是	添加的节点数量

Node

参数名称	类型	说明
nodeSpec	NodeSpec	节点配置
nodeStatus	NodeStatus	节点状态

NodeSpec

参数名称	类型	说明
region	String	地域
zone	String	可用区
flavorName	String	套餐规格
resourcePoolId	String	资源池ID
accountId	String	账号ID
userId	String	用户ID
instanceName	String	实例名称
nodeName	String	节点名称
instanceType	String	实例类型
instanceSpec	String	实例类型
instanceId	String	实例ID
resourceChargingOption	ResourceChargingOption	付费类型

NodeStatus

参数名称	类型	说明
ehcClusterId	String	ehc 集群 ID
statusPhase	String	节点状态
createdAt	String	创建时间
billResourceId	String	billing资源ID
resourceQueueId	String	队列ID
nodeIntervention	NodeIntervention	节点干预
nodeResource	NodeResource	节点资源

NodeIntervention

参数名称	类型	说明
cordon	Boolean	是否封锁

NodeResource

参数名称	类型	说明
isAcclerator	Boolean	是否是加速卡节点
accleratorType	String	加速卡类型
accleratorDescriptor	String	加速卡描述符
capacity	NodeResourceAmount	资源容量

NodeResourceAmount

参数名称	类型	说明
milliCPUcores	Number	CPU，单位 millicore
memoryBytes	Number	内存，单位 byte
cpuCores	Number	CPU，单位 core
memoryGi	Number	内存，单位 Gi
gpuNum	Number	GPU 卡数

ServiceGroupBrief

参数名称	类型	说明
id	String	服务组 ID
groupName	String	服务组名称
resourcePoolId	String	资源池 ID，由后端根据队列自动填写
resourcePoolType	String	资源池类型，固定为serverless
creator	String	创建者，即发起创建请求的 IAM 用户名
createdAt	Int	创建时间，Unix 秒
updatedAt	Int	更新时间，Unix 秒
trafficStrategyId	String	关联的流量策略 ID，没有关联时不返回
trafficStrategyPort	Int	流量策略端口，没有关联时不返回

ServiceGroupDetail

参数名称	类型	说明
serviceGroupId	String	服务组 ID
groupName	String	服务组名称
resourcePool	ResourcePoolConf	资源池信息。resourcePoolId、resourcePoolType 有值；queueName、resourcePoolName 目前不返回具体内容（为空串）
aiGateway	AiGatewayConf	网关配置：aigwId、aigwName、enableAuth、version，以及创建时填写过的 strategy、protocol
roles	Array of RoleBrief	各角色概要
llmScheduler	LlmSchedulerConf	调度器配置，没有创建调度器时为空对象
llmSchedulerInstances	Array of llmSchedulerInsInfo	调度器实例列表，没有创建调度器时没有该字段
publicModel	PublicModelConf	公共模型配置
creator	String	创建者，即发起创建请求的 IAM 用户名
createdAtt	Int	创建时间，Unix 秒
updatedAt	Int	更新时间，Unix 秒
trafficStrategyId	String	关联了流量策略时才返回
trafficStrategyPort	Int	关联了流量策略时才返回

RoleBrief

参数名称	类型	说明
id	String	角色对应的服务 ID
roleName	String	角色名
workloadType	String	工作负载类型，空时不返回
resourceSpec	ResourceSpec	单个实例的资源：cpus、memory、acceleratorCount、acceleratorType
briefStat	ServiceBriefStat	角色状态概要：status（状态码）、availableIns（可用实例数）、totalIns（总实例数）、reason（异常原因）
createdAt	Int	创建时间，Unix 秒
updatedAt	Int	更新时间，Unix 秒

llmSchedulerInsInfo

参数名称	类型	说明
instanceId	String	实例 ID
status	InsStauts	实例状态：status、availableContainers、totalContainers、podIP、nodeIP、blocked、reason、createdAt
containers	Array of ContainerConf	实例里的容器，每项含containerId、container（容器配置）和 status（containerStatus、reason、createdAt）

LlmSchedulerConf

参数名称	参数类型	是否必须	参数说明
enable	Bool	否	是否启用调度器。只有为true 时才校验下面的字段
replicas	Int	条件必填	启用时必填，副本数，必须大于等于 1
resourceSpec	SchedResSpec	条件必填	启用时必填，cpus（核数）和 memory（GB）都必须大于 0
policy	String	否	调度策略，可选pd_split、random、round_robin、prefix_aware
pdPolicy	PdSchedPolicy	否	policy 为 pd_split 时生效，prefillPolicy、decodePolicy 分别指定 prefill、decode 的调度策略，取值不合法报 InvalidParameter
inferEngine	String	是	推理引擎，仅支持sglang。

SchedResSpec

参数名称	参数类型	是否必须	参数说明
cpus	Int	是	LLM 调度器的 CPU 使用量
memory	Int	是	LLM 调度器的内存使用量

PdSchedPolicy

参数名称	参数类型	是否必须	参数说明
prefillPolicy	String	是	调度策略，可选random、round_robin、prefix_aware，推荐prefix_aware
decodePolicy	String	是	调度策略，可选random、round_robin、prefix_aware

模型评测相关接口

查询评测集的默认配置

查询评测集的默认配置 DescribeEmbodiedEvalBenchmarks

接口描述

查询一个模型在各评测集、子集上的默认配置。每个 (modelName, name, subBench) 组合返回一项，defaultBenchmarkConfig 可以直接作为创建请求 benchmarkConfigs 的一项。

权限说明

具备百舸（AIHC）的任一权限即可。

请求结构

```
{
 "method": "GET",
 "uri": "/?action=DescribeEmbodiedEvalBenchmarks&modelName={modelName}&benchmarkName={benchmarkName}&subBench={subBench}&source={source}",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "{authorization}",
   "version": "v2"
 }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
modelName	String	是	Query参数	模型名称
benchmarkName	String	否	Query参数	评测集名称，不传时返回该模型的全部评测集
subBench	String	否	Query参数	子集名称，不传时返回评测集的全部子集
source	String	否	Query参数	同 DescribeEmbodiedEvalModels，默认 public。用自己的权重或镜像时传 custom

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
benchmarks	BenchmarkMetadata[]	每个组合的默认配置

请求示例

```
{
 "method": "GET",
 "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEvalBenchmarks&modelName=openvla_7b&benchmarkName=libero&subBench=libero_spatial",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
   "x-bce-date": "2026-11-04T08:00:00Z",
   "version": "v2"
 }
}
```

响应示例

```
{
 "requestId": "a2b3c4d5-e6f7-4089-b2ad-1e2f3a4b5c6d",
 "benchmarks": [
  {
   "modelName": "openvla_7b",
   "name": "libero",
   "subBench": "libero_spatial",
   "minHardware": {
    "acceleratorTypes": ["baidu.com/l20_cgpu", "baidu.com/rtx_4090_cgpu", "baidu.com/a10_24g_cgpu", "baidu.com/a800_80g_cgpu"],
    "acceleratorCount": 0,
    "cpu": 8,
    "memory": 16
   },
   "defaultBenchmarkConfig": {
    "name": "libero",
    "subBench": "libero_spatial",
    "resources": [
     {"name": "baidu.com/l20_cgpu", "quantity": 1},
     {"name": "cpu", "quantity": 8},
     {"name": "memory", "quantity": 32}
    ],
    "evalType": "sync",
    "parallelShards": 1,
    "episodesPerTask": 50,
    "params": {"seed": 7, "num_steps_wait": 10, "absolute_action": false, "max_steps": null, "env_seed": null, "suite": "libero_spatial"},
    "recordVideo": false,
    "modelOverride": {
     "ckptPath": "openvla/openvla-7b-finetuned-libero-spatial",
     "chunkSize": 1,
     "extraArgs": {"unnorm_key": "libero_spatial", "jpeg_roundtrip": true, "center_crop": true}
    }
   },
   "paramsSchema": {
    "requiredArgs": [],
    "optionalArgs": ["seed", "num_steps_wait", "send_wrist_image", "send_state", "absolute_action", "max_steps", "env_seed"],
    "properties": {
     "seed": {"type": "integer", "default": 7, "description": "环境初始化随机种子"},
     "max_steps": {"type": "integer", "default": null, "nullable": true, "description": "覆盖 suite 默认最大步数；null 时按子集默认值"}
    }
   },
   "maxEpisodesPerTask": 50
  }
 ]
}
```


注意事项

- defaultBenchmarkConfig.resources 中的卡型、episodesPerTask 是推荐值，按需修改。episodesPerTask 不能超过 maxEpisodesPerTask（为 0 时不限）。
- minHardware.acceleratorCount 为 0 表示该评测集可以不用 GPU（CPU 渲染），此时 resources 里可以不放 GPU。
- modelOverride.ckptPath、extraArgs 是预置模型在该子集上的推荐权重和参数。用自己的权重时把 ckptPath 换成自己的路径。
- 模型不存在返回 404；模型存在但按 benchmarkName、subBench 过滤后没有组合时，返回空列表。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	没有传 modelName；source 取值不合法
404	NotFound	模型不存在

查询模型的默认配置

查询模型的默认配置 DescribeEmbodiedEvalModel

接口描述

查询一个模型的默认配置和最低硬件要求。defaultModelConfig 可以直接作为创建请求的 modelConfig。

权限说明

具备百舸（AIHC）的任一权限即可。

请求结构


```
{
 "method": "GET",
 "uri": "/?action=DescribeEmbodiedEvalModel&modelName={modelName}",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "{authorization}",
   "version": "v2"
 }
}
```


请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
modelName	String	是	Query参数	模型名称，取自 DescribeEmbodiedEvalModels

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
modelName / displayName	String	模型名称、展示名称
minHardware	MinHardware	模型服务单副本的最低硬件要求
defaultModelConfig	ModelConfig	推荐的模型服务配置
extraArgsSchema	ParamSchema	benchmarkConfigs[].modelOverride.extraArgs 可以填的参数
chunkSizeLimit	Integer	modelOverride.chunkSize 的建议上限
maxBatchSizeLimit	Integer	modelConfig.maxBatchSize 的建议上限

请求示例

```
{
 "method": "GET",
 "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEvalModel&modelName=openvla_7b",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
   "x-bce-date": "2026-11-04T08:00:00Z",
   "version": "v2"
 }
}
```

响应示例

```
{
 "requestId": "91a2b3c4-d5e6-4f78-a19c-0d1e2f3a4b5c",
 "modelName": "openvla_7b",
 "displayName": "OpenVLA",
 "minHardware": {
   "acceleratorTypes": ["baidu.com/l20_cgpu", "baidu.com/rtx_4090_cgpu", "baidu.com/a10_24g_cgpu", "baidu.com/a800_80g_cgpu"],
   "acceleratorCount": 1,
   "cpu": 4,
   "memory": 16
 },
 "defaultModelConfig": {
   "modelType": "public",
   "modelName": "openvla_7b",
   "replicas": 1,
   "maxBatchSize": 1,
   "resources": [
     {"name": "baidu.com/l20_cgpu", "quantity": 1},
     {"name": "cpu", "quantity": 8},
     {"name": "memory", "quantity": 32}
   ]
 },
 "extraArgsSchema": {
   "requiredArgs": ["unnorm_key"],
   "optionalArgs": ["unnorm_key", "jpeg_roundtrip", "center_crop", "action_ensemble"],
   "properties": {
     "unnorm_key": {"type": "string", "default": null, "nullable": true, "enum": ["libero_spatial", "libero_object", "libero_goal", "libero_10", "bridge_orig"], "description": "动作反归一化使用哪一组训练集统计量"},
     "jpeg_roundtrip": {"type": "boolean", "default": false, "description": "模拟 OpenVLA 官方评测的 JPEG 预处理"},
     "center_crop": {"type": "boolean", "default": false, "description": "模拟 OpenVLA 官方评测的中心裁剪预处理"},
     "action_ensemble": {"type": "string", "default": "newest", "enum": ["newest", "average", "ema"], "description": "动作 chunk 跨步重叠时的融合策略"}
   }
 },
 "chunkSizeLimit": 8,
 "maxBatchSizeLimit": 1
}
```

注意事项

- defaultModelConfig.resources 里的卡型只是示例，要换成自己队列里的卡型，且必须在 minHardware.acceleratorTypes 里。
- chunkSizeLimit、maxBatchSizeLimit 是建议值，超过时创建不会报错，但可能导致显存不足或结果异常。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	没有传 modelName
404	NotFound	模型不存在

查询可评测的模型及其评测集

查询可评测的模型及其评测集 DescribeEmbodiedEvalModels

接口描述

查询可评测的模型，以及每个模型可以跑的评测集和子集。

权限说明

具备百舸（AIHC）的任一权限即可。

请求结构

```
{
 "method": "GET",
 "uri": "/?action=DescribeEmbodiedEvalModels&source={source}",
 "headers": {
  "Host": "aihc.bj.baidubce.com",
  "Authorization": "{authorization}",
  "version": "v2"
 }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
source	String	否	Query参数	public：平台预置模型及其经过验证的评测集组合，默认值。custom：使用自己的权重或镜像时用，每个模型都列出全部评测集

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
models	ModelItem[]	模型列表

请求示例

```
{
 "method": "GET",
 "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEvalModels&source=public",
 "headers": {
  "Host": "aihc.bj.baidubce.com",
  "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
  "x-bce-date": "2026-11-04T08:00:00Z",
  "version": "v2"
 }
}
```

响应示例

```
{
 "requestId": "8091a2b3-c4d5-4e67-f08b-9c0d1e2f3a4b",
 "models": [
  {
   "modelName": "openvla_7b",
   "displayName": "OpenVLA",
   "benchmarks": [
    {"name": "libero", "displayName": "LIBERO", "subBenches": ["libero_spatial", "libero_object", "libero_goal", "libero_10"]}
   ]
  },
  {
   "modelName": "pi05_libero",
   "displayName": "Pi0.5",
   "benchmarks": [
    {"name": "libero", "displayName": "LIBERO", "subBenches": ["libero_spatial", "libero_object", "libero_goal", "libero_10"]}
   ]
  }
 ]
}
```

注意事项

- 用自己的权重（modelType=customCKPT）时，modelName 选一个和自己权重同架构的预置模型，平台用它的推理服务镜像加载你的权重；评测集组合用 source=custom 查。
- 用自己的镜像（modelType=customImage）时不需要 modelName，评测集同样用 source=custom 查，查默认配置时 modelName 填任一模型即可。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	source 不是 public / custom

查询回放视频播放地址

查询回放视频播放地址 DescribeEmbodiedEvalVideo

接口描述

获取一条 episode 回放视频的临时播放地址。只有创建时 recordVideo 为 true 的评测集会产出视频，videoId 取自 DescribeEmbodiedEvalResult 返回的 benchmarks[].details.tasks[].videos[].video_id。

权限说明

需要对该任务的查看权限。

请求结构

```
{
  "method": "GET",
  "uri": "/?action=DescribeEmbodiedEvalVideo&evalId={evalId}&videoId={videoId}",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "{authorization}",
    "version": "v2"
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
evalId	String	是	Query参数	评测任务 ID
videoId	String	是	Query参数	视频 ID，以 v- 开头

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
videoId	String	视频 ID
streamUrl	String	视频的临时访问地址，支持 HTTP Range 请求，可直接在浏览器或播放器里打开
expiresIn	Integer	streamUrl 的有效期，单位秒，目前为 900
mimeType	String	视频格式，如 video/mp4
sizeBytes	Integer	视频大小，单位字节

请求示例

```
{
  "method": "GET",
  "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEvalVideo&evalId=em-6o004i76d88e&videoId=v-ZW0tNm8wMDRpNzZkODhlL3ZpZGVvcy9saWJlcm8tbGliZXJvX3NwYXRpYWwvc2hhcmQwL3Rhc2swMDAwX2VwMDAwMC5tcDQ",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
    "x-bce-date": "2026-11-04T08:00:00Z",
    "version": "v2"
  }
}
```

响应示例

```
{
  "requestId": "7f8091a2-b3c4-4d56-ef7a-8b9c0d1e2f3a",
  "videoId": "v-ZW0tNm8wMDRpNzZkODhlL3ZpZGVvcy9saWJlcm8tbGliZXJvX3NwYXRpYWwvc2hhcmQwL3Rhc2swMDAwX2VwMDAwMC5tcDQ",
  "streamUrl": "https://aihc-eval-xxxx-bj.bj.bcebos.com/results/...?authorization=...",
  "expiresIn": 900,
  "mimeType": "video/mp4",
  "sizeBytes": 1048576
}
```

注意事项

- streamUrl 本身带有访问凭证，持有链接即可下载视频，不要转发给无关人员。过期后重新调用本接口获取。
- videos[].state 为 failed 的视频没有可播放的文件，调用会返回 404。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	缺少 evalId 或 videoId；videoId 格式不对或不属于该任务
403	Forbidden	没有查看该任务的权限
404	NotFound	任务不存在；视频文件不存在

查询评测结果

查询评测结果 DescribeEmbodiedEvalResult

接口描述

查询评测结果。任务运行期间结果文件每隔一段时间刷新一次，可以拿到各评测集的进度；任务成功后返回最终评分和每个任务的明细。

权限说明

需要对该任务的查看权限。

请求结构

```
{
 "method": "GET",
 "uri": "/?action=DescribeEmbodiedEvalResult&evalId={evalId}",
 "headers": {
  "Host": "aihc.bj.baidubce.com",
  "Authorization": "{authorization}",
  "version": "v2"
 }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
evalId	String	是	Query参数	评测任务 ID

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
evalId	String	评测任务 ID
result	EvalResult	评测结果。结果文件还没有产出时为 null

请求示例

```
{
 "method": "GET",
 "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEvalResult&evalId=em-6o004i76d88e",
 "headers": {
  "Host": "aihc.bj.baidubce.com",
  "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
  "x-bce-date": "2026-11-04T08:00:00Z",
  "version": "v2"
 }
}
```

响应示例

```
{
 "requestId": "6e7f8091-a2b3-4c45-de6f-7a8b9c0d1e2f",
 "evalId": "em-6o004i76d88e",
 "result": {
  "version": "1",
  "job_id": "job-a1b2c3d4e5f6",
  "started_at": "2026-11-04T02:01:05Z",
  "finished_at": "2026-11-04T02:43:02Z",
  "video_summary": {"enabled": true, "total": 100, "ready": 100, "failed": 0},
  "benchmarks": [
   {
    "id": "libero-libero_spatial",
    "name": "libero",
    "sub_bench": "libero_spatial",
    "status": "succeeded",
    "progress": {"completed_episodes": 100, "total_episodes": 100, "num_errors": 0, "num_timeouts": 3},
    "config": {
     "mode": "sync", "num_shards": 1, "episodes_per_task": 10, "max_steps": 220, "max_tasks": null, "tasks": null,
     "chunk_size": 1,
     "params": {"seed": 7, "num_steps_wait": 10, "absolute_action": false, "max_steps": null, "env_seed": null, "suite": "libero_spatial"}
    },
    "metrics": {"mean_success": 0.84, "num_tasks": 10},
    "details": {
     "tasks": [
      {
       "task": "pick up the black bowl between the plate and the ramekin and place it on the plate",
       "mean_success": 0.9,
       "avg_steps": 132.4,
       "num_episodes": 10,
       "num_errors": 0,
       "num_timeouts": 0,
       "videos": [
        {"video_id": "v-ZW0tNm8wMDRpNzZkODhlL3ZpZGVvcy9saWJlcm8tbGliZXJvX3NwYXRpYWwvc2hhcmQwL3Rhc2swMDAwX2VwMDAwMC5tcDQ", "episode_index": 0, "success": true, "state": "ready"}
       ]
      }
     ]
    }
   }
  ]
 }
}
```

注意事项

- 任务成功前 result 可能是 null，也可能是运行中的进度快照（benchmarks[].metrics、details 为 null，finished_at 为 null）。以任务状态为 Succeeded 后取到的结果为准。
- 部分分片失败时，任务仍可能成功结束，对应评测集的 status 为 partial，评分只统计成功跑完的分片，使用前请检查。
- result 内部字段随评测集变化，本文只保证 EvalResult 中列出的字段。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	没有传 evalId
403	Forbidden	没有查看权限，或任务已成功但结果文件读取被拒绝
404	NotFound	任务不存在；或任务已成功但结果文件不存在
500	InternalError	任务已成功但没有结果路径

删除评测任务

删除评测任务 DeleteEmbodiedEval

接口描述

删除一个评测任务。运行中的任务会先被终止，再删除记录。删除后任务不能再被查询。

权限说明

资源池管理员可以删除池内任意任务；队列成员只能删除自己创建的任务。只有只读权限的用户不能删除。

请求结构

```
{
  "method": "POST",
  "uri": "/?action=DeleteEmbodiedEval&evalId={evalId}",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "{authorization}",
    "version": "v2",
    "Content-Type": "application/json"
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
evalId	String	是	Query参数	评测任务 ID，一次只能删除一个任务
clientToken	String	否	Query参数	幂等键，不超过 64 字节

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
evalId	String	被删除的任务 ID

请求示例

```
{
  "method": "POST",
  "url": "https://aihc.bj.baidubce.com/?action=DeleteEmbodiedEval&evalId=em-6o004i76d88e",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
    "x-bce-date": "2026-11-04T08:00:00Z",
    "version": "v2",
    "Content-Type": "application/json; charset=utf-8"
  }
}
```

响应示例

```
{
  "requestId": "5d6e7f80-91a2-4b34-cd5e-6f7a8b9c0d1e",
  "evalId": "em-6o004i76d88e"
}
```

注意事项

- 删除成功后再删除同一个任务返回 404，重试时可以把 404 当作已删除。
- 任务删除后，它使用过的 clientToken 可以重新使用。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	没有传 evalId
403	Forbidden	没有删除该任务的权限
404	NotFound	任务不存在或已删除
503	ResourceBusy	任务还在提交中
500	InternalError	删除失败，可重试

创建评测任务

创建评测任务 CreateEmbodiedEval

接口描述

创建一个评测任务：用一个模型服务配置，跑 1～20 个评测集（子集）。创建成功只表示任务已提交，任务会先排队再运行，用 DescribeEmbodiedEval 查进度。

权限说明

调用人需要是目标队列的成员，或者是该资源池的管理员。只有只读权限的用户不能创建。

请求结构

```
{
  "method": "POST",
  "uri": "/?action=CreateEmbodiedEval&clientToken={clientToken}",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "{authorization}",
    "version": "v2",
    "Content-Type": "application/json"
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
clientToken	String	否	Query参数	幂等键，长度不超过 64 字节，规则见下文
name	String	是	RequestBody参数	任务名称，不超过 64 字节
resourcePoolId	String	是	RequestBody参数	资源池 ID
queueId	String	是	RequestBody参数	队列 ID
resourcePoolType	String	否	RequestBody参数	资源池类型。使用全托管（Serverless）资源池时填 serverless，否则不传
priority	String	否	RequestBody参数	任务优先级：high / normal / low，默认 normal
visibleScope	Integer	否	RequestBody参数	可见范围：1 队列内成员可见，0 仅创建人可见。默认 1
dataSources	DataSource[]	否	RequestBody参数	挂载到任务里的数据源，用于读取自己的模型权重。目前只支持 bos 类型
modelConfig	ModelConfig	是	RequestBody参数	模型服务配置
benchmarkConfigs	BenchmarkConfig[]	是	RequestBody参数	评测集配置，1～20 个

clientToken 的使用规则：

- 每次创建生成一个新的值（如 UUID），只有重试**同一次**创建时才传回同一个值。不传时不做幂等，每次调用都新建任务。
- 同一账号下，同一个 clientToken 只会创建一个任务：重试时返回原来的 evalId；参数和第一次不一样时返回 403 IdempotentParameterMismatch。
- clientToken 没有有效期，只有对应任务被删除或创建失败后才能再用。不要给所有请求写死同一个值，否则从第二个任务起都会返回第一个任务的 evalId 或报参数冲突。

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
evalId	String	评测任务 ID，格式为 em- 加 12 位字符

请求示例

用平台预置的 OpenVLA 模型跑 LIBERO 的 libero_spatial 子集，每个任务 10 个 episode。资源池、队列换成自己的；GPU 卡型要和队列里的卡一致，且在 DescribeEmbodiedEvalModel 返回的 minHardware.acceleratorTypes 里。clientToken 每次创建生成一个新值，重试同一次创建时传回同一个值。

```
{
 "method": "POST",
 "url": "https://aihc.bj.baidubce.com/?action=CreateEmbodiedEval&clientToken=4a7f3c2e-0b8d-4f5a-9c61-2d3e4f5a6b7c",
 "headers": {
  "Host": "aihc.bj.baidubce.com",
  "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
  "x-bce-date": "2026-11-04T08:00:00Z",
  "version": "v2",
  "Content-Type": "application/json; charset=utf-8"
 },
 "body": {
  "name": "openvla-libero-spatial",
  "resourcePoolId": "cce-xxxxxxxx",
  "queueId": "default",
  "priority": "normal",
  "modelConfig": {
   "modelType": "public",
   "modelName": "openvla_7b",
   "replicas": 1,
   "maxBatchSize": 1,
   "resources": [
    {"name": "baidu.com/a800_80g_cgpu", "quantity": 1},
    {"name": "cpu", "quantity": 8},
    {"name": "memory", "quantity": 32}
   ]
  },
  "benchmarkConfigs": [
   {
    "name": "libero",
    "subBench": "libero_spatial",
    "resources": [
     {"name": "baidu.com/a800_80g_cgpu", "quantity": 1},
     {"name": "cpu", "quantity": 8},
     {"name": "memory", "quantity": 32}
    ],
    "parallelShards": 1,
    "episodesPerTask": 10,
    "recordVideo": true,
    "modelOverride": {"chunkSize": 1}
   }
  ]
 }
}
```

响应示例

```
{
 "requestId": "1f2e3d4c-5b6a-4789-8a9b-0c1d2e3f4a5b",
 "evalId": "em-6o004i76d88e"
}
```

注意事项

- 最省事的写法是先调 DescribeEmbodiedEvalModel 取 defaultModelConfig、调 DescribeEmbodiedEvalBenchmarks 取 defaultBenchmarkConfig，再填上资源池、队列和名称。
- modelConfig.replicas 必须是 benchmarkConfigs 个数的整数倍，模型服务副本会平均分给各评测集。
- 每个评测集的 parallelShards 不能超过 episodesPerTask。部分评测集按 GPU 卡数分片（如 RoboDojo 要求 parallelShards 等于该评测集的 GPU 卡数，且 CPU、内存、共享内存不低于单分片最低配置乘以分片数），不满足时返回 400 并给出应填的值。
- 同一个评测集可以出现多次，每次用不同的 subBench。
- 跑哪些任务只由 subBench 决定，不要在 params 里填 suite、task_name、tasks 这类由子集决定的键。
- 使用自己的权重（customCKPT）时，先在 dataSources 里挂载 BOS 路径，再把 ckptPath 写成挂载点下的路径，见 ModelConfig。
- 使用自己的模型服务镜像（customImage）时，镜像必须满足总览文档《百舸具身评测 OpenAPI 参考》附录「自定义镜像要求」，否则任务会失败或产出错误分数。
- 使用自建推理服务（selfHosted）时，平台不启动模型服务，modelConfig 只填 modelType 和 selfHosted，不能带 replicas、maxBatchSize、resources、ckptPath、customImage；benchmarkConfigs[].modelOverride 不传或全空。评测节点必须能直连 url，服务要在 attachTimeoutSec 内就绪。
- selfHosted 带 bearerToken 时，每个评测集都要支持鉴权头：当前公共评测集都支持，自定义评测集不支持，返回 400 benchmarkConfs[i] does not support bearerToken。带 bearerToken 时 url 的 query 不能再带 token、access_token、auth、key 这类凭据参数。
- selfHosted 的 modelConfig 示例（benchmarkConfigs 与上面的请求示例相同，去掉 modelOverride）：

```
{
 "modelType": "selfHosted",
 "selfHosted": {
  "endpoints": [{"url": "wss://10.0.0.8:8443/infer", "serverTimeoutSec": 120, "attachTimeoutSec": 600, "bearerToken": "<token>"}]
 }
}
```

- 以下挂载点由平台使用，dataSources[].mountPath 不能用：/mnt/aihc-eval 及其下的子路径、/mnt/aihc-eval-result。
- 请求体不是合法 JSON、为空，或字段类型不对（如 params 不是对象）时返回 400。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	必填字段缺失、取值超出范围、模型或评测集不存在、模型与评测集不兼容、卡型不支持、挂载点被占用等，原因见 message
403	Forbidden	没有该队列的操作权限
403	IdempotentParameterMismatch	同一个 clientToken 带了不同参数

查询评测任务列表

查询评测任务列表 DescribeEmbodiedEvals

接口描述

分页查询当前账号下自己有权限查看的评测任务。创建失败的任务不出现在列表里，可以用 DescribeEmbodiedEval 按 ID 查看。

权限说明

只返回调用人有权查看的任务：visibleScope=1 的任务对同队列成员可见，visibleScope=0 的任务只对创建人和资源池管理员可见。

请求结构

```
{
 "method": "GET",
 "uri": "/?action=DescribeEmbodiedEvals&pageNumber=1&pageSize=10",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "{authorization}",
   "version": "v2"
 }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
pageNumber	Integer	否	Query参数	页码，从 1 开始，默认 1
pageSize	Integer	否	Query参数	每页条数，1～100，默认 10
keywordType	String	否	Query参数	关键字匹配的字段：name 任务名称、evalId 任务 ID、creator 创建人。传 keyword 时必须传
keyword	String	否	Query参数	关键字，按 keywordType 指定的字段模糊匹配
status	String	否	Query参数	按状态过滤，取值见总览文档《百舸具身评测 OpenAPI 参考》附录「任务状态」，多个用英文逗号分隔，如 Running,Failed
modelName	String	否	Query参数	按模型名称精确过滤，多个用英文逗号分隔
priority	String	否	Query参数	按优先级过滤，多个用英文逗号分隔
orderBy	String	否	Query参数	排序字段：createdAt、finishedAt、runTimeSeconds、priority，默认 createdAt
order	String	否	Query参数	排序方向：ASC / DESC，只接受大写，默认 DESC

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
totalCount	Integer	符合条件的任务总数
evals	EvalSummary[]	当前页的任务

请求示例

```
{
 "method": "GET",
 "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEvals&pageNumber=1&pageSize=10&status=Running,Succeeded",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
   "x-bce-date": "2026-11-04T08:00:00Z",
   "version": "v2"
 }
}
```

响应示例

```
{
 "requestId": "2a3b4c5d-6e7f-4801-9a2b-3c4d5e6f7a8b",
 "totalCount": 1,
 "evals": [
  {
   "evalId": "em-6o004i76d88e",
   "name": "openvla-libero-spatial",
   "status": "Running",
   "resourcePoolId": "cce-xxxxxxxx",
   "queueId": "default",
   "resourcePoolType": "",
   "priority": "normal",
   "visibleScope": 1,
   "creator": "zhangsan",
   "modelName": "openvla_7b",
   "modelDisplayName": "OpenVLA",
   "benchmarks": [{"name": "libero", "displayName": "LIBERO", "subBench": "libero_spatial"}],
   "resources": [
    {"name": "baidu.com/a800_80g_cgpu", "quantity": 2},
    {"name": "cpu", "quantity": 16},
    {"name": "memory", "quantity": 64}
   ],
   "runTimeSeconds": 1260,
   "createdAt": "2026-11-04T02:00:00Z",
   "finishedAt": "",
   "region": "bj"
  }
 ]
}
```


注意事项

- 列表返回时会刷新未结束任务的状态，`pageSize` 越大响应越慢。轮询单个任务请用 `DescribeEmbodiedEval`。
- 按 `status=CreateFailed` 过滤得到的是空列表。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	`pageSize` 超出 1～100、`order` 不是大写、`orderBy/keywordType/status` 取值不合法、只传了 `keyword` 没传 `keywordType`

查询评测任务详情

查询评测任务详情 DescribeEmbodiedEval

接口描述

查询一个评测任务的详情。返回的配置字段和 `CreateEmbodiedEval` 的请求体结构相同，可以修改后作为新任务的请求体。

权限说明

需要对该任务的查看权限，规则同 DescribeEmbodiedEvals。

请求结构

```
{
 "method": "GET",
 "uri": "/?action=DescribeEmbodiedEval&evalId={evalId}",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "{authorization}",
   "version": "v2"
 }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
evalId	String	是	Query参数	评测任务 ID

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
evalId	String	评测任务 ID
name	String	任务名称
status	String	任务状态，见总览文档《百舸具身评测 OpenAPI 参考》附录「任务状态」
jobId	String	任务对应的百舸分布式训练任务 ID，任务还没提交时为空
resourcePoolId / queueId / resourcePoolType	String	同创建请求
priority	String	优先级
visibleScope	Integer	可见范围
creator	String	创建人用户名
region	String	地域
runTimeSeconds	Integer	累计运行秒数，没开始运行时为 0
createdAt / updatedAt / finishedAt	String	创建、更新、结束时间
dataSources	DataSource[]	创建时传入的数据源，不包含平台自动挂载的数据源
modelConfig	ModelConfig	模型服务配置
benchmarkConfigs	BenchmarkConfig[]	评测集配置。params、modelOverride.extraArgs 是创建时传入的值，没传或为空对象时可能不返回该字段，按空对象处理即可

请求示例

```
{
 "method": "GET",
 "url": "https://aihc.bj.baidubce.com/?action=DescribeEmbodiedEval&evalId=em-6o004i76d88e",
 "headers": {
   "Host": "aihc.bj.baidubce.com",
   "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
   "x-bce-date": "2026-11-04T08:00:00Z",
   "version": "v2"
 }
}
```

响应示例

```
{
 "requestId": "3b4c5d6e-7f80-4912-ab3c-4d5e6f7a8b9c",
 "evalId": "em-6o004i76d88e",
 "name": "openvla-libero-spatial",
 "jobId": "job-a1b2c3d4e5f6",
 "status": "Succeeded",
 "resourcePoolId": "cce-xxxxxxxx",
 "queueId": "default",
 "resourcePoolType": "",
 "priority": "normal",
 "visibleScope": 1,
 "creator": "zhangsan",
 "region": "bj",
 "runTimeSeconds": 2410,
 "createdAt": "2026-11-04T02:00:00Z",
 "finishedAt": "2026-11-04T02:43:12Z",
 "updatedAt": "2026-11-04T02:43:20Z",
 "modelConfig": {
   "modelType": "public",
   "modelName": "openvla_7b",
   "replicas": 1,
   "maxBatchSize": 1,
   "resources": [
     {"name": "baidu.com/a800_80g_cgpu", "quantity": 1},
     {"name": "cpu", "quantity": 8},
     {"name": "memory", "quantity": 32}
   ]
 },
 "benchmarkConfigs": [
   {
     "name": "libero",
     "subBench": "libero_spatial",
     "resources": [
       {"name": "baidu.com/a800_80g_cgpu", "quantity": 1},
       {"name": "cpu", "quantity": 8},
       {"name": "memory", "quantity": 32}
     ],
     "evalType": "sync",
     "parallelShards": 1,
     "episodesPerTask": 10,
     "recordVideo": true,
     "modelOverride": {"chunkSize": 1}
   }
 ]
}
```

注意事项

- 自定义镜像任务的 modelConfig.customImage.imageConfig.password 固定返回 ******。拿详情重新创建任务时要重新填写真实密码，否则返回 400。
- 自定义镜像任务不返回 modelConfig.modelName。
- 自建推理服务任务的 modelConfig.modelName 是 selfHosted，原样回传创建不会被拒。modelConfig.selfHosted.endpoints[] 返回 url、serverTimeoutSec、attachTimeoutSec、hasBearerToken：url 的 query 整段换成 ?***，两个超时是生效值（创建时不传或传 0 的回填为 60 / 300），不返回 bearerToken。拿详情重新创建任务时，hasBearerToken 为 true 要重新传 bearerToken，url 带 ?*** 要重新传完整地址。
- 任务不存在、已删除或属于其他账号时返回 404；同账号但没有查看权限时返回 403。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	没有传 evalId
403	Forbidden	没有查看该任务的权限
404	NotFound	任务不存在或已删除

停止评测任务

停止评测任务 StopEmbodiedEval

接口描述

停止一个运行中或排队中的评测任务。停止后任务进入 Stopping，随后变为 ManualTermination。停止前已经写出的进度和结果仍可通过 DescribeEmbodiedEvalResult 查询。

权限说明

资源池管理员可以停止池内任意任务；队列成员只能停止自己创建的任务。只有只读权限的用户不能停止。

请求结构

```
{
  "method": "POST",
  "uri": "/?action=StopEmbodiedEval&evalId={evalId}",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "{authorization}",
    "version": "v2",
    "Content-Type": "application/json"
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	说明
evalId	String	是	Query参数	评测任务 ID，一次只能停止一个任务
clientToken	String	否	Query参数	幂等键，不超过 64 字节

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名称	类型	说明
requestId	String	请求 ID
evalId	String	被停止的任务 ID

请求示例

```
{
  "method": "POST",
  "url": "https://aihc.bj.baidubce.com/?action=StopEmbodiedEval&evalId=em-6o004i76d88e",
  "headers": {
    "Host": "aihc.bj.baidubce.com",
    "Authorization": "bce-auth-v1/<AccessKeyId>/2026-11-04T08:00:00Z/1800/host;x-bce-date/<signature>",
    "x-bce-date": "2026-11-04T08:00:00Z",
    "version": "v2",
    "Content-Type": "application/json; charset=utf-8"
  }
}
```

响应示例

```
{
  "requestId": "4c5d6e7f-8091-4a23-bc4d-5e6f7a8b9c0d",
  "evalId": "em-6o004i76d88e"
}
```

注意事项

- 任务已经是 ManualTermination 时再次停止返回成功，可以放心重试。
- 任务已经是 Succeeded、Failed、CreateFailed 时返回 400。
- 任务刚创建、还在提交时返回 503 ResourceBusy，等几秒后重试。

错误码

HTTP 状态码	code	场景
400	InvalidParameter	没有传 evalId；任务已经结束
403	Forbidden	没有停止该任务的权限
404	NotFound	任务不存在或已删除
503	ResourceBusy	任务还在提交中
500	InternalError	停止失败，可重试

数据结构

ModelConfig

模型服务配置，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）、查询模型的默认配置（响应 defaultModelConfig）

平台按此配置启动 replicas 个推理服务副本，评测集通过它们调用模型；modelType=selfHosted 时平台不启动副本，评测集直连 selfHosted.endpoints 里的地址。

参数名称	类型	必选	说明
modelType	String	是	模型来源：public 平台预置模型和权重；customCKPT 平台预置模型 + 自己的权重；customImage 自己的推理服务镜像；selfHosted 自建推理服务，平台不拉起模型服务
modelName	String	条件必须	public、customCKPT 必填，取自查询可评测的模型及其评测集；customImage 不能填；selfHosted 不填或填 selfHosted
ckptPath	String	条件必须	模型权重路径，对所有评测集生效；某个评测集的 modelOverride.ckptPath 非空时以后者为准。customCKPT 时它和每个评测集的 modelOverride.ckptPath 至少要填一个。以 / 开头表示容器内路径，必须在 dataSources 的某个 mountPath 下；否则按 Hugging Face 仓库名处理（如 openvla/openvla-7b）
replicas	Integer	条件必须	推理服务副本数，大于 0，且是 benchmarkConfigs 个数的整数倍；selfHosted 不能传，其他类型必填
maxBatchSize	Integer	条件必须	单副本最大批大小，大于 0；selfHosted 不能传，其他类型必填
resources	List<Resource>	条件必须	单副本的资源。GPU 卡型必须在模型支持的卡型内（查询模型的默认配置返回的 minHardware.acceleratorTypes）；selfHosted 不能传，其他类型必填
customImage	CustomImage	条件必须	modelType=customImage 时必填，其他类型不能传
selfHosted	SelfHosted	条件必须	modelType=selfHosted 时必填，其他类型不能传

CustomImage

自己的推理服务镜像，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

镜像必须满足总览文档《百舸具身评测 OpenAPI 参考》附录「自定义镜像要求」。

参数名称	类型	必选	说明
image	String	是	镜像地址，包含仓库和 tag，不超过 512 字节。填写 imageConfig 时镜像必须在 ccr-registry.baidubce.com 上
command	String	是	启动命令，不超过 4096 字节。按空格切分后直接执行（不经过 shell），不能写死监听端口
envs	List<EnvVar>	否	环境变量，最多 32 个，单个 value 不超过 1024 字节，全部合计不超过 8192 字节
imageConfig	ImageConfig	否	私有镜像仓库凭据

- 查询评测任务详情中自定义镜像任务不返回 modelConfig.modelName。

EnvVar

自定义镜像的环境变量，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

参数名称	类型	必选	说明
name	String	是	变量名，只能包含字母、数字、-、_、.，不能以数字开头，不能重名
value	String	是	变量值，不超过 1024 字节

- 以下名称由平台使用，不能填写（不区分大小写）：EVAL_SPEC、EVAL_SPEC_VERSION、EVAL_RESULT_FILE、EVAL_PLATFORM_MOUNT、EVAL_SRC_MOUNT、EVAL_RESULT_MOUNT、CUDA_VISIBLE_DEVICES、PYTHONPATH、MS_PORT、HTTP_PROXY、HTTPS_P 以及以 RAY_ 开头的名称。
- envs 的值在详情里原样返回，**不要在 envs 里放密码、密钥**。镜像仓库密码只放在 imageConfig.password。

ImageConfig

私有镜像仓库凭据，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

参数名称	类型	必选	说明
username	String	是	镜像仓库用户名，不超过 256 字节
password	String	是	镜像仓库密码，不超过 256 字节，不能是 ******

- 两项同时填或同时不填。
- 查询评测任务详情中 password 固定返回 ******。拿详情重新创建任务时要重新填写真实密码，否则返回 400。

SelfHosted

自建推理服务配置，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

平台不拉起模型服务，评测节点直连你提供的地址。

参数名称	类型	必选	说明
endpoints	List<SelfHostedEndpoint>	是	推理服务地址。当前只支持 1 个，数量不对返回 400 endpoints count must be in [1, 1]。用数组是为了以后放开数量时不改协议

SelfHostedEndpoint

一个自建推理服务地址，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

参数名称	类型	必选	说明
url	String	是	ws:// 或 wss:// 开头，不超过 512 字节，不能带 userinfo 和 #fragment。host 填 IP 或公共顶级域下的域名；不接受 localhost、回环、link-local（含 169.254.169.254）、未指定、组播、广播地址，以及 .local、.svc、单段名这类集群内部名；私网 IP 可以用。私有 DNS 区里的服务请直接填 IP
serverTimeoutSec	Integer	否	单帧推理超时，1～900 秒，不传或 0 按 60
attachTimeoutSec	Integer	否	等待服务就绪的总时长，30～900 秒，不传或 0 按 300
bearerToken	String	否	评测节点建连时放进 Authorization: Bearer <token>。16～4096 字节，字符集是 RFC 6750 b64token，不要带 Bearer 前缀。只写不读
hasBearerToken	Boolean	-	只出现在查询评测任务详情的响应中，表示创建时是否传了 bearerToken

- 查询评测任务详情中 url 的 query 整段换成 ?***，两个超时返回生效值，不返回 bearerToken。拿详情重新创建任务时，hasBearerToken 为 true 要重新传 bearerToken，url 带 ?*** 要重新传完整地址。
- 带 bearerToken 时，url 的 query 不能再带 token、access_token、auth、key 这类凭据参数；自定义评测集不支持鉴权头，返回 400 benchmarkConfs[i] does not support bearerToken。

BenchmarkConfig

一个评测集（子集）的配置，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）、查询评测集的默认配置（响应 defaultBenchmarkConfig）

参数名称	类型	必选	说明
name	String	是	评测集名称，如 libero
subBench	String	是	子集名称，如 libero_spatial，必须属于该评测集，且与模型兼容（source=public 时以查询可评测的模型及其评测集为准）
resources	List<Resource>	是	评测集仿真环境的资源，所有分片共用。GPU 卡型必须在评测集支持的卡型内；GPU 为 0 只适用于 minHardware.acceleratorCount 为 0 的评测集
parallelShards	Integer	是	并行分片数，大于 0，不超过 episodesPerTask。分片越多跑得越快，需要的 CPU、内存也越多
episodesPerTask	Integer	是	每个任务（task）跑的 episode 数，大于 0，不超过 maxEpisodesPerTask
evalType	String	否	评测方式：sync 同步（默认）、realtime 实时
params	Object	否	评测集参数，可填的键见查询评测集的默认配置返回的 paramsSchema。没填的键使用默认值。不要填 suite、task_name、tasks 等由子集决定的键
recordVideo	Boolean	否	是否录制每个 episode 的回放视频，默认 false。开启后可用查询回放视频播放地址接口播放
modelOverride	ModelOverride	条件必须	本评测集对模型的参数；modelType=selfHosted 时不传或全空，其他类型必填

- 查询评测任务详情中，params、modelOverride.extraArgs 是创建时传入的值，没传或为空对象时可能不返回该字段，按空对象处理即可。

ModelOverride

评测集级的模型参数，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

参数名称	类型	必选	说明
chunkSize	Integer	是	模型每次推理输出的动作步数，大于 0，建议不超过 chunkSizeLimit
ckptPath	String	否	本评测集使用的权重路径，规则同 ModelConfig.ckptPath。不填时依次使用 modelConfig.ckptPath、平台推荐权重
extraArgs	Object	否	模型推理参数，可填的键见查询模型的默认配置返回的 extraArgsSchema。没填的键使用平台推荐值

DataSource

挂载到评测任务里的数据源，被如下接口引用：创建评测任务（请求）、查询评测任务详情（响应）

目前只支持 BOS，用于读取自己的模型权重。

参数名称	类型	必选	说明
type	String	是	数据源类型，目前只能是 bos
sourcePath	String	是	BOS 路径，格式 <bucket>/<prefix>，如 my-bucket/ckpt/openvla-ft
mountPath	String	是	容器内的挂载路径，以 / 开头，每一段只能包含字母、数字、_、-，如 /mnt/ckpt
name	String	否	数据源名称
id	String	否	数据源 ID
options	Object	否	挂载选项。BOS 可以填 {"readOnly": true}

- 当前账号需要对该 BOS 路径有读权限，建议桶与任务在同一地域。
- /mnt/aihc-eval 及其下的子路径、/mnt/aihc-eval-result 由平台使用，不能作为 mountPath。
- 查询评测任务详情只返回创建时传入的数据源，不包含平台自动挂载的数据源。

Resource

资源配额，被如下接口引用：创建评测任务（请求）、查询评测任务列表（响应）、查询评测任务详情（响应）

参数名称	类型	必选	说明
name	String	是	资源名称：cpu（核）、memory（GB）、sharedMemory（共享内存，GB，不填时为 10）、GPU 卡型（如 baidu.com/a800_80g_cgpu、baidu.com/l20_cgpu）
quantity	Number	是	数量，大于 0。cpu、memory、sharedMemory 保留一位小数

- GPU 卡型需要与队列里实际的卡一致。各模型、评测集支持的卡型见元数据接口返回的 minHardware.acceleratorTypes，一次评测里模型和评测集使用同一种卡。

EvalSummary

评测任务摘要，被如下接口引用：查询评测任务列表（响应 evals）

参数名称	类型	说明
evalId	String	评测任务 ID
name	String	任务名称
status	String	任务状态，见任务状态
resourcePoolId / queueId / resourcePoolType	String	资源池、队列、资源池类型
priority	String	优先级
visibleScope	Integer	可见范围
creator	String	创建人用户名
modelName / modelDisplayName	String	模型名称和展示名称；自定义镜像任务为 `custom`
benchmarks	List	评测集和子集，每项 `{name, displayName, subBench}`
resources	List<Resource>	整个任务申请的资源合计
runTimeSeconds	Integer	累计运行秒数
createdAt / finishedAt	String	创建时间、结束时间
region	String	地域

任务状态

评测任务的状态取值，被如下接口引用：查询评测任务列表（请求 `status` 过滤、响应 `evals[].status`）、查询评测任务详情（响应 `status`）

状态	是否终态	说明
Created	否	已创建，正在提交
ToBeSubmitted	否	等待提交
Scheduled	否	已提交，排队等待资源
Running	否	运行中
Scaling	否	扩缩容中
Restarting	否	重启中
Preempted	否	被高优先级任务抢占，等待重新调度
Stopping	否	停止中
Abnormal	否	运行异常，或底层任务已不存在
Unknown	否	状态未知，稍后再查
Succeeded	是	运行成功，可以查询结果
Failed	是	运行失败
ManualTermination	是	已手动停止
CreateFailed	是	创建失败，只在详情中出现，不出现在列表中

EvalResult

评测结果，被如下接口引用：查询评测结果（响应 `result`）

字段名为下划线风格。`result` 内部字段随评测集变化，只保证下表列出的字段。

参数名称	类型	说明
version	String	结果格式版本，目前为 `"1"`
job_id	String	底层任务 ID
started_at	String	开始时间，UTC
finished_at	String	结束时间，UTC；运行中为 `null`
video_summary	Object	视频统计：`enabled` 是否开启录像，`total` 视频总数，`ready` 可播放数，`failed` 不可播放数
benchmarks	List<EvalResultBenchmark>	每个评测集（子集）一项

- 任务成功前 `result` 可能是 `null`，也可能是运行中的进度快照（`benchmarks[].metrics`、`details` 为 `null`，`finished_at` 为 `null`）。以任务状态为 `Succeeded` 后取到的结果为准。

EvalResultBenchmark

一个评测项（评测集 + 子集）的结果，被如下接口引用：查询评测结果（响应 `result.benchmarks`）

参数名称	类型	说明
id	String	评测项 ID，格式 `<评测集>-<子集>`
name	String	评测集名称
sub_bench	String	子集名称
status	String	`running` 运行中、`succeeded` 全部分片成功、`partial` 部分分片失败、`failed` 失败、`stopped` 已停止
progress	Object	`completed_episodes` 已完成 episode 数、`total_episodes` 总数、`num_errors` 出错数、`num_timeouts` 达到最大步数仍未成功的数量
config	Object	本评测项实际使用的配置：`mode`、`num_shards`、`episodes_per_task`、`max_steps`、`chunk_size`、`params` 等
metrics	Object	评分：`mean_success` 平均成功率（0～1）、`num_tasks` 任务数。运行中为 `null`
details	Object	明细：`tasks[]` 每个任务的 `task` 任务描述、`mean_success`、`avg_steps`、`num_episodes`、`num_errors`、`num_timeouts`、`videos`（List<EvalResultVideo>）。运行中为 `null`

EvalResultVideo

一条 episode 回放视频，被如下接口引用：查询评测结果（响应 `result.benchmarks[].details.tasks[].videos`）、查询回放视频播放地址（请求 `videoId` 取自这里）

参数名称	类型	说明
video_id	String	视频 ID，传给查询回放视频播放地址
episode_index	Integer	episode 序号
success	Boolean	该 episode 是否成功
state	String	ready 可播放、failed 不可播放

ModelItem

可评测的模型，被如下接口引用：查询可评测的模型及其评测集（响应 models）

参数名称	类型	说明
modelName	String	模型名称，用于创建评测任务请求的 modelConfig.modelName
displayName	String	展示名称
benchmarks	List	可以跑的评测集，每项 {name, displayName, subBenches}，subBenches 是子集名称列表

MinHardware

最低硬件要求，被如下接口引用：查询模型的默认配置（响应 minHardware，模型服务单副本）、查询评测集的默认配置（响应 benchmarks[].minHardware，评测集仿真环境单分片）

参数名称	类型	说明
acceleratorTypes	List	支持的 GPU 卡型
acceleratorCount	Integer	最少 GPU 卡数，0 表示可以不用 GPU
cpu	Integer	最少 CPU 核数
memory	Integer	最少内存，单位 GB

ParamSchema

params 或 extraArgs 可以填的参数，被如下接口引用：查询模型的默认配置（响应 extraArgsSchema）、查询评测集的默认配置（响应 benchmarks[].paramsSchema）

参数名称	类型	说明
requiredArgs	List	必须有值的参数。平台推荐配置里已经给了值时可以不填
optionalArgs	List	可以填的参数
properties	Object	每个参数的说明，键是参数名，值包含 type（string、integer、number、boolean 等）、default 默认值、description 说明，以及可选的 enum 取值范围、nullable 是否可为 null

BenchmarkMetadata

一个「模型 × 评测集 × 子集」组合的默认配置，被如下接口引用：查询评测集的默认配置（响应 benchmarks）

参数名称	类型	说明
modelName / name / subBench	String	模型、评测集、子集
minHardware	MinHardware	评测集仿真环境单分片的最低硬件要求
defaultBenchmarkConfig	BenchmarkConfig	推荐配置
paramsSchema	ParamSchema	params 可以填的参数
maxEpisodesPerTask	Integer	episodesPerTask 的上限，0 表示不限

SDK参考

GO SDK

概述

本文档主要介绍AIHC GO SDK的安装和使用。在使用本文档前，您需要先了解AIHC的一些基本知识，并已经开通了AIHC服务。若您还不了解AIHC，可以参考产品描述。

安装SDK工具包

运行环境

Go SDK 可以在 Go 1.11 及以上环境下运行。

安装 SDK

直接从 GitHub 下载

使用 go get 工具从 GitHub 下载 AIHC Go SDK：

```
go get github.com/baidubce/bce-sdk-go
```

SDK 目录结构

```
bce-sdk-go
├─── auth            // BCE 签名相关类
├─── http            // BCE 的 HTTP 通信相关类
├─── services
|    └─── aihc        // AIHC 服务相关类
├─── util            // BCE 公用工具类
└─── examples
     └─── aihc       // AIHC 各种对象 SDK 使用示例
```

卸载 SDK

卸载 Go SDK 时，删除已下载到 GOPATH 下的 bce-sdk-go 源码目录即可。

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于服务域名的部分，理解Endpoint相关的概念。百度云目前开放了多区域支持，请参考区域选择说明。

访问区域	对应Endpoint
北京	bj
广州	gz
苏州	su
香港	hkg
武汉	fwh
保定	bd

获取密钥

要使用百度云百舸服务，您需要拥有一个有效的AK(Access Key ID)和SK(Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问百舸服务做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

注册百度云账号

创建AK/SK

新建 Aihc Client

AihcClient是百舸服务的客户端，为开发者访问接口提供了一系列的方法。

```
 import (
"github.com/baidubce/bce-sdk-go/services/aihc"
)

func main() {
 // 用户的Access Key ID和Secret Access Key
 ACCESS_KEY_ID, SECRET_ACCESS_KEY := <your-access-key-id>, <your-secret-access-key>

 // 用户指定的Endpoint
 ENDPOINT := <domain-name>

 // 初始化一个AihcClient
 client, err := aihc.NewClient(AK, SK, ENDPOINT)
}
```

在上面代码中，ACCESS_KEY_ID对应控制台中的“Access Key ID”，SECRET_ACCESS_KEY对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 如何获取AKSK》。第三个参数ENDPOINT为用户自己填写指定域名，如果设置为空字符串，会使用默认域名作为百舸服务的服务地址。

使用STS创建 Aihc Client 申请STS token

百舸服务可以通过STS机制实现第三方的临时授权访问。STS（Security Token Service）是百度云提供的临时授权服务。通过STS，您可以为第三方用户颁发一个自定义时效和权限的访问凭证。第三方用户可以使用该访问凭证直接调用百度云的API或SDK访问百度云资源。

通过STS方式访问百舸服务，用户需要先通过STS的client申请一个认证字符串。

用STS token新建 Aihc Client

申请好STS后，可将STS Token配置到 Aihc Client中，从而实现通过STS Token创建 Aihc Client。

代码示例

GO SDK实现了STS服务的接口，用户可以参考如下完整代码，实现申请STS Token和创建 Aihc Client对象：

```
import (
"fmt"

"github.com/baidubce/bce-sdk-go/auth"                //导入认证模块
"github.com/baidubce/bce-sdk-go/services/aihc"       //导入百舸服务模块
"github.com/baidubce/bce-sdk-go/services/sts"        //导入STS服务模块
)

func main() {
// 创建STS服务的Client对象，Endpoint使用默认值
AK, SK := <your-access-key-id>, <your-secret-access-key>
stsClient, err := sts.NewClient(AK, SK)
if err != nil {
 fmt.Println("create sts client object :", err)
 return
}

// 获取临时认证token，有效期为60秒，ACL为空
stsObj, err := stsClient.GetSessionToken(60, "")
if err != nil {
 fmt.Println("get session token failed:", err)
 return
  }
fmt.Println("GetSessionToken result:")
fmt.Println("  accessKeyId:", stsObj.AccessKeyId)
fmt.Println("  secretAccessKey:", stsObj.SecretAccessKey)
fmt.Println("  sessionToken:", stsObj.SessionToken)
fmt.Println("  createTime:", stsObj.CreateTime)
fmt.Println("  expiration:", stsObj.Expiration)
fmt.Println("  userId:", stsObj.UserId)

// 使用申请的临时STS创建百舸服务服务的Client对象，Endpoint使用默认值
aihcClient, err := aihc.NewClientWithSTS(stsObj.AccessKeyId, stsObj.SecretAccessKey, stsObj.SessionToken, "aihc.bj.baidubce.com")
if err != nil {
 fmt.Println("create aihc client failed:", err)
 return
}
}
```

配置HTTPS协议访问百舸服务

百舸服务支持HTTPS传输协议，您可以通过在创建 Aihc Client对象时指定的Endpoint中指明HTTPS的方式，在Aihc GO SDK中使用HTTPS访问百舸服务：

```
import "github.com/baidubce/bce-sdk-go/aihc"

ENDPOINT := "https://aihc.bj.baidubce.com" //指明使用HTTPS协议
AK, SK := <your-access-key-id>, <your-secret-access-key>
AihcClient, _ := aihc.NewClient(AK, SK, ENDPOINT)
```

配置Aihc Client

如果用户需要配置Aihc Client的一些细节的参数，可以在创建Aihc Client对象之后，使用该对象的导出字段Config进行自定义配置，可以为客户端配置代理，最大连接数等参数。

使用代理 下面一段代码可以让客户端使用代理访问百舸服务：

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"

//创建Aihc Client对象
AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "aihc.bj.baidubce.com"
client, _ := aihc.NewClient(AK, SK, ENDPOINT)

//代理使用本地的8080端口
client.Config.ProxyUrl = "127.0.0.1:8080"
```

设置网络参数 用户可以通过如下的示例代码进行网络参数的设置：

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "aihc.bj.baidubce.com"
client, _ := aihc.NewClient(AK, SK, ENDPOINT)

// 配置不进行重试，默认为Back Off重试
client.Config.Retry = bce.NewNoRetryPolicy()

// 配置连接超时时间为30秒
client.Config.ConnectionTimeoutInMillis = 30 * 1000
```

配置生成签名字符串选项

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "aihc.bj.baidubce.com"
client, _ := aihc.NewClient(AK, SK, ENDPOINT)

// 配置签名使用的HTTP请求头为`Host`
headersToSign := map[string]struct{}{"Host": struct{}{}}
client.Config.SignOption.HeadersToSign = HeadersToSign

// 配置签名的有效期为30秒
client.Config.SignOption.ExpireSeconds = 30
```

参数说明

用户使用GO SDK访问百舸服务时，创建的 Aihc Client对象的Config字段支持的所有参数如下表所示：

配置项名称	类型	含义
Endpoint	string	请求服务的域名
ProxyUrl	string	客户端请求的代理地址
Region	string	请求资源的区域
UserAgent	string	用户名称，HTTP请求的User-Agent头
Credentials	*auth.BceCredentials	请求的鉴权对象，分为普通AK/SK与STS两种
SignOption	*auth.SignOptions	认证字符串签名选项
Retry	RetryPolicy	连接重试策略
ConnectionTimeoutInMillis	int	连接超时时间，单位毫秒，默认20分钟

说明：

1. Credentials字段使用auth.NewBceCredentials与auth.NewSessionBceCredentials函数创建，默认使用前者，后者为使用STS鉴权时使用。
2. SignOption字段为生成签名字符串时的选项，详见下表说明：

名称	类型	含义
HeadersToSign	map[string]struct{}	生成签名字符串时使用的HTTP头
Timestamp	int64	生成的签名字符串中使用的时间戳，默认使用请求发送时的值
ExpireSeconds	int	签名字符串的有效期

说明：

1. 其中，HeadersToSign默认为Host，Content-Type，Content-Length，Content-MD5；TimeStamp一般为零值，表示使用调用生成认证字符串时的时间戳，用户一般不应该明确指定该字段的值；ExpireSeconds默认为1800秒即30分钟。
2. Retry字段指定重试策略，目前支持两种：NoRetryPolicy和BackOffRetryPolicy。默认使用后者，该重试策略是指定最大重试次数、最长重试时间和重试基数，按照重试基数乘以2的指数级增长的方式进行重试，直到达到最大重试测试或者最长重试时间为止。

资源池相关

资源池列表

使用以下代码可以列出资源池

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.ListResourcePool(&v1.ListResourcePoolRequest{
 PageNo:   2,
 PageSize: 2,
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为资源池列表

资源池详情

使用以下代码可以获取资源池详情

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
resourcePoolID := "xxxxx"

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetResourcePool(resourcePoolID)

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为资源池详情

资源池节点列表

使用以下代码可以获取资源池节点列表

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
resourcePoolID := "xxxxx"

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.ListNodeByResourcePoolID(resourcePoolID, &v1.ListResourcePoolNodeRequest{
 PageNo:   1,
 PageSize: 2,
})
if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为资源池节点列表

队列列表

使用以下代码可以获取资源池队列列表

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
resourcePoolID := "xxxxx"

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.ListQueue(resourcePoolID, &v1.ListQueueRequest{
 PageNo:   1,
 PageSize: 2,
})
if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为队列列表

队列详情

使用以下代码可以获取资源池节点列表

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
resourcePoolID, queueName := "xxxxx", "xxxxx"

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetQueue(resourcePoolID, queueName)
if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为队列详情

分布式训练相关

查询训练任务列表

使用以下代码可以查询训练任务列表。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
req := &v1.OpenAPIJobListRequest{
ResourcePoolID: RESOURCE_POOL_ID,
PageNo:         1,
PageSize:       3,
}
result, err := client.ListJobs(req)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务列表

创建训练任务

使用以下代码可以创建训练任务。

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
resourcePoolID := RESOURCE_POOL_ID

jobConfig := &v1.OpenAPIJobCreateRequest{
Name: AIJobName,
JobSpec: v1.OpenAPIAIJobSpec{
Command:  `echo "hello sdk"; sleep infinity`,
Replicas: 1,
Image:    ImageID,
Resources: []v1.OpenAPIResource{
{
Name:     "cpu",
Quantity: 1,
},
},
EnableRDMA: false,
},
EnableBccl: false,
}
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.CreateJob(jobConfig, resourcePoolID)

if err != nil {
panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为创建训练任务

查询训练任务详情

使用以下代码可以查询训练任务详情。

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
resourcePoolID, JobID := RESOURCE_POOL_ID, AIJobID

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetJob(JobID, resourcePoolID)

if err != nil {
panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务详情

更新训练任务

使用以下代码可以更新训练任务。

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
resourcePoolID := RESOURCE_POOL_ID
jobID := AIJobID

jobConfig := &v1.OpenAPIJobUpdateRequest{
Priority: "high",
}
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.UpdateJob(jobConfig, jobID, resourcePoolID)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为更新训练任务

停止训练任务

使用以下代码可以停止训练任务。

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
resourcePoolID := RESOURCE_POOL_ID
jobID := AIJobID

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.StopJob(jobID, resourcePoolID)
log.Infof("stop job result: %v", result)
if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为停止训练任务

删除训练任务

使用以下代码可以删除训练任务。

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
resourcePoolID, JobID := RESOURCE_POOL_ID, AIJobID

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DeleteJob(JobID, resourcePoolID)

if err != nil {
panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为删除训练任务

查询训练任务事件

使用以下代码可以查询训练任务事件。

```
// import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test

req := &v1.GetJobEventsRequest{
Namespace:      "",
JobFramework:   "PyTorchJob",
StartTime:      "",
EndTime:        "",
JobID:          AIJobID,
ResourcePoolID: RESOURCE_POOL_ID,
}

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetTaskEvent(req)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务事件

查询训练任务日志

使用以下代码可以查询训练任务日志。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test

req := &v1.GetPodLogsRequest{
JobID:          AIJobID,
ResourcePoolID: RESOURCE_POOL_ID,
PodName:        PodName,
Namespace:      "default",
StartTime:      "",
EndTime:        "",
MaxLines:       "",
Container:      "",
Chunk:          "",
}

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetPodLogs(req)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务日志

查询训练任务Pod事件

使用以下代码可以查询训练任务Pod事件。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
req := &v1.GetPodEventsRequest{
JobID:          AIJobID,
ResourcePoolID: RESOURCE_POOL_ID,
Namespace:      "",
JobFramework:   "PyTorchJob",
StartTime:      "",
EndTime:        "",
PodName:        PodName,
}

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetPodEvents(req)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务Pod事件

查询训练任务监控

使用以下代码可以查询训练任务监控。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
req := &v1.GetTaskMetricsRequest{
StartTime:      "",
ResourcePoolID: RESOURCE_POOL_ID,
EndTime:        "",
TimeStep:       "",
MetricType:     MetricType,
JobID:          AIJobID,
Namespace:      "",
RateInterval:   "",
}

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetTaskMetrics(req)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务监控

查询训练任务所在节点列表

使用以下代码可以查询训练任务所在节点列表。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
resourcePoolID := RESOURCE_POOL_ID
jobID := AIJobID
namespace := ""

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetJobNodesList(jobID, resourcePoolID, namespace)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为查询训练任务所在节点列表

获取训练任务WebTerminal地址

使用以下代码可以获取训练任务WebTerminal地址。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc"
// import "github.com/baidubce/bce-sdk-go/services/aihc/api/v1"
ak, sk, endpoint := ak_test, sk_test, endpoint_test

req := &v1.GetWebShellURLRequest{
JobID:                AIJobID,
ResourcePoolID:       RESOURCE_POOL_ID,
PodName:              PodName,
Namespace:            "",
PingTimeoutSecond:    "",
HandshakeTimeoutSecond: "",
}

client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.GetWebSSHUrl(req)

if err != nil {
panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档去填写具体的访问参数，接口链接为获取训练任务WebTerminal地址

在线服务部署相关

> 在线推理服务SDK中相关接口为最新的v2版本百舸API,相关接口文档详见 API参考/在线服务部署相关接口

创建服务

使用以下代码可以创建在线部署任务

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
client, _ := api.NewClient(ak, sk, endpoint)
createServiceArgs := &api.ServiceConf{
    Name:            "test",
    InstanceCount:   1,
    AcceleratorType: "",
    WorkloadType:    "",
    Containers: []api.ContainerConf{
        {
            Name:             "test-container",
            Cpus:             1,
            Memory:           1,
            AcceleratorCount: 0,
            Command:          []string{"/bin/sh", "-c", "sleep inf"},
            Image: api.ImageConf{
                ImageType: 1,
                ImageUrl:  "your image url",
            },
            Ports: []api.PortConf{
                {
                    Name: "HTTP",
                    Port: 8080,
                },
            },
        },
    },
    ResourcePool: api.ResourcePoolConf{
        ResourcePoolId:   "your resource pool id",
        ResourcePoolName: "your resource pool name",
        QueueName:        "your queue name",
    },
    Misc: api.Misc{
        GracePeriodSec: 30,
        FedPodsPerIns:  0,
    },
    Deploy: api.DeployConf{
        Schedule: api.ScheduleConf{
            Priority: "high",
        },
    },
    Access: api.AccessConf{
        PublicAccess: false,
        NetworkType:  "aiGateway",
        AiGateway: api.AiGatewayConf{
            EnableAuth: true,
        },
    },
}

result, err := client.CreateService(createServiceArgs, "your clientToken")

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

查询服务列表

使用以下代码可以查询服务列表

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ListService(&api.ListServiceArgs{
    PageSize:   10,
    PageNumber: 1,
})
if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

查询服务状态

使用以下代码可以查询服务状态

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ListServiceStats(&api.ListServiceStatsArgs{
    ServiceId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

查询服务详情

使用以下代码可以查询服务详情

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ServiceDetails(&api.ServiceDetailsArgs{
    ServiceId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

升级服务

使用以下代码可以对服务进行升级更新

```
 // import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
client, _ := api.NewClient(ak, sk, endpoint)
updateServiceArgs := &api.ServiceConf{
    Name:            "test",
    InstanceCount:   1,
    AcceleratorType: "",
    WorkloadType:    "",
    Containers: []api.ContainerConf{
        {
            Name:             "test-container",
            Cpus:             1,
            Memory:           1,
            AcceleratorCount: 0,
            Command:          []string{"/bin/sh", "-c", "sleep inf"},
            Image: api.ImageConf{
                ImageType: 1,
                ImageUrl:  "your image url",
            },
            Ports: []api.PortConf{
                {
                    Name: "HTTP",
                    Port: 8080,
                },
            },
        },
    },
    ResourcePool: api.ResourcePoolConf{
        ResourcePoolId:   "your resource pool id",
        ResourcePoolName: "your resource pool name",
        QueueName:        "your queue name",
    },
    Misc: api.Misc{
        GracePeriodSec: 30,
        FedPodsPerIns:  0,
    },
    Deploy: api.DeployConf{
        Schedule: api.ScheduleConf{
            Priority: "high",
        },
    },
    Access: api.AccessConf{
        PublicAccess: false,
        NetworkType:  "aiGateway",
        AiGateway: api.AiGatewayConf{
            EnableAuth: true,
        },
    },
}

result, err := client.UpdateService(&api.UpdateServiceArgs{
    ServiceId:   "you serviceId",
    ServiceConf: *updateServiceArgs,
    Description: "your description",
 })

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

服务扩缩容

使用以下代码可以对服务实例进行扩缩容

```
 // import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ScaleService(&api.ScaleServiceArgs{
    ServiceId:     "",
    InstanceCount: 1,
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

配置公网访问

使用以下代码可以配置开启和关闭公网

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.PubAccess(&api.PubAccessArgs{
    ServiceId:    "",
    PublicAccess: false,
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

查询变更记录

使用以下代码可以查询服务变更记录

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ListChange(&api.ListChangeArgs{
    ServiceId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

查询变更详情

使用以下代码可以查询服务变更详情

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ChangeDetail(&api.ChangeDetailArgs{
    ChangeId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

删除服务

使用以下代码可以删除服务

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.DeleteService(&api.DeleteServiceArgs{
    ServiceId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

查询服务实例组列表

使用以下代码可以查询服务实例组列表（仅限开启分布式的服务）

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ListPodGroups(&api.ListPodGroupsArgs{
    ServiceId: "",
})
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

查询服务pod列表

使用以下代码可以查询服务pod列表

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.ListPod(&api.ListPodArgs{
    ServiceId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

摘除服务pod流量

使用以下代码可以摘除和开启pod流量

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.BlockPod(&api.BlockPodArgs{
    ServiceId:  "",
    InstanceId: "",
    Block:      true,
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

删除服务pod并触发重建

使用以下代码可以删除pod并重建

```
// import api "github.com/baidubce/bce-sdk-go/services/aihc/inference/v2"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := api.NewClient(ak, sk, endpoint)
result, err := client.DeletePod(&api.DeletePodArgs{
    ServiceId:  "",
    InstanceId: "",
})

if err != nil {
    panic(err)
}

jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数

开发机相关

接口文档

> 开发机SDK中相关接口为最新的v2版本百舸API,相关接口文档详见 API参考/开发机相关接口

创建开发机 使用以下代码可以创建开发机。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.CreateDevInstance(&dev.CreateDevInstanceArgs{
    Name: "your_dev_instance_name",
        Conf: &DevInstanceConf{
            ResourcePool: &ResourcePool{
                ResourcePoolID:   "your_resource_pool_id",
                ResourcePoolName: "your_resource_pool_name",
                QueueName:        "queue_name",
            },
            Resources: &Resources{
                CPUs:   1,
                Memory: 2,
            },
            Image: &Image{
                ImageType: 0,
                ImageURL:  "your_dev_instance_image_url",
                Username:  "",
                Password:  "",
            },
            ScheduleConf: &ScheduleConf{
                CPUNodeAffinity: true,
                Priority:        "high",
            },
            VolumnConfs: []*VolumnConf{
                {
                    VolumnType: "cds",
                    MountPath:  "/.rootfs",
                    ReadOnly:   false,
                    CDS: &CDS{
                        Capacity: 100,
                    },
                },
            },
        },
        VisibleScope: &VisibleScope{
            Type: 1,
        },
        Creator:   "your_dev_instance_owner",
        CreatorID: "your_dev_instance_owner_user_id",
    }
)

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

查询开发机列表 使用以下代码可以查询开发机列表。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.ListDevInstance(&dev.ListDevInstanceArgs{
    PageNumber: 1,
    PageSize:   10,
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档填写具体的访问参数

查询开发机详情 使用以下代码可以查询开发机状态。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.QueryDevInstanceDetail(&dev.QueryDevInstanceDetailArgs{
    DevInstanceId: "your_dev_instance_id",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

更新开发机 使用以下代码可以更新开发机配置。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"
client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.UpdateDevInstance(&dev.CreateDevInstanceArgs{
    DevInstanceId: "your_dev_instance_id",
    Name: "your_dev_instance_name",
        Conf: &DevInstanceConf{
            ResourcePool: &ResourcePool{
                ResourcePoolID:   "your_resource_pool_id",
                ResourcePoolName: "your_resource_pool_name",
                QueueName:        "queue_name",
            },
            Resources: &Resources{
                CPUs:   2,
                Memory: 4,
            },
            Image: &Image{
                ImageType: 0,
                ImageURL:  "your_dev_instance_image_url",
                Username:  "",
                Password:  "",
            },
            ScheduleConf: &ScheduleConf{
                CPUNodeAffinity: true,
                Priority:        "high",
            },
            VolumnConfs: []*VolumnConf{
                {
                    VolumnType: "cds",
                    MountPath:  "/.rootfs",
                    ReadOnly:   false,
                    CDS: &CDS{
                        Capacity: 100,
                    },
                },
            },
        },
        VisibleScope: &VisibleScope{
            Type: 1,
        },
        Creator:   "your_dev_instance_owner",
        CreatorID: "your_dev_instance_owner_user_id",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

开启开发机实例 使用以下代码可以开启开发机实例。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.StartDevInstance(&dev.StartDevInstanceArgs{
    DevInstanceId: "your_dev_instance_id",

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

停止开发机实例 使用以下代码可以停止开发机实例。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.StopDevInstance(&dev.StopDevInstanceArgs{
    DevInstanceId: "your_dev_instance_id",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

删除开发机 使用以下代码可以删除开发机。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.DeleteDevInstance(&dev.DeleteDevInstanceArgs{
    DevInstanceId: "your_dev_instance_id",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

定时停止开发机实例 使用以下代码可以定时停止开发机实例。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.TimedStopDevInstance(&dev.TimedStopDevInstanceArgs{
    DevInstanceId: "your_dev_instance_id",
    DelaySec:      3600,
    Enable:        true,
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

制作镜像 使用以下代码可以制作开发机镜像。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)

result, err := client.CreateDevInstanceImagePackJob(&dev.CreateDevInstanceImagePackJobArgs{
    DevInstanceID: "your_dev_instance_id",
    ImageName:     "your_image_name",
    ImageTag:      "your_image_tag",
    Namespace:     "your_registry_namespace",
    Password:      "your_registry_password",
    Registry:      "your_registry",
    Username:      "your_registry_username",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

制作镜像任务详情 使用以下代码可以查询制作镜像任务详情。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)
result, err := client.DevInstanceImagePackJobDetail(&dev.DevInstanceImagePackJobDetailArgs{
    ImagePackJobId: "your_image_pack_job_id",
    DevInstanceId:  "your_dev_instance_id",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数开发机事件

查询开发机事件 使用以下代码可以查询开发机事件。

```
 // import "github.com/baidubce/bce-sdk-go/services/aihc/dev"
ak, sk, endpoint := "Your ak", "Your sk", "aihc.bj.baidubce.com"

client, _ := dev.NewClient(ak, sk, endpoint)

result, err := client.ListDevInstanceEvent(&dev.ListDevInstanceEventArgs{
    DevInstanceId: "your_dev_instance_id",
    StartTime:     "2025-05-18T17:12:20.761Z",
    EndTime:       "2025-06-04T05:30:23.337Z",
})

if err != nil {
    panic(err)
}
```

注意:

- 根据接口文档去填写具体的访问参数

数据集相关

获取数据集列表

使用以下代码可以查询数据集列表

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DescribeDatasets(&v2.DescribeDatasetsOptions{
    PageNumber: 1,
    PageSize:   10,
})
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集列表

获取数据集详情

使用以下代码可以查询数据集详情

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
DatasetID := "dataset-123"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DescribeDataset(DatasetID)
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意:

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集详情

创建数据集

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)

req := &v2.CreateDatasetRequest{
    Name:           "test-dataset-" + time.Now().Format("20060102150405"),
    StorageType:    "BOS",
    StorageInstance: "agilecloud-image",
    ImportFormat:   "FOLDER",
    Description:    "test dataset",
    VisibilityScope: "ONLY_OWNER",
    InitVersionEntry: v2.DatasetVersionEntry{
        Description: "test dataset version",
        StoragePath: "/",
        MountPath:   "/",
    },
}

result, err := client.CreateDataset(req)
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/创建数据集

修改数据集

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
DatasetID := "dataset-123"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.ModifyDataset(DatasetID, &v2.ModifyDatasetRequest{
    Name: "test-dataset-modify-" + time.Now().Format("20060102150405"),
})
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/修改数据集

删除数据集

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
DatasetID := "dataset-123"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DeleteDataset(DatasetID)
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/删除数据集

获取数据集版本列表

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
DatasetID := "dataset-123"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DescribeDatasetVersions(DatasetID, &v2.DescribeDatasetVersionsOptions{
    PageNumber: 1,
    PageSize:   10,
})
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集版本列表

获取数据集版本详情

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
DatasetID := "dataset-123"
DatasetVersionID := "dataset-version-123"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DescribeDatasetVersion(DatasetID, DatasetVersionID)
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集版本详情

创建数据集版本

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.CreateDatasetVersion(DatasetID, &v2.CreateDatasetVersionRequest{
    StoragePath: "/test-dataset-version",
    MountPath:   "/test-dataset-version",
    Description: "test dataset version",
})
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/创建数据集版本

删除数据集版本

```
 // import (
//  "encoding/json"
//  "fmt"
//  "time"

//  "github.com/baidubce/bce-sdk-go/services/aihc"
//  v2 "github.com/baidubce/bce-sdk-go/services/aihc/api/v2"
// )
DatasetID := "dataset-123"
DatasetVersionID := "dataset-version-123"
ak, sk, endpoint := ak_test, sk_test, endpoint_test
client, _ := aihc.NewClient(ak, sk, endpoint)
result, err := client.DeleteDatasetVersion(DatasetID, DatasetVersionID)
if err != nil {
    panic(err)
}
jsonBytes, _ := json.Marshal(result)
fmt.Println(string(jsonBytes))
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/删除数据集版本

JAVA SDK

概述

本文档主要介绍百度百舸JAVA语言版的开发者工具包（SDK），用户可基于该SDK使用JAVA语言接入百舸平台的服务。

SDK封装了便捷的调用接口，保持了多种编程语言版的使用方式、调用接口相似，提供了统一的错误码和返回格式，方便开发者调试。

安装SDK工具包

安装SDK工具包

运行环境

JAVA SDK可以在Java1.7，Java1.8环境下运行。

前提条件

AK/SK：SDK 认证时必须传入 AK/SK 参数，在安全认证页面 获取 Access Key和Secret Key。

用户可以通过两种方式与百度智能云进行交互，包括认证方式和匿名方式。对于认证方式，需要通过使用Access Key / Secret Key加密的方法来验证某个请求的发送者身份。Access Key(AK) 用于标示用户，Secret Key(SK) 是用户用于加密认证字符串和百度智能云用来验证认证字符串的密钥。

安装SDK

SDK目录结构

```
com.baidubce
├── auth                              //BCE签名相关类
├── http                              //BCE的Http通信相关类
├── internal                          //SDK内部类
├── model                             //BCE公用model类
├── services
│   └── aihc                          //AIHC服务相关类
│   └── ...                           //其它服务相关类
├── util                              //BCE公用工具类
├── BceClientConfiguration.class      //对BCE的HttpClient的配置
├── BceClientException.class          //BCE客户端的异常类
├── BceServiceException.class         //与BCE服务端交互后的异常类
├── ErrorCode.class                   //BCE通用的错误码
└── Region.class                      //BCE提供服务的区域
```

使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>{version}</version>
</dependency>
```

直接使用JAR包

1. 从百度云SDK中心下载Java SDK压缩包。
2. 将下载的bce-java-sdk-version.zip解压后，复制到工程文件夹中。
3. 在Eclipse右键"工程 -> Properties -> Java Build Path -> Add JARs"。
4. 添加SDK工具包lib/bce-java-sdk-version.jar和第三方依赖工具包third-party/*.jar。
5. 其中，version为版本号，经过上面几步之后，用户就可以在工程中使用BCE Java SDK。

卸载SDK

预期卸载SDK时，删除下载的源码即可。

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于服务域名的部分，理解Endpoint相关的概念。百度云目前开放了多区域支持，请参考区域选择说明。

访问区域	对应Endpoint
北京	bj
广州	gz
苏州	su
香港	hkg
武汉	fwh
保定	bd

获取密钥

要使用百度云百舸服务，您需要拥有一个有效的AK(Access Key ID)和SK(Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问百舸服务做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

注册百度云账号

创建AK/SK

创建Client对象

每种具体的服务都有一个Client对象，为开发者与对应的服务进行交互封装了一系列易用的方法。

调用功能接口

开发者基于创建的对应服务的Client对象，即可调用相应的功能接口，使用百度云产品的功能。

示例

下面给出一个基本的使用示例，详细使用说明请参考各服务的详细说明文档。

```
public class Sample {
  public static void main(String[] args) {
      String ak = <your-access-key-id>;              // 用户的Access Key ID
      String sk = <your-secret-access-key>;          // 用户的Secret Access Key
      String endpoint = "aihc.bj.baidubce.com";              // 修改区域标识可以切换不同region的域名
      // 初始化一个AihcClient
      AihcClientConfiguration config = new AihcClientConfiguration();
      config.setCredentials(new DefaultBceCredentials(ak, sk));
      config.setEndpoint(endpoint);
      config.setProtocol(Protocol.HTTPS);
      AihcClient client = new AihcClient(config);
      }
}
```

在上面代码中，变量ak与sk是由系统分配给用户的，均为字符串，用于标识用户，为访问百舸做签名验证。 其中ak对应控制台中的“Access Key ID”，sk对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 获取ACCESSKEY》。

上面的方式使用北京区域的域名作为百舸的默认服务地址，如果用户需要自己指定域名，可以通过修改域名中的区域信息来指定。

配置

使用HTTPS协议

该SDK支持使用HTTPS协议访问百度云的服务产品，您可以通过如下两种方式在BCC Java SDK中使用HTTPS访问BCC服务：

- 在endpoint中指明https:

```
 String ak = "ak";
String sk = "sk";
BceClientConfiguration config = new BceClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
AihcClient client = new AihcClient(config);
```

- 通过调用setProtocol方法设置https协议:

```
 String endpoint = "aihc.bj.baidubce.com"; // endpoint中不包含protocol
String ak = "ak";
String sk = "sk";
BceClientConfiguration config = new BceClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endpoint);
config.setProtocol(Protocol.HTTPS); // 如果不指明, 则使用http
AihcClient client = new AihcClient(config);
```

> 注意：如果在endpoint中指明了protocol, 则endpoint中的生效, 另外单独再调用setProtocol()不起作用。

```
 String endpoint = "http://aihc.bj.baidubce.com";
String ak = "ak";
String sk = "sk";
BceClientConfiguration config = new BceClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endpoint);
config.setProtocol(Protocol.HTTPS); // endpoint中已经指明, 此为无效操作, 对http也是如此
AihcClient client = new AihcClient(config);
```

配置AihcClient

如果用户需要配置AihcClient的一些细节的参数，可以在构造AihcClient的时候传入BceClientConfiguration对象。 BceClientConfiguration是服务的公用配置类，可以为客户端配置代理，最大连接数等参数。

使用代理

下面一段代码可以让客户端使用代理访问BCC服务：

```
 String ak = <your-access-key-id>;              // 用户的Access Key ID
String sk = <your-secret-access-key>;          // 用户的Secret Access Key
String endpoint = <domain-name>;               // 用户自己指定的域名

// 创建BceClientConfiguration实例
BceClientConfiguration config = new BceClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

// 创建AihcClient客户端
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endpoint);
AihcClient client = new AihcClient(config);
```

使用上面的代码段，客户端的所有操作都会通过127.0.0.1地址的8080端口做代理执行。 对于有用户验证的代理，可以通过下面的代码段配置用户名和密码：

```
 // 创建BceClientConfiguration实例
BceClientConfiguration config = new BceClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

//设置用户名和密码
config.setProxyUsername(<username>);                    //用户名
config.setProxyPassword(<password>);                   //密码
```

设置网络参数

用户可以用BceClientConfiguration对基本网络参数进行设置：

```
 BceClientConfiguration config = new BceClientConfiguration();

// 设置HTTP最大连接数为10
config.setMaxConnections(10);

// 设置TCP连接超时为5000毫秒
config.setConnectionTimeout(5000);

// 设置Socket传输数据超时的时间为2000毫秒
config.setSocketTimeout(2000);
```

参数说明

通过BceClientConfiguration能指定的所有参数如下表所示：

参数	说明
UserAgent用户代理，指HTTP的User-Agent头	
Protocol	连接协议类型
ProxyDomain	访问NTLM验证的代理服务器的Windows域名
ProxyHost	代理服务器主机地址
ProxyPort	代理服务器端口
ProxyUsername	代理服务器验证的用户名
ProxyPassword	代理服务器验证的密码
ProxyPreemptiveAuthenticationEnabled	是否设置用户代理认证
ProxyWorkstation	NTLM代理服务器的Windows工作站名称
LocalAddress	本地地址
ConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
SocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
MaxConnections	允许打开的最大HTTP连接数
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小
StreamBufferSize	流文件缓冲区大小

错误处理

BCC异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServerException	服务器异常

用户可以使用try获取某个事件所产生的异常。

SDK日志

BCC Java SDK发布版本中增加了logback作为slf4j的实现，如果用户没有自己的实现可以直接用，如果工程中有其他的如log4j则可以替代。

默认日志

如果用户使用默认的logback，则需要配置logback.xml到classpath中。如果没有这个配置文件，日志级别默认为DEBUG。

```
<configuration>
  <property name="LOG_HOME" value="./log/"/>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>${LOG_HOME}/BccUnitTest.%d{yyyy-MM-dd}.log</FileNamePattern>
      <MaxHistory>30</MaxHistory>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
    </encoder>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <MaxFileSize>10MB</MaxFileSize>
    </triggeringPolicy>
  </appender>

  <root level="info">
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="FILE"/>
  </root>
</configuration>
```

自有日志模块

若用户使用自己的日志实现模块，例如项目依赖于Maven，则可以类似下面的配置到pom.xml中来去除logback。

```
<?xml version="1.0" encoding="utf-8"?>

<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>${bce.sdk.version}</version>
  <exclusions>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
    </exclusion>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

数据集相关

获取数据集列表

使用以下代码可以查询数据集列表

```
DescribeDatasetsRequest request = new DescribeDatasetsRequest();
try {
    // 方式1：传统调用方式
    System.out.println("--- 方式1：传统调用方式 ---");
    DescribeDatasetsResponse response = client.dataset().describeDatasets(request);
    System.out.println("数据集总数: " + (response.getDatasets() != null ? response.getDatasets().size() : 0));

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);
} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意:

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集列表

获取数据集详情

使用以下代码可以查询数据集详情

```
 DescribeDatasetRequest request = new DescribeDatasetRequest();
// 设置必选参数
request.setDatasetId(System.getenv("DATASET_ID"));  // 数据集ID

try {
    // 方式1：传统调用方式（向后兼容）
    System.out.println("--- 方式1：传统调用方式 ---");
    DescribeDatasetResponse response = client.dataset().describeDataset(request);
    System.out.println("数据集ID: " + response.getId());
    System.out.println("数据集名称: " + response.getName());
    System.out.println("存储类型: " + response.getStorageType());
    System.out.println("存储实例: " + response.getStorageInstance());
    System.out.println("导入格式: " + response.getImportFormat());
    System.out.println("描述: " + response.getDescription());
    System.out.println("所有者: " + response.getOwner());
    System.out.println("所有者名称: " + response.getOwnerName());
    System.out.println("可见范围: " + response.getVisibilityScope());
    System.out.println("权限: " + response.getPermission());
    System.out.println("最新版本ID: " + response.getLatestVersionId());
    System.out.println("最新版本号: " + response.getLatestVersion());
    System.out.println("创建时间: " + response.getCreatedAt());
    System.out.println("更新时间: " + response.getUpdatedAt());

    // 打印最新版本信息
    if (response.getLatestVersionEntry() != null) {
        DescribeDatasetResponse.DatasetVersionEntry latestVersionEntry = response.getLatestVersionEntry();
        System.out.println("\n=== 最新版本信息 ===");
        System.out.println("版本ID: " + latestVersionEntry.getId());
        System.out.println("版本号: " + latestVersionEntry.getVersion());
        System.out.println("版本描述: " + latestVersionEntry.getDescription());
        System.out.println("存储路径: " + latestVersionEntry.getStoragePath());
        System.out.println("挂载路径: " + latestVersionEntry.getMountPath());
        System.out.println("创建用户: " + latestVersionEntry.getCreateUser());
        System.out.println("创建用户名: " + latestVersionEntry.getCreateUserName());
        System.out.println("版本创建时间: " + latestVersionEntry.getCreatedAt());
        System.out.println("版本更新时间: " + latestVersionEntry.getUpdatedAt());
    }

    // 打印用户权限列表
    if (response.getVisibilityUser() != null && !response.getVisibilityUser().isEmpty()) {
        System.out.println("\n=== 用户权限列表 ===");
        for (DescribeDatasetResponse.PermissionEntry user : response.getVisibilityUser()) {
            System.out.println("用户ID: " + user.getId() + ", 用户名: " + user.getName() + ", 权限: " + user.getPermission());
        }
    }

    // 打印用户组权限列表
    if (response.getVisibilityGroup() != null && !response.getVisibilityGroup().isEmpty()) {
        System.out.println("\n=== 用户组权限列表 ===");
        for (DescribeDatasetResponse.PermissionEntry group : response.getVisibilityGroup()) {
            System.out.println("用户组ID: " + group.getId() + ", 用户组名: " + group.getName() + ", 权限: " + group.getPermission());
        }
    }

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);
} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意:

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集详情

创建数据集

```
 CreateDatasetRequest request = new CreateDatasetRequest();

// 设置必选参数
request.setName("test-dataset-11111");  // 数据集名称
request.setStorageType("BOS");  // 存储类型：PFS 或 BOS
request.setStorageInstance("bucket1");  // 存储实例ID
request.setImportFormat("FOLDER");  // 导入格式：FILE 或 FOLDER
request.setVisibilityScope("USER_GROUP");  // 可见范围：ALL_PEOPLE、ONLY_OWNER、USER_GROUP

// 设置可选参数
request.setDescription("test dataset");  // 数据集描述
request.setOwner("xxx");  // 修改为实际的owner id（可选）

// 设置用户权限列表（可选）
List<CreateDatasetRequest.PermissionEntry> visibilityUser = new ArrayList<>();
CreateDatasetRequest.PermissionEntry user1 = new CreateDatasetRequest.PermissionEntry();
user1.setId("xxxxx"); // 修改为实际的user id
user1.setName("lisi");
user1.setPermission("r");  // 只读权限
visibilityUser.add(user1);

CreateDatasetRequest.PermissionEntry user2 = new CreateDatasetRequest.PermissionEntry();
user2.setId("xxxxx"); // 修改为实际的user id
user2.setName("wangwu");
user2.setPermission("rw");  // 读写权限
visibilityUser.add(user2);

request.setVisibilityUser(visibilityUser);

// 设置用户组权限列表（可选）
List<CreateDatasetRequest.PermissionEntry> visibilityGroup = new ArrayList<>();
CreateDatasetRequest.PermissionEntry group1 = new CreateDatasetRequest.PermissionEntry();
group1.setId("group1");
group1.setName("开发组");
group1.setPermission("r");
visibilityGroup.add(group1);

request.setVisibilityGroup(visibilityGroup);

// 设置初始版本信息（必选）
CreateDatasetRequest.DatasetVersionEntry initVersionEntry = new CreateDatasetRequest.DatasetVersionEntry();
initVersionEntry.setDescription("dataset first version");
initVersionEntry.setStoragePath("/path/to/dir");
initVersionEntry.setMountPath("/mnt/datasets/test1");
request.setInitVersionEntry(initVersionEntry);

try {
    System.out.println("=== 开始创建数据集 ===");
    System.out.println("数据集名称: " + request.getName());
    System.out.println("存储类型: " + request.getStorageType());
    System.out.println("存储实例: " + request.getStorageInstance());
    System.out.println("导入格式: " + request.getImportFormat());
    System.out.println("可见范围: " + request.getVisibilityScope());
    System.out.println("描述: " + request.getDescription());

    // 方式1：传统调用方式（向后兼容）
    System.out.println("\n--- 方式1：传统调用方式 ---");
    CreateDatasetResponse response = client.dataset().createDataset(request);
    System.out.println("数据集创建成功！");
    System.out.println("数据集ID: " + response.getId());
    System.out.println("Request ID: " + response.getMetadata().getBceRequestId());

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);

    System.out.println("\n=== 创建的数据集信息 ===");
    System.out.println("数据集ID: " + response.getId());
    System.out.println("数据集名称: " + request.getName());
    System.out.println("存储类型: " + request.getStorageType());
    System.out.println("存储实例: " + request.getStorageInstance());
    System.out.println("导入格式: " + request.getImportFormat());
    System.out.println("可见范围: " + request.getVisibilityScope());
    System.out.println("描述: " + request.getDescription());
    System.out.println("初始版本描述: " + initVersionEntry.getDescription());
    System.out.println("存储路径: " + initVersionEntry.getStoragePath());
    System.out.println("挂载路径: " + initVersionEntry.getMountPath());

} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/创建数据集

修改数据集

```
 ModifyDatasetRequest request = new ModifyDatasetRequest();

// 设置必选参数
request.setDatasetId(System.getenv("DATASET_ID"));  // 数据集ID

// 设置可选参数（根据需要修改）
request.setName("my-dataset-1");  // 数据集名称
request.setDescription("my dataset test modify");  // 数据集描述
request.setVisibilityScope("ALL_PEOPLE");  // 可见范围：ALL_PEOPLE、ONLY_OWNER、USER_GROUP

// 设置用户权限列表（可选）
List<ModifyDatasetRequest.PermissionEntry> visibilityUser = new ArrayList<>();
ModifyDatasetRequest.PermissionEntry user1 = new ModifyDatasetRequest.PermissionEntry();
user1.setId("xxxxx");
user1.setName("lisi");
user1.setPermission("r");  // 只读权限
visibilityUser.add(user1);

ModifyDatasetRequest.PermissionEntry user2 = new ModifyDatasetRequest.PermissionEntry();
user2.setId("xxxxx");
user2.setName("wangwu");
user2.setPermission("rw");  // 读写权限
visibilityUser.add(user2);

// request.setVisibilityUser(visibilityUser);

// 设置用户组权限列表（可选）
List<ModifyDatasetRequest.PermissionEntry> visibilityGroup = new ArrayList<>();
ModifyDatasetRequest.PermissionEntry group1 = new ModifyDatasetRequest.PermissionEntry();
group1.setId("group1");
group1.setName("开发组");
group1.setPermission("r");
visibilityGroup.add(group1);

// request.setVisibilityGroup(visibilityGroup);

try {
    System.out.println("=== 开始修改数据集 ===");
    System.out.println("数据集ID: " + request.getDatasetId());
    System.out.println("新名称: " + request.getName());
    System.out.println("新描述: " + request.getDescription());
    System.out.println("新可见范围: " + request.getVisibilityScope());

    // 方式1：传统调用方式（向后兼容）
    System.out.println("\n--- 方式1：传统调用方式 ---");
    ModifyDatasetResponse response = client.dataset().modifyDataset(request);
    System.out.println("数据集修改成功！");
    System.out.println("Request ID: " + response.getMetadata().getBceRequestId());

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);

    System.out.println("\n=== 修改的数据集信息 ===");
    System.out.println("数据集ID: " + request.getDatasetId());
    System.out.println("数据集名称: " + request.getName());
    System.out.println("数据集描述: " + request.getDescription());
    System.out.println("可见范围: " + request.getVisibilityScope());

    if (request.getVisibilityUser() != null && !request.getVisibilityUser().isEmpty()) {
        System.out.println("用户权限列表:");
        for (ModifyDatasetRequest.PermissionEntry user : request.getVisibilityUser()) {
            System.out.println("  - 用户ID: " + user.getId() + ", 用户名: " + user.getName() + ", 权限: " + user.getPermission());
        }
    }

    if (request.getVisibilityGroup() != null && !request.getVisibilityGroup().isEmpty()) {
        System.out.println("用户组权限列表:");
        for (ModifyDatasetRequest.PermissionEntry group : request.getVisibilityGroup()) {
            System.out.println("  - 用户组ID: " + group.getId() + ", 用户组名: " + group.getName() + ", 权限: " + group.getPermission());
        }
    }

} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/修改数据集

删除数据集

```
 DeleteDatasetRequest request = new DeleteDatasetRequest();
// 设置必选参数
request.setDatasetId(System.getenv("DATASET_ID"));  // 数据集ID

try {
    System.out.println("=== 开始删除数据集 ===");
    System.out.println("数据集ID: " + request.getDatasetId());

    // 方式1：传统调用方式（向后兼容）
    System.out.println("\n--- 方式1：传统调用方式 ---");
    DeleteDatasetResponse response = client.dataset().deleteDataset(request);
    System.out.println("数据集删除成功！");
    System.out.println("Request ID: " + response.getMetadata().getBceRequestId());

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);

    System.out.println("\n=== 注意事项 ===");
    System.out.println("删除操作是不可逆的，数据集及其所有版本将被永久删除。");
    System.out.println("请确保您要删除的数据集不再被使用。");

} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/删除数据集

获取数据集版本列表

```
 DescribeDatasetVersionsRequest request = new DescribeDatasetVersionsRequest();
// 设置必选参数
request.setDatasetId(System.getenv("DATASET_ID"));  // 数据集ID
// 设置可选参数
request.setPageNumber(1);  // 第1页
request.setPageSize(10);   // 每页10条

try {
    // 方式1：传统调用方式（向后兼容）
    System.out.println("--- 方式1：传统调用方式 ---");
    DescribeDatasetVersionsResponse response = client.dataset().describeDatasetVersions(request);
    System.out.println("数据集版本总数: " + response.getTotalCount());
    System.out.println("当前页版本数量: " + (response.getVersions() != null ? response.getVersions().size() : 0));

    // 打印版本列表
    if (response.getVersions() != null && !response.getVersions().isEmpty()) {
        System.out.println("\n=== 数据集版本列表 ===");
        for (int i = 0; i < response.getVersions().size(); i++) {
            DescribeDatasetVersionsResponse.DatasetVersionEntry version = response.getVersions().get(i);
            System.out.println("\n--- 版本 " + (i + 1) + " ---");
            System.out.println("版本ID: " + version.getId());
            System.out.println("版本号: " + version.getVersion());
            System.out.println("版本描述: " + version.getDescription());
            System.out.println("存储路径: " + version.getStoragePath());
            System.out.println("挂载路径: " + version.getMountPath());
            System.out.println("创建用户: " + version.getCreateUser());
            System.out.println("创建用户名: " + version.getCreateUserName());
            System.out.println("创建时间: " + version.getCreatedAt());
            System.out.println("更新时间: " + version.getUpdatedAt());
        }
    } else {
        System.out.println("\n=== 数据集版本列表 ===");
        System.out.println("暂无版本数据");
    }

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);
} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集版本列表

获取数据集版本详情

```
 DescribeDatasetVersionRequest request = new DescribeDatasetVersionRequest();
// 设置必选参数
request.setDatasetId(System.getenv("DATASET_ID"));  // 数据集ID
request.setVersionId(System.getenv("DATASET_VERSION_ID"));  // 数据集版本ID

try {
    // 方式1：传统调用方式（向后兼容）
    System.out.println("--- 方式1：传统调用方式 ---");
    DescribeDatasetVersionResponse response = client.dataset().describeDatasetVersion(request);
    System.out.println("数据集ID: " + response.getId());

    System.out.println("数据集名称: " + response.getName());
    System.out.println("存储类型: " + response.getStorageType());
    System.out.println("存储实例: " + response.getStorageInstance());
    System.out.println("导入格式: " + response.getImportFormat());
    System.out.println("描述: " + response.getDescription());
    System.out.println("所有者: " + response.getOwner());
    System.out.println("所有者名称: " + response.getOwnerName());
    System.out.println("可见范围: " + response.getVisibilityScope());
    System.out.println("权限: " + response.getPermission());
    System.out.println("创建时间: " + response.getCreatedAt());
    System.out.println("更新时间: " + response.getUpdatedAt());

    // 打印版本信息
    if (response.getVersionEntry() != null) {
        DescribeDatasetVersionResponse.DatasetVersionEntry versionEntry = response.getVersionEntry();
        System.out.println("\n=== 版本信息 ===");
        System.out.println("版本ID: " + versionEntry.getId());
        System.out.println("版本号: " + versionEntry.getVersion());
        System.out.println("版本描述: " + versionEntry.getDescription());
        System.out.println("存储路径: " + versionEntry.getStoragePath());
        System.out.println("挂载路径: " + versionEntry.getMountPath());
        System.out.println("创建用户: " + versionEntry.getCreateUser());
        System.out.println("创建用户名: " + versionEntry.getCreateUserName());
        System.out.println("版本创建时间: " + versionEntry.getCreatedAt());
        System.out.println("版本更新时间: " + versionEntry.getUpdatedAt());
    }

    // 打印用户权限列表
    if (response.getVisibilityUser() != null && !response.getVisibilityUser().isEmpty()) {
        System.out.println("\n=== 用户权限列表 ===");
        for (DescribeDatasetVersionResponse.PermissionEntry user : response.getVisibilityUser()) {
            System.out.println("用户ID: " + user.getId() + ", 用户名: " + user.getName() + ", 权限: " + user.getPermission());
        }
    }

    // 打印用户组权限列表
    if (response.getVisibilityGroup() != null && !response.getVisibilityGroup().isEmpty()) {
        System.out.println("\n=== 用户组权限列表 ===");
        for (DescribeDatasetVersionResponse.PermissionEntry group : response.getVisibilityGroup()) {
            System.out.println("用户组ID: " + group.getId() + ", 用户组名: " + group.getName() + ", 权限: " + group.getPermission());
        }
    }

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);
} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/获取数据集版本详情

创建数据集版本

```
 CreateDatasetVersionRequest request = new CreateDatasetVersionRequest();

// 设置必选参数
request.setDatasetId(System.getenv("DATASET_ID"));  // 数据集ID
request.setStoragePath("/path/to/version2");  // 存储路径
request.setMountPath("/mnt/datasets/my-dataset-1/v2");  // 默认挂载路径

// 设置可选参数
request.setDescription("new version");  // 版本描述

try {
    System.out.println("=== 开始创建数据集版本 ===");
    System.out.println("数据集ID: " + request.getDatasetId());
    System.out.println("版本描述: " + request.getDescription());
    System.out.println("存储路径: " + request.getStoragePath());
    System.out.println("挂载路径: " + request.getMountPath());

    // 方式1：传统调用方式（向后兼容）
    System.out.println("\n--- 方式1：传统调用方式 ---");
    CreateDatasetVersionResponse response = client.dataset().createDatasetVersion(request);
    System.out.println("数据集版本创建成功！");
    System.out.println("版本ID: " + response.getId());
    System.out.println("Request ID: " + response.getMetadata().getBceRequestId());

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);

    System.out.println("\n=== 创建的数据集版本信息 ===");
    System.out.println("数据集ID: " + request.getDatasetId());
    System.out.println("版本ID: " + response.getId());
    System.out.println("版本描述: " + request.getDescription());
    System.out.println("存储路径: " + request.getStoragePath());
    System.out.println("挂载路径: " + request.getMountPath());

    System.out.println("\n=== 注意事项 ===");
    System.out.println("新创建的版本将成为数据集的最新版本。");
    System.out.println("请确保存储路径中的数据已经准备好。");
    System.out.println("挂载路径将在使用数据集时生效。");

} catch (BceClientException e) {
    System.out.println("API调用失败: " + e.getMessage());
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/创建数据集版本

删除数据集版本

```
 // 创建请求对象
DeleteDatasetVersionRequest request = new DeleteDatasetVersionRequest();

// 设置必选参数
String datasetId = System.getenv("DATASET_ID");  // 数据集ID - 请替换为实际的数据集ID
String versionId = System.getenv("DATASET_VERSION_ID");  // 数据集版本ID - 请替换为实际的版本ID

request.setDatasetId(datasetId);
request.setVersionId(versionId);

try {
    System.out.println("=== 开始删除数据集版本 ===");
    System.out.println("数据集ID: " + request.getDatasetId());
    System.out.println("版本ID: " + request.getVersionId());
    System.out.println();

    // 确认删除操作
    System.out.println("警告：此操作将永久删除指定的数据集版本！");
    System.out.println("请确认您要删除的版本信息是否正确。");
    System.out.println();

    // 方式1：传统调用方式（向后兼容）
    System.out.println("--- 方式1：传统调用方式 ---");
    DeleteDatasetVersionResponse response = client.dataset().deleteDatasetVersion(request);
    System.out.println("数据集版本删除成功！");
    System.out.println("Request ID: " + response.getMetadata().getBceRequestId());

    // 打印完整的JSON响应
    System.out.println("\n=== API返回的JSON数据 ===");
    String jsonResponse = JsonUtils.toJsonString(response);
    System.out.println(jsonResponse);

    System.out.println("\n=== 操作完成 ===");
    System.out.println("数据集版本删除操作已成功完成");
    System.out.println("删除的版本信息：");
    System.out.println("   - 数据集ID: " + datasetId);
    System.out.println("   - 版本ID: " + versionId);
    System.out.println("   - 请求ID: " + response.getMetadata().getBceRequestId());

} catch (BceClientException e) {
    System.err.println("API调用失败: " + e.getMessage());
    System.err.println();
    System.err.println("可能的原因：");
    System.err.println("1. 访问密钥无效或权限不足");
    System.err.println("2. 数据集ID或版本ID不存在");
    System.err.println("3. 网络连接问题");
    System.err.println("4. 服务端点不可用");
    System.err.println();
    System.err.println("建议的解决方案：");
    System.err.println("1. 检查环境变量设置是否正确");
    System.err.println("2. 验证数据集ID和版本ID是否存在");
    System.err.println("3. 确认网络连接正常");
    System.err.println("4. 检查账户权限是否足够");
} catch (Exception e) {
    System.err.println("程序执行异常: " + e.getMessage());
    e.printStackTrace();
}
```

注意：

- 根据接口文档填写具体的访问参数，相关接口文档详见 API参考/数据集相关接口/删除数据集版本

Python SDK

概述

本文档主要介绍百度百舸Python语言版的开发者工具包（SDK），用户可基于该SDK使用Python语言接入百舸平台的服务。

SDK封装了便捷的调用接口，保持了多种编程语言版的使用方式、调用接口相似，提供了统一的错误码和返回格式，方便开发者调试。

> 注意：当前python sdk仅支持分布式训练、在线推理服务、资源池和数据集功能模块接口，其他功能模块暂不支持。

安装SDK工具包

安装SDK工具包

运行环境 在开始安装前，请确保您的系统满足以下要求：

- **Python 版本**：SDK 当前仅支持 Python 3。为确保兼容性和安全性，我们强烈建议使用 **Python 3.7 或更高版本**的稳定发行版。
- **SDK 版本**：请确保安装的 SDK 版本大于 `0.9.50`，以包含您所需的功能和最新的安全更新。

> **提示**：您可以在命令行中运行 `python --version`（或 `python3 --version`）来检查您当前的 Python 版本。

安装SDK

您可以选择以下两种方式之一进行安装。我们**推荐使用 pip 源安装**，这是最简便的方法。

方式一：pip源安装（推荐）

1. 打开终端/命令提示符。
2. 执行以下命令即可完成安装：

```
pip install bce-python-sdk>0.9.50
```

方式二：从SDK中心下载，手动安装 安装步骤

1. **从SDK中心下载代码**： 从百度智能云官方网站的SDK中心下载Python SDK的源代码压缩包（.zip格式），并将其解压到您选择的本地目录。

2. **打开终端/命令提示符**： 打开命令行工具（如 Terminal、CMD 或 PowerShell），然后使用 cd 命令切换到 SDK 代码所在的根目录。该目录下应包含 setup.py 文件和 baidubce 文件夹。

3. **执行安装命令**： 在命令行中运行以下命令，该命令会自动处理所有依赖并完成安装：

```
python setup.py install
```

验证安装

无论使用哪种安装方式，安装完成后，都建议进行验证以确保SDK已正确安装。

1. 打开 Python 解释器或创建一个新的 Python 脚本文件。

2. 尝试导入 baidubce 模块：

```
# 在 Python 交互界面或脚本中运行
import baidubce
```

3. 如果命令行没有报错，并成功打印出提示信息，则说明 SDK 已经准备就绪。

现在，您可以开始使用百度智能云 Python SDK 进行开发了。请参考官方文档获取详细的使用方法和示例代码。

SDK目录结构

```
python/
├── baidubce/services/aihc/          # AIHC核心SDK模块
│   ├── base/                        # 基础模块
│   │   ├── __init__.py
│   │   ├── aihc_base_client.py      # 基础客户端类
│   │   └── aihc_common.py           # 公共工具函数
│   ├── modules/                     # 业务模块目录
│   │   ├── dataset/                 # 数据集模块
│   │   │   ├── __init__.py
│   │   │   ├── dataset_client.py    # 数据集相关接口
│   │   │   └── dataset_model.py     # 数据集模型定义
│   │   ├── job/                     # 训练任务模块
│   │   │   ├── __init__.py
│   │   │   ├── job_client.py        # 任务相关接口
│   │   │   └── job_model.py         # 任务模型定义
│   │   ├── resource_pool/           # 资源池模块
│   │   │   ├── __init__.py
│   │   │   └── resource_pool_client.py # 资源池相关接口
│   │   └── service/                 # 在线推理服务模块
│   │       ├── __init__.py
│   │       ├── service_client.py    # 服务相关接口
│   │       └── service_model.py     # 服务模型定义
├── sample/aihc/                     # AIHC示例代码目录
│   ├── aihc_dataset_sample.py       # 数据集操作示例
│   ├── aihc_job_sample.py           # 训练任务操作示例
│   ├── aihc_resource_pool_sample.py # 资源池操作示例
│   ├── aihc_sample_conf.py          # 示例配置文件
│   ├── aihc_service_sample.py       # 在线推理服务操作示例
```

卸载SDK 使用pip uninstall bce-python-sdk即可卸载sdk工具包

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于服务域名的部分，理解Endpoint相关的概念。百度云目前开放了多区域支持，请参考区域选择说明。

访问区域	对应Endpoint
北京	bj
广州	gz
苏州	su
香港	hkg
武汉	fwh
保定	bd

获取密钥

要使用百度云百舸服务，您需要拥有一个有效的AK(Access Key ID)和SK(Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问百舸服务做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

注册百度云账号

创建AK/SK

新建 Aihc Client

AihcClient 是百舸AIHC服务的客户端，为开发者与百舸AIHC服务交互提供了一系列的方法。

初始化客户端示例

```
import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 配置日志记录（可选）
logger = logging.getLogger('baidubce.http.bce_http_client')
fh = logging.FileHandler('sample.log')
fh.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
logger.setLevel(logging.DEBUG)
logger.addHandler(fh)

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 初始化AihcClient实例
aihc_client = AihcClient(config)
```

参数说明

- **AK（Access Key ID）**：对应百度智能云控制台中的"Access Key ID"，用于标识用户身份。
- **SK（Secret Access Key）**：对应百度智能云控制台中的"Access Key Secret"，用于API请求签名认证。
- **HOST（服务端点）**：百舸AIHC服务的访问地址。如果设置为空字符串或None，SDK会使用默认域名作为服务地址。

环境变量配置（推荐） 建议使用环境变量管理敏感信息，创建`.env`文件：

```
##### .env 文件示例
HOST=https://aihc.bj.baidubce.com
AK=your-actual-access-key-id
SK=your-actual-secret-access-key
```

日志配置 代码中包含了日志配置，会将SDK的详细请求日志记录到`sample.log`文件中，便于调试和问题排查。在实际生产环境中，可以根据需要调整日志级别和输出方式。

完成初始化后，您就可以使用`aihc_client`实例调用百舸AIHC服务的各种接口了。

资源池相关

查询资源池列表

使用以下代码可以查询资源池列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 aihc client
aihc_client = AihcClient(config)

##### 查询资源池列表
try:
    __logger.info('--------------------------------DescribeResourcePools start--------------------------------')
    # resourcePoolType: 资源池类型：1、通用资源池：common，2、托管资源池：dedicatedV2
    response = aihc_client.resource_pool.DescribeResourcePools(resourcePoolType='common')
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询资源池列表

查询资源池详情

使用以下代码可以查询资源池详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询资源池详情
try:
    __logger.info('----------------------DescribeResourcePool start--------------------------------')
    resource_pool_id = "cce-xxx" # 替换为实际的资源池ID
    response = aihc_client.resource_pool.DescribeResourcePool(resourcePoolId=resource_pool_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询资源池详情

查询资源池概览

使用以下代码可以查询资源池概览。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询资源池概览
try:
    __logger.info('--------------------------------DescribeResourcePoolOverview start--------------------------------')
    response = aihc_client.resource_pool.DescribeResourcePoolOverview()
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询资源池概览

查询资源池配置

使用以下代码可以查询资源池配置。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询资源池配置
try:
    __logger.info('----------DescribeResourcePoolConfiguration start----------')
    resource_pool_id = "cce-xxx" # 替换为实际的资源池ID
    response = aihc_client.resource_pool.DescribeResourcePoolConfiguration(resourcePoolId=resource_pool_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询资源池配置

查询队列列表

使用以下代码可以查询队列列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询队列列表
try:
    __logger.info('--------------------------------DescribeQueues start--------------------------------')
    resource_pool_id = "cce-xxx" # 替换为实际的资源池ID
    response = aihc_client.resource_pool.DescribeQueues(resourcePoolId=resource_pool_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询队列列表

查询队列详情

使用以下代码可以查询队列详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询队列详情
try:
    __logger.info('--------------------------------DescribeQueue start--------------------------------')
    queue_id = "queue-xxx" # 替换为实际的队列ID
    response = aihc_client.resource_pool.DescribeQueue(queueId=queue_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询队列详情

分布式训练相关

查询训练任务列表

使用以下代码可以查询训练任务列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 aihc client
aihc_client = AihcClient(config)

##### 查询训练任务列表
try:
    __logger.info('--------------------------------DescribeJobs start--------------------------------')
    resource_pool_id = "cce-xxx"  # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"          # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    keyword = ""                  # 搜索关键字（可选）
    page_number = 1               # 页码，从 1 开始
    page_size = 10                # 每页条数
    response = aihc_client.job.DescribeJobs(
        resourcePoolId=resource_pool_id,
        queue=queue_id,
        keyword=keyword,
        pageNumber=page_number,
        pageSize=page_size,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务列表

查询训练任务详情

使用以下代码可以查询训练任务详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询训练任务详情
try:
    __logger.info('--------------------------------DescribeJob start--------------------------------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    need_detail = True
    response = aihc_client.job.DescribeJob(resourcePoolId=resource_pool_id, queueID=queue_id, jobId=job_id,
                                           needDetail=need_detail)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务详情

删除训练任务

使用以下代码可以删除训练任务。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 删除训练任务
try:
    __logger.info('--------------------------------DeleteJob start--------------------------------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    response = aihc_client.job.DeleteJob(resourcePoolId=resource_pool_id, queueID=queue_id, jobId=job_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为删除训练任务

更新训练任务

使用以下代码可以更新训练任务（如修改优先级）。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 更新训练任务
try:
    __logger.info('----------ModifyJob start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    priority = "high"            # 新的优先级
    response = aihc_client.job.ModifyJob(resourcePoolId=resource_pool_id, queueID=queue_id, jobId=job_id, priority=priority)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为更新训练任务

查询训练任务事件

使用以下代码可以查询训练任务事件。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询训练任务事件
try:
    __logger.info('----------DescribeJobEvents start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    start_time = "1758532230"    # 起始时间戳（可选）
    end_time = "1758618650"      # 结束时间戳（可选）
    response = aihc_client.job.DescribeJobEvents(resourcePoolId=resource_pool_id, queueID=queue_id, jobId=job_id,
                                                 startTime=start_time, endTime=end_time)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务事件

查询训练任务日志

使用以下代码可以查询训练任务日志。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询训练任务日志
try:
    __logger.info('----------DescribeJobLogs start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    pod_name = "xxx-test-copy2-master-0" # 替换为实际的pod名称
    keywords = "xxx"             # 搜索关键字
    response = aihc_client.job.DescribeJobLogs(resourcePoolId=resource_pool_id,
                                               queueID=queue_id,
                                               jobId=job_id,
                                               keywords=keywords,
                                               podName=pod_name)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务日志

查询训练任务Pod事件

使用以下代码可以查询训练任务Pod事件。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询训练任务Pod事件
try:
    __logger.info('----------DescribeJobPodEvents start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    pod_name = "job-xxx-master-0" # 替换为实际的pod名称
    job_id = "job-xxx"           # 替换为实际的任务ID
    response = aihc_client.job.DescribeJobPodEvents(resourcePoolId=resource_pool_id,
                                            queueID=queue_id,
                                            jobId=job_id,
                                            podName=pod_name)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务Pod事件

停止训练任务

使用以下代码可以停止训练任务。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 停止训练任务
try:
    __logger.info('stop job')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    response = aihc_client.job.StopJob(resourcePoolId=resource_pool_id, queueID=queue_id, jobId=job_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为停止训练任务

查询训练任务所在节点列表

使用以下代码可以查询训练任务所在节点列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询训练任务所在节点列表
try:
    __logger.info('----------DescribeJobNodes start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    response = aihc_client.job.DescribeJobNodes(resourcePoolId=resource_pool_id, queueID=queue_id, jobId=job_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务所在节点列表

获取训练任务WebTerminal地址

使用以下代码可以获取训练任务WebTerminal地址。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 获取训练任务WebTerminal地址
try:
    __logger.info('----------DescribeJobWebterminal start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    job_id = "job-xxx"           # 替换为实际的任务ID
    podName = "xxx-test-bb1-rerun1-master-0" # 替换为实际的pod名称
    handshake_timeout_second = "30"  # 可选参数
    ping_timeout_second = "900"      # 可选参数
    response = aihc_client.job.DescribeJobWebterminal(
        resourcePoolId=resource_pool_id,
        queueID=queue_id,
        jobId=job_id,
        podName=podName,
        pingTimeoutSecond=ping_timeout_second,
        handshakeTimeoutSecond=handshake_timeout_second,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为获取训练任务WebTerminal地址

创建训练任务

使用以下代码可以创建训练任务。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.services.aihc.modules.job.job_model import Datasource, Env, JobSpec
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 创建训练任务
try:
    __logger.info('------------------------------CreateJob start------------------------------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    queue_id = "default"         # 训练任务所属队列，通用资源池须填入队列名称，托管资源池须填入队列Id
    name = "python-sdk-test-xxx" # 任务名称
    command = "sleep 5m"         # 启动命令

    # 配置环境变量
    envs = [Env(name="NCCL_DEBUG", value="DEBUG"), Env(name="NCCL_IB_DISABLE", value="0")]
```

```
    # 配置数据源
    data_source = Datasource(type="pfs", name="pfs-xxx", mountPath="/mnt/cluster")  # pfs数据源挂载示例
    # data_source = Datasource(type="hostpath", name="hostpath-1", sourcePath="/a"
    #                          , mountPath="/b", options={"readOnly": True})  # 本地盘挂载示例
    # data_source = Datasource(type="bos", sourcePath="xxx-test-bos-bucket",
    #                          mountPath="/test", options={"readOnly": False})  # bos数据源挂载示例
    # data_source = Datasource(type="cfs", name="", mountPath="/data", sourcePath="/data",
    # options={"cfsInstanceId": "cfs-xxx", "cfsMountPoint": "cfs-xxx.lb-xxx.cfs.xx.baidubce.com"})  # cfs存储挂载示例

    # 配置任务规格
    job_spec = JobSpec(
        image="registry.baidubce.com/aihc-aiak/aiak-megatron:ubuntu20.04-cu11.8-torch1.14.0-py38_v1.2.7.12_release",
        replicas=1,  # 实例副本数
        # 资源配置示例，可根据实际需求进行调整
        resources=[
            # # 1核CPU
            # {
            #     "name": "cpu",
            #     "quantity": 1
            # },
            # # 内存4G
            # {
            #     "name": "memory",
            #     "quantity": 4
            # },
            # # A100-SXM4-40GB显卡1张
            # {
            #     "name": "baidu.com/a100_40g_cgpu",
            #     "quantity": 1
            # },
            # # 共享内存32G
            # {
            #     "name": "sharedMemory",
            #     "quantity": 32
            # }
        ],
        envs=envs,
        enableRDMA=False  # 是否开启RDMA
    )

    job_type = "PyTorchJob"
    labels = []  # 任务标签
    priority = "normal"  # 任务优先级
    dataSources = [
        data_source,
    ]
    enable_bccl = False  # 是否开启BCCL自动注入
    fault_tolerance = False  # 是否开启容错
    fault_tolerance_args = ""  # 容错参数设置
    tensorboard_config = {}  # tensorboard配置
    retention_period = ""  # 任务保留时间

    response = aihc_client.job.CreateJob(
        resourcePoolId=resource_pool_id,
        queueID=queue_id,
        name=name,
        command=command,
        jobSpec=job_spec,
        jobType=job_type,
        labels=labels,
        priority=priority,
        dataSources=dataSources,
        enableBccl=enable_bccl,
        faultTolerance=fault_tolerance,
        faultToleranceArgs=fault_tolerance_args,
        tensorboardConfig=tensorboard_config,
        retentionPeriod=retention_period,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为创建训练任务

查询训练任务监控

使用以下代码可以查询训练任务监控数据。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询训练任务监控
try:
    __logger.info('----------DescribeJobMetrics start----------')
    resource_pool_id = "cce-xxx" # 自运维资源池传递资源池唯一标识（示例：cce-1uji3ib5），托管资源池传递 aihc-serverless
    job_id = "job-xxx"           # 替换为实际的任务ID
    start_time = "1758359060"    # 起始时间戳
    end_time = "1758445563"      # 结束时间戳
    time_step = "5m"             # 监控数据间隔
    metric_type = "GpuUsage"     # 监控指标类型
    rate_interval = "5m"         # 指标变化周期频率
    response = aihc_client.job.DescribeJobMetrics(resourcePoolId=resource_pool_id,
                                                  jobId=job_id,
                                                  metricType=metric_type,
                                                  startTime=start_time,
                                                  endTime=end_time,
                                                  timeStep=time_step,
                                                  rateInterval=rate_interval)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询训练任务监控

在线服务部署相关

拉取服务列表

使用以下代码可以拉取服务列表。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------------------DescribeServices start--------------------------------')
    page_number = 1
    page_size = 10
    order_by = "createdAt"
    order = "asc"

    response = aihc_client.service.DescribeServices(
        pageNumber=page_number,
        pageSize=page_size,
        orderBy=order_by,
        order=order
    )
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为拉取服务列表

查询服务详情

使用以下代码可以查询服务详情。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------------------DescribeService start--------------------------------')
    service_id = "s-xxxxxxxx"  # 替换为实际的服务 ID
    response = aihc_client.service.DescribeService(serviceId=service_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询服务详情

获取服务状态

使用以下代码可以获取服务状态。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('----------------------DescribeServiceStatus start--------------------------------')
    service_id = "s-xxxxxxxx"  # 替换为实际的服务 ID
    response = aihc_client.service.DescribeServiceStatus(serviceId=service_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为获取服务状态

创建在线服务

使用以下代码可以创建在线服务。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError
from baidubce.services.aihc.modules.service.service_model import ResourcePoolConf, ImageConf, ContainerConf

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------------------CreateService start--------------------------------')
    # 构造资源池配置
    resource_pool_conf = ResourcePoolConf(
        resourcePoolId="cce-xxx",
        queueName='default',
        resourcePoolType="",
        resourcePoolName="pool-xxx"
    )

    # 构造镜像配置
    image_conf = ImageConf(
        imageType=1,
        imageUrl="registry.baidubce.com/csm-online/sglang-router:0.2.0",
    )

    # 构造容器配置
    containers = [ContainerConf(
        name="container-test",
        cpus=16,
        memory=64,
        acceleratorCount=1,
        image=image_conf,
        command=[
            "python", "-m", "sglang_router.launch_router",
            "--host", "0.0.0.0", "--port", "30000"
        ],
    )]

    # 创建服务
    response = aihc_client.service.CreateService(
        serviceConf={
            "name": "pythonsdk-aihc-service-test",
            "acceleratorType": "NVIDIA H20-3e",
            "workloadType": "",
            "instanceCount": 1,
            "resourcePool": resource_pool_conf,
            "containers": containers
        }
    )
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为创建在线服务

删除在线服务

使用以下代码可以删除在线服务。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('----------DeleteService start----------')
    service_id = "s-xxxxxxxx"
    response = aihc_client.service.DeleteService(serviceId=service_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为删除在线服务

升级在线服务

使用以下代码可以升级在线服务。

```python
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError
from baidubce.services.aihc.modules.service.service_model import ResourcePoolConf, ImageConf, ContainerConf, ModifyServiceConf

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------------------ModifyService start--------------------------------')
    service_id = "s-xxxxxxxx"

    # 构造更新后的配置
    resource_pool_conf = ResourcePoolConf(
        resourcePoolId="cce-xxx",
        queueName='default',
        resourcePoolType="",
        resourcePoolName="pool-xxx"
    )

    service_conf = ModifyServiceConf(
        name="service-test-updated",
        acceleratorType="NVIDIA H20-3e",
        instanceCount=3,
        resourcePool=resource_pool_conf,
        containers=[ContainerConf(
            name="custom-container",
            cpus=16,
            memory=64,
            acceleratorCount=1,
            image=ImageConf(
                imageType=1,
                imageUrl="registry.baidubce.com/csm-online/sglang-router:0.2.0",
            ),
            command=["python", "launch.py"]
        )]
    )

    response = aihc_client.service.ModifyService(serviceId=service_id, serviceConf=service_conf)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为升级在线服务

服务扩缩容

使用以下代码可以服务扩缩容。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('----------------------ModifyServiceReplicas start----------------------')
    service_id = "s-xxxxxxxx"
    instance_count = 2  # 目标实例数

    response = aihc_client.service.ModifyServiceReplicas(
        serviceId=service_id,
        instanceCount=instance_count
    )
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为服务扩缩容

拉取服务pod列表

使用以下代码可以拉取服务pod列表。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('------------------------DescribeServicePods start--------------------------------')
    service_id = "s-xxxxxxxx"
    response = aihc_client.service.DescribeServicePods(serviceId=service_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为拉取服务pod列表

删除Pod并重建

使用以下代码可以删除Pod并重建。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------------------DeleteServicePod start--------------------------------')
    service_id = "s-xxxxxxxx"
    instance_id = "pod-xxxxxxxx"  # Pod 实例 ID

    response = aihc_client.service.DeleteServicePod(
        serviceId=service_id,
        instanceId=instance_id
    )
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为删除Pod并重建

摘除Pod流量

使用以下代码可以摘除Pod流量。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------------------DisableServicePod start--------------------------------')
    service_id = "s-xxxxxxxx"
    instance_id = "pod-xxxxxxxx"
    block = False  # 是否阻塞等待

    response = aihc_client.service.DisableServicePod(
        serviceId=service_id,
        instanceId=instance_id,
        block=block
    )
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为摘除Pod流量

配置公网访问

使用以下代码可以配置公网访问。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('----------ModifyServiceNetConfig start----------')
    service_id = "s-xxxxxxxx"
    public_access = False  # False 关闭，True 开启

    response = aihc_client.service.ModifyServiceNetConfig(
        serviceId=service_id,
        publicAccess=public_access
    )
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为配置公网访问

获取实例组列表

使用以下代码可以获取实例组列表。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('------------------DescribeServicePodGroups start--------------------------------')
    service_id = "s-xxxxxxxx"
    response = aihc_client.service.DescribeServicePodGroups(serviceId=service_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为获取实例组列表

拉取服务变更记录

使用以下代码可以拉取服务变更记录。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('--------------------DescribeServiceChangelogs start--------------------------------')
    service_id = "s-xxxxxxxx"
    response = aihc_client.service.DescribeServiceChangelogs(serviceId=service_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为拉取服务变更记录

查询服务变更详情

使用以下代码可以查询服务变更详情。

```
 import logging
import os
import json

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

##### 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 AihcClient 实例
aihc_client = AihcClient(config)

##### 调用接口
try:
    __logger.info('------------------------DescribeServiceChangelog start----------------------------')
    change_id = "ch-xxxxxxxx"  # 变更 ID
    response = aihc_client.service.DescribeServiceChangelog(changeId=change_id)
    print(json.dumps(response.__dict__, default=lambda o: o.__dict__, ensure_ascii=False))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询服务变更详情

数据集相关

获取数据集列表

使用以下代码可以获取数据集列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 aihc client
aihc_client = AihcClient(config)

##### 查询数据集列表
try:
    __logger.info('--------------------DescribeDatasets start--------------------')
    response = aihc_client.dataset.DescribeDatasets()
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为获取数据集列表

获取数据集详情

使用以下代码可以获取数据集详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询数据集详情
try:
    __logger.info('--------------------DescribeDataset start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    response = aihc_client.dataset.DescribeDataset(datasetId=dataset_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为获取数据集详情

获取数据集版本列表

使用以下代码可以获取数据集版本列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 获取数据集版本列表
try:
    __logger.info('--------------------DescribeDatasetVersions start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    response = aihc_client.dataset.DescribeDatasetVersions(datasetId=dataset_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为获取数据集版本列表

获取数据集版本详情

使用以下代码可以获取数据集版本详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 获取数据集版本详情
try:
    __logger.info('--------------------DescribeDatasetVersion start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    version_id = "v-xxx" # 替换为实际的数据集版本ID
    response = aihc_client.dataset.DescribeDatasetVersion(datasetId=dataset_id, versionId=version_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为获取数据集版本详情

修改数据集

使用以下代码可以修改数据集。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 修改数据集
try:
    __logger.info('--------------------ModifyDataset start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    response = aihc_client.dataset.ModifyDataset(datasetId=dataset_id, name='test-dataset-xxx')
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为修改数据集

删除数据集

使用以下代码可以删除数据集。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 删除数据集
try:
    __logger.info('--------------------DeleteDataset start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    response = aihc_client.dataset.DeleteDataset(datasetId=dataset_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为删除数据集

创建数据集

使用以下代码可以创建数据集。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.services.aihc.modules.dataset.dataset_model import DatasetVersionEntry
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 创建数据集
try:
    __logger.info('--------------------CreateDataset start--------------------')
    response = aihc_client.dataset.CreateDataset(
        name='sdk-test-dataset-xxx-xxx',
        storageType='PFS',
        storageInstance='pfs-xxx',
        importFormat='FOLDER',
        visibilityScope='ALL_PEOPLE',
        initVersionEntry=DatasetVersionEntry(
            storagePath='/',
            mountPath='/xx'
        )
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为创建数据集

创建数据集版本

使用以下代码可以创建数据集版本。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 创建数据集版本
try:
    __logger.info('--------------------CreateDatasetVersion start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    response = aihc_client.dataset.CreateDatasetVersion(
        datasetId=dataset_id, storagePath='/xxx-xxx',
        mountPath='/xxx/xxx', description='Test dataset version'
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为创建数据集版本

删除数据集版本

使用以下代码可以删除数据集版本。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 删除数据集版本
try:
    __logger.info('--------------------DeleteDatasetVersion start--------------------')
    dataset_id = "d-xxx" # 替换为实际的数据集ID
    version_id = "v-xxx" # 替换为实际的数据集版本ID
    response = aihc_client.dataset.DeleteDatasetVersion(datasetId=dataset_id, versionId=version_id)
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为删除数据集版本

工作流相关

创建工作流模版

使用以下代码可以创建工作流模版。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量（可选，推荐使用）
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息和服务端点
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'  # 百舸AIHC服务地址
AK = os.getenv('AK') or 'your-access-key-id'                # 您的Access Key ID
SK = os.getenv('SK') or 'your-secret-access-key'            # 您的Secret Access Key

##### 创建BCE客户端配置
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

##### 创建 aihc client
aihc_client = AihcClient(config)

##### 创建工作流模版
try:
    __logger.info('--------------------------------CreatePipeline start--------------------------------')
    name = "python-sdk-test-pipeline"  # 工作流模版名称
    description = "示例 Pipeline"      # 工作流模版描述
    pipeline_type = "normal"           # 工作流模版类型
    resource_pool_id = "aihc-xxx"      # 替换为实际的资源池ID
    concurrency_policy = "Allow"       # 并发策略：Allow / Forbid / Replace
    manifest = (                       # 工作流模版 YAML 定义，详见接口文档
        "version: v1\n"
        "kind: PipelineTemplate\n"
        "taskTemplates:\n"
        "  - name: custom-task-template\n"
        "    type: CustomTask\n"
    )
    response = aihc_client.workflow.CreatePipeline(
        name=name,
        manifest=manifest,
        description=description,
        pipelineType=pipeline_type,
        resourcePoolId=resource_pool_id,
        concurrencyPolicy=concurrency_policy,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为创建工作流模版

查询工作流模版列表

使用以下代码可以查询工作流模版列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询工作流模版列表
try:
    __logger.info('--------------------------------DescribePipelines start--------------------------------')
    page_number = 1  # 页码，从 1 开始
    page_size = 10   # 每页条数
    response = aihc_client.workflow.DescribePipelines(
        pageNumber=page_number,
        pageSize=page_size,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询工作流模版列表

查询工作流模版详情

使用以下代码可以查询工作流模版详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询工作流模版详情
try:
    __logger.info('--------------------------------DescribePipeline start--------------------------------')
    pipeline_id = "pipe-xxx"  # 替换为实际的工作流模版ID
    response = aihc_client.workflow.DescribePipeline(
        pipelineId=pipeline_id,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为查询工作流模版详情

更新工作流模版

使用以下代码可以更新工作流模版。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 更新工作流模版
try:
    __logger.info('--------------------------------UpdatePipeline start--------------------------------')
    pipeline_id = "pipe-xxx"          # 替换为实际的工作流模版ID
    new_name = "updated-pipeline"     # 更新后的模版名称
    new_description = "更新后的描述"  # 更新后的描述
    response = aihc_client.workflow.UpdatePipeline(
        pipelineId=pipeline_id,
        name=new_name,
        description=new_description,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为更新工作流模版

删除工作流模版

使用以下代码可以删除工作流模版。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 删除工作流模版
try:
    __logger.info('--------------------------------DeletePipeline start--------------------------------')
    pipeline_id = "pipe-xxx"  # 替换为实际的工作流模版ID
    response = aihc_client.workflow.DeletePipeline(
        pipelineId=pipeline_id,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为删除工作流模版

运行工作流

使用以下代码可以运行工作流（创建工作流运行实例）。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 运行工作流
try:
    __logger.info('--------------------------------CreatePipelineRun start--------------------------------')
    pipeline_id = "pipe-xxx"  # 替换为实际的工作流模版ID
    response = aihc_client.workflow.CreatePipelineRun(
        pipelineId=pipeline_id,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

注意: 根据接口文档去填写具体的访问参数，接口链接为运行工作流

查询工作流运行列表

使用以下代码可以查询工作流运行列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询工作流运行列表
try:
    __logger.info('--------------------------------DescribePipelineRuns start--------------------------------')
    pipeline_id = "pipe-xxx"  # 替换为实际的工作流模版ID
    page_number = 1           # 页码，从 1 开始
    page_size = 10            # 每页条数
    response = aihc_client.workflow.DescribePipelineRuns(
        pipelineId=pipeline_id,
        pageNumber=page_number,
        pageSize=page_size,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询工作流运行列表

查询工作流运行详情

使用以下代码可以查询工作流运行详情。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询工作流运行详情
try:
    __logger.info('--------------------------------DescribePipelineRun start--------------------------------')
    run_id = "run-xxx"  # 替换为实际的工作流运行ID
    response = aihc_client.workflow.DescribePipelineRun(
        runId=run_id,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询工作流运行详情

查询工作流运行节点列表

使用以下代码可以查询工作流运行节点列表。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 查询工作流运行节点列表
try:
    __logger.info('--------------------------------DescribePipelineRunNodes start--------------------------------')
    run_id = "run-xxx"  # 替换为实际的工作流运行ID
    response = aihc_client.workflow.DescribePipelineRunNodes(
        runId=run_id,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为查询工作流运行节点列表

重试工作流运行

使用以下代码可以重试工作流运行。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 重试工作流运行
try:
    __logger.info('--------------------------------RetryPipelineRun start--------------------------------')
    run_id = "run-xxx"  # 替换为实际的工作流运行ID
    mode = "full"       # 重试模式：full（全部重试）或 failed（仅重试失败节点）
    response = aihc_client.workflow.RetryPipelineRun(
        runId=run_id,
        mode=mode,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为重试工作流运行

停止工作流运行

使用以下代码可以停止工作流运行。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 停止工作流运行
try:
    __logger.info('--------------------------------StopPipelineRun start--------------------------------')
    run_id = "run-xxx"     # 替换为实际的工作流运行ID
    terminate = False      # 是否强制终止：True 表示强制终止，False 表示优雅停止
    message = "手动停止"   # 停止原因（可选）
    response = aihc_client.workflow.StopPipelineRun(
        runId=run_id,
        terminate=terminate,
        message=message,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为停止工作流运行

删除工作流运行实例

使用以下代码可以删除工作流运行实例。

```
 import logging
import os

import dotenv

from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.services.aihc.aihc_client import AihcClient
from baidubce.exception import BceHttpClientError, BceServerError

##### 加载环境变量
dotenv.load_dotenv()

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', force=True)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("baidubce").setLevel(logging.INFO)
__logger = logging.getLogger(__name__)
__logger.setLevel(logging.INFO)

##### 配置认证信息
HOST = os.getenv('HOST') or 'https://aihc.bj.baidubce.com'
AK = os.getenv('AK') or 'your-access-key-id'
SK = os.getenv('SK') or 'your-secret-access-key'

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
aihc_client = AihcClient(config)

##### 删除工作流运行实例
try:
    __logger.info('--------------------------------DeletePipelineRun start--------------------------------')
    run_id = "run-xxx"  # 替换为实际的工作流运行ID
    response = aihc_client.workflow.DeletePipelineRun(
        runId=run_id,
    )
    print(response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        __logger.error('send request failed. Response %s, code: %s, msg: %s'
                       % (e.last_error.status_code, e.last_error.code, str(e.last_error)))
    else:
        __logger.error('send request failed. Unknown exception: %s' % e)
```

> 注意: 根据接口文档去填写具体的访问参数，接口链接为删除工作流运行实例

常用工具

AIHC CLI

概览

为了方便开发者用户使用百舸资源池、队列、分布式训练任务OpenAPI，百舸平台提供了命令行工具AIHCCLI，支持用户通过命令行方式进行核心业务操作

当前支持使用CLI工具执行资源池、队列、任务相关操作

安装AIHCCLI

支持Mac 、Linux、Windows 三种操作系统上使用

安装包下载

选择对应的版本，下载编译好的二进制可执行文件：

> 当前最新版本v1.6.5

平台	下载地址
windows-amd64	https://aihc-public.bj.bcebos.com/cli/release/v1.6.5/windows-amd64/aihc.exe
darwin-arm64	https://aihc-public.bj.bcebos.com/cli/release/v1.6.5/darwin-arm64/aihc
darwin-amd64	https://aihc-public.bj.bcebos.com/cli/release/v1.6.5/darwin-amd64/aihc
linux-amd64	https://aihc-public.bj.bcebos.com/cli/release/v1.6.5/linux-amd64/aihc
linux-arm64	https://aihc-public.bj.bcebos.com/cli/release/v1.6.5/linux-arm64/aihc

操作步骤

Windows环境

1. 将工具添加到系统环境变量$PATH 中

a. 打开“系统属性”对话框（右键点击“此电脑”或“计算机”，选择“属性”，然后点击“高级系统设置”和“环境变量”）。

b. 在“系统变量”区域找到Path变量，点击“编辑”，然后点击“新建”并输入工具可执行文件的路径。

c. 点击“确定”保存更改。

d. 重新打开一个命令行窗口使新的PATH设置生效。

2. **验证安装**

a. 打开一个新的命令提示符窗口（可以通过按Win + R，输入cmd，然后按回车来打开）。

b. 输入aihc -h 并按回车。如果工具正确安装并配置，它应该会显示一些帮助信息或启动该工具

Linux环境

将工具存放路径添加到环境变量$PATH 中

```
 $ echo $PATH

##### 工具存放路径
$ export PATH="$PATH:/usr/local/bin"
$ mv aihc /usr/local/bin/ && cd /usr/local/bin/
$ chmod +x aihc

##### 执行bash命令刷新终端以保证设置生效
$ bash

##### 检查指令是否生效，如果正常显示帮助信息则表示工具配置成功
$ aihc -h
```

Mac环境

1. 将工具存放路径添加到环境变量$PATH中

```
 ##### 查看当前环境变量
$ echo $PATH

##### 将/usr/local/bin加入到环境变量，如果已经存在不需要重复执行
$ export PATH="$PATH:/usr/local/bin"

##### 进入aihc可执行程序所在目录,将可执行程序移动到/usr/local/bin/路径下
$ cd /Users/xxx/Downloads/aihc
$ mv aihc /usr/local/bin/

##### 进入/usr/local/bin/路径进行权限设置
$ cd /usr/local/bin/
$ chmod +x aihc

##### 检查指令是否生效，如果正常显示帮助信息则表示工具配置成功
$ aihc -h
```

2. 设置执行权限（可选），如果遇到「未打开“aihc”，Apple无法校验....」提示，执行以下操作信任工具

```
 ##### 移除aihc文件的隔离属性
$ sudo xattr -r -d com.apple.quarantine /usr/local/bin/aihc

##### 添加执行权限（如果尚未授权）,执行完成后检测是否生效
$ sudo chmod +x /usr/local/bin/aihc

$ aihc -h
```

配置AIHCCLI

使用 AIHC CLI 前需配置百度智能云的 AccessKey（AK）、SecretKey（SK）以及服务地域（region）等信息。配置以 YAML 文件保存，读取时遵循下文「配置文件优先级」。

> 版本：v1.6.3（aihc-cli/golang/v1.6.3）。配置文件中的 endpoint、useragent、retry 由 CLI 根据其它字段自动生成，无需手动填写。

一、配置文件优先级

CLI 在执行任何命令时，按以下顺序确定使用哪个配置文件，命中即停止（高优先级在前）：

优先级	来源	说明
1	全局 -C/--config 标志指定的文件	对所有命令生效。例：aihc job list -C /path/to/config.yaml
2	ConfigManager 当前活动配置	由 aihc config use <path> 设置，记录在 ~/.aihc/config_manager.yaml
3	本地配置 ./.aihc/config	项目级配置，适合在某个项目目录下覆盖默认值
4	全局默认 ~/.aihc/config	用户级兜底配置

要点：

- -C/--config 是根命令的全局标志，优先级最高，可临时指定任一配置文件。
- aihc config use <path> 用于在多套配置间切换当前活动配置（见下文）。
- 本地配置 ./.aihc/config 仅在当前工作目录下生效，适合隔离不同项目的配置。

二、配置文件结构

配置文件为 YAML，推荐字段如下：

```
 region: cn-bj                  # 必填，服务地域（见 region 列表）
credentials:
   accesskey: ALTAKeB*********** # 必填，Access Key ID
   secretkey: b903d4***********  # 必填，Secret Access Key
defaultpool: ""                 # 可选，默认资源池 ID（可用 aihc config 设置）
defaultqueue: ""                 # 可选，默认队列标识（可用 aihc config 设置，托管资源池填队列id，自运维资源池填队列名称）
aihcusername: ""                 # 可选，作业创建者（仅bhcmp平台生效，非bhcmp平台不用设置，默认为ak、sk用户名）
```

字段说明：

字段	是否必填	说明
region	必填	服务地域，取值见下方 region 列表。同时支持简称（cn-bj）与全称（cn-beijing）
credentials.accesskey	必填	Access Key ID
credentials.secretkey	必填	Secret Access Key
defaultpool	可选	默认资源池 ID。设置后命令可省略 -p/--pool
defaultqueue	可选	默认队列标识。设置后命令可省略 -q/--queue（具体存 name 还是 ID 见「aihc pool set 存储规则」）
aihcusername	可选	作业创建者

> 自动字段：endpoint（由 region 映射得出）、useragent、retry 由 CLI 自动生成，**请勿手动填写**。

支持的 region

简称	全称	地域
cn-bj	cn-beijing	华北-北京
cn-gz	cn-guangzhou	华南-广州
cn-su	cn-suzhou	华东-苏州
cn-bd	cn-baoding	华北-保定
cn-fwh	cn-wuhan	华中-武汉
cn-yq	cn-yangquan	华北-阳泉

三、aihc config 命令用法

aihc config 用于配置 AK/SK、region 等，并支持多套配置文件管理。

根命令

- 不带任何标志：进入**交互式**配置，逐项提示输入（回车保留当前值）。
- 带标志：以**命令式**方式写入指定字段，适合脚本化。

获取帮助信息：

```
 $ aihc config -h
Configure Client credentials and settings. You can:
1. Run without flags for interactive mode
2. Set specific values using flags:
  - Set credentials: aihc config --ak <AccessKey> --sk <SecretKey>
  - Set region: aihc config --region <region>
  - Specify config directory: aihc config --path /path/to/directory (creates config file in the directory)
  - Specify config file: aihc config --path /path/to/config.yaml (uses exact file path)

Available Regions:
"cn-bj":   华北-北京
"cn-gz":   华南-广州
"cn-su":   华东-苏州
"cn-bd":   华北-保定
"cn-fwh":  华中-武汉
"cn-yq":   华北-阳泉

Usage:
  aihc config [flags]
  aihc config [command]

Available Commands:
  create      Create a new configuration file
  get         Get current configuration
  list        List all configuration files
  set         Set configuration values
  use         Switch to a different configuration file

Flags:
      --access-key string      Access Key ID
      --aihc-username string   Job creator
      --ak string              Access Key ID
      --default-pool string    Default Pool
      --default-queue string   Default Queue
  -h, --help                   help for config
      --path string            set config directory path
      --region string          Region
      --secret-key string      Secret Access Key
      --sk string              Secret Access Key

Global Flags:
  -C, --config string   Global configuration file. (default "/Users/xxx/.aihc/config")

Use "aihc config [command] --help" for more information about a command.
```

交互式配置示例

```
 $ aihc config

Current configuration:
region: cn-bj
credentials:
    accesskey: ""
    secretkey: ""

Press Enter to keep current value or input new value:
Access Key ID [""]: <在此输入 AK 后回车>
Secret Access Key [""]: <在此输入 SK 后回车>
Region [cn-bj]: <回车保留，或输入新值>
Default Pool [""]:
Default Queue [""]:
AIHC Username [""]:
Configuration updated successfully in /Users/xxx/.aihc/config
```

命令式配置示例

```
 ##### 一次性设置凭证与 region
$ aihc config --ak <AccessKey> --sk <SecretKey> --region cn-bj

##### 写入指定目录/文件（目录会自动在其下创建 config 文件）
$ aihc config --path /path/to/dir --ak <AccessKey> --sk <SecretKey>
$ aihc config --path /path/to/my.yaml --region cn-gz
```

根命令支持的全部标志：

标志	说明
`--ak` / `--access-key`	Access Key ID
`--sk` / `--secret-key`	Secret Access Key
`--region`	服务地域
`--default-pool`	默认资源池 ID
`--default-queue`	默认队列标识
`--aihc-username`	作业创建者
`--path`	指定配置目录或文件路径

子命令：config get 查看当前配置

查看当前生效的配置文件内容：

```
 $ aihc config get
Current config file: /Users/xxx/.aihc/config

Configuration content:

region: cn-bj
credentials:
    accesskey: ALTAKeB***********csC9GO
    secretkey: b903d4***********c7c315063
endpoint: aihc.bj.baidubce.com
retry: 3
defaultpool: ""
defaultqueue: ""
```

可用 `--path` 查看指定路径的配置：`aihc config get --path /path/to/config.yaml`

子命令：config set 修改配置

修改某个配置文件的字段（需要 `--path` 指定文件）：

```
 ##### 仅修改 region（其它字段保持不变）
$ aihc config set --path ~/.aihc/config --region cn-gz

##### 同时设置 AK/SK
$ aihc config set --path ~/.aihc/config --ak <AccessKey> --sk <SecretKey>
```

> 若仅写 aihc config set 而不带 `--path`，会提示使用 `--path`。set 的标志与根命令一致，支持 `--ak`/`--access-key`、`--sk`/`--secret-key`、`--region`、`--default-pool`、`--default-queue`、`--aihc-username`。

子命令：config create 创建新配置文件

创建一个新的配置文件（默认创建在 ./.aihc/config）：

```
 ##### 交互式创建（默认 ./.aihc/config）
$ aihc config create

##### 命令式创建
$ aihc config create --ak <AccessKey> --sk <SecretKey> --region cn-bj

##### 指定目录或文件
$ aihc config create --path /path/to/dir --ak <AccessKey> --sk <SecretKey> --region cn-bj
```

子命令：config use 切换当前活动配置

切换当前活动配置，写入 `~/.aihc/config_manager.yaml`，后续命令自动使用该配置（见优先级第 2 项）：

```
##### 使用指定配置文件
$ aihc config use ~/.aihc/config

##### 使用目录下的 config 文件
$ aihc config use ~/.aihc/

##### 使用具名配置文件
$ aihc config use /etc/aihc/prod.yaml
Successfully switched to configuration file: /etc/aihc/prod.yaml
```

子命令：config list 列出所有配置文件

列出 ConfigManager 管理的全部配置文件，并标记当前活动配置（*）：

```
$ aihc config list
Current configuration: /etc/aihc/prod.yaml

Available configurations:
* 1. /etc/aihc/prod.yaml
  2. /Users/xxx/.aihc/config
  3. /Users/xxx/work/.aihc/config

* = current configuration
```

四、设置默认资源池（aihc pool set）

设置默认资源池与队列后，后续命令若不指定则自动使用默认值（即可省略 -p/--pool；托管资源池还可省略 -q/--queue）。

交互式示例

```
$ aihc pool set
Available pools:
Resource pool type is: common
1. cce-rdy (hpas-runtime-02)
2. cce-4kvu (hpas-runtime)
Resource pool type is: dedicatedV2
3. cce-5uvro (workflow)

Select pool (enter number): 1

Available queues:
1. 63a9f0ea7bb96b649e85481845 (default)
2. 8b2c1f7d3a6e0c5b49217f38a1 (test)

Select queue (enter number): 1

Default pool set to: cce-rdy (hpas-runtime-02)
Default queue set to: 63a9f0ea7bb96b649e85481845 (default)
```

输出说明：

- 资源池**按类型分组**显示，每组先打印 Resource pool type is: <类型>，随后列出该类型下的池：序号. <资源池ID> (<名称>)。
- 队列列表格式：序号. <队列ID> (<队列名称>)。

五、直接编辑配置文件

也可以用 vim/nano 等编辑器直接修改配置文件，保存后重新执行命令即生效（注意遵循「配置文件优先级」，编辑的文件需为当前生效的那一个）。

```
vim ~/.aihc/config
```

```
region: cn-bj          # 必填
credentials:
  accesskey: xxxxxxx   # 必填
  secretkey: xxxxxxx   # 必填
defaultpool: ""        # 可选，默认资源池 ID
defaultqueue: ""       # 可选，默认队列标识
aihcusername: ""       # 可选，作业创建者
```

通过CLI使用百舸服务

使用命令行工具操作百舸与用户在百舸操作具有相同的权限控制，AK/SK所属用户需要在百舸控制台获得对应的操作权限（如资源池、队列、训练任务的开发权限、管理员权限）才能对资源池、队列、训练任务进行管理操作

通用语法和参数

```
$ aihc [资源] <command> [flags] [options]
```

- 全局通用参数 --help/-h：所有命令可以通过此参数获得帮助信息，查看使用说明
- 列表页通用参数 --order ：列表页接口支持排序参数，可选值asc, desc，默认值desc

--orderBy：设置排序字段，默认值createTime

--page：支持分页的接口可以指定页数，默认值1

-s, --size：分页大小，默认值100

p/n/q：带有翻页的命令，输p/n/q可以实现翻页或退出（上一页：p；下一页：n；退出：q）

查看参数说明

在输入任何命令之后添加 -h ，即可查看对应命令的参数说明，例如 aihc job list -h

```
##### 获取任务列表命令的参数说明
$ aihc job list -h
列出所有的训练任务。

Usage:
  aihc job list [flags]

Aliases:
  list, ls

Flags:
  -h, --help              help for list
  -n, --name string       任务名称关键字
      --order string      排序方式(asc/desc) (default "desc")
  -o, --order-by string   排序字段 (default "createdAt")
      --page int          页码 (default 1)
  -p, --pool string       资源池ID
  -q, --queue string      队列ID
      --size int          每页显示数量 (default 100)

Global Flags:
  -C, --config string   Global configuration file. (default "/Users/luyuchao/.aihc/config")
```

资源池相关操作

获取资源池列表

```
 $ aihc pool list [flags]
Flags:
  -h, --help                      help for list
  -w, --keyWord string            keyWord
  -k, --keyWordType string        keyWordType (default "resourcePoolName")
  -l, --limit int                 Number of resource pools to return (default 100)
      --offset int                Offset of the first resource pool to return
      --order string              Sort order (asc, desc) (default "ASC")
  -o, --orderBy string            Order by field (name, createTime) (default "createTime")
  -p, --pageNo int                Page number (default 1)
  -s, --pageSize int              Page size (default 50)
  -t, --resourcePoolType string   (common-自运维资源池,dedicatedV2-全托管资源池) (default "common")
```

- 命令示例

```
##### 当存在多页数据时，可以输入 p/n 进行上一页/下一页翻页；输入 q 退出当前命令
$ aihc pool list

Page 1 of 1 (Total items: 4)

NAME                                ID              STATUS        NODE_COUNT     GPU_COUNT      CREATED_AT
-------------------------------------------------------------------------------------------------------------
jun-gateway                         cce-mj***fzk    running       1/2            4/8            2025-03-06 06:41:33
lingang-hpas-runtime-02             cce-zq***i4dy   running       1/1            0/0            2025-02-10 13:44:45
lingang-hpas-runtime                cce-4k***u9u    running       4/4            0/0            2025-02-10 12:19:42
workflow-lijipeng                   cce-5u***vro    running       4/4            0/0            2025-01-21 11:22:09

Navigation:
  q - Quit

Enter command (p/n/q):
```

获取资源池详情

```
 $ aihc pool get [flags]
-h, --help          help for get
-p, --pool string   Resource pool ID 指定资源池ID
```

- 命令示例（获取指定资源池详情）

```
 $ aihc pool get -p cce-mjtzk

resourcePool:
  metadata:
    createdAt: "2025-03-06T06:41:33Z"
    id: cce-mjtzk
    name: jun-gateway
    updatedAt: "2025-03-06T08:34:59Z"
  spec:
    associatedCpromIds: []
    associatedPfsId: ""
    createdBy: unpeng
    description: ""
    forbidDelete: true
    k8sVersion: 1.24.4
    region: bj
  status:
    gpuCount:
      total: 8
      used: 4
    gpuDetail:
      - gpudescriptor: baidu.com/l20_cgpu
        total: 8
        used: 4
    nodeCount:
      total: 2
      used: 1
    phase: running
```

- 命令示例（获取默认资源池详情）

```
 ##### 当设置了默认资源池时，可以省略 -p/--pool 参数，工具直接使用默认资源池作为 -p/--pool参数
$ aihc pool get

resourcePool:
  metadata:
    createdAt: "2023-09-17T15:48:04Z"
    id: cce-4h***1m
    name: IAT-Regression-pei
    updatedAt: "2025-02-11T08:21:50Z"
  spec:
    associatedCpromIds:
      - cprom-xm1***3k3s7
    associatedPfsId: pfs-p***jz
    createdBy: ""
    description: ""
    forbidDelete: true
    k8sVersion: 1.20.8
    region: bj
  status:
    gpuCount:
      total: 16
      used: 1
    gpuDetail:
      - gpudescriptor: baidu.com/xpu
        total: 8
        used: 0
      - gpudescriptor: baidu.com/a800_80g_cgpu
        total: 8
        used: 1
    nodeCount:
      total: 3
      used: 1
    phase: running
```

队列相关操作

获取队列列表

```
 aihc queue list/ls [flags]
Flags:
  -h, --help                help for list
  -k, --keyword string      Keyword
  -t, --keywordtype string  Keyword type
  -r, --order string        Sort order
  -o, --orderby string      Sort field (default "dec")
  -n, --pagenumber int      Page number (default 3)
  -s, --pagesize int        Page size (default 100)
  -p, --pool string         Resource pool ID
```

- 命令示例（获取指定资源池的队列列表）

```
 $ aihc queue list -p cce-mjtzk

name                            state    queueType    reclaimable    disableOversell    createdTime
openapi-regular--dtea           Open     Elastic      True           False              2025-03-16T19:47:18Z
63a9f0ea7bb98050796b649e85481845    Open     Elastic      True           False              2025-03-14T03:47:13Z
default                         Open     Elastic      True           False              2025-03-14T03:47:12Z
```

- 命令示例（获取默认资源池队的列列表）

```
##### 当设置了默认资源池时，可以省略 -p/--pool 参数，工具直接使用默认资源池作为 -p/--pool参数
$ aihc queue list

name                              state     queueType    reclaimable    disableOversell    createdTime
openapi-regular--dtea               Open      Elastic      True           False              2025-03-16T19:47:18Z
63a9f0ea7bb98050796b649e85481845    Open      Elastic      True           False              2025-03-14T03:47:13Z
default                           Open      Elastic      True           False              2025-03-14T03:47:12Z
```

获取队列详情

```
$ aihc queue get queueID [flags]
-h, --help          help for get
-p, --pool string   Resource pool ID 指定资源池ID
```

- 命令示例

```
$ aihc queue get 63a9f0ea7bb98050796b649e85481845 -p cce-2k9sw0cjaihc

$ aihc queue get 63a9f0ea7bb98050796b649e85481845
```

任务相关操作

获取任务列表

```
$ aihc job list/ls [flags]

Flags:
-h, --help              help for list
-n, --name string       任务名称关键字
    --order string      排序方式(asc/desc) (default "desc")
-o, --order-by string   排序字段 (default "createdAt")
    --page int          页码 (default 1)
-p, --pool string       资源池ID
-q, --queue string      队列ID
    --size int          每页显示数量 (default 100)
```

- 命令示例（指定全托管资源池）

> 查询托管资源池下任务需指定 -p 托管资源池id -q 队列id

```
$ aihc job list -p aihc-sxdfsd -q aihcq-dcfadas

Page 1 of 1 (Total items: 6)


NAME                              JOB_ID                                      STATUS              CREATED_AT
aihc-createjob-test                 pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59    Created             2025-02-17T07:59:35Z
qwen2-vl-test-4-copy1               pytorchjob-a53c413c-8e96-416b-8d6a-bc3e1b16a406    Succeeded           2025-01-01T13:22:19Z
qwen2-vl-test-4                   pytorchjob-0389039d-aae5-4601-8e9e-fb230867baa4    ManualTermination   2025-01-01T12:00:52Z
qwen2-vl-test-one-copy1             pytorchjob-c9ac1567-4306-4085-8432-5305f4dc600f    ManualTermination   2024-12-31T15:22:26Z
qwen2-vl-test-2                   pytorchjob-5c6c22c4-2c8d-49e7-8c2e-46e5fc2caa24    ManualTermination   2024-12-31T14:26:05Z
qwen2-vl-test-one                   pytorchjob-ba8b2830-06c6-462f-aaf7-8c2513733482    Succeeded           2024-12-31T14:18:47Z

Navigation:
q - Quit

Enter command (p/n/q):
```

> 查询自运维资源池下任务需指定 -p 自运维资源池id -q 队列名

- 命令示例（指定自运维资源池）

```
$ aihc job list -p cce-2k0cj -q test-queue-name

Page 1 of 1 (Total items: 6)


NAME                              JOB_ID                                      STATUS              CREATED_AT
aihc-createjob-test                 pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59    Created             2025-02-17T07:59:35Z
qwen2-vl-test-4-copy1               pytorchjob-a53c413c-8e96-416b-8d6a-bc3e1b16a406    Succeeded           2025-01-01T13:22:19Z
qwen2-vl-test-4                   pytorchjob-0389039d-aae5-4601-8e9e-fb230867baa4    ManualTermination   2025-01-01T12:00:52Z
qwen2-vl-test-one-copy1             pytorchjob-c9ac1567-4306-4085-8432-5305f4dc600f    ManualTermination   2024-12-31T15:22:26Z
qwen2-vl-test-2                   pytorchjob-5c6c22c4-2c8d-49e7-8c2e-46e5fc2caa24    ManualTermination   2024-12-31T14:26:05Z
qwen2-vl-test-one                   pytorchjob-ba8b2830-06c6-462f-aaf7-8c2513733482    Succeeded           2024-12-31T14:18:47Z

Navigation:
q - Quit

Enter command (p/n/q):
```

- 命令示例（默认资源池）

```
 $ aihc job list

Page 1 of 1 (Total items: 6)


NAME                              JOB_ID                                    STATUS             CREATED_AT
aihc-createjob-test               pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59   Created            2025-02-17T07:59:35Z
qwen2-vl-test-4-copy1             pytorchjob-a53c413c-8e96-416b-8d6a-bc3e1b16a406   Succeeded          2025-01-01T13:22:19Z
qwen2-vl-test-4                   pytorchjob-0389039d-aae5-4601-8e9e-fb230867baa4   ManualTermination  2025-01-01T12:00:52Z
qwen2-vl-test-one-copy1           pytorchjob-c9ac1567-4306-4085-8432-5305f4dc600f   ManualTermination  2024-12-31T15:22:26Z
qwen2-vl-test-2                   pytorchjob-5c6c22c4-2c8d-49e7-8c2e-46e5fc2caa24   ManualTermination  2024-12-31T14:26:05Z
qwen2-vl-test-one                 pytorchjob-ba8b2830-06c6-462f-aaf7-8c2513733482   Succeeded          2024-12-31T14:18:47Z

Navigation:
q - Quit

Enter command (p/n/q):
```

单任务创建

> 当前版本的AIHC CLI工具创建任务是基于百舸OpenAPI v1实现，支持的任务参数、校验规则、格式与API一致，建议客户首先阅读 API参考(v1)/训练任务相关接口/创建训练任务 有助于快速上手创建任务命令

直接传参方式创建任务

```
 ##### 直接传参方式创建任务
aihc job create [flags]

Flags:
      --code-dir string              代码在容器中的挂载路径，默认为 /workspace
      --code-url string              代码URL,使用代码上传命令返回的URL
      --command string               指定训练任务的入口命令 (与--script-file二选一)
      --ds-id string                 数据集ID（public_dataset类型仅需id+mountPath；亦可与其他ds-*组合）
      --ds-mountpath string          数据源挂载路径
      --ds-name string               数据源名称
      --ds-pfsl3-instance-id string  pfsl3 实例ID（--ds-type pfsl3 时必填）
      --ds-pfsl3-mount-point string  pfsl3 挂载地址（--ds-type pfsl3 时必填）
      --ds-sourcepath string         数据源路径
      --ds-type string               数据源类型（支持 pfs、pfsl3、dataset、bos、cfs、hostpath 等）
      --ds-version string            数据集版本（配合 --ds-id 使用）
      --enable-bccl                  是否启用BCCL,默认false
      --enable-fault-tolerance       是否启用容错功能,默认false
      --enable-rdma                  是否启用RDMA,默认false
      --env strings                  环境变量 (key=value)
      --fault-tolerance-args string  容错功能的详细参数
      --framework string             任务框架类型（支持pytorch、mpi、tensorflow、ray，默认pytorch）
      --gpu strings                  GPU资源 (type=count)

  -h, --help                         help for create
      --host-network                 是否启用主机网络,默认false
      --idle-task                    是否提交闲时任务（借用其他队列闲置资源，可能被回收；仅全托管资源池，服务端会强制将优先级改为lowest）
      --image string                 容器镜像
  -f, --job-file strings             任务配置文件路径json/yaml
      --label strings                标签 (key=value)
      --local-code string            本地代码路径，创建任务时会先上传代码
      --metrics-path string          metrics 采集路径（配合 --metrics-port 使用，默认 /metrics）
      --metrics-port int32           监控监听端口（自定义 metrics 采集），设置后启用 portConfigs (default 2333)
      --name string                  任务名称
  -p, --pool string                  资源池ID (如未指定则使用配置文件中的默认值)
      --pre-init-command string      前置命令（仅RayJob可用）
      --priority string              任务优先级 (low/normal/high) (default "normal")
      --privileged                   是否启用特权模式,默认false
      --replicas int32               任务副本数 (default 1)
  -r, --retention-period string      任务保留时间，格式1m、1h、1d
      --runtime-env string           Ray runtimeEnv（JSON字符串，仅RayJob）
      --script-file string           命令脚本的路径，以脚本的方式指定训练任务的入口命令 (与--command二选一)
      --submitter-backoff-limit int32 Ray submitter最大重试次数（仅RayJob）
      --redis-instance-id string     SCS Redis 实例 ID，配置后开启 head 节点 GCS 容错高可用（仅 RayJob）
      --redis-username string        Redis 访问用户名，缺省由后端填充 default（仅 RayJob）
      --redis-password string        Redis 访问密码，配置 --redis-instance-id 时必填（仅 RayJob）
  -u, --username string              用户自定义的创建人名称
      --visible-scope int            可见范围：0=仅创建人 / 1=队列内 (default 1)
      --worker-gpu strings           Worker GPU 资源 type=count（仅 RayJob；可重复；）
      --worker-name strings          Worker 角色名（仅 RayJob；可重复，与 --worker-replicas/--worker-gpu 按序号对齐）
      --worker-replicas ints         Worker 副本数（仅 RayJob；可重复；未指定 --worker-name 时取首值作为默认 Worker 角色副本数）
```

- 命令示例（PyTorch任务）

```
 $ aihc job create --local-code Aihc \
 --name cli-codeupload-test \
 --image registry.baidubce.com/inf-qa/nginx:latest \
 --framework PyTorchJob \
 --command "sleep 1d" \
 --replicas 4 \
 --privileged=true \
 --fault-tolerance-args="--enable-replace=true --enable-hang-detection=true --hang-detection-log-timeout-minutes=7 --hang-detection-startup-toleration-minutes=15 --hang-detection-stack-timeout-minutes=3 --max-num-of-unconditional-retry=2 --unconditional-retry-observe-seconds=3600 --custom-log-patterns=timeout1 --custom-log-patterns=timeout2 --enable-use-nodes-of-last-job=true --enable-checkpoint-migration=true --internal-fault-tolerance-alarm-phone=10086,10010" \
 --priority high \
 --enable-bccl=false \
 --enable-fault-tolerance=true \
 --local-code /codeDir/file --code-dir /workspace #代码路径  #代码上传挂载目录
 -p cce-cm1jjxrq
```

- 命令示例（RayJob任务 - 单 Worker）

> RayJob 使用 --framework ray 指定框架类型。与 PyTorch 任务不同，RayJob 的 jobSpec 按角色（Head/Worker）划分，Head 副本数固定为 1。 RayJob 不支持的功能：--enable-fault-tolerance、--enable-bccl、--retention-period、--host-network

```
$ aihc job create --name syf-test-ray-cli-byflag \
--framework ray \
--replicas 1 \
--pre-init-command "echo 'syf-test-log'" \
--runtime-env "{}" \
--submitter-backoff-limit 2 \
--redis-instance-id scs-bj-xxxxxxxx \
--redis-username default \
--redis-password xxxxxxx \
--visible-scope 1 \
--enable-rdma true \
--env "env-name=env-value" \
--command "sleep 1d" \
--worker-gpu "baidu.com/rtx_3090_cgpu=1" \
--worker-replicas 1 \
--ds-id "ds-69GJT3J" \
--ds-type "dataset" \
--ds-version "1" \
--ds-mountpath "/data" \
--ds-sourcepath "/huggingface/dataset/RoboCOIN/leju_robot_part_placement/main" \
--image ccr-registry.baidubce.com/aihc/rayproject/ray/ray-cpu:2.9.3-py3.10-ubuntu20.04
```

- 命令示例（RayJob任务 - 多 Worker 自定义角色）

> 通过 --worker-name + --worker-replicas + --worker-gpu 三个可重复标志按序号对齐，可一次性创建多个不同名字/副本数/GPU 配置的自定义 Worker 角色。 每个自定义 Worker 与 Head 共享 image、command、envs、enable-rdma、privileged、pre-init-command 等参数。 Worker 角色名必须以 worker 前缀开头（大小写不敏感），且不能重复，不能命名为 Head。

```
$ aihc job create --name syf-test-ray-multi-worker \
--framework ray \
--command "python train.py" \
--image ccr-registry.baidubce.com/aihc/rayproject/ray/ray-cpu:2.9.3-py3.10-ubuntu20.04 \
--enable-rdma true \
--env "MODEL_NAME=demo" \
--pre-init-command "echo 'init'" \
--worker-name worker-a --worker-replicas 2 --worker-gpu "baidu.com/a800_80g_cgpu=8" \
--worker-name worker-b --worker-replicas 4 --worker-gpu "baidu.com/a800_80g_cgpu=4" \
-p cce-xxxx -q default
```

上述命令将创建：

- Head 角色（replicas=1，无 GPU）
- worker-a（replicas=2，8×A800）
- worker-b（replicas=4，4×A800）

所有 Worker 角色继承 Head 的 image、command、envs、enable-rdma、privileged 等设置。

> **RayJob 专属参数说明：**

参数	类型	说明
--worker-name	[]string	Worker 角色名（可重复，与 --worker-replicas/--worker-gpu 按序号对齐）；未指定时使用默认名Worker
--worker-replicas	[]int	Worker 副本数（可重复）；未指定--worker-name 时取首值作为默认 Worker 角色副本数
--worker-gpu	[]string	Worker GPU 资源，格式type=count；一条内多 GPU 用逗号分隔（如 A100=8,cpu=8），多角色时可重复此标志
--runtime-env	string	Ray runtimeEnv，JSON字符串
--submitter-backoff-limit	int32	submitter最大重试次数
--redis-instance-id	string	SCS Redis 实例 ID，配置后开启 head 节点 GCS 容错高可用
--redis-username	string	Redis 访问用户名，缺省由后端填充 default
--redis-password	string	Redis 访问密码，配置 --redis-instance-id 时必填
--pre-init-command	string	前置命令，仅RayJob可用

> **多 Worker 模式使用规则：**
>
> - 提供 --worker-name 时进入**多 Worker 模式**：--worker-name、--worker-replicas、--worker-gpu 的出现次数必须相等，按索引一一对应。
> - 未提供 --worker-name 时进入**兼容模式**：--worker-replicas 取首值，--worker-gpu 全部条目合并到默认名为 Worker 的单一角色。
> - Worker 角色名必须以 worker 前缀（大小写不敏感），不可命名为 Head，不可重复。
> - Head 的 --gpu 与 Worker 的 --worker-gpu 独立设置；--gpu 仅作用于 Head。

- 命令示例（设置监控监听端口）

> 通过 --metrics-port 指定容器监听端口用于自定义 metrics 采集，采集路径由 --metrics-path 指定（缺省 /metrics）。 仅在显式传入 --metrics-port 时生效；只传 --metrics-path 会报错。该配置与框架无关，PyTorch、RayJob 等均可使用。 端口区间与保留端口由服务端校验，instanceId/agentId 由服务端自动补齐，无需传入。

```
$ aihc job create --name job-with-metrics \
--image <镜像> \
--command "sleep 1d" \
--metrics-port 8080 \
--metrics-path /custom/metrics \
-p cce-xxxx -q default
```

- 命令示例（挂载 pfsl3 存储）

> pfsl3 通过 `--ds-type pfsl3` 挂载，需同时提供 `--ds-pfsl3-instance-id`（实例ID）和 `--ds-pfsl3-mount-point`（挂载地址）。 pfsl3 实例需在 PFS 侧预先创建；不要设置 `--ds-name`，实例统一用 pfsl3 实例ID 标识。

```
$ aihc job create --name job-with-pfsl3 \
--image <镜像> \
--command "sleep 1d" \
--ds-type pfsl3 \
--ds-pfsl3-instance-id pfs-xxxxxx \
--ds-pfsl3-mount-point pfs-xxxxxx.lb-xxxx.pfs.bj.baidubce.com \
--ds-sourcepath / \
--ds-mountpath /mnt/pfsl3 \
-p cce-xxxx -q default
```

- 命令示例（提交闲时任务）

> 闲时任务借用其他队列的闲置资源，资源紧张时可能被回收。仅全托管资源池、且队列支持闲时提交时可用；提交后服务端会将任务优先级强制设为 lowest。与框架无关，PyTorch、RayJob 均可使用。

```
$ aihc job create --name job-idle-task \
--image <镜像> \
--command "sleep 1d" \
--idle-task \
-p aihc-xxxx -q <队列ID>
```

使用任务模板创建任务

- 命令示例（使用JSON格式模板）

```
 ##### 支持使用json格式文件传递参数，详见参数模板，创建任务支持的参数详见创建任务接口参数：https://cloud.baidu.com/doc/AIHC/s/jm56inxn7
##### 接口请求body参数的json文件，需将在命令行运行的主机上先创建好任务信息
$ aihc job create -f job_info.json

##### 使用command文件保存启动命令
$ aihc job create -f job_info.json --script-file command.txt

##### 创建任务时上传代码

$ aihc job create -f job-info.yaml --local-code /file/path #本地代码路径
```

- 上传代码

```
 ##### 使用命令上传代码
aihc code upload --folder /code/filepath

Usage:
  aihc code upload [flags]

Flags:
  -f, --folder string   指定要上传的代码文件夹路径
  -h, --help            help for upload
  -p, --pool string     指定资源池ID，如未指定则使用配置文件中的默认值
  -q, --queue string    指定队列ID
```

JSON格式任务参数参考模板：

```
{
 "name": "qwen2-vl-test-4-copy2",
 "queue": "",
 "jobFramework": "pytorch",
 "jobSpec": {
  "command": "cd /data \u0026\u0026 mkdir qwen2-vl\n\ncp /mnt/ca-p800-poc/llava-en-zh-300k.tar.gz /data/qwen2-vl/\n#cp /mnt/ca-p800-poc/llava-en-zh-2k.tar.gz /data/qwen2-vl/\ncd /data/qwen2-vl\n\ntar -zxvf llava-en-zh-300k.tar.gz \u0026\u0026 rm llava-en-zh-300k.tar.gz\n#tar -zxvf llava-en-zh-2k.tar.gz \u0026\u0026 rm llava-en-zh-2k.tar.gz\n\n# code\ncp /mnt/ca-p800-poc/models/qwen2-vl.tar.gz /home/\ncd /home/\ntar -zxvf qwen2-vl.tar.gz \u0026\u0026 rm qwen2-vl.tar.gz\n\n# model weights\ncp /mnt/ca-p800-poc/Qwen2-VL-7B-Instruct.tar.gz .\ntar -zxvf Qwen2-VL-7B-Instruct.tar.gz \u0026\u0026 rm Qwen2-VL-7B-Instruct.tar.gz\n\n#train lora\ncd /home/qwen2-vl/\nconda activate llamafactory_env\nexport PATH=/root/miniconda/envs/llamafactory_env/bin:$PATH\nexport PYTHONPATH=$PYTHONPATH:/home/qwen2-vl/\n\napt-get install dnsutils -y\npip install deepspeed==0.14.5\n\nsource env.sh\nnohup bash dist_train.sh lora \u0026\n\nsleep 100000",
  "image": "ccr-2ccrtest-vpc.cnc.bj.baidubce.com/yetao04-ca-test/qwen2vl_p800_image:v1.0",
  "imageConfig": {
   "username": "",
   "password": ""
  },
  "replicas": 4,
  "resources": [
   {
    "name": "kunlunxin.com/xpu",
    "quantity": 8
   },
   {
    "name": "sharedMemory",
    "quantity": 1024
   }
  ],
  "envs": [
   {
    "name": "AIHC_JOB_NAME",
    "value": "qwen2-vl-test-4-copy1"
   },
   {
    "name": "NCCL_IB_DISABLE",
    "value": "0"
   }
  ],
  "enableRDMA": true,
  "hostNetwork": false
 },
 "faultTolerance": false,
 "labels": [
  {
   "key": "aijob.cce.baidubce.com/ai-user-id",
   "value": "ac3553acbb8d4c5e9b212fc0a04c8f7d"
  },
  {
   "key": "aijob.cce.baidubce.com/ai-user-name",
   "value": "daichaonan"
  },
  {
   "key": "aijob.cce.baidubce.com/create-from-aihcp",
   "value": "true"
  },
  {
   "key": "aijob.cce.baidubce.com/openapi-jobid",
   "value": "pytorchjob-a53c413c-8e96-416b-8d6a-bc3e1b16a406"
  }
 ],
 "priority": "high",
 "datasources": [
  {
   "type": "pfsl1",
   "sourcePath": "/yetao04",
   "mountPath": "/mnt",
   "name": "pfs-zesqWP",
   "options": {
    "sizeLimit": 1000,
    "medium": "",
    "readOnly": false,
    "pfsL1ClusterIp": "172.16.0.221",
    "pfsL1ClusterPort": "8888"
   }
  },
  {
   "type": "hostpath",
   "sourcePath": "/ssd1",
   "mountPath": "/data",
   "name": "hostpath-1",
   "options": {
    "sizeLimit": 0,
    "medium": "",
    "readOnly": false
   }
  }
 ],
 "faultToleranceConfig": {
  "enabledHangDetection": false,
  "hangDetectionTimeoutMinutes": 0,
  "faultToleranceLimit": 0,
  "customFaultTolerancePattern": null
 },
 "alertConfig": null,
 "enableBccl": false
}
```


- 命令示例（使用YAML格式模板）

支持使用yaml格式文件传递参数，详见参数模板，创建任务支持的参数详见创建任务接口参数：https://cloud.baidu.com/doc/AIHC/s/jm56inxn7

```
##### 接口请求body参数的yaml文件
$ aihc job create -f ./job_info.yaml

##### 创建任务时上传代码及使用command文件保存启动命令
$ aihc job create -f job_info.yaml --local-code folder --script-file command.txt
```

YAML格式任务参数参考模板（PyTorch任务）：

```
name: qwen2-vl-test-4-cli-test
queue: ""
jobFramework: pytorch
jobSpec:
  command: |
    sleep 100000
  image: ccr-2ccrtest-vpc.cnc.bj.baidubce.com/yetao04-ca-test/qwen2vl_p800_image:v1.0
  imageConfig:
    username: ""
    password: ""
  replicas: 4
  resources:
    - name: kunlunxin.com/xpu
      quantity: 8
    - name: sharedMemory
      quantity: 1024
  envs:
    - name: AIHC_JOB_NAME
      value: qwen2-vl-test-4-copy1
    - name: NCCL_IB_DISABLE
      value: "0"
  enableRDMA: true
  hostNetwork: false
faultTolerance: true
labels:
  - key: aijob.cce.baidubce.com/ai-user-id
    value: ac3553acbb8d4c5e9b212fc0a04c8f7d
  - key: aijob.cce.baidubce.com/ai-user-name
    value: daichaonan
priority: high
codeSource:
  mountPath: /workspace
datasources:
  - type: hostpath
    sourcePath: /
    mountPath: /mnt/rapidfs
    name: rapidfs-p800
  - type: bos
    sourcePath: aihc-aijob-log/joblog/job-xxxx/
    mountPath: /test
    name: ""
    options:
      readOnly: true
  - type: cfs
    sourcePath: /data
    mountPath: /data
    name: ""
    options:
      readOnly: true
      cfsInstanceId: cfs-xxx
      cfsMountPoint: cfs-xxx.lb-xxx.cfs.xx.baidubce.com
  - type: pfs
    sourcePath: ""
    mountPath: /mnt/cluster
    name: "pfs-xxx"
    options:
      readOnly: true
  - type: pfsl3
    sourcePath: /
    mountPath: /mnt/pfsl3
    options:
      readOnly: false
      pfsL3InstanceId: pfs-xxxxxx
      pfsL3MountPoint: pfs-xxxxxx.lb-xxxx.pfs.bj.baidubce.com
  - id: ds-xxx
    type: dataset
    sourcePath: //huggingface/dataset/example/main
    mountPath: /data
    name: ""
    options:
      readOnly: false
      datasetVersion: "1"
  - id: ds-xxx
    type: public_dataset
    sourcePath: ""
    mountPath: /opt/aihc/models
    name: ""
    options:
      readOnly: false
      datasetVersion: "v1"
enableBccl: false
idleTask: false                    # 是否提交闲时任务（仅全托管资源池；true 时服务端强制优先级为 lowest）
faultToleranceArgs: "--enable-replace=true --enable-hang-detection=true --hang-detection-log-timeout-minutes=7 --hang-detection-startup-toleration-minutes=15 --hang-detection-stack-timeout-minutes=3 --max-num-of-unconditional-retry=2 --unconditional-retry-observe-seconds=3600 --custom-log-patterns=timeout1 --custom-log-patterns=timeout2 --enable-use-nodes-of-last-job=true --enable-checkpoint-migration=true --internal-fault-tolerance-alarm-phone=10086,10010"
```

datasources 数据源挂载说明：

任务模板支持多种数据源类型，通过 `datasources` 数组配置挂载。每项包含以下字段：

字段	说明
id	数据集ID（dataset/public_dataset类型必填）
type	数据源类型：hostpath、bos、cfs、pfs、pfsl1、pfsl2、pfsl3、dataset、public_dataset
sourcePath	数据源路径
mountPath	容器内挂载路径
name	数据源名称（hostpath/pfs类型需要；pfsl3 必须留空，用 pfsL3InstanceId 标识）
options.readOnly	是否只读挂载
options.cfsInstanceId	CFS实例ID（cfs类型必填）
options.cfsMountPoint	CFS挂载点（cfs类型必填）
options.pfsL3InstanceId	pfsl3 实例ID（pfsl3类型必填）
options.pfsL3MountPoint	pfsl3 挂载地址（pfsl3类型必填）
options.datasetVersion	数据集版本（dataset/public_dataset类型可选）

监控端口配置（portConfigs）：

模板可通过顶层 `portConfigs` 字段配置自定义 metrics 采集端口，与框架无关。`usage` 固定为 `Metrics`，`path` 缺省 `/metrics`；`instanceId/agentId` 由服务端自动补齐，无需填写。当前仅支持 1 条 `portConfig`。

```yaml
portConfigs:
portConfig:
  - containerPort: 8080
    usage: Metrics
    path: /custom/metrics
```

YAML格式RayJob参考模板：

RayJob 的 `jobSpec` 按角色（Head/Worker）划分，各角色独立配置资源和镜像。Head 副本数固定为 1。

```yaml
name: syf-test-ray-template
queue: syf-test
jobFramework: RayJob
jobSpec:
  Head:
    command: sleep 1d
    image: ccr-registry.baidubce.com/aihc/rayproject/ray/ray-cpu:2.9.3-py3.10-ubuntu20.04
    imageConfig:
      username: ""
      password: ""
    replicas: 1
    resources:
      - name: cpu
        quantity: 4
      - name: memory
        quantity: 64
      - name: sharedMemory
        quantity: 32
      - name: baidu.com/a100_40g_cgpu
        quantity: 1
    envs:
      - name: ENV_TEST_NAME
        value: env_test_value
    enableRDMA: true
  Worker:
    command: sleep 1d
    image: ccr-registry.baidubce.com/aihc/rayproject/ray/ray-cpu:2.9.3-py3.10-ubuntu20.04
    imageConfig:
      username: ""
      password: ""
    replicas: 2
    resources:
      - name: cpu
        quantity: 4
      - name: memory
        quantity: 64
      - name: sharedMemory
        quantity: 32
      - name: baidu.com/a100_40g_cgpu
        quantity: 2
    envs:
      - name: ENV_TEST_NAME
        value: env_test_value
    enableRDMA: true
  advancedSettings:
    runtimeEnv: "{}"
    submitterBackoffLimit: 1
    redisInstanceId: scs-bj-xxxxxxxx
    redisUsername: default
    redisPassword: xxxxxxxx
  preInitCommand: echo "123"
  labels:
    - key: aihc.baidubce.com/label-test-key
      value: label-test-value
  priority: normal
  datasources:
    - type: hostpath
      sourcePath: /a
      mountPath: /b
      name: "hostpath-1"
      options:
        readOnly: true
    - type: bos
      sourcePath: aihc-aijob-log/joblog/job-xxxx/
      mountPath: /test
      name: ""
      options:
        readOnly: true
    - id: ds-xxx
      type: dataset
      sourcePath: //huggingface/dataset/example/main
      mountPath: /data
      name: ""
      options:
        readOnly: false
        datasetVersion: "1"
```

RayJob 模板校验规则：

- Head 角色的 `replicas` 必须为 1
- Head 角色必须包含 `command`
- 每个角色必须指定 `image`
- `runtimeEnv` 必须为合法 JSON 字符串
- 不可设置 `faultTolerance=true`、`enableBccl=true`、`retentionPeriod`、`hostNetwork=true`
- 非 Head 角色名必须以 `worker` 前缀开头（大小写不敏感），可包含多个自定义 Worker 角色（如 `worker-a`、`worker-b`、`WorkerGPU` 等）

命令行参数覆盖配置文件

使用 -f 配置文件创建任务时，可以同时传递命令行参数来覆盖配置文件中的值。

覆盖规则：

- 优先级：命令行显式传入的参数 > 配置文件中的值
- 仅用户显式传入的参数会覆盖，未传入的参数保留配置文件中的值
- 切片类参数（`--gpu`、`--env`、`--label`）传入时将完全替换文件中的列表
- 布尔参数支持 `--flag=false` 显式关闭
- 命令示例

```
##### 使用配置文件创建，但覆盖任务名称和副本数
$ aihc job create -f job.yaml --name new-job-name --replicas 8

##### 覆盖镜像和优先级
$ aihc job create -f job.yaml --image new-image:v2 --priority high

##### RayJob 场景：覆盖单 Worker 副本数和 GPU（兼容模式）
$ aihc job create -f ray_job.yaml --worker-replicas 4 --worker-gpu "baidu.com/a800_80g_cgpu=4"

##### RayJob 场景：多 Worker 模式重建所有非 Head 角色
$ aihc job create -f ray_job.yaml \
    --worker-name worker-a --worker-replicas 2 --worker-gpu "baidu.com/a800_80g_cgpu=8" \
    --worker-name worker-b --worker-replicas 4 --worker-gpu "baidu.com/v100=4,cpu=8"

##### RayJob 场景：覆盖 Head 共享字段将同步到所有 Worker（image、env、enable-rdma、privileged）
$ aihc job create -f ray_job.yaml --image new-image:v2 --env "K=V" --enable-rdma=true

##### 显式关闭布尔参数
$ aihc job create -f job.yaml --enable-fault-tolerance=false

##### 覆盖监控端口配置（文件已有 portConfigs 时按字段覆盖，没有时由 --metrics-port 新建）
$ aihc job create -f job.yaml --metrics-port 9000 --metrics-path /custom/metrics
```

> **RayJob 覆盖行为说明：**
>
> - `--worker-name` 传入时（多 Worker 模式）：完全重建配置文件中的所有非 Head 角色，Head 保留。
> - 仅传 `--worker-replicas`/`--worker-gpu`（无 `--worker-name`）：以单 `Worker` 角色为目标进行覆盖（兼容旧行为）。
> - Head 共享字段（`--image`、`--env`、`--enable-rdma`、`--privileged`、`--command`、`--pre-init-command`）覆盖时：会同步写入到配置文件中的所有非 Head 角色。
> - `--gpu` 仅覆盖 Head 的资源列表，不影响 Worker。

批量任务提交

```
aihc job create -f job-1.yaml -f job-2.yaml ..-f ..
```

- 命令示例

```
$ aihc job create -f ./job-1.yaml -f ./job-2.yaml
```

删除训练任务

```
aihc job delete jobID [flags]
-h, --help          help for delete
-p, --pool string   资源池ID
```

- 示例命令

```
##### 删除指定资源池下任务
$ aihc job delete pytorchjob-c0d3504cfd-b44b-6dd7c39dde59 -p cce-2k0cj

##### 删除默认资源池下任务
$ aihc job delete pytorchjob-c0d3504cfd-b44b-6dd7c39dde59
```

停止训练任务

```
aihc job stop jobID [flags]

Flags:
  -h, --help          help for stop
  -p, --pool string   资源池ID
```

- 示例命令

```
##### 停止指定资源池下任务
$ aihc job stop pytorchjob-c0d3504cfd-b44b-6dd7c39dde59 -p cce-2k0cj

##### 停止默认资源池下任务
$ aihc job stop pytorchjob-c0d3504cfd-b44b-6dd7c39dde59
```

获取任务详情

```
 aihc job get jobID [flags]
Flags:
  -h, --help          help for get
      --pods          是否展示作业的实例信息
  -p, --pool string   资源池ID
  -s, --status        显示任务状态简要信息
```

- 示例命令

```
 ##### 获取默认资源池下指定任务详情
$ aihc job get pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59

##### 获取指定资源池下指定任务详情
$ aihc job get pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 -p cce-2k9sw0cj

Using default pool ID from config: cce-4hw7gn1m
failed to get job details: [Code: cce.warning.GetAIJobByJobIdFailed, Message: get job by jobid failed, err: pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 not found job, RequestId: a17a6ab1-9045-4346-9f1a-53733c621321]
(base) luyuchao@luyuchaodeMacBook-Pro ~ % aihc job get pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 -p cce-2k9sw0cj
openapigetjobresponseresult:
    jobid: pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59
    name: aihc-createjob-test
    resourcepoolid: cce-2k9sw0cj
    command: sleep 100
    createdat: "2025-02-17T07:59:35Z"
    finishedat: ""
    runningat: ""
    scheduledat: ""
    datasources: []
    enablefaulttolerance: false
    customfaulttolerancepattern: []
    labels:
        - key: aijob.cce.baidubce.com/ai-user-id
          value: eca97e148cb74e9683d7b7240829d1ff
        - key: aijob.cce.baidubce.com/ai-user-name
          value: root
        - key: aijob.cce.baidubce.com/create-from-aihcp-api
          value: "true"
        - key: aijob.cce.baidubce.com/openapi-jobid
          value: pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59
        - key: key
          value: value
    priority: high
    queue: default
    status: Created
    image: registry.baidubce.com/aihc-aiak/aiak-training-llm
    resources:
        - name: baidu.com/a100_80g_cgpu
          quantity: 8
    enablerdma: false
    hostnetwork: false
    replicas: 2
    envs:
        - name: AIHC_JOB_NAME
          value: aihc-createjob-test
        - name: AIHC_TENSORBOARD_LOG_PATH
          value: ""
        - name: CUDA_DEVICE_MAX_CONNECTIONS
          value: "1"
        - name: NCCL_DEBUG
          value: INFO
    jobframework: pytorch
    queueingsequence: null
    enablebccl: false
    enablebcclstatus: unknown
    enablebcclerrorreason: ""
    k8suid: 3168a331-b584-46c6-b6cc-b01214e217a7
    k8snamespace: default
podlist:
    listmeta:
        totalitems: 0
    pods: []
```

查询任务状态

```
 aihc job get jobID --status/-s
```

- 示例命令

```
 $ aihc job get pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 -p cce-2k9sw0cj -s
name                priority  pool/queue              replicas  status   runningAt  scheduledAt  createdAt
aihc-createjob-test  high      cce-2k9sw0cj/default  2         Created                          2025-02-17T07:59:35Z
```

查询任务实例列表

```
 aihc pod list jobID [flags]

Flags:
  -h, --help          help for list
  -p, --pool string   资源池ID
  -q, --queue string  队列ID
```

- 示例命令

```
##### 查询指定资源池下指定任务的实例列表
$ aihc pod list pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 -p cce-2k9sw0cj

##### 查询默认资源池下指定任务的实例列表
$ aihc pod list pytorchjob-a3a7f6cf-43c9-4792-b0e7-fdb454a7555b

replicaType    name                          namespace    podPhase    status    creationTimestamp
master         pxy-moe-48hours-cpu-master-0  default      Running     Running   2024-12-17T13:22:41Z
```

连接任务实例

```
aihc job exec jobID [flags]

Flags:
  -c, --container string   容器名称
  -h, --help               help for exec
  -i, --interactive        保持标准输入打开
  -n, --namespace string   命名空间 (默认为 default)
  -p, --pool string        资源池ID
  -t, --tty                分配伪终端
```

- 示例命令

```
$ aihc job exec pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 -it pxy-moe-48hours-cpu-master-0  bin/bash
```

复制任务配置

```
aihc job export jobID [flags]

Flags:
  -h, --help          help for export
  -p, --pool string   资源池ID
  -q, --queue string  队列ID
```

> 导出指定任务的配置为JSON文件，可直接用于 `aihc job create -f` 重新创建任务。 导出文件保存路径：`workspace/aihc/<任务名>-copy.json` 推荐使用文档中的任务模版修改后创建训练任务

- 示例命令

```
##### 导出指定资源池下的任务配置
$ aihc job export pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59 -p cce-2k9sw0cj -q my-queue
workspace/aihc/aihc-createjob-test-copy.json

##### 导出默认资源池下的任务配置
$ aihc job export pytorchjob-c0d35053-229d-4cfd-b44b-6dd7c39dde59

##### 使用导出的配置文件创建新任务
$ aihc job create -f workspace/aihc/aihc-createjob-test-copy.json
```

> **RayJob 导出说明：**
>
> - 导出的 JSON 中 `jobSpec` 为角色映射格式（{"Head": {...}, "Worker": {...}}）
> - 自动剥离 RayJob 不兼容的字段（faultTolerance、enableBccl、retentionPeriod 等）
> - Head replicas 固定为 1，Worker replicas 按原始配置保留
> - `preInitCommand` 保留在顶层

查询任务日志

```
aihc job logs jobID [flags]

Flags:
  -c, --chunk string       输出日志按着chunk数进行汇聚,例如将10行日志为1条记录,默认0
      --filepath string    日志文件路径
  -h, --help               help for logs
      --log-source string  日志来源,例如node
      --marker string      日志的起始位置,用于分页查询
  -m, --max-lines string   日志的最大行数
  -n, --namespace string   命名空间 (默认为 default)
      --podname string     Pod名称
  -p, --pool string        资源池ID
  -q, --queue string       队列ID
  -s, --start-time string  日志的起始时间,unix时间戳
```

- 示例命令

```
$ aihc job logs pytorchjob-a3a7f6cf-43c9-4792-b0e7-fdb454a7555b --podname ui-test-test-running-master-0 -p cce-hcuw9ybk
```

开发机相关操作

生成开发机创建模板

> 纯客户端生成，不调用API。

```
 $ aihc dev template [flags]

Flags:
 -f, --format string   模板格式：json 或 yaml（默认 json） (default "json")
 -h, --help            help for template
 -p, --path string     模板生成目录，未指定则生成到当前目录
```

- 命令示例

```
 ##### 默认生成 dev_conf.json 到当前目录
$ aihc dev template
dev create template generated: ./dev_conf.json

##### 生成 YAML 格式模板到指定目录
$ aihc dev template -f yaml -p /tmp
dev create template generated: /tmp/dev_conf.yaml
```

创建开发机

> 支持两种方式：模板文件创建（-f）和逐参数创建。开发机API要求至少提供一个数据卷（`--cds-size` 或 `--volumn-confs`），否则会返回错误。

```
 $ aihc dev create [flags]

Flags:
      --accelerator-count int32     加速卡数量
      --accelerator-type string     加速卡类型（如 NVIDIA_A100）
      --blb-id string               BLB 实例 ID（开启外网访问时必填）
      --cds-mount string            CDS 数据卷挂载路径 (default "/home/work")
      --cds-size int32              CDS 数据卷容量 (GB)
      --cpu int32                   CPU 核数
      --creator string              创建者名称
      --creator-id string           创建者 ID
      --envs stringArray            环境变量（key=value，可重复）
  -f, --file string                 开发机创建参数文件 (yaml/json)，缺省读取当前目录的 dev_conf.yaml
  -h, --help                        help for create
      --image-url string            镜像地址
      --memory int32                内存 (GB)
  -n, --name string                 开发机名称（标志模式必填）
      --port-info stringArray       自定义端口（name=xxx,port=NNN,access-port=NNN，可重复）
      --priority string             调度优先级 (low/normal/high)
  -q, --queue-name string           队列名称
  -p, --resource-pool-id string     资源池 ID
      --resource-pool-name string   资源池名称（部分校验场景需填）
      --resource-pool-type string   资源池类型（如 common）
      --shm-size int32              共享内存大小 (GB)
      --ssh-enable                  是否启用 SSH 访问
      --ssh-rsa-pub-key string      SSH RSA 公钥内容
      --start-cmd string            开发机启动后执行的命令
      --volumn-confs stringArray    存储卷（type=cds,capacity=100,mount=/foo 或 type=pfs,instance-id=...,source=/x,mount=/y[,read-only=true]，可重复）
      --workspace-dir string        工作目录路径
```

- 命令示例（模板方式创建）

```
 ##### 使用JSON格式模板创建
$ aihc dev create -f dev_conf.json

##### 使用YAML格式模板创建
$ aihc dev create -f dev_conf.yaml
```

JSON格式开发机创建参数参考模板（由 `aihc dev template` 生成）：

```
{
 "name": "my-dev",
 "conf": {
  "resourcePool": {
   "resourcePoolType": "serverless",
   "resourcePoolId": "aihc-xxxxx",
   "resourcePoolName": "",
   "queueName": "aihcq-xxxxx"
  },
  "resources": {
   "acceleratorType": "",
   "acceleratorCount": 0,
   "cpus": 4,
   "memory": 16,
   "shmSize": 8
  },
  "image": {
   "imageType": 0,
   "imageUrl": "registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04",
   "username": "",
   "password": ""
  },
  "access": {
   "blbId": "",
   "sshEnable": false,
   "sshRSAPubKey": "",
   "portInfo": []
  },
  "scheduleConf": {
   "cpuNodeAffinity": false,
   "priority": "high"
  },
  "workspaceDir": "$HOME/workspace",
  "envs": {},
  "startCmd": "",
  "volumnConfs": [
   {
    "volumnType": "cds",
    "pfs": null,
    "dataset": null,
    "cfs": null,
    "cds": {
     "capacity": 20
    },
    "bos": null,
    "mountPath": "/.rootfs",
    "readOnly": true
   }
  ],
  "updatedAt": 0
 },
 "visibleScope": {
  "type": 1
 },
 "notify": {
  "notifyRuleId": "",
  "isOpen": false
 },
 "isPublicMgmt": false,
 "creator": "",
 "creatorId": ""
}
```


YAML格式开发机创建参数参考模板（由 `aihc dev template -f yaml` 生成）：

```
conf:
  access:
    blbId: ""
    portInfo: []
    sshEnable: false
    sshRSAPubKey: ""
  envs:
    k1: v1
    k2: v2
  image:
    imageType: 0
    imageUrl: registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04
    password: ""
    username: ""
  resourcePool:
    queueName: aihcq-xxxxx
    resourcePoolId: aihc-xxxxx
    resourcePoolName: ""
    resourcePoolType: serverless
  resources:
    acceleratorCount: 0
    acceleratorType: ""
    cpus: 4
    memory: 16
    shmSize: 8
  scheduleConf:
    cpuNodeAffinity: false
    priority: high
  startCmd: ""
  updatedAt: 0
  volumnConfs:
    - bos: null
      cds:
        capacity: 20
      cfs: null
      dataset: null
      mountPath: /.rootfs
      pfs: null
      readOnly: true
      volumnType: cds
  workspaceDir: $HOME/workspace
creator: ""
creatorId: ""
isPublicMgmt: false
name: my-dev
notify:
  isOpen: false
  notifyRuleId: ""
visibleScope:
  type: 1
```

- 命令示例（逐参数方式创建）

```
 ##### 基础参数创建
$ aihc dev create -n dev-cli-test-01 -p aihc-1oobshtbcwip --resource-pool-type serverless -q aihcq-j5ncoe0w0m2o --image-url ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04 --cpu 2 --memory 8 --cds-size 50

##### 带GPU + 优先级 + SSH
$ aihc dev create -n dev-cli-test-02 -p aihc-1oobshtbcwip --resource-pool-type serverless -q aihcq-j5ncoe0w0m2o --image-url ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04 --cpu 2 --memory 8 --accelerator-type kunlunxin.com/xpu --accelerator-count 1 --priority high --ssh-enable --cds-size 50

##### 带共享内存 + 工作目录 + 启动命令
$ aihc dev create -n dev-cli-test-03 -p aihc-1oobshtbcwip --resource-pool-type serverless -q aihcq-j5ncoe0w0m2o --image-url ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04 --cpu 2 --memory 8 --shm-size 8 --workspace-dir /workspace --start-cmd "echo hello" --cds-size 50

##### 带环境变量 + 端口映射
$ aihc dev create -n dev-cli-test-04 -p aihc-1oobshtbcwip --resource-pool-type serverless -q aihcq-j5ncoe0w0m2o --image-url ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04 --cpu 2 --memory 8 --envs KEY1=val1 --envs KEY2=val2 --port-info name=test,port=8080,access-port=30080 --cds-size 50

##### 带存储卷
$ aihc dev create -n dev-cli-test-05 -p aihc-1oobshtbcwip --resource-pool-type serverless -q aihcq-j5ncoe0w0m2o --image-url ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04 --cpu 2 --memory 8 --volumn-confs "type=pfs,instance-id=pfs-Eo2rEd,source=/,mount=/mnt/pfs" --cds-size 50

##### 带创建者
$ aihc dev create -n dev-cli-test-06 -p aihc-1oobshtbcwip --resource-pool-type serverless -q aihcq-j5ncoe0w0m2o --image-url ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04 --cpu 2 --memory 8 --creator xxx --cds-size 50
```

获取开发机列表

```
 $ aihc dev list/ls [flags]

Flags:
 -h, --help                  help for list
     --name string           按开发机名称模糊过滤
     --only-my-devs          仅显示我创建的开发机
     --order string          排序方向：asc(升序) / desc(降序)
     --order-by string       排序字段：createdAt(创建时间) / updatedAt(更新时间) / name(名称)
     --pageNo int            页码（从 1 开始） (default 1)
 -s, --pageSize int          每页数量 (default 50)
     --query-key string      查询字段名（高级过滤；与 --name 互斥）
     --query-val string      查询字段值（高级过滤；与 --name 互斥）
 -q, --queue-name string     按队列名称过滤
 -p, --resource-pool-id string  按资源池 ID 过滤
     --status string         按状态过滤（数字状态码，例如 3=运行中、5=已关机）
```

- 命令示例

```
##### 列出所有开发机
$ aihc dev ls

##### 按资源池和名称过滤
$ aihc dev list --resource-pool-id cce-xxxxx --name dev-test --pageNo 1 --pageSize 10

##### 只显示我的开发机
$ aihc dev ls --only-my-devs -s 5
```

获取开发机详情

```
$ aihc dev get <devID> [flags]

Flags:
  -h, --help            help for get
  -o, --output string   输出格式：table / json / yaml（缺省为 yaml）
```

- 命令示例

```
##### 默认YAML格式输出
$ aihc dev get dv-216e683e1d

##### JSON格式输出
$ aihc dev get dv-f259223fa7 -o json

##### 表格格式输出
$ aihc dev get dv-216e683e1d -o table
```

输出示例（表格）：

```
ID                  dv-c914a799a7
Name                commit-test
Status              running
Status Reason
Region              bj
Creator             xxx (4ab0a886c9774dfc842db8d481479705)
Created At          2026-05-27 09:48:24
Updated At          2026-06-01 14:08:26
Pool/Queue          / aihcq-oh1yq6xr0xak
Resources           4C/16G
Image               ccr-registry.baidubce.com/aihc/aihc-daft-gpu:0.4.0-cu12.1-py3.11-ubuntu22.04
NodeIP/PodName      10.0.1.8 / aibox-rc09b3405e6d-d5c9748bb-b88jw
Jupyter URL         https://console.bce.baidu.com/api/aihcbox/proxy/v1/webide/dv-c914a799a7/bj/jupyter
VSCode URL          https://console.bce.baidu.com/api/aihcbox/proxy/v1/webide/dv-c914a799a7/bj/vscode/
```

删除开发机

```
$ aihc dev delete/rm <devID> [flags]

Flags:
  -h, --help   help for delete
```

- 命令示例

```
$ aihc dev delete dv-216e683e1d
```

开关机

```
$ aihc dev start <devID> [flags]
$ aihc dev stop <devID> [flags]

Flags:
  -h, --help   help for start/stop
```

- 命令示例

```
##### 开机
$ aihc dev start dv-bccbdd9db8

##### 关机
$ aihc dev stop dv-bccbdd9db8
```

定时停止开发机

```
$ aihc dev timed-stop <devID> [flags]

Flags:
      --delay-sec int32   延迟关机秒数（启用定时关机时必填）
      --enable            启用/取消定时关机：--enable=true 启用，--enable=false 取消 (default true)
  -h, --help              help for timed-stop
```

- 命令示例

```
##### 设置1小时后定时停止
$ aihc dev timed-stop dv-bccbdd9db8 --delay-sec 3600

##### 取消定时停止
$ aihc dev timed-stop dv-bccbdd9db8 --enable=false
```

更新开发机配置

> 需通过 -f 指定完整的配置文件 (yaml/json)，建议先使用 aihc dev export <devID> 导出后修改再提交。

```
$ aihc dev update <devID> [flags]

Flags:
 -f, --file string   更新配置文件 (yaml/json)，必填
 -h, --help          help for update
```

- 命令示例

```
##### 导出当前配置
$ aihc dev export dv-f4cdd6964d -o /tmp/dev_upd.json -f json

##### 修改配置文件后更新
$ aihc dev update dv-f4cdd6964d -f /tmp/dev_upd.json
```

变更参数文件示例（dev_update.yaml）：

```
name: new-dev-name
conf:
  image:
    imageUrl: registry.baidubce.com/aihc/pytorch:2.0.0
  resources:
    cpus: 16
    memory: 64
```

提交开发机为镜像

```
$ aihc dev commit <devID> [flags]

Flags:
 -h, --help                help for commit
     --image-name string   镜像名称（必填）
     --image-tag string    镜像 Tag (default "latest")
     --namespace string    镜像命名空间（必填）
     --password string     镜像仓库密码
     --registry string     镜像仓库地址（必填）
     --username string     镜像仓库用户名
```

- 命令示例

```
$ aihc dev commit dv-e78be903d7 \
  --image-name dev-cli-test-img2 \
  --namespace zzq \
  --image-tag v1 \
  --registry ccr-registry.baidubce.com
```

查看镜像制作任务详情

> 通过 --image-pack-job-id 指定镜像任务ID（一般在 aihc dev commit 时返回）。

```
$ aihc dev img-list <devID> [flags]

Flags:
 -h, --help                    help for img-list
     --image-pack-job-id string   镜像任务 ID（必填）
```

- 命令示例

```
$ aihc dev img-list dv-e78be903d7 --image-pack-job-id tk-6db402058f
```

查看开发机事件

```
$ aihc dev events <devID> [flags]

Flags:
     --end-time string     结束时间
 -t, --event-type string   事件类型（如 Normal/Warning）
 -h, --help                help for events
 -i, --interactive         启用交互式翻页 (n/p/q)
     --message string      按消息内容过滤
     --pageNo int          页码（从 1 开始） (default 1)
 -s, --pageSize int        页大小 (default 10)
     --start-time string   开始时间
```

- 命令示例

```
$ aihc dev events dv-e78be903d7 \
  --start-time "2026-05-21T00:00:00Z" \
  --end-time "2026-05-27T23:59:59Z" \
  -t Warning --message "probe" \
  --pageNo 1 -s 3
```

导出开发机配置

> 导出配置可用于 create -f 和 update -f，实现导出→修改→创建/更新闭环。

```
 $ aihc dev export <devID> [flags]

Flags:
 -f, --format string   输出格式：json / yaml（默认 json，大小写不敏感）(default "json")
 -h, --help            help for export
 -o, --output string   输出文件路径
```

- 命令示例

```
 ##### 导出为JSON格式
$ aihc dev export dv-c914a799a7 -o /tmp/dev_exp.json -f json
/tmp/dev_exp.json

##### 导出为YAML格式
$ aihc dev export dv-c914a799a7 -o /tmp/dev_exp.yaml -f yaml
/tmp/dev_exp.yaml
```

推理服务相关操作

生成推理服务创建模板

> 纯客户端生成，不调用API。

```
 $ aihc service template [flags]

Flags:
 -f, --format string   模板格式：json 或 yaml（默认 json）(default "json")
 -h, --help            help for template
 -p, --path string     模板生成目录，未指定则生成到当前目录
```

- 命令示例

```
 ##### 默认生成 service_conf.json 到当前目录
$ aihc service template
service create template generated: ./service_conf.json

##### 生成 YAML 格式模板到指定目录
$ aihc service template -f yaml -p /tmp
service create template generated: /tmp/service_conf.yaml
```

YAML格式推理服务创建参数参考模板（由 `aihc service template -f yaml` 生成）：

```
##### 推理服务创建模板。提交时使用: aihc service create -f service_conf.yaml
name: ""                        # 服务名称（必填）
instanceCount: 1                # 实例副本数
acceleratorType: ""             # 加速芯片类型（如 NVIDIA-A100）

##### === 容器配置 ===
containers:
  - name: main                  # 容器名称
    cpus: 8                     # CPU 核数
    memory: 32                  # 内存 GB
    acceleratorCount: 1         # GPU 卡数
    command:                    # 启动命令
      - python
      - /workspace/app.py
    image:
      imageType: 1              # 0=公共镜像 1=自定义镜像
      imageUrl: ""              # 镜像地址
      username: ""
      password: ""
    ports:                      # 端口映射
      - name: HTTP
        port: 8000
    envs:                       # 环境变量
      MODEL_NAME: deepseek-r1

##### === 存储配置 ===
storage:
  shmSize: 10                   # 共享内存大小 (GB)
  volumns:
    - volumeType: pfs
      volumnName: model-pfs
      pfs:
        sourcePath: /mnt/models
      mountPath: /workspace/models

##### === 资源池配置 ===
resourcePool:
  resourcePoolId: ""            # 资源池 ID
  resourcePoolName: ""          # 资源池名称
  queueName: ""                 # 队列名称

##### === 调度配置 ===
deploy:
  schedule:
    priority: normal            # low | normal | high

##### === 流量接入 ===
access:
  publicAccess: false
  networkType: aiGateway        # aiGateway | blb
  aiGateway:
    enableAuth: true

##### === 高级配置 ===
misc:
  gracePeriodSec: 30
  fedPodsPerIns: 0
```

创建推理服务

> 支持两种方式：模板文件创建（-f）和逐参数创建。-f 支持 JSON 和 YAML 格式，通过文件内容自动判断格式（以 { 或 [开头按 JSON 处理，否则按 YAML 处理）。

```
$ aihc service create [flags]

Flags:
      --accelerator-count int32     加速卡数量（第一个容器）
      --accelerator-type string     加速卡类型（如 NVIDIA-A100）
      --command string              启动命令（第一个容器）
      --containers string           自定义容器 JSON 字符串（多容器场景，与逐参数标志互斥）
      --cpus int32                  CPU 核数（第一个容器）
      --envs stringArray            环境变量（key=value，可重复）
  -f, --file string                 推理服务创建参数文件 (yaml/json)，缺省读取当前目录的 service_conf.yaml
  -h, --help                        help for create
      --image-url string            镜像地址（作用于第一个容器）
      --instance-count int32        实例副本数 (default 1)
      --memory int32                内存 GB（第一个容器）
  -n, --name string                 服务名称（标志模式必填）
      --network-type string         网络类型：aiGateway / blb
      --ports stringArray           端口映射（name=HTTP,port=8000，可重复）
      --priority string             调度优先级 (low/normal/high)
      --public-access               开启公网访问
  -q, --queue-name string           队列名称
  -p, --resource-pool-id string     资源池 ID
      --resource-pool-name string   资源池名称
      --resource-pool-type string   资源池类型（空字符串=自运维, serverless=全托管）
      --volumns stringArray         存储卷（type=pfs,source=/mnt,mount=/workspace，可重复）
```

- 命令示例（模板方式创建）

```
##### 使用JSON文件创建
$ aihc service create -f test_service.json

##### 使用YAML文件创建
$ aihc service create -f test_service.yaml
```

- 命令示例（逐参数方式创建）

```
##### 基础参数创建
$ aihc service create --name test-cli-create-01 \
  --resource-pool-id cce-1ov7onxr --queue-name default \
  --image-url registry.baidubce.com/aiak-inference/aiak-inference-comfyui:ubuntu22.04-cu12.4-torch2.5.1-py310_v1.2 \
  --cpus 4 --memory 16 --instance-count 1

##### 完整参数创建（GPU + 命令 + 环境变量 + 端口 + 优先级 + 网络类型）
$ aihc service create --name test-cli-create-02 \
  --resource-pool-id cce-1ov7onxr --queue-name default \
  --image-url registry.baidubce.com/aiak-inference/aiak-inference-comfyui:ubuntu22.04-cu12.4-torch2.5.1-py310_v1.2 \
  --cpus 4 --memory 16 --accelerator-type NVIDIA-A100 --accelerator-count 1 \
  --instance-count 1 --command "python /workspace/app.py" \
  --envs MODEL_NAME=test-model --envs DEBUG=true \
  --ports name=HTTP,port=8000 --priority high --network-type aiGateway

##### 带存储卷
$ aihc service create --name test-cli-create-03 \
  --resource-pool-id cce-1ov7onxr --queue-name default \
  --image-url registry.baidubce.com/aiak-inference/aiak-inference-comfyui:ubuntu22.04-cu12.4-torch2.5.1-py310_v1.2 \
  --cpus 2 --memory 8 --instance-count 1 \
  --volumns "type=pfs,source=/mnt/models,mount=/workspace/models,instance-id=pfs-xxxx"

##### 带资源池名称
$ aihc service create --name test-cli-create-rpn \
  --resource-pool-id cce-1ov7onxr --resource-pool-name aihclite-regress-infer \
  --queue-name default \
  --image-url registry.baidubce.com/aiak-inference/aiak-inference-comfyui:ubuntu22.04-cu12.4-torch2.5.1-py310_v1.2 \
  --cpus 2 --memory 8 --accelerator-type NVIDIA-A100 --accelerator-count 1 --instance-count 1
```

获取推理服务列表

> 注意：OpenAPI当前不返回服务状态信息，故列表中无状态列。

```
$ aihc service list/ls [flags]

Flags:
  -h, --help                  help for list
  -n, --name string             按名称关键字过滤（客户端过滤）
      --order string           排序方向：asc(升序)/desc(降序)
      --order-by string         排序字段：createdAt(创建时间)/updatedAt(更新时间)/name(名称)
      --pageNo int              页码（从 1 开始） (default 1)
  -s, --pageSize int            每页数量 (default 50)
  -p, --resource-pool-id string   按资源池 ID 过滤（客户端过滤）
```

- 命令示例

```
##### 列出指定资源池的服务
$ aihc service list --resource-pool-id cce-1ov7onxr --name test --pageNo 1 --pageSize 10

##### 使用短标志和排序
$ aihc service ls -p cce-1ov7onxr -n ctest --pageNo 1 -s 5 --order-by createdAt --order desc

##### 升序排列
$ aihc service list -p cce-1ov7onxr -n test -s 5 --order-by createdAt --order asc
```

获取推理服务详情

```
$ aihc service get <serviceID> [flags]

Flags:
  -h, --help            help for get
  -o, --output string   输出格式：table / json / yaml（缺省为 yaml，大小写不敏感）
```

- 命令示例

```
##### 默认YAML格式输出
$ aihc service get s-ref4993b9dd5

##### JSON格式输出
$ aihc service get s-ref4993b9dd5 -o json

##### 表格格式输出
$ aihc service get s-r53254a7b418 -o table
```

删除推理服务

> 只支持删除标准推理服务，不支持删除多角色推理服务。

```
$ aihc service delete/rm <serviceID> [flags]

Flags:
  -h, --help    help for delete
```

- 命令示例

```
$ aihc service delete s-r36caeaf78fa
```

更新推理服务

> 需通过 -f 指定完整的配置文件 (yaml/json)，建议先使用 `aihc service export <serviceID>` 导出后修改再提交。注意：仅支持修改 acceleratorType/instanceCount/storage/containers/log/deploy/misc，name 和 resourcePool 不可修改；serverless 资源池的服务可能因 API 限制无法更新。

```
$ aihc service update <serviceID> [flags]

Flags:
  -d, --description string   更新描述
  -f, --file string          更新配置文件 (yaml/json)，必填
  -h, --help                 help for update
```

- 命令示例

```
##### 导出当前配置
$ aihc service export s-r53254a7b418 -o /tmp/update_test.json -f json

##### 修改配置文件后更新
$ aihc service update s-r53254a7b418 -f /tmp/update_test.json -d "test update cpus to 6"
```

变更参数文件示例（service_update.yaml）：

```
description: "升级到 v2.0 模型"
containers:
  - name: main
    image:
      imageUrl: registry.baidubce.com/aihc/vllm:v2.0.0
```

扩缩容

```
$ aihc service scale <serviceID> [flags]

Flags:
  -h, --help                 help for scale
      --instance-count int32   目标实例数量（必填，0 表示缩容至零） (default -1)
```

- 命令示例

```
##### 扩容到2个实例
$ aihc service scale s-r53254a7b418 --instance-count 2
```

配置公网访问

> 仅 AI 原生网关（aiGateway）类型支持，BLB 负载均衡类型不支持。开启公网访问需绑定 EIP，否则 API 会返回错误。

```
$ aihc service net-config <serviceID> [flags]

Flags:
      --eip string           绑定的弹性公网 IP（可选）
  -h, --help                 help for net-config
      --public-access string   开启/关闭公网访问 (true/false) (default "false")
```

- 命令示例

```
##### 开启公网访问
$ aihc service net-config s-r15dcaf6a0a4 --public-access true

##### 关闭公网访问
$ aihc service net-config s-r15dcaf6a0a4 --public-access false

##### 开启公网访问并绑定EIP
$ aihc service net-config s-r15dcaf6a0a4 --public-access true --eip "1.2.3.4"
```

重启推理服务

> 重启操作通过删除 Pod 触发平台重建实现。不指定 -i 时重启整个服务（逐个重建所有 Pod），指定 -i 时重建指定实例。

```
$ aihc service restart <serviceID> [flags]

Flags:
  -h, --help                 help for restart
  -i, --instance-id string   指定实例 ID
```

- 命令示例

```
##### 重启整个服务
$ aihc service restart s-r53254a7b418

##### 重启指定实例
$ aihc service restart s-r53254a7b418 -i s-r53254a7b418-67477dc4b8-7r65v
```

查看Pod列表

> 不支持多角色服务。API不支持分页，故CLI无 `--pageNo`/`--pageSize` 参数。

```
 $ aihc service pods <serviceID> [flags]

Flags:
 -h, --help   help for pods
```

- 命令示例

```
 $ aihc service pods s-r53254a7b418
```

查看实例组列表

> 仅用于开启分布式部署的标准服务，不支持多角色服务。

```
 $ aihc service pod-groups <serviceID> [flags]

Flags:
 -h, --help   help for pod-groups
```

- 命令示例

```
 $ aihc service pod-groups s-f41e42d5c6fe
```

删除Pod（触发重建）

```
 $ aihc service pod-delete <serviceID> [flags]

Flags:
 -h, --help                help for pod-delete
 -i, --instance-id string  实例 ID（必填）
```

- 命令示例

```
 ##### 先查看Pod列表获取实例ID
$ aihc service pods s-r53254a7b418

##### 删除指定Pod，将自动触发重建
$ aihc service pod-delete s-r53254a7b418 -i s-r53254a7b418-98bdc687-wcr7b
```

摘流/恢复

```
 $ aihc service disable-service-pod <serviceID> [flags]

Flags:
     --block               true=摘除流量 / false=恢复流量 (default true)
 -h, --help                help for disable-service-pod
 -i, --instance-id string  实例 ID（必填）
```

- 命令示例

```
 ##### 摘流
$ aihc service disable-service-pod s-r70a6f998e10 -i s-r70a6f998e10-677c7d89b-j49dj --block=true

##### 恢复流量
$ aihc service disable-service-pod s-r70a6f998e10 -i s-r70a6f998e10-677c7d89b-j49dj --block=false
```

进入服务容器终端

> 通过 WebSocket 连接进入容器终端。需 Pod 处于 Running 状态，且容器中需有 `/bin/bash`。

```
 $ aihc service exec <serviceID> [flags]

Flags:
     --handshake-timeout-second int32   建连超时（秒），默认 30 (default 30)
 -h, --help                             help for exec
 -i, --instance-id string               实例 ID（必填）
     --ping-timeout-second int32        心跳超时（秒），默认 900 (default 900)
 -q, --queue string                     队列名称或 ID
 -p, --resource-pool-id string          资源池 ID
     --resource-pool-type string        资源池类型
```

- 命令示例

```
 $ aihc service exec s-r640c92f3d2a -i s-r640c92f3d2a-7bc6f8f859-749tg -p aihc-serverless -q aihcq-tkdtuz8q6ud9 --resource-pool-type serverless
```

查看推理服务日志

> 支持分页（通过 marker 翻页）和实时跟踪（`--follow`）。与 `job logs` 对齐，支持 follow 和非 follow 两种模式。

```
 $ aihc service logs <serviceID> [flags]

Flags:
  -c, --container string     容器名称
  -e, --end-time string      日志结束时间（RFC3339 格式，如 2026-03-27T01:40:35Z）
  -f, --follow               实时跟踪日志
  -h, --help                 help for logs
  -i, --instance-id string   实例 ID（必填）
      --marker string        日志起始位置，用于分页（取上一次返回的 nextMarker）
  -m, --max-lines int32      最大行数 (default 100)
  -n, --namespace string     命名空间（默认为 default）
  -s, --start-time string    日志起始时间（RFC3339 格式，如 2026-03-20T01:40:35Z）
```

- 命令示例

```
 ##### 查看指定时间范围的日志
$ aihc service logs s-r045298c51cb \
   -i s-r045298c51cb-5b558cf964-bb2cs \
   -s "2026-05-27T00:00:00Z" \
   -e "2026-05-27T20:00:00Z" \
   -m 50

##### 持续输出日志（follow模式）
$ aihc service logs s-r045298c51cb \
   -i s-r045298c51cb-5b558cf964-bb2cs \
   -f
```

查看变更记录

```
 $ aihc service changelog <serviceID> [changeID] [flags]

Flags:
  -h, --help           help for changelog
      --pageNo int     页码（从 1 开始） (default 1)
  -s, --pageSize int   每页数量 (default 50)
```

- 命令示例

```
 ##### 查看变更记录列表
$ aihc service changelog s-r53254a7b418

##### 查看某条变更详情
$ aihc service changelog s-r8294f7229c4 ch-2b4a8b7a33d0 --pageNo 1 -s 20
```

导出推理服务配置

> 导出配置可用于 `create -f` 和 `update -f`，实现导出→修改→创建/更新闭环。

```
 $ aihc service export <serviceID> [flags]

Flags:
  -f, --format string   输出格式：json / yaml（默认 json，大小写不敏感） (default "json")
  -h, --help            help for export
  -o, --output string   输出文件路径
```

- 命令示例

```
 ##### 默认导出JSON格式到当前目录
$ aihc service export s-ref4993b9dd5
./service_export.json

##### 导出JSON到指定路径
$ aihc service export s-ref4993b9dd5 -o /tmp/test_j.json -f json
/tmp/test_j.json

##### 导出YAML到指定路径
$ aihc service export s-ref4993b9dd5 -o /tmp/test_y.yaml -f yaml
/tmp/test_y.yaml
```

透传请求OpenAPI

透传请求 AIHC 的 OpenAPI，自动附带 BCE V1 签名。用法与原生 curl 保持一致，仅自动处理百度云鉴权签名。

支持请求百舸已开放的全部OpenAPI，接口参数详见API参考

```
 aihc curl <URL> [flags]

Flags:
  -d, --data string      请求body数据（JSON字符串）
  -H, --header strings   自定义header，格式 'Key: Value'，可多次指定
  -h, --help             help for curl
  -X, --request string   请求method: GET/POST/PUT/DELETE (default "GET")

Global Flags:
  -C, --config string   Global configuration file. (default "./.aihc/config")
```

- 命令示例

```
##### GET 请求（查询数据集列表）
aihc curl '/?action=DescribeDatasets' -X GET -H "version: v2"

##### GET 请求带分页参数
aihc curl '/?action=DescribeDatasets&pageSize=10&pageNumber=1' -X GET -H "version: v2"

##### POST 请求（创建数据集）
aihc curl '/?action=CreateDataset' -X POST -H "version: v2" -d '{"name":"test","storageType":"PFS"}'

##### 指定完整 host
aihc curl 'http://aihc.bj.baidubce.com/?action=DescribeDatasets' -X GET -H "version: v2"
```

版本变更记录

- [2026-10-26] 版本号1.6.5
  - 创建训练任务支持 RayJob Head 节点容错
- [2026-09-21] 版本号1.6.4
  - 创建训练任务支持开启闲时任务（仅全托管资源池）
- [2026-09-17] 版本号1.6.3
  - 创建训练任务支持挂载 pfsl3 存储、设置监控端口
- [2026-08-13] 版本号1.6.2
  - 支持Ray任务的创建、复制
  - 修复不指定 framework 标志时创建任务报错的问题
  - 修复创建训练任务的代码上传问题
  - 修复 aihc pool set 命令中队列信息的展示问题：托管资源池将 defaultQueue 设置为队列 ID，自运维资源池设置为队列名称
  - 修复 defaultpool 参数在任务接口与资源池接口间冲突的问题——defaultpool 统一传递真实资源池 ID，托管资源池调用训练接口时自动补充为 aihc-serverless
  - 新增创建训练任务时的数据集挂载标志 ds-id 和任务可见范围标志 visible-scope
  - aihc job list支持显示训练任务框架
  - 同步更新官网文档
- [2026-07-29] 版本号1.6.1
  - 开发机、推理服务、训练任务支持通过命令行 flag 覆盖 yaml/json 文件中的配置，优先级为「命令行 flag > yaml/json 文件」
  - 修复训练任务导出命令中 datasource 里多余的共享内存参数影响后续训练任务创建的问题
- [2026-05-27] 版本号1.6.0
  - 新增开发机与推理服务的 CLI 命令
- [2026-05-20] 版本号1.4.0
  - 新增 `aihc curl` 命令，透传请求 AIHC 的 OpenAPI，自动附带 BCE V1 签名，用法与原生 curl 保持一致
- [2026-03-18] 版本号1.3.2
  - aihc pod list 命令支持显示 podIP 和 nodeName
- [2026-02-27] 版本号1.3.1
  - 修复创建训练任务挂载数据集报错
- [2025-11-05] 版本号1.3.0
  - 新增配置文件管理和切换当前配置功能
  - 修复aihc job list命令报错问题
- [2025-10-21] 版本号1.2.2
  - 新增创建训练任务时设置任务保留时间的支持，可通过 -r/--retention-period 标志或 yaml/json 文件配置。
- [2025-09-03] 版本号1.2.1
  - 修复日志命令因缺少 queueID 导致的报错问题。
  - 修复代码上传操作因缺少 queueID 导致的报错问题。
- [2025-09-01] 版本号1.2.0
  - 新增查询托管资源池接口支持（新增参数--resourcePoolType，默认值为common）
  - 移除查询资源池节点列表命令
- [2025-04-08] 版本号1.1.0
  - 新增创建任务支持容错
  - 新增代码上传操作
  - 新增进入容器操作
  - 新增日志按照路径查询
- [2025-03-06] 版本号1.0.0
  - 首发版本，支持资源池、队列、任务常用操作

AIHC Skill

概述

百舸 Skill 能力面向希望通过本地 Agent 使用百舸平台的用户，提供一组可复用的操作入口，帮助用户更自然地完成开发机访问、快速开始体验、训练任务执行、模型评估、性能调优、资源查询等操作。

通过 Skill，用户可以将百舸平台中的常见操作流程沉淀为 Agent 可理解、可调用、可检查的能力。用户不再需要在控制台、文档和终端之间反复切换，可以直接通过自然语言向本地 Agent 描述目标，由 Agent 结合 Skill 完成步骤拆解、命令生成、远程执行和结果解释。

使用方式

目前，您可以将 Skill 复制到本地 Agent 中使用。

计费说明

使用本地 Agent 运行 Skill 本身不产生百舸平台费用。通过 Skill 执行训练、部署、开发机创建、资源变配等操作时，会使用用户账号下已有的百舸资源，相关费用以实际使用的百舸资源为准。

具身智能

Isaac

isaac-lab-install

你可以将以下代码以如下形式放到您的 agent 目录中，即可通过/isaac-sim-install + 参数 的形式调用该 Skill。

```
.claude/skills/<install-skill>/
├── SKILL.md
└── scripts/
    └── install_isaac_lab.sh
```

SKILL.md：

```
 ---
name: isaac-lab-install
description: >
  在已有 Isaac Sim Python 上用内置、固定 recipe 安装 Isaac Lab。支持本机或 SSH 远端、临时代理和可选冒烟测试；
  不安装或重装 Isaac Sim。默认使用 /root/workspace/env_isaacsim/bin/python。
argument-hint: [local|ssh <ssh-target>] [--profile lab-3.0|lab-main|lab-2.3] [--install-root /root/workspace] [--sim-python <path>|--venv <path>] [--proxy <url>|--no-proxy] [--run-mode wait|background|schedule-check] [--smoke-test]
disable-model-invocation: true
allowed-tools: Read, Bash, AskUserQuestion, CronCreate
---

**Isaac Lab 安装 Skill**

你是 Isaac Lab 环境安装工程师。这个 skill 只在已有 Isaac Sim Python 上用内置、固定 recipe 安装 Isaac Lab；不安装、不重装 Isaac Sim。没有 recipe 的 Lab 版本不能自动安装，也不能现场猜 branch、commit 或安装命令。

**使用边界**

- 只安装 Isaac Lab；不调用 Isaac Sim 安装脚本。
- 必须依赖已有 Isaac Sim Python，默认是 `<install-root>/env_isaacsim/bin/python`。
- 找不到 Isaac Sim Python 时停止，让用户先运行 `isaac-sim-install` 或显式传 `--sim-python` / `--venv`。
- 安装逻辑固定在 `scripts/install_isaac_lab.sh`，通过 `--profile` 选择 recipe。
- 支持本机和 SSH 远端；参数以 `ssh ...` 开头时，所有检查和安装都在远端执行。
- 代理只临时用于 git/pip 下载；不要写入 shell 配置、安装记录或仓库文件。
- 默认安装根目录是 `/root/workspace`，但交互流程必须向用户确认。
- 成功且验证通过后，远端过程目录 `<install-root>/isaac-lab-install-skill` 必须删除。

**内置 Recipe**

| profile | Isaac Lab ref | commit | 期望 Sim | Python | 用途 |
| --- | --- | --- | --- | --- | --- |
| `lab-3.0` | `release/3.0.0-beta2` | `a8554968f604e54e4d0862c5922ed0f9b49aaa92` | Sim 6 | `3.12` | 默认推荐 |
| `lab-main` | `main` | `f58361c8f1ea3380ded6ddfde029c40b0e7203ca` | Sim 6 | `3.12` | 实验组合；不要作为默认推荐 |
| `lab-2.3` | `release/2.3.0` | `5c2ec81cb17532d32f7922dd7fcaae40d123b71a` | Sim 5 | `3.11` | 兼容 Sim 5.1 线 |

脚本仍接受旧组合 alias，例如 `sim-6.0.0+lab-3.0`、`sim-6.0.0+lab-main`、`sim-5.1.0+lab-2.3`，但新调用优先使用 Lab-only profile。

如果用户要求未列出的版本，例如 `lab-3.1` 或未来 `lab-4.0`，先进入 recipe 扩展流程：查询官方 IsaacLab repo release/tag/branch 和精确 commit、官方 Sim 兼容信息、Python 要求、PyTorch/CUDA pins、`source/` package layout，以及是否会拉偏 Sim 依赖；生成候选 `normalize_profile()` alias 和 `recipe_lab()` 分支后，先运行 `bash -n` 和 `--dry-run`，向用户说明来源、缺口和候选 recipe，得到确认后才写入脚本并允许安装。`lab-main` 也必须固定 commit；更新 main 属于 recipe 更新，不是无钉安装。

**环境约束**

这些约束是安装前预检和最终风险说明的来源，不用于现场改 recipe：

- Isaac Lab 必须复用已有 Isaac Sim Python；不能自动补装 Isaac Sim。
- Python 版本严格跟随 Lab recipe；不能用系统默认 Python 代替不匹配的小版本。
- 已安装的 `isaacsim` major 必须和 Lab recipe 的期望 Sim major 对照；不匹配时报告风险。
- 安装默认使用已有 venv `<install-root>/env_isaacsim`，不要装进系统 Python。
- Isaac Lab 通过 editable 子包安装，不安装仓库根 `pyproject.toml`，避免它临时改写 Sim 依赖。
- 安装 Lab 后必须恢复 recipe 中需要保护的 Sim pins；Sim 6 当前恢复 `coverage==7.4.4`、`numpy==2.3.1`、`Pillow==12.1.1`、`psutil==5.9.8`、`packaging==26.0`。
- `--smoke-test` 必须设置 `OMNI_KIT_ACCEPT_EULA=YES` 和 `OMNI_KIT_ALLOW_ROOT=1`，优先运行 `<install-root>/IsaacLab/scripts/environments/list_envs.py`。
- 如果 import/smoke test 通过但 `pip check` 或日志有 dependency conflict，最终报告要说明“基础运行可用，但依赖元数据不一致”。

**安装前决策**

用户调用示例：

```text
/isaac-lab-install ssh -p <port> <user>@<host> --profile lab-3.0 --sim-python /root/workspace/env_isaacsim/bin/python --proxy http://<proxy-host>:<proxy-port> --run-mode wait --smoke-test
```

参数和交互规则：

1. 目标：参数以 `ssh` 开头时，把后续 SSH 参数保存为远端命令前缀；没有 `ssh` 或参数是 `local` 时，在本机执行。
2. 目录：`--install-root` 默认 `/root/workspace`；用户未显式传入时必须确认默认目录是否合适，或让用户用 Other 输入已有/挂载目录。
3. Sim Python：默认 `<install-root>/env_isaacsim/bin/python`；已有 Sim 不在默认位置时，必须让用户传 `--sim-python` 或 `--venv`。
4. 代理：`--proxy <url>` 直接传给脚本；`--no-proxy` 只表示用户选择直连，脚本不接收该参数。
5. 运行方式：`--run-mode wait` 持续追踪到成功/失败；`background` 只启动后台任务并返回日志/PID；`schedule-check` 启动后台任务并设置本会话定时检查。
6. 版本：`--profile` 默认 `lab-3.0`；`lab-main` 是实验组合，不作为默认推荐。
7. 冒烟测试：`--smoke-test` 启用 Lab import 和官方 headless 示例验证。
```

7. 冒烟测试：`--smoke-test` 启用 Lab import 和官方 headless 示例验证。
8. 禁用参数：不支持 `--component`；这个 skill 永远只装 Lab。

只有用户显式给全目标、profile、proxy/no-proxy、run-mode 后，才可以跳过交互直接安装。用户只给 SSH 目标或只确认 host key 时，不要自动开始安装。不允许自动运行 `isaac-sim-install`；缺少 Sim 时停止并说明下一步。

交互顺序

一次交互最多问 1–4 个问题；远端首次连接可能需要 host key 信任时，把它合并进安装配置问题。

1. 目标和 host key：装本机还是远端；首次 SSH host key 是否信任。信任 host key 只表示允许连接，不等于允许开始安装。
2. 安装目录和 Sim Python：默认 `/root/workspace` 与 `/root/workspace/env_isaacsim/bin/python`；不同位置用 Other 输入 `--sim-python` 或 `--venv` 对应路径。
3. 网络代理：默认不假设用户有内网或预置代理。选项包含"不使用代理（直连）""我已有代理 URL（选 Other 填写）""不确定，先检测直连"。
4. 运行方式：`wait`、`background`、`schedule-check`。
5. Lab 版本：默认推荐 `lab-3.0`；实验线可选 `lab-main`；旧线可选 `lab-2.3`。
6. 是否启用 `--smoke-test`。

安装脚本

```bash
bash .claude/skills/isaac-lab-install/scripts/install_isaac_lab.sh \
  --profile lab-3.0 \
  --install-root /root/workspace \
  --sim-python /root/workspace/env_isaacsim/bin/python \
  --proxy http://host:port \
  --smoke-test
```

脚本职责：

1. 读取 `recipe_lab()` 中的固定分支、commit、Python/PyTorch 版本和 package dirs。
2. 验证已有 Isaac Sim Python、Python 小版本和 `isaacsim` major。
3. 按权限和参数安装或跳过 apt build dependencies。
4. clone 或 checkout `<install-root>/IsaacLab` 到固定 ref/commit。
5. 安装 recipe 中列出的 Isaac Lab editable 子包。
6. 恢复 recipe 钉住的 PyTorch 版本和 Sim pins。
7. 可选运行 import 和官方 headless 示例测试。
8. 写 `<install-root>/isaac-lab-install.record`。

远端执行模板

远端不一定已有本地 skill 脚本；必须先复制固定脚本，再执行。不要用 `cat <<EOF` 临时拼远端脚本。下面模板里的 `<install-root>` 使用用户确认的安装根目录；未显式指定时展开为 `/root/workspace`。

复制脚本并确认 Sim Python：

```bash
ssh <target> 'mkdir -p <install-root>/isaac-lab-install-skill/scripts <install-root>/isaac-lab-install-skill/logs'
scp -P <port> .claude/skills/isaac-lab-install/scripts/install_isaac_lab.sh <target>:<install-root>/isaac-lab-install-skill/scripts/
ssh <target> 'bash -n <install-root>/isaac-lab-install-skill/scripts/install_isaac_lab.sh && test -x <install-root>/env_isaacsim/bin/python'
```

启动后台安装：

```bash
ssh <target> 'nohup bash -lc '\''set -o pipefail; START=$(date +%s); echo $START > <install-root>/isaac-lab-install-skill/logs/lab.start; <install-root>/isaac-lab-install-skill/scripts/install_isaac_lab.sh --profile lab-3.0 --install-root <install-root> --sim-python <install-root>/env_isaacsim/bin/python --smoke-test > <install-root>/isaac-lab-install-skill/logs/lab.log 2>&1; STATUS=$?; END=$(date +%s); echo $STATUS > <install-root>/isaac-lab-install-skill/logs/lab.status; echo $END > <install-root>/isaac-lab-install-skill/logs/lab.end; exit $STATUS'\'' ><install-root>/isaac-lab-install-skill/logs/lab.nohup 2>&1 & echo $! > <install-root>/isaac-lab-install-skill/logs/lab.pid'
```

检查状态：

```bash
ssh <target> 'cat <install-root>/isaac-lab-install-skill/logs/lab.status 2>/dev/null || echo running; tail -n 80 <install-root>/isaac-lab-install-skill/logs/lab.log'
```

成功并验证通过后清理：

```bash
ssh <target> 'rm -rf <install-root>/isaac-lab-install-skill'
```

只删除过程目录；不能删除 `<install-root>/env_isaacsim`、`<install-root>/IsaacLab`、`<install-root>/isaac-lab-install.record`、Isaac Sim 文件、用户数据或挂载资产。

成功判定

不能只看脚本退出码。最终成功必须同时满足：

- 前台执行时脚本退出码为 `0`；远端后台执行时 `<install-root>/isaac-lab-install-skill/logs/lab.status` 为 `0`。
- `<install-root>/isaac-lab-install.record` 存在且字段完整。
- `<install-root>/env_isaacsim/bin/python` 可执行，且 `isaacsim` 仍可导入。
- `<install-root>/IsaacLab` 存在，且 repo ref/commit 符合 recipe。
- Python 能导入 `isaaclab`；启用 `--smoke-test` 时，官方 headless 示例通过。
- `<install-root>/IsaacLab/isaaclab.sh -p` 可作为 Lab 入口；验证时要显式设置 `VIRTUAL_ENV=<install-root>/env_isaacsim` 或清理抢占的 `CONDA_PREFIX`，避免入口误用当前 shell 的 conda Python。
- 有安装日志时，日志没有 `Traceback`、`ImportError`、`ModuleNotFoundError`、extension startup failure、pip resolver failure 或安装失败信号。
- `python -m pip check` 结果必须检查；如有 dependency conflict，报告为风险，不能声称环境完全干净。

短验证命令模板：

```bash
<install-root>/env_isaacsim/bin/python - <<'PY'
import importlib.metadata as metadata
import isaacsim
import isaaclab
print('isaacsim', metadata.version('isaacsim'))
print('isaaclab-import-ok')
PY
<install-root>/env_isaacsim/bin/python -m pip check
```

启用 smoke test 时额外验证：

```bash
OMNI_KIT_ACCEPT_EULA=YES OMNI_KIT_ALLOW_ROOT=1 timeout 180 \
  <install-root>/env_isaacsim/bin/python <install-root>/IsaacLab/scripts/environments/list_envs.py
```

远端后台安装时额外检查日志风险信号：

```bash
grep -Ein '(Traceback|ImportError|ModuleNotFoundError|ResolutionImpossible|dependency conflict|failed building|Could not install|No matching distribution|subprocess-exited-with-error|extension startup)'
<install-root>/isaac-lab-install-skill/logs/lab.log || true
```

平台快速开始适配

- `/isaac-sim/python.sh <script.py>` 通常由 `isaac-sim-install` 创建，内部执行 `<install-root>/env_isaacsim/bin/python <script.py>`。
- 安装 Isaac Lab 后，可用 `VIRTUAL_ENV=<install-root>/env_isaacsim <install-root>/IsaacLab/isaaclab.sh -p <script.py>`；如果当前 shell 有 `CONDA_PREFIX`，先清理或显式设置 `VIRTUAL_ENV`。
- `~/workspace/quick_start` 是平台样例目录，不是 recipe 自动创建目录；缺失时让用户提供样例脚本，或从已安装的 Isaac Lab 示例中选择替代脚本。
- `/mnt/bos/isaac-sim` 属于平台资产挂载；缺失时报资产挂载问题，而不是 Lab 安装入口问题。

扩展新版本

新增 Isaac Lab 版本时，必须先生成并审查固定 recipe，再改 `scripts/install_isaac_lab.sh`：

- 在 `normalize_profile()` 中添加 profile alias，例如 `lab-3.1`、`isaaclab-3.1`。
- 在 `recipe_lab()` 中添加 canonical `LAB_PROFILE` 和精确 `LAB_VERSION`。
- 固定 Isaac Lab repo、branch/tag 和 commit；不能只跟随 branch head。
- 从官方兼容信息确认期望 Isaac Sim major 或具体 Sim 线。
- 从官方要求确认 Python 版本。
- 固定 PyTorch 三件套和 index，不能只写宽松范围。
- 检查 `source/` 下 editable package dirs，不能默认沿用旧列表。
- 检查该 Lab 版本是否会拉偏 Sim 依赖；需要时添加对应 `SIM_PIN_PACKAGES`。

扩展后先跑：

```bash
bash scripts/install_isaac_lab.sh --profile <new-profile> --sim-python /root/workspace/env_isaacsim/bin/python --dry-run
bash -n scripts/install_isaac_lab.sh
```

`--dry-run` 必须只展示 recipe、路径、package dirs、pins、命令和 smoke-test 计划，不创建文件、不 clone、不安装包、不写 record、不创建 smoke 脚本。dry-run 不证明目标 Sim Python 和 runtime 可用，除非在目标机器上用真实路径单独验证。

最终输出要求

最终只输出有效安装记录，不输出大段日志：

- 目标机器。
- Lab profile。
- 安装根目录，以及是否使用用户确认的目录或默认 `/root/workspace`。
- 使用的 Isaac Sim Python 路径。
- 检测到的 Isaac Sim 版本，或版本检查警告。
- Isaac Lab repo、ref、commit、安装路径。
- editable package dirs。
- 是否恢复 PyTorch pins 和 Sim pins。
- `<install-root>/IsaacLab/isaaclab.sh -p` 是否可作为入口。
- 使用的脚本命令摘要。
- 安装脚本完成状态。
- 冒烟测试结果，或跳过原因。
- 验证结果；必须包含日志风险扫描和 `pip check` 结果。
- 远端过程目录清理结果；成功并验证通过后应已删除。
- 如果有 OS/driver 警告或依赖冲突，单独列为“风险”，并明确是否影响已执行的 smoke test。

Skill 测试报告要求

当用户要求“测试这个 skill”时，最终报告必须包含：

- 测试目标、目标机器、profile。
- 起止时间和每个阶段耗时。
- 实际执行的脚本路径和命令摘要。
- 成功项、失败项、远端日志位置。
- 失败原因判断：环境问题、网络/SSH 问题、recipe 问题或脚本问题。
- 已做的 skill 迭代，以及仍需用户或环境侧处理的事项。

install_isaac_lab.sh：

```
  **!/usr/bin/env bash**
set -euo pipefail

PROFILE="lab-3.0"
INSTALL_ROOT="/root/workspace"
VENV_PATH=""
SIM_PYTHON=""
PROXY_URL=""
DRY_RUN=0
INSTALL_SYSTEM_DEPS=1
APT_USE_PROXY=0
SMOKE_TEST=0
FORCE=0
PIP_FLAGS=(--progress-bar off --no-input)
SIM_PIN_PACKAGES=()
DETECTED_SIM_VERSION=""
export DEBIAN_FRONTEND=noninteractive

usage() {
  cat <<'USAGE'
Install Isaac Lab from a pinned recipe using an existing Isaac Sim Python.
```

```
Usage:
  install_isaac_lab.sh [options]

Recipe:
  --profile <name>          Recipe name: lab-3.0, lab-main, lab-2.3. Combined aliases are accepted for compatibility.
  --sim-python <path>       Python executable from an Isaac Sim install. Overrides --venv.
  --venv <path>             Existing Isaac Sim virtualenv. Default: <install-root>/env_isaacsim

Install location:
  --install-root <path>     Install root directory. Default: /root/workspace

Network and system deps:
  --proxy <url>             Temporary proxy for git/pip downloads only.
  --skip-system-deps        Do not install apt build dependencies.
  --apt-use-proxy           Let apt inherit --proxy. Default: apt runs without that proxy.

Execution:
  --smoke-test              Run import and minimal headless startup checks.
  --force                   Re-checkout recipe branch in existing IsaacLab repo.
  --dry-run                 Print commands without executing.
  --help                    Show this help.

Add a new Isaac Lab version by adding one branch in recipe_lab().
USAGE
}

log() { printf '[LAB] %s\n' "$*"; }
warn() { printf '[LAB][WARN] %s\n' "$*" >&2; }
die() { printf '[LAB][ERROR] %s\n' "$*" >&2; exit 1; }

supported_profiles() {
  printf 'lab-3.0, lab-main, lab-2.3'
}

normalize_profile() {
  local profile="$1"
  profile="$(printf '%s' "$profile" | tr '[:upper:]' '[:lower:]')"
  case "$profile" in
    lab-3.0|lab-3.0.0|lab-3.0.0-beta2|isaaclab-3.0|isaaclab-3.0.0|sim-6.0.0+lab-3.0|arena-0.2)
      printf 'lab-3.0'
      ;;
    lab-main|isaaclab-main|sim-6.0.0+lab-main)
      printf 'lab-main'
      ;;
    lab-2.3|lab-2.3.0|isaaclab-2.3|isaaclab-2.3.0|sim-5.1.0+lab-2.3|arena-0.1)
      printf 'lab-2.3'
      ;;
    lab-[0-9]*|isaaclab-[0-9]*)
      printf '%s' "$profile"
      ;;
    *)
      printf '%s' "$profile"
      ;;
  esac
}

run() {
  if [[ "$DRY_RUN" == "1" ]]; then
    printf '+ '
    printf '%q ' "$@"
    printf '\n'
  else
    "$@"
  fi
}

run_bash() {
  if [[ "$DRY_RUN" == "1" ]]; then
    printf '+ bash -lc %q\n' "$1"
  else
    bash -lc "$1"
  fi
}

recipe_lab() {
  REQUESTED_PROFILE="$1"
  NORMALIZED_PROFILE="$(normalize_profile "$REQUESTED_PROFILE")"

  case "$NORMALIZED_PROFILE" in
    lab-3.0)
      LAB_PROFILE="lab-3.0"
      LAB_VERSION="3.0.0-beta2"
      EXPECTED_SIM_MAJOR="6"
      REPO_URL="https://github.com/isaac-sim/IsaacLab.git"
      REF_KIND="branch"
      REF_NAME="release/3.0.0-beta2"
      REF_COMMIT="a8554968f604e54e4d0862c5922ed0f9b49aaa92"
      PYTHON_REQUIRED="3.12"
      TORCH_VERSION="2.11.0"
      TORCHVISION_VERSION="0.26.0"
      TORCHAUDIO_VERSION="2.11.0"
      PYTORCH_INDEX="https://download.pytorch.org/whl/cu128"
      SIM_PIN_PACKAGES=(
        "coverage==7.4.4"
        "numpy==2.3.1"
        "Pillow==12.1.1"
        "psutil==5.9.8"
        "packaging==26.0"
      )
      PACKAGE_DIRS=(
```

```
      source/isaaclab
      source/isaaclab_assets
      source/isaaclab_rl
      source/isaaclab_tasks
      source/isaaclab_mimic
    )
    ;;
  lab-main)
    LAB_PROFILE="lab-main"
    LAB_VERSION="main"
    EXPECTED_SIM_MAJOR="6"
    REPO_URL="https://github.com/isaac-sim/IsaacLab.git"
    REF_KIND="branch"
    REF_NAME="main"
    REF_COMMIT="f58361c8f1ea3380ded6ddfde029c40b0e7203ca"
    PYTHON_REQUIRED="3.12"
    TORCH_VERSION="2.11.0"
    TORCHVISION_VERSION="0.26.0"
    TORCHAUDIO_VERSION="2.11.0"
    PYTORCH_INDEX="https://download.pytorch.org/whl/cu128"
    SIM_PIN_PACKAGES=(
      "coverage==7.4.4"
      "numpy==2.3.1"
      "Pillow==12.1.1"
      "psutil==5.9.8"
      "packaging==26.0"
    )
    PACKAGE_DIRS=(
      source/isaaclab
      source/isaaclab_assets
      source/isaaclab_rl
      source/isaaclab_tasks
      source/isaaclab_mimic
    )
    ;;
  lab-2.3)
    LAB_PROFILE="lab-2.3"
    LAB_VERSION="2.3.0"
    EXPECTED_SIM_MAJOR="5"
    REPO_URL="https://github.com/isaac-sim/IsaacLab.git"
    REF_KIND="branch"
    REF_NAME="release/2.3.0"
    REF_COMMIT="5c2ec81cb17532d32f7922dd7fcaae40d123b71a"
    PYTHON_REQUIRED="3.11"
    TORCH_VERSION="2.7.0"
    TORCHVISION_VERSION="0.22.0"
    TORCHAUDIO_VERSION="2.7.0"
    PYTORCH_INDEX="https://download.pytorch.org/whl/cu128"
    PACKAGE_DIRS=(
      source/isaaclab
      source/isaaclab_assets
      source/isaaclab_rl
      source/isaaclab_tasks
    )
    ;;
  *)
    die "No pinned Isaac Lab recipe for '$REQUESTED_PROFILE' (normalized: '$NORMALIZED_PROFILE'). Supported profiles: $(supported_profiles). Add a recipe_lab() branch before installing this version."
    ;;
  esac
}

parse_args() {
  while [[ $# -gt 0 ]]; do
    case "$1" in
      --profile) PROFILE="$2"; shift 2 ;;
      --install-root) INSTALL_ROOT="$2"; shift 2 ;;
      --venv) VENV_PATH="$2"; shift 2 ;;
      --sim-python) SIM_PYTHON="$2"; shift 2 ;;
      --proxy) PROXY_URL="$2"; shift 2 ;;
      --skip-system-deps) INSTALL_SYSTEM_DEPS=0; shift ;;
      --apt-use-proxy) APT_USE_PROXY=1; shift ;;
      --smoke-test) SMOKE_TEST=1; shift ;;
      --force) FORCE=1; shift ;;
      --dry-run) DRY_RUN=1; shift ;;
      --help|-h) usage; exit 0 ;;
      *) die "Unknown argument: $1" ;;
    esac
  done
}

apply_proxy() {
  if [[ -n "$PROXY_URL" ]]; then
    export http_proxy="$PROXY_URL" https_proxy="$PROXY_URL" HTTP_PROXY="$PROXY_URL" HTTPS_PROXY="$PROXY_URL"
  fi
}

resolve_python() {
  if [[ -n "$SIM_PYTHON" ]]; then
    if [[ "$DRY_RUN" != "1" ]]; then
      [[ -x "$SIM_PYTHON" ]] || die "Isaac Sim Python is not executable: $SIM_PYTHON"
    fi
    return
  fi

  if [[ -z "$VENV_PATH" ]]; then
    VENV_PATH="${INSTALL_ROOT%/}/env_isaacsim"
  fi
  SIM_PYTHON="${VENV_PATH%/}/bin/python"
  if [[ "$DRY_RUN" != "1" ]]; then
    [[ -x "$SIM_PYTHON" ]] || die "Isaac Sim Python not found: ${SIM_PYTHON}. Run install_isaac_sim.sh first or pass --sim-python."
```

```
  [[ -x "$SIM_PYTHON" ]] || die "Isaac Sim Python not found: ${SIM_PYTHON}. Run install_isaac_sim.sh first or pass --sim-python."
  fi
}

check_python() {
  [[ "$DRY_RUN" == "1" ]] && return 0
  local actual
  actual="$($SIM_PYTHON - <<'PY'
import sys
print(f"{sys.version_info.major}.{sys.version_info.minor}")
PY
)"
  [[ "$actual" == "$PYTHON_REQUIRED" ]] || die "Profile ${LAB_PROFILE} requires Python ${PYTHON_REQUIRED}, got ${actual} from ${SIM_PYTHON}."
}

check_isaac_sim() {
  [[ "$DRY_RUN" == "1" ]] && return 0
  DETECTED_SIM_VERSION="$($SIM_PYTHON -m pip show isaacsim 2>/dev/null | awk '/^Version:/{print $2}')"
  [[ -n "$DETECTED_SIM_VERSION" ]] || die "isaacsim is not installed in ${SIM_PYTHON}."
  if [[ "$DETECTED_SIM_VERSION" != ${EXPECTED_SIM_MAJOR}.* ]]; then
    warn "Isaac Sim ${DETECTED_SIM_VERSION} may not match ${LAB_PROFILE}, expected major ${EXPECTED_SIM_MAJOR}."
  fi
}

install_system_deps() {
  [[ "$INSTALL_SYSTEM_DEPS" == "1" ]] || return 0

  local apt_prefix="env -u http_proxy -u https_proxy -u HTTP_PROXY -u HTTPS_PROXY"
  [[ "$APT_USE_PROXY" == "1" ]] && apt_prefix="env"

  if [[ "$DRY_RUN" == "1" ]]; then
    run_bash "$apt_prefix apt-get update"
    run_bash "$apt_prefix apt-get install -y git cmake build-essential"
    return 0
  fi

  if [[ "$(id -u)" != "0" ]]; then
    warn "Not root; skipping apt build dependencies."
    return 0
  fi

  run_bash "$apt_prefix apt-get update"
  run_bash "$apt_prefix apt-get install -y git cmake build-essential"
}

checkout_repo() {
  LAB_DIR="${INSTALL_ROOT%/}/IsaacLab"
  if [[ ! -d "$LAB_DIR/.git" ]]; then
    run git clone --branch "$REF_NAME" "$REPO_URL" "$LAB_DIR"
  else
    run git -C "$LAB_DIR" fetch origin "$REF_NAME"
    if [[ "$FORCE" == "1" ]]; then
      run git -C "$LAB_DIR" checkout -f "$REF_NAME"
    else
      run git -C "$LAB_DIR" checkout "$REF_NAME"
    fi
  fi

  if [[ -n "$REF_COMMIT" ]]; then
    run git -C "$LAB_DIR" checkout "$REF_COMMIT"
  fi
}

install_packages() {
  run "$SIM_PYTHON" -m pip install "${PIP_FLAGS[@]}" --index-url https://pypi.org/simple wheel toml "setuptools<82.0.0"
  for package_dir in "${PACKAGE_DIRS[@]}"; do
    if [[ "$DRY_RUN" != "1" ]]; then
      [[ -d "$LAB_DIR/$package_dir" ]] || die "Recipe package path missing: $LAB_DIR/$package_dir"
    fi
    run "$SIM_PYTHON" -m pip install "${PIP_FLAGS[@]}" --index-url https://pypi.org/simple --extra-index-url https://pypi.nvidia.com --find-links https://py.mujoco.org/ --no-build-isolation --editable
"$LAB_DIR/$package_dir"
  done
  run "$SIM_PYTHON" -m pip install "${PIP_FLAGS[@]}" "torch==${TORCH_VERSION}" "torchvision==${TORCHVISION_VERSION}" "torchaudio==${TORCHAUDIO_VERSION}" --index-url "$PYTORCH_INDEX"
}

restore_sim_pins() {
  [[ "${#SIM_PIN_PACKAGES[@]}" -gt 0 ]] || return 0
  run "$SIM_PYTHON" -m pip install "${PIP_FLAGS[@]}" --index-url https://pypi.org/simple "${SIM_PIN_PACKAGES[@]}"
}

print_recipe_summary() {
  log "Recipe summary: requested=${REQUESTED_PROFILE}, normalized=${NORMALIZED_PROFILE}, profile=${LAB_PROFILE}, lab_version=${LAB_VERSION}."
  log "Isaac Lab source: ${REPO_URL} ${REF_KIND}:${REF_NAME} commit=${REF_COMMIT}."
  log "Expected Isaac Sim major: ${EXPECTED_SIM_MAJOR}; required Python: ${PYTHON_REQUIRED}."
  log "PyTorch pins: torch=${TORCH_VERSION}, torchvision=${TORCHVISION_VERSION}, torchaudio=${TORCHAUDIO_VERSION}, index=${PYTORCH_INDEX}."
  log "Install root: ${INSTALL_ROOT}; Lab path: ${INSTALL_ROOT%/}/IsaacLab; Sim Python: ${SIM_PYTHON}."
  log "Editable package dirs: ${PACKAGE_DIRS[*]}."
  if [[ "${#SIM_PIN_PACKAGES[@]}" -gt 0 ]]; then
    log "Sim pins to restore: ${SIM_PIN_PACKAGES[*]}."
  else
    log "Sim pins to restore: none."
  fi
  if [[ "$SMOKE_TEST" == "1" ]]; then
    log "Smoke test: enabled."
  else
    log "Smoke test: disabled."
  fi
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Dry-run skips real Sim Python, isaacsim package, and runtime validation."
  fi
```

```
}

smoke_test() {
  [[ "$SMOKE_TEST" == "1" ]] || return 0
  run env OMNI_KIT_ACCEPT_EULA=YES OMNI_KIT_ALLOW_ROOT=1 "$SIM_PYTHON" - <<'PY'
import isaaclab
print("ISAAC_LAB_IMPORT_OK")
PY

  local smoke_script
  smoke_script="${LAB_DIR}/scripts/environments/list_envs.py"
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Would run official Isaac Lab smoke test ${smoke_script}."
    return 0
  fi
  [[ -f "$smoke_script" ]] || die "Official smoke test script not found: $smoke_script"

  OMNI_KIT_ACCEPT_EULA=YES OMNI_KIT_ALLOW_ROOT=1 timeout 180 "$SIM_PYTHON" "$smoke_script"
}

write_record() {
  local record_path="$INSTALL_ROOT/isaac-lab-install.record"
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Would write install record to ${record_path}"
    return
  fi
  cat > "$record_path" <<EOF
component=isaac-lab
profile=${LAB_PROFILE}
lab_version=${LAB_VERSION}
expected_sim_major=${EXPECTED_SIM_MAJOR}
detected_sim_version=${DETECTED_SIM_VERSION}
repo=${REPO_URL}
ref_kind=${REF_KIND}
ref_name=${REF_NAME}
ref_commit=${REF_COMMIT}
path=${LAB_DIR}
python=${SIM_PYTHON}
torch=${TORCH_VERSION}
torchvision=${TORCHVISION_VERSION}
torchaudio=${TORCHAUDIO_VERSION}
pytorch_index=${PYTORCH_INDEX}
packages=${PACKAGE_DIRS[*]}
sim_pin_packages=${SIM_PIN_PACKAGES[*]}
EOF
}

main() {
  parse_args "$@"
  recipe_lab "$PROFILE"
  apply_proxy
  resolve_python
  check_python
  check_isaac_sim
  print_recipe_summary
  install_system_deps
  run mkdir -p "$INSTALL_ROOT"
  checkout_repo
  install_packages
  restore_sim_pins
  smoke_test
  write_record
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Dry run complete: no filesystem or package changes were made."
    exit 0
  fi
  log "Isaac Lab installed: ${LAB_DIR}"
}

main "$@"
```

isaac-sim-install

你可以将以下代码以如下形式放到您的 agent 目录中，即可通过/isaac-sim-install + 参数 的形式调用该 Skill。

```
.claude/skills/<install-skill>/
├── SKILL.md
└── scripts/
    └── install_isaac_sim.sh
```

SKILL.md：

```
---
name: isaac-sim-install
description: >
  用内置、固定 recipe 安装 Isaac Sim。支持本机或 SSH 远端，安装前确认代理、安装根目录和运行方式；
  只安装 Isaac Sim，不安装 Isaac Lab。适合先准备 Isaac Sim 基础环境或基础镜像。
argument-hint: [local|ssh <ssh-target>] [--profile sim-6.0.0|sim-5.1.0|sim-4.5.0] [--install-root /root/workspace] [--proxy <url>|--no-proxy] [--run-mode wait|background|schedule-check]
disable-model-invocation: true
allowed-tools: Read, Bash, AskUserQuestion, CronCreate
---

**Isaac Sim 安装 Skill**

你是 Isaac Sim 环境安装工程师。这个 skill 只用内置、固定 recipe 安装 Isaac Sim；没有 recipe 的版本不能自动安装，也不能现场查文档拼临时命令。

**使用边界**
```

- 只安装 Isaac Sim；不安装、不补装 Isaac Lab。
- 安装逻辑固定在 `scripts/install_isaac_sim.sh`，通过 `--profile` 选择 recipe。
- 支持本机和 SSH 远端；参数以 `ssh ...` 开头时，所有检查和安装都在远端执行。
- 代理只临时用于 git/pip 下载；不要写入 shell 配置、安装记录或仓库文件。
- 默认安装根目录是 `/root/workspace`，但交互流程必须向用户确认。
- 成功且验证通过后，远端过程目录 `<install-root>/isaac-sim-install-skill` 必须删除。

内置 Recipe

profile	Isaac Sim recipe	Python	PyTorch	repo ref	用途
`sim-6.0.0`	pip `isaacsim[all,extscache]==6.0.0.1`	`3.12`	`2.11.0`	`v6.0.0`	默认推荐
`sim-5.1.0`	pip `isaacsim[all,extscache]==5.1.0`	`3.11`	`2.7.0`	`v5.1.0`	兼容 Lab 2.3 线
`sim-4.5.0`	pip `isaacsim[all,extscache]==4.5.0.0`	`3.10`	`2.5.1`	无固定 repo tag，pip-only	旧版兼容线

脚本仍接受旧组合 alias，例如 `sim-6.0.0+lab-3.0`、`sim-5.1.0+lab-2.3`，但新调用优先使用 Sim-only profile。

如果用户要求未列出的版本，例如 `sim-7.0`，先进入 recipe 扩展流程：查询官方 pip 包版本、Python 约束、PyTorch/CUDA 组合、系统/driver 要求和官方 `IsaacSim` repo tag；生成候选 `recipe_sim()` 分支后，先运行 `bash -n` 和 `--dry-run`，向用户说明来源和缺口，得到确认后才写入脚本并允许安装。若官方 repo 没有对应 tag，可以明确做成 pip-only recipe，不能伪造 repo ref。

环境约束

这些约束是安装前预检和最终风险说明的来源，不用于现场改 recipe：

- Python 版本严格跟随 recipe；不能用系统默认 Python 代替不匹配的小版本。
- GPU/driver 通过 `nvidia-smi` 检查；缺失或低于 recipe 测试版本时只警告，不自动换 profile。
- Ubuntu 版本必须和 recipe 的测试集合对照；不在集合内时只报告风险。
- 安装默认使用独立 venv `<install-root>/env_isaacsim`，不要装进系统 Python。
- Isaac Sim wheel 和扩展缓存包体积很大；远端安装优先后台运行并写日志、PID、状态文件。
- root 安装会创建 `/isaac-sim/python.sh` 文档兼容入口；非 root 安装只保证 venv Python 可用。

安装前决策

用户调用示例：

```text
/isaac-sim-install ssh -p <port> <user>@<host> --profile sim-6.0.0 --proxy http://<proxy-host>:<proxy-port> --run-mode wait
```

参数和交互规则：

1. 目标：参数以 `ssh` 开头时，把后续 SSH 参数保存为远端命令前缀；没有 `ssh` 或参数是 `local` 时，在本机执行。
2. 目录：`--install-root` 默认 `/root/workspace`；用户未显式传入时必须确认默认目录是否合适，或让用户用 Other 输入已有/挂载目录。
3. 代理：`--proxy <url>` 直接传给脚本；`--no-proxy` 只表示用户选择直连，脚本不接收该参数。
4. 运行方式：`--run-mode wait` 持续追踪到成功/失败；`background` 只启动后台任务并返回日志/PID；`schedule-check` 启动后台任务并设置本会话定时检查。
5. 版本：`--profile` 默认 `sim-6.0.0`；如果脚本输出 driver/OS 警告，不要自动换版本。
6. Lab：用户要求"一起装 Lab"时，引导两步执行：先装 Sim，再用 `isaac-lab-install` 复用 `<install-root>/env_isaacsim/bin/python`。
7. 禁用参数：不支持 `--component`；这个 skill 永远只装 Sim。

只有用户显式给全目标、profile、proxy/no-proxy、run-mode 后，才可以跳过交互直接安装。用户只给 SSH 目标或只确认 host key 时，不要自动开始安装。

交互顺序

一次交互最多问 1–4 个问题；远端首次连接可能需要 host key 信任时，把它合并进安装配置问题。

1. 目标和 host key：装本机还是远端；首次 SSH host key 是否信任。信任 host key 只表示允许连接，不等于允许开始安装。
2. 安装目录：默认 `/root/workspace`，也可以用 Other 输入已有目录、BOS 挂载或平台预置目录。
3. 网络代理：默认不假设用户有内网或预置代理。选项包含"不使用代理（直连）""我已有代理 URL（选 Other 填写）""不确定，先检测直连"。
4. 运行方式：`wait`、`background`、`schedule-check`。
5. Sim 版本：默认推荐 `sim-6.0.0`；旧线可选 `sim-5.1.0` 或 `sim-4.5.0`。

安装脚本

```bash
bash .claude/skills/isaac-sim-install/scripts/install_isaac_sim.sh \
  --profile sim-6.0.0 \
  --install-root /root/workspace \
  --proxy http://host:port
```

脚本职责：

1. 读取 `recipe_sim()` 中的固定版本。
2. 检查 Ubuntu、Python、NVIDIA driver；不因警告擅自改 recipe。
3. 按权限和参数安装或跳过 apt 运行库。
4. 创建 `<install-root>/env_isaacsim`。
5. 安装固定版本的 PyTorch 和 Isaac Sim pip 包。
6. clone 或 checkout 官方 `https://github.com/isaac-sim/IsaacSim.git` 到 `<install-root>/IsaacSim`。
7. root 安装时创建 `/isaac-sim/python.sh`，映射到 `<install-root>/env_isaacsim/bin/python`。
8. 写 `<install-root>/isaac-sim-install.record`。

远端执行模板

远端不一定已有本地 skill 脚本；必须先复制固定脚本，再执行。不要用 `cat <<EOF` 临时拼远端脚本。下面模板里的 `<install-root>` 使用用户确认的安装根目录；未显式指定时展开为 `/root/workspace`。

复制脚本：

```bash
ssh <target> 'mkdir -p <install-root>/isaac-sim-install-skill/scripts <install-root>/isaac-sim-install-skill/logs'
scp -P <port> .claude/skills/isaac-sim-install/scripts/install_isaac_sim.sh <target>:<install-root>/isaac-sim-install-skill/scripts/
ssh <target> 'bash -n <install-root>/isaac-sim-install-skill/scripts/install_isaac_sim.sh'
```

启动后台安装：

```bash
ssh <target> 'nohup bash -lc '\''set -o pipefail; START=$(date +%s); echo $START > <install-root>/isaac-sim-install-skill/logs/sim.start; <install-root>/isaac-sim-install-skill/scripts/install_isaac_sim.sh --profile sim-6.0.0 --install-root <install-root> > <install-root>/isaac-sim-install-skill/logs/sim.log 2>&1; STATUS=$?; END=$(date +%s); echo $STATUS > <install-root>/isaac-sim-install-skill/logs/sim.status; echo $END > <install-root>/isaac-sim-install-skill/logs/sim.end; exit $STATUS'\'' ><install-root>/isaac-sim-install-skill/logs/sim.nohup 2>&1 & echo $! > <install-root>/isaac-sim-install-skill/logs/sim.pid'
```

检查状态：

```bash
ssh <target> 'cat <install-root>/isaac-sim-install-skill/logs/sim.status 2>/dev/null || echo running; tail -n 80 <install-root>/isaac-sim-install-skill/logs/sim.log'
```

成功并验证通过后清理：

```bash
ssh <target> 'rm -rf <install-root>/isaac-sim-install-skill'
```

只删除过程目录；不能删除 `<install-root>/env_isaacsim`、`<install-root>/IsaacSim`、`<install-root>/isaac-sim-install.record` 或 `/isaac-sim/python.sh`。

成功判定

不能只看脚本退出码。最终成功必须同时满足：

- 前台执行时脚本退出码为 `0`；远端后台执行时 `<install-root>/isaac-sim-install-skill/logs/sim.status` 为 `0`。
- `<install-root>/isaac-sim-install.record` 存在且字段完整。
- `<install-root>/env_isaacsim/bin/python` 可执行。
- root 安装时 `/isaac-sim/python.sh` 可执行，并映射到 venv Python；非 root 安装时明确说明兼容入口被跳过，并验证 venv Python。
- Python 能导入并读取 `isaacsim` 和 `torch` 版本。
- `python -m pip check` 无 broken requirements。
- 有安装日志时，日志没有 `Traceback`、`ImportError`、`ModuleNotFoundError`、依赖冲突或安装失败信号。

短验证命令模板：

```bash
<install-root>/env_isaacsim/bin/python - <<'PY'
import importlib.metadata as metadata
import torch
print('isaacsim', metadata.version('isaacsim'))
print('torch', torch.__version__)
PY
<install-root>/env_isaacsim/bin/python -m pip check
```

root 安装时额外验证兼容入口：

```bash
/isaac-sim/python.sh - <<'PY'
print('entrypoint-ok')
PY
```

远端后台安装时额外检查日志风险信号：

```bash
grep -Ein '(Traceback|ImportError|ModuleNotFoundError|ResolutionImpossible|dependency conflict|failed building|Could not install|No matching distribution|subprocess-exited-with-error)' <install-root>/isaac-sim-install-skill/logs/sim.log || true
```

平台快速开始适配

如果用户给出平台文档中的 `/isaac-sim/python.sh` 或 `~/workspace/quick_start` 示例，不要因为路径不存在就判定无法执行。先按实际安装记录适配：

- 官方 Isaac Sim 项目位于 `<install-root>/IsaacSim`，快速开始脚本优先从 `<install-root>/IsaacSim/source/standalone_examples/tutorials/getting_started/` 取用。
- `/isaac-sim/python.sh <script.py>` 由安装脚本创建，内部执行 `<install-root>/env_isaacsim/bin/python <script.py>`。
- pip 安装的 Isaac Sim CLI 是 `<install-root>/env_isaacsim/bin/isaacsim`。
- `/isaac-sim/python.sh` 默认设置 `OMNI_KIT_ACCEPT_EULA=YES` 和 `OMNI_KIT_ALLOW_ROOT=1`。
- `~/workspace/quick_start` 是平台样例目录，不是 recipe 自动创建目录；缺失时让用户提供样例脚本，或从已安装的 Isaac Sim 示例中选择替代脚本。
- `/mnt/bos/isaac-sim` 属于平台资产挂载；缺失时报资产挂载问题，而不是安装入口问题。

扩展新版本

新增 Isaac Sim 版本时，必须先生成并审查固定 recipe，再改 `scripts/install_isaac_sim.sh`：

- 在 `normalize_profile()` 中添加 profile alias。
- 在 `recipe_sim()` 中添加 `set_recipe` 分支。
- 从官方包元数据确认 `PYTHON_REQUIRED`。
- 固定 PyTorch 三件套和 index，不能只写宽松范围。
- 从官方系统要求确认支持 Ubuntu 和测试 driver。
- 从官方仓库确认 `ISAACSIM_REPO_REF`；没有对应 tag 时设为空并声明 pip-only。

扩展后先跑：

```bash
bash scripts/install_isaac_sim.sh --profile <new-profile> --dry-run
bash -n scripts/install_isaac_sim.sh
```

最终输出要求

最终只输出有效安装记录，不输出大段日志：

- 目标机器。
- profile。
- 安装根目录，以及是否使用用户确认的目录或默认 `/root/workspace`。
- Isaac Sim 版本、venv 路径、Python 路径、官方项目 repo/ref/path。
- 平台文档入口适配：`/isaac-sim/python.sh` 是否已创建，以及实际 Python 路径。
- 使用的脚本命令摘要。
- 安装脚本完成状态。
- 验证结果；必须包含日志风险扫描结果。
- 远端过程目录清理结果；成功并验证通过后应已删除。
- 如果有 OS/driver 警告或依赖冲突，单独列为“风险”。

Skill 测试报告要求

当用户要求"测试这个 skill"时，最终报告必须包含：

- 测试目标、目标机器、profile。
- 起止时间和每个阶段耗时。
- 实际执行的脚本路径和命令摘要。
- 成功项、失败项、远端日志位置。
- 失败原因判断：环境问题、网络/SSH 问题、recipe 问题或脚本问题。
- 已做的 skill 迭代，以及仍需用户或环境侧处理的事项。

install_isaac_sim.sh：

```
 **!/usr/bin/env bash**
set -euo pipefail

PROFILE="sim-6.0.0"
INSTALL_ROOT="/root/workspace"
VENV_NAME="env_isaacsim"
PYTHON_BIN=""
PROXY_URL=""
DRY_RUN=0
INSTALL_SYSTEM_DEPS=1
APT_USE_PROXY=0
ACCEPT_EULA=1
PIP_FLAGS=(--progress-bar off --no-input)
ISAACSIM_REPO_BASE_URL="https://github.com/isaac-sim/IsaacSim.git"
export DEBIAN_FRONTEND=noninteractive

usage() {
  cat <<'USAGE'
Install Isaac Sim from a pinned recipe.

Usage:
  install_isaac_sim.sh [options]

Recipe:
  --profile <name>          Recipe name: sim-6.0.0, sim-5.1.0, sim-4.5.0. Combined aliases are accepted for compatibility.
  --python <path>           Python executable to create the venv. Defaults to recipe Python.

Install location:
  --install-root <path>     Install root directory. Default: /root/workspace
  --venv-name <name>        Virtualenv directory name under install root. Default: env_isaacsim

Network and system deps:
  --proxy <url>             Temporary proxy for git/pip downloads only. Apt uses it only with --apt-use-proxy.
  --skip-system-deps        Do not install apt runtime libraries.
  --apt-use-proxy           Let apt inherit --proxy. Default: apt runs without that proxy.

Execution:
  --dry-run                 Print commands without executing.
  --help                    Show this help.

Add a new Isaac Sim version by adding one branch in recipe_sim().
USAGE
}

log() { printf '[SIM] %s\n' "$*"; }
warn() { printf '[SIM][WARN] %s\n' "$*" >&2; }
die() { printf '[SIM][ERROR] %s\n' "$*" >&2; exit 1; }

supported_profiles() {
  printf 'sim-6.0.0, sim-5.1.0, sim-4.5.0'
}

normalize_profile() {
  local profile="$1"
  profile="$(printf '%s' "$profile" | tr '[:upper:]' '[:lower:]')"
  case "$profile" in
    sim6.0|sim-6.0|sim6.0.0|sim-6.0.0|isaacsim6.0.0|isaacsim-6.0.0|sim-6.0.0+lab-3.0|sim-6.0.0+lab-main|arena-0.2)
      printf 'sim-6.0.0'
      ;;
    sim5.1|sim-5.1|sim5.1.0|sim-5.1.0|isaacsim5.1.0|isaacsim-5.1.0|sim-5.1.0+lab-2.3|arena-0.1)
      printf 'sim-5.1.0'
      ;;
    sim4.5|sim-4.5|sim4.5.0|sim-4.5.0|isaacsim4.5.0|isaacsim-4.5.0)
      printf 'sim-4.5.0'
      ;;
    sim[0-9]*|sim-[0-9]*|isaacsim[0-9]*|isaacsim-[0-9]*)
      printf '%s' "$profile"
      ;;
    *)
      printf '%s' "$profile"
      ;;
  esac
}

run() {
  if [[ "$DRY_RUN" == "1" ]]; then
    printf '+ '
    printf '%q ' "$@"
    printf '\n'
  else
    "$@"
  fi
}

run_bash() {
  if [[ "$DRY_RUN" == "1" ]]; then
    printf '+ bash -lc %q\n' "$1"
  else
```

```
  else
    bash -lc "$1"
  fi
}

set_recipe() {
  SIM_PROFILE="$1"
  SIM_VERSION="$2"
  ISAACSIM_PIP_VERSION="$3"
  PYTHON_REQUIRED="$4"
  TORCH_VERSION="$5"
  TORCHVISION_VERSION="$6"
  TORCHAUDIO_VERSION="$7"
  PYTORCH_INDEX="$8"
  ISAACSIM_EXTRA="$9"
  ISAACSIM_REPO_REF="${10}"
  SUPPORTED_UBUNTU="${11}"
  TESTED_DRIVER="${12}"
}

recipe_sim() {
  local requested_profile="$1"
  local normalized_profile
  normalized_profile="$(normalize_profile "$requested_profile")"

  case "$normalized_profile" in
    sim-6.0.0)
      set_recipe sim-6.0.0 6.0.0 6.0.0.1 3.12 2.11.0 0.26.0 2.11.0 https://download.pytorch.org/whl/cu128 all,extscache v6.0.0 "22.04 24.04" 580.95.05
      ;;
    sim-5.1.0)
      set_recipe sim-5.1.0 5.1.0 5.1.0 3.11 2.7.0 0.22.0 2.7.0 https://download.pytorch.org/whl/cu128 all,extscache v5.1.0 "22.04 24.04" 580.65.06
      ;;
    sim-4.5.0)
      set_recipe sim-4.5.0 4.5.0 4.5.0.0 3.10 2.5.1 0.20.1 2.5.1 https://download.pytorch.org/whl/cu124 all,extscache "" "20.04 22.04" 535.129.03
      ;;
    *)
      die "No pinned Isaac Sim recipe for '$requested_profile' (normalized: '$normalized_profile'). Supported profiles: $(supported_profiles). Add a recipe_sim() branch before installing this version."
      ;;
  esac
}

version_ge() {
  [[ "$(printf '%s\n%s\n' "$2" "$1" | sort -V | head -n1)" == "$2" ]]
}

parse_args() {
  while [[ $# -gt 0 ]]; do
    case "$1" in
      --profile) PROFILE="$2"; shift 2 ;;
      --install-root) INSTALL_ROOT="$2"; shift 2 ;;
      --venv-name) VENV_NAME="$2"; shift 2 ;;
      --python) PYTHON_BIN="$2"; shift 2 ;;
      --proxy) PROXY_URL="$2"; shift 2 ;;
      --skip-system-deps) INSTALL_SYSTEM_DEPS=0; shift ;;
      --apt-use-proxy) APT_USE_PROXY=1; shift ;;
      --dry-run) DRY_RUN=1; shift ;;
      --help|-h) usage; exit 0 ;;
      *) die "Unknown argument: $1" ;;
    esac
  done
}

apply_proxy() {
  if [[ -n "$PROXY_URL" ]]; then
    export http_proxy="$PROXY_URL" https_proxy="$PROXY_URL" HTTP_PROXY="$PROXY_URL" HTTPS_PROXY="$PROXY_URL"
  fi
}

select_python() {
  if [[ -n "$PYTHON_BIN" ]]; then
    if [[ "$DRY_RUN" != "1" ]]; then
      [[ -x "$PYTHON_BIN" ]] || die "Python executable is not executable: $PYTHON_BIN"
    fi
    SELECTED_PYTHON="$PYTHON_BIN"
    return
  fi

  if [[ "$DRY_RUN" == "1" ]]; then
    SELECTED_PYTHON="python${PYTHON_REQUIRED}"
    return
  fi

  if command -v "python${PYTHON_REQUIRED}" >/dev/null 2>&1; then
    SELECTED_PYTHON="python${PYTHON_REQUIRED}"
  elif command -v python3 >/dev/null 2>&1; then
    SELECTED_PYTHON="python3"
  else
    die "No Python found. Install Python ${PYTHON_REQUIRED} for profile ${SIM_PROFILE}."
  fi

  local actual
  actual="$($SELECTED_PYTHON - <<'PY'
import sys
print(f"{sys.version_info.major}.{sys.version_info.minor}")
PY
)"
  [[ "$actual" == "$PYTHON_REQUIRED" ]] || die "Profile ${SIM_PROFILE} requires Python ${PYTHON_REQUIRED}, got ${actual} from ${SELECTED_PYTHON}."
}

check_host() {
```

```
  if [[ -r /etc/os-release ]]; then
    . /etc/os-release
    local version_id="${VERSION_ID:-unknown}"
    if [[ " $SUPPORTED_UBUNTU " != *" $version_id "* ]]; then
      warn "Ubuntu ${version_id} is not in the tested set for ${SIM_PROFILE}: ${SUPPORTED_UBUNTU}."
    fi
  fi

  if command -v nvidia-smi >/dev/null 2>&1; then
    local driver
    driver="$(nvidia-smi --query-gpu=driver_version --format=csv,noheader | head -n1 | tr -d ' ')"
    if ! version_ge "$driver" "$TESTED_DRIVER"; then
      warn "Driver ${driver} is below recipe tested driver ${TESTED_DRIVER}; install can proceed but runtime may fail."
    fi
  else
    warn "nvidia-smi not found; GPU/driver compatibility was not verified."
  fi
}

install_system_deps() {
  [[ "$INSTALL_SYSTEM_DEPS" == "1" ]] || return 0
  if [[ "$(id -u)" != "0" ]]; then
    warn "Not root; skipping apt runtime libraries. Re-run as root or install them manually if Isaac Sim cannot start."
    return 0
  fi

  local apt_prefix="env -u http_proxy -u https_proxy -u HTTP_PROXY -u HTTPS_PROXY"
  [[ "$APT_USE_PROXY" == "1" ]] && apt_prefix="env"

  run_bash "$apt_prefix apt-get update"
  run_bash "$apt_prefix apt-get install -y libgl1 libglu1-mesa libxt6 libxrender1 libxrandr2 libxinerama1 libxcursor1 libxi6 libxkbcommon-x11-0 libsm6 libvulkan1"
}

create_venv() {
  run mkdir -p "$INSTALL_ROOT"
  if [[ ! -d "$VENV_PATH" ]]; then
    run "$SELECTED_PYTHON" -m venv "$VENV_PATH"
  fi
}

python_in_venv() {
  run "${VENV_PATH}/bin/python" "$@"
}

accept_eula_value() {
  if [[ "$ACCEPT_EULA" == "1" ]]; then
    printf 'YES'
  else
    printf 'NO'
  fi
}

upgrade_pip() {
  python_in_venv -m pip install "${PIP_FLAGS[@]}" --upgrade pip
}

install_pytorch() {
  python_in_venv -m pip install "${PIP_FLAGS[@]}" \
    "torch==${TORCH_VERSION}" \
    "torchvision==${TORCHVISION_VERSION}" \
    "torchaudio==${TORCHAUDIO_VERSION}" \
    --index-url "$PYTORCH_INDEX"
}

install_isaacsim_packages() {
  run env OMNI_KIT_ACCEPT_EULA="$(accept_eula_value)" \
    "${VENV_PATH}/bin/python" -m pip install "${PIP_FLAGS[@]}" \
    "isaacsim[${ISAACSIM_EXTRA}]==${ISAACSIM_PIP_VERSION}" \
    --index-url https://pypi.org/simple \
    --extra-index-url https://pypi.nvidia.com
}

verify_isaacsim_package() {
  python_in_venv -m pip show isaacsim >/dev/null
}

install_python_packages() {
  log "Installing Python packages into ${VENV_PATH}."
  upgrade_pip
  install_pytorch
  install_isaacsim_packages
  [[ "$DRY_RUN" == "1" ]] || verify_isaacsim_package
}

checkout_isaacsim_repo() {
  ISAACSIM_REPO_URL="$ISAACSIM_REPO_BASE_URL"
  ISAACSIM_REPO_DIR="${INSTALL_ROOT%/}/IsaacSim"
  if [[ -z "$ISAACSIM_REPO_REF" ]]; then
    ISAACSIM_REPO_URL=""
    ISAACSIM_REPO_DIR=""
    log "No official IsaacSim repo ref is pinned for ${SIM_PROFILE}; using pip packages only."
    return
  fi
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Would checkout official IsaacSim repo ${ISAACSIM_REPO_REF} into ${ISAACSIM_REPO_DIR}."
    return
  fi
  if [[ ! -d "$ISAACSIM_REPO_DIR/.git" ]]; then
    run git clone --branch "$ISAACSIM_REPO_REF" "$ISAACSIM_REPO_URL" "$ISAACSIM_REPO_DIR"
  else
```

```
    run git -C "$ISAACSIM_REPO_DIR" fetch origin --tags
    run git -C "$ISAACSIM_REPO_DIR" checkout "$ISAACSIM_REPO_REF"
  fi
}

install_compat_layout() {
  local compat_dir="/isaac-sim"
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Would create compatibility entrypoint ${compat_dir}/python.sh"
    return
  fi
  if [[ "$(id -u)" != "0" ]]; then
    warn "Not root; skipping ${compat_dir}/python.sh compatibility entrypoint. Use ${VENV_PATH}/bin/python instead."
    return
  fi
  mkdir -p "$compat_dir"
  cat > "${compat_dir}/python.sh" <<EOF
**!/usr/bin/env bash**
set -euo pipefail
export OMNI_KIT_ACCEPT_EULA=\${OMNI_KIT_ACCEPT_EULA:-YES}
export OMNI_KIT_ALLOW_ROOT=\${OMNI_KIT_ALLOW_ROOT:-1}
exec "${VENV_PATH}/bin/python" "\$@"
EOF
  chmod +x "${compat_dir}/python.sh"
}

write_record() {
  local record_path="$INSTALL_ROOT/isaac-sim-install.record"
  if [[ "$DRY_RUN" == "1" ]]; then
    log "Would write install record to ${record_path}"
    return
  fi
  cat > "$record_path" <<EOF
component=isaac-sim
profile=${SIM_PROFILE}
sim_version=${SIM_VERSION}
isaacsim_pip_version=${ISAACSIM_PIP_VERSION}
python=${VENV_PATH}/bin/python
venv=${VENV_PATH}
repo=${ISAACSIM_REPO_URL}
repo_ref=${ISAACSIM_REPO_REF}
repo_path=${ISAACSIM_REPO_DIR}
compat_python=/isaac-sim/python.sh
torch=${TORCH_VERSION}
torchvision=${TORCHVISION_VERSION}
torchaudio=${TORCHAUDIO_VERSION}
pytorch_index=${PYTORCH_INDEX}
tested_driver=${TESTED_DRIVER}
EOF
}

print_summary() {
  log "Isaac Sim installed: ${VENV_PATH}/bin/python"
  if [[ -n "$ISAACSIM_REPO_REF" ]]; then
    log "Official IsaacSim repo: ${ISAACSIM_REPO_DIR} (${ISAACSIM_REPO_REF})"
  else
    log "Official IsaacSim repo: not pinned for ${SIM_PROFILE}; pip packages only"
  fi
  log "Compatibility entrypoint: /isaac-sim/python.sh"
}

main() {
  parse_args "$@"
  recipe_sim "$PROFILE"
  VENV_PATH="${INSTALL_ROOT%/}/${VENV_NAME}"

  apply_proxy
  select_python
  check_host

  log "Using recipe ${SIM_PROFILE}: Isaac Sim ${ISAACSIM_PIP_VERSION}, Python ${PYTHON_REQUIRED}, torch ${TORCH_VERSION}."
  install_system_deps
  create_venv
  install_python_packages
  checkout_isaacsim_repo
  install_compat_layout
  write_record

  [[ "$DRY_RUN" == "1" ]] || print_summary
}

main "$@"~~~~
```

通过 Skill 在百舸开发机快速安装 Isaac Sim 与 Isaac Lab

面向具身智能仿真、机器人训练与评测场景，百舸提供 Isaac Sim 与 Isaac Lab 快速安装 Skill，帮助用户在百舸开发机或远程 GPU 环境中，以交互式方式完成安装配置，降低仿真环境准备成本。

功能介绍 当前提供两个安装 Skill：

- `isaac-sim-install`：用于安装 NVIDIA Isaac Sim 仿真平台。
- `isaac-lab-install`：用于安装 Isaac Lab 机器人学习与仿真实验框架。

用户可以通过本地或 SSH 方式调用 Skill，在目标开发机上完成安装路径选择、网络代理配置、运行方式选择和版本选择等步骤。

> 您可以参考该文档配置您的开发机：SSH远程连接开发机。 同时，百舸平台已经在官方 bos 存储中预置了 Isaac Sim 资产库，您可以通过将 aihc-rdw-bj/isaac-sim/ 挂载至 /mnt/bos/isaac-sim 来直接使用相关资产，如果需要加速运行您可以将相关资产转存至您私有的 bos或 pfs 中。

适用场景

- 在百舸开发机中快速搭建 Isaac Sim 仿真环境。
- 为机器人强化学习、模仿学习、仿真评测任务准备 Isaac Lab 环境。
- 在远程 GPU 机器上统一安装具身智能实验依赖。
- 降低手动配置 CUDA、Python、依赖包和启动脚本等环境成本。

典型使用方式 以远程开发机安装 Isaac Sim 为例：

```
/isaac-sim-install ssh -p <端口> <用户名>@<开发机地址>
```

示例：

```
/isaac-sim-install ssh -p 8560 root@xxx.xxx.xxx.xxx
```

安装 Isaac Lab 时，可使用：

```
/isaac-lab-install ssh -p <端口> <用户名>@<开发机地址>
```

如需在本地环境执行，可使用对应 Skill 的 `local` 调用方式。

安装流程 调用 Skill 后，将按交互式流程完成以下配置（请注意，基于您不同的本地 Agent 以及模型，所产生的对话会有所不同，以下截图仅作参考）：

1. 信任 ssh 连接。



2. 选择安装路径：指定 Isaac Sim / Isaac Lab 的安装目录，建议磁盘预留充足空间。



3. 配置网络代理：安装过程中可能需要访问 Git、pip、NVIDIA 相关资源，建议按网络环境配置代理（注意此处需要在选项 4 中输入）。



4. 选择运行方式：根据实际使用习惯选择启动方式，例如持续监听服务或按需启动。



5. 选择安装版本：根据任务需求选择 Isaac Sim 或 Isaac Lab 的目标版本，建议优先选择 Skill 推荐版本。



6. 执行安装与基础验证：Skill 会自动执行安装流程，并进行基础冒烟验证。

- 安装完成并验证通过。
  - 目标机器：`ssh -p 8560 root@106.12.167.108`
  - Profile：`sim-6.0.0`
  - 安装根目录：`/root/workspace`，使用你确认的默认目录
  - Isaac Sim：`6.0.0.1`
  - venv：`/root/workspace/env_isaacsim`
  - Python：`/root/workspace/env_isaacsim/bin/python`
  - 官方 repo：`https://github.com/isaac-sim/IsaacSim.git`，ref `v6.0.0`，路径 `/root/workspace/IsaacSim`
  - 平台入口：`/isaac-sim/python.sh` 已创建，映射到 `/root/workspace/env_isaacsim/bin/python`
  - 命令摘要：复制 .claude/skills/isaac-sim-install/scripts/install_isaac_sim.sh 到远端后，以 `--profile sim-6.0.0 --install-root /root/workspace --proxy <user-provided-url>` 后台执行
  - 完成状态：安装脚本退出码 `0`
  - 验证结果：`/isaac-sim/python.sh -c "import isaacsim"` 成功；显式 EULA 环境下直接 Python 导入成功；`pip check` 为 `No broken requirements found.`
  - 清理结果：远端过程目录 `/root/workspace/isaac-sim-install-skill` 已删除，安装环境、repo、record 和入口均保留
  - 风险：直接运行 venv Python 导入 `isaacsim` 会触发 NVIDIA EULA 交互；使用 `/isaac-sim/python.sh` 或设置 `OMNI_KIT_ACCEPT_EULA=YES OMNI_KIT_ALLOW_ROOT=1` 可正常非交互运行。

注意事项

- Isaac Lab 与 Isaac Sim 存在版本依赖关系，建议使用 Skill 推荐的版本组合。
- 当前 Isaac Lab 官方推荐依赖组合在部分环境下可能存在依赖冲突；基础冒烟验证可通过，但复杂任务仍建议结合实际 workload 进一步验证。
- 安装前请确认开发机具备可用 GPU、充足磁盘空间以及可访问必要软件源的网络环境，例如如安装过程中 Git、pip、NVIDIA 相关资源下载较慢，建议优先配置网络代理。
- 如果目标机器为百舸开发机，请确认 SSH 连接信息、端口和账号权限可用。

后续操作 您可以继续使用使用该环境探索以下实践：

Isaac Sim 可视化

Isaac Lab遥操环境搭建和图形可视化

最佳实践

使用产品事件设置告警通知规则

百舸平台各功能模块的产品事件已接入云监控 BCM ，用户可以使用 BCM 的告警通知功能自定义设置重要事件报警通知规则，当监控到指定事件发生时，BCM 会向用户预设的联系方式发送告警通知。

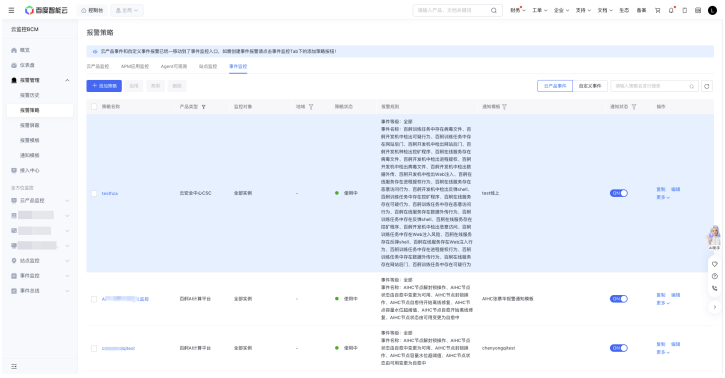
典型场景

- 资源节点被封锁时通知资源池管理员
- 训练任务停止时通知任务提交用户
- 开发机被停止时通知开发机所有者

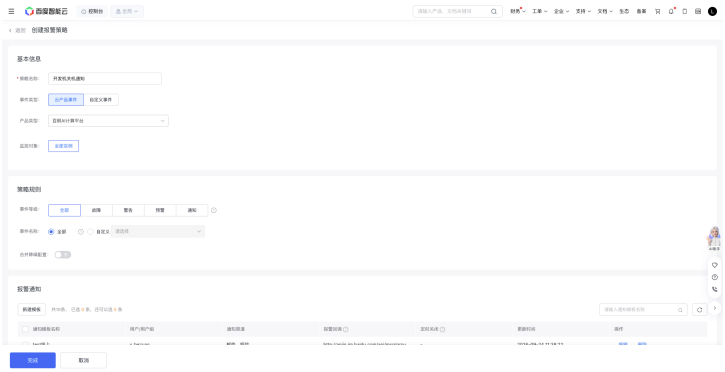
操作步骤

> 以下以开发机被停止事件为例，说明如何设置告警通知规则。

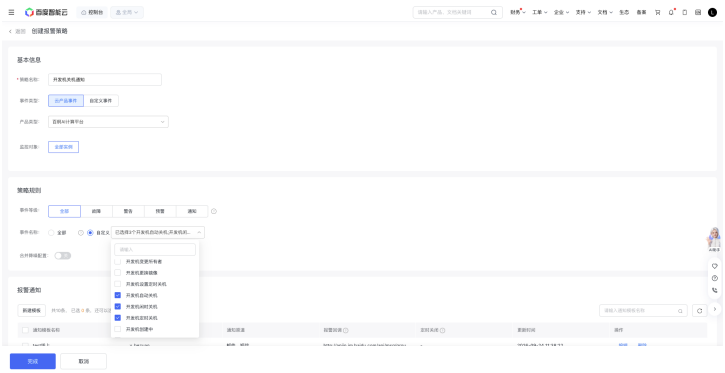
1. 登录 云监控BCM控制台 ，在左侧菜单中选择“报警管理/报警策略“，点击进入报警策略设置页面。



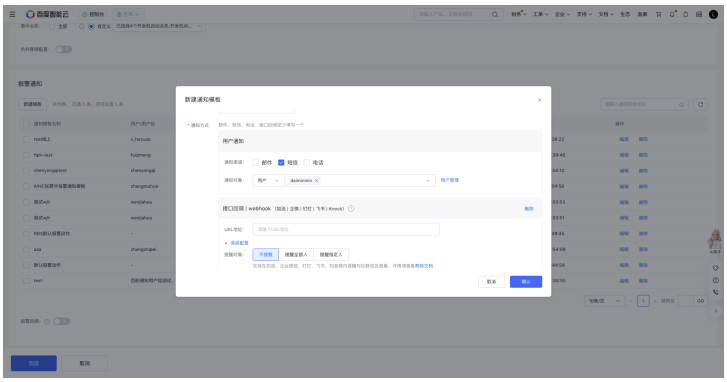
2. 切换到【事件监控】Tab页，点击【添加策略】按钮进入“创建报警策略”页面。填写基本信息，输入策略名称，“事件类型”选择“云产品事件”，从产品类型下拉列表中选择“百舸AI计算平台”。



4. 配置策略规则，“事件等级”选择“全部”；“事件名称”选择“自定义”，从下拉列表中选择与开发机关机相关的全部产品事件（如开发机自动关机、开发机闲时关机、开发机定时关机）



5. 配置报警通知模板，可以选择已有模板或者“新建模版”，以“新建模版为例”，填写模版名称及需用通知的用户和通知渠道点击“确定”



6. 点击【完成】提交返回列表即可在列表中看到新建的告警策略，在列表中可以开启或关闭通知。

百舸平台支持产品事件清单

详见"操作指南"中全托管资源池、分布式训练、在线服务部署、开发机各功能模块的相关文档。

更多告警配置

更多告警配置可参考云监控BCM/操作指南。

```
 ## 容器实例开启访问公网


大模型开发或训练部分场景中，如果用户需要在容器（训练任务、推理服务、开发机）中访问公网下载数据，则需要开启访问公网能力，本文介绍如何实现容器实例支持访问公网。

##### 创建弹性公网IP（EIP）

1. 登录百度智能云EIP控制台https://console.bce.baidu.com/eip
2. 选择**地域**，并选择**实例** > **创建实例**

| 配置项 | 说明 |
| --- | --- |
| 付费方式 | 按需选择预付费或者后付费 |
| 名称 | 自定义EIP实例名称 |
| 线路类型 | 默认选择"标准型BGP"即可 |
| 公网带宽 | 按需选择 |
| 原生防护及其他 | 默认即可 |
![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_81b2431.png)
![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_f46bdce.png)
3. 点击确认订单即可

##### 创建NAT网关实例
1. 登录百度云VPC-NAT网关控制台https://console.bce.baidu.com/network/#/vpc/nat
2. **地域**，并选择**NAT网关** > **创建NAT网关**
![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_95c5401.png)
3. 配置相关配置项，点击"确认订单"，完成NAT网关创建

| 配置项 | 说明 |
| --- | --- |
| 付费方式 | 按需选择预付费或者后付费 |
| 所在网络 | 选择与计算资源同一网络 |
| NAT网关名称 | 自定义NAT网关名称 |
| 性能容量 | 按需配置 |
| SNAT公网IP | 选择"弹性公网IP"，并在下拉菜单中选择上一步创建的EIP |
![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_3f53cab.png)

4. 配置路由表，选择**路由表** > **所在网络** > **路由表名称**，进入路由表管理页面
![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_2021983.png)

5. 为路由表添加路由，配置完成后，点击确定即可。

| 配置项 | 说明 |
| --- | --- |
| 网段类型 | IPv4 |
| 源网段 | 选择与计算资源同一网段 |
| 目标网段 | 可填写0.0.0.0/0 |
| 路由类型 | NAT网关 |
| 下一跳实例 | 选择上步创建的 NAT 网关 |

![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_3b4ffa2.png)

6. 配置SNAT，选择**网络连接** > **NAT网关** > 对应NAT实例操作列中**设置SNAT**，进入SNAT列表，点击**添加SNAT条目**，配置完成后确定即可。

注意：要保证集群内机器的子网均在SNAT 列表中 （如 所有机器均在一个子网则配置一条SNAT即可， 如果在n个子网需要配置n 条 SNAT）

| 配置项 | 说明 |
| --- | --- |
| 条目名称 | 自定义 |
| 源网段 | 选择与计算资源同一网段 |
| 公网IP地址 | 选择上一步创建的EIP |

![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_2cd33c0.png)

![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_faa5866.png)

## 分布式多机部署DeepSeek R1模型

<font style="color:#1c1d1f;">随着模型参数量不断增加，单台GPU服务器已经无法满足大规模模型的推理需求。分布式多机推理可以将单实例分配到多台服务器部署，利用并行计算加速推理过程，提高大规模模型和高并发场景的推理效率。本文将介绍如何通过百舸平台在H20 2机分布式部署DeepSeek R1模型。</font>

##### 前置条件
```

准备资源
1. 购买2台支持RDMA的H20机器创建百舸自运维资源池（H20需开通白名单，请联系百度售前工程师）
2. 开通PFS或CFS存储并与资源池绑定

准备数据
平台已提供DeepSeek R1模型权重文件存储在BOS对象存储中，您可以从对应地域的BOS路径中下载模型。

1. 登录节点安装BOSCMDhttps://cloud.baidu.com/doc/BOS/s/qjwvyqegc并完成BOSCMD配置https://cloud.baidu.com/doc/BOS/s/Ejwvyqe55
2. 从平台提供的BOS地址中拷贝权重文件到PFS或CFS存储中：

```
./bcecmd bos sync bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1 /mnt/model/DeepSeek-R1
```

地域	**BOS路径**
北京	bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1
苏州	bos:/aihc-models-su/deepseek-ai/DeepSeek-R1
广州	bos:/aihc-models-gz/deepseek-ai/DeepSeek-R1

镜像准备
仅支持vllm和sglang推理框架，vllm 版本 >= 0.7.3 sglang版本 >= 0.4.3，平台已预置vllm和sglang 官方镜像，建议直接使用平台提供的镜像。

部署服务
1. 登录**百舸在线服务部署**控制台，选择**自定义部署**，点击**部署服务；**
2. 输入服务名称、选择资源池/队列；
3. 开启分布式推理，单实例Pod数设置为2，开启RDMA；

4. **服务镜像：**选择百舸预置镜像中的 sglang v0.4.3.post4-cu125 或 vllm 0.7.3


5. **设置启动命令：**根据选择的镜像填写对应的启动命令

sglang启动命令：

```
MASTER_IP=$(cut -d ',' -f 1 $FED_MEMBER_FILE)
python3 -m sglang.launch_server \
    --model-path /mnt/pfs/deepseek-ai/DeepSeek-R1 \
    --tp 16 \
    --dist-init-addr $MASTER_IP:5000 \
    --nnodes $FED_POD_NUM \
    --node-rank $FED_POD_INDEX \
    --trust-remote-code \
    --host 0.0.0.0 \
    --port 8088
```

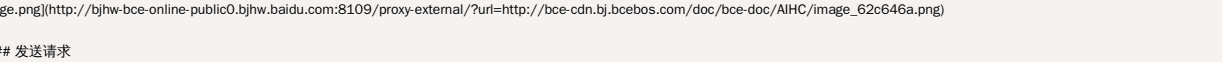
vllm 启动命令

```
MASTER_IP=$(cut -d ',' -f 1 $FED_MEMBER_FILE)
if [ "$FED_POD_INDEX" -eq 0 ]; then
    /vllm-workspace/ray_init.sh leader --ray_cluster_size=$FED_POD_NUM
    python3 -m vllm.entrypoints.openai.api_server \
        --port 8088 \
        --model /mnt/model/DeepSeek-R1 \
        --tensor-parallel-size 16 \
        --trust_remote_code
else
    echo "Master IP: $MASTER_IP"
    /vllm-workspace/ray_init.sh worker --ray_address=$MASTER_IP
fi
```

6. **资源申请：**加速芯片数量设置为8卡
7. **存储挂载：**建议您使用PFS L2存储类型，提高模型加载和服务异常恢复速度；以PFS存储类型为例，填写存储模型权重文件的PFS或CFS源路径和容器目标路径，启动命令中的路径会根据目标路径动态更新；

8. **流量接入：**
+ 设置服务端口设置为8088，启动命令会根据此配置动态更新
+ 开启云原生AI网关（云原生AI网关仅支持http请求）
9. **健康检查：**若部署生产服务建议开启健康检查配置探针
+ 参考以下配置设置启动探针

+ 参考以下配置就绪探针和存活探针



发送请求
服务部署成功后，可在列表查看，服务状态均为"运行中"即可调用；在调用信息中可获取HTTP调用地址和Token信息；

请求示例：

```
curl --location '180.76.137.48/auth/ap-f6d303295/8088' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6ImNzbSIsInR5cCI6IkpXVCJ9.eyJpYXQiO***' \
--data '{
```

```
    "model": "deepseek r1",
    "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}],
    "max_tokens": 10,
    "temperature": 0
}'
```

查看服务监控
可在服务详情中查看服务资源、流量监控


在百舸平台部署Llama-4推理服务

Llama 4模型是Llama系列模型中首批采用混合专家（MoE）架构的模型。Llama 4 模型还针对视觉识别、图像推理、字幕生成以及回答关于图像的问题进行了优化。本文将指导您如何在百舸平台部署Llama-4模型。

前置条件
准备资源
1. 购买1台H20机器并创建资源池（H20需要开通白名单，请联系百度售前工程师）；
2. 开通存储产品，PFS、CFS或BOS，PFS需与资源池绑定。

准备模型文件
平台已提供 Llama-4-Scout-17B-16E-Instruct 模型权重文件存储在BOS对象存储中，您可以从BOS下载模型。

1. 登录节点安装BOSCMD https://cloud.baidu.com/doc/BOS/s/qjwvyqegc 并完成BOSCMD配置置 https://cloud.baidu.com/doc/BOS/s/Ejwvyqe55
2. 从平台提供的BOS地址中拷贝权重文件到PFS存储中：

```
./bcecmd bos sync bos:/aihc-models-bj/meta-llama/Llama-4-Scout-17B-16E-Instruct
```

部署服务
1. 登录百舸异构计算平台，在左侧导航中选择 **在线服务部署** 部署自定义服务；

2. 进入自定义部署页面点击 **部署服务**,填写部署服务的相关参数；

3. 资源申请中的加速芯片卡数设置为8；

4. 服务镜像选择 **CCR企业版镜像>百舸预置镜像** 平台提供vLLM镜像 **vllm-openai v0.8.3**

5. 设置端口及流量接入，填写调用时的服务访问端口，如需公网访问请开启公网开关，默认仅支持VPC内调用

6. 设置存储挂载

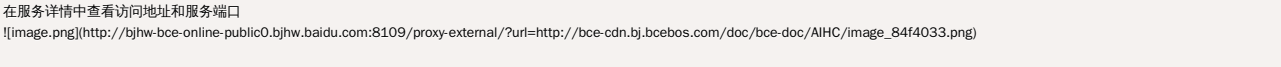
源路径填写存储模型权重文件的CFS、PFS或BOS路径，如无特殊需求目标路径可填写为 /mnt/model；

7. 设置启动命令

PATH_TO_MODEL替换为容器目标路径; PORT替换为服务端口

```
VLLM_DISABLE_COMPILE_CACHE=1 vllm serve PATH_TO_MODEL --host 0.0.0.0 --port PORT --tensor-parallel-size 8 --max-model-len 262144 --trust-remote-code --override-generation-config='{"attn_temperature_tuning": true}' --served-model-name Llama-4-Scount-17B-16E-Instruct
```

8.部署服务，等待服务状态为"运行中"即可调用。

调用模型服务
在服务详情中查看访问地址和服务端口


请求示例

```
curl --location '<访问地址>/v1/chat/completions' \
--header 'Content-Type: application/json' \
--header 'Authorization: <TOKEN>' \
--data '{
    "model": "Llama-4-Scount-17B-16E-Instruct",
    "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}],
    "max_tokens": 1024,
    "temperature": 0
}'
```

快速微调rdt-1b模型

robotics-diffusion-transformer/rdt-1b（简称rdt-1b）是一款基于扩散Transformer（Diffusion Transformer, DiT）的双臂机器人操作基础模型，专为复杂双臂协调任务设计，是目前全球最大的双臂操作扩散模型之一。

目前百舸AI计算平台提供了 rdt-1b 模型的训练模版，内置训练代码和镜像，您可以在平台[分布式训练](https://console.bce.baidu.com/aihc/tasks)模块中，快速发起训练。

前置条件

算力和存储资源准备

算力资源

下表给出了不同模型在百舸默认参数和数据集的情况下，训练所依赖的最低配置：

模型名称	训练方式	训练资源配置
robotics-diffusion-transformer/rdt-1b	Pretrain & Finetune	H800/A800*8卡

存储资源

为保证训练的性能，推荐使用[并行文件存储PFS](https://cloud.baidu.com/product/pfs.html)。

数据准备

rdt-1b Pretrain

多模态编码器准备

- `t5-v1_1-xxl`: [link](https://huggingface.co/google/t5-v1_1-xxl/tree/main)，可通过百舸平台的数据下载功能从huggingface导入并使用数据转储功能转存到PFS。
- `siglip`: [link](https://huggingface.co/google/siglip-so400m-patch14-384)，可通过百舸平台的数据下载功能从huggingface导入并使用数据转储功能转存到PFS。

百舸平台已经在对象存储BOS中预置上述编码器，方便用户使用。请参考[下载百舸平台预置的公共数据到PFS](https://cloud.baidu.com/doc/AIHC/s/Mmfjj3i9f)

```
t5-v1_1-xxl :
bos:/aihc-models-bj/robotics-diffusion-transformer/rdt-1b/google/t5-v1_1-xxl

siglip :
bos:/aihc-models-bj/robotics-diffusion-transformer/rdt-1b/google/siglip-so400m-patch14-384
```

数据集准备

数据集要求：TFRecord格式，如果数据集不满足要求需要转换。

百舸平台已经在对象存储BOS中，预置训练数据集，方便用户快速开始。您可以从对应地域的BOS路径中下载数据。

```
bos:/aihc-rdw-bj/austin_sailor_dataset_converted_externally_to_rlds
```

训练数据加载文件准备（可选）

根据需要自定义训练数据加载文件（L71和L72请不要修改），训练时修改`DATA_CONFIG`为该文件路径即可。

```
import tensorflow as tf

from data.utils import clean_task_instruction, euler_to_quaternion, rotation_matrix_to_ortho6d

def process_step(step: dict) -> dict:
    """
    Unify the action format and clean the task instruction.

    DO NOT use python list, use tf.TensorArray instead.
    """
    # Convert raw action to our action

    origin_action = step['action']
    step['action']={}
    action=step['action']
    action['terminate'] = step['is_terminal']

    eef_delta_pos = origin_action[:3]
    eef_ang=origin_action[3:6]
    eef_ang = euler_to_quaternion(eef_ang)
    # gripper_open: -1-open, 1-closed
    grip_open=tf.where(tf.equal(origin_action[6:],tf.constant(-1.0)),tf.constant(1.0),tf.constant(0.0))

    # No base found

    # Concatenate the action
    action['arm_concat'] = tf.concat([eef_delta_pos,eef_ang,grip_open],axis=0)

    # Write the action format
    action['format'] = tf.constant(
        "eef_delta_pos_x,eef_delta_pos_y,eef_delta_pos_z,eef_delta_angle_x,eef_delta_angle_y,eef_delta_angle_z,eef_delta_angle_w,gripper_open")

    # Convert raw state to our state
    state = step['observation']
    # Concatenate the state
    eef_mat = tf.transpose(tf.reshape(state['state'][8:], (4, 4)))
    eef_pos = eef_mat[:3, 3]
    rotaion_matrix = eef_mat[:3, :3]
    eef_ang = rotation_matrix_to_ortho6d(rotaion_matrix)
    joint_pos = state['state'][:7]
    grip_open = state['state'][7:8] * 12.5 # rescale to [0, 1]
    state['arm_concat'] = tf.concat([joint_pos,grip_open,eef_pos,eef_ang],axis=0)

    # Write the state format
    state['format'] = tf.constant(

"arm_joint_0_pos,arm_joint_1_pos,arm_joint_2_pos,arm_joint_3_pos,arm_joint_4_pos,arm_joint_5_pos,arm_joint_6_pos,gripper_joint_0_pos,eef_pos_x,eef_pos_y,eef_pos_z,eef_angle_0,eef_angle_1,eef_angle_2,
)

    # Clean the task instruction
    # Define the replacements (old, new) as a dictionary
    replacements = {
        '_': ' ',
        '1f': ' ',
        '4f': ' ',
```

```
        '-': ' ',
        '50': ' ',
        '55': ' ',
        '56': ' ',

    }
    instr = step['language_instruction']
    instr = clean_task_instruction(instr, replacements)
    step['observation']['natural_language_instruction'] = instr

    return step


if __name__ == "__main__":
    import tensorflow_datasets as tfds
    from data.utils import dataset_to_path

    DATASET_DIR = 'data/datasets/openx_embod'
    DATASET_NAME = 'rdt_data'
    # Load the dataset
    dataset = tfds.builder_from_directory(
        builder_dir=dataset_to_path(
            DATASET_NAME, DATASET_DIR))
    dataset = dataset.as_dataset(split='all')

    # Inspect the dataset
    for episode in dataset:
        for step in episode['steps']:
            print(step)
```

rdt-1b Finetune

数据集准备

用户数据集要求：HDF5格式，如果数据集不满足要求需要转换。

可使用官方数据集：https://huggingface.co/datasets/robotics-diffusion-transformer/rdt-ft-data

百舸平台已经在对象存储BOS中预置上述训练数据集，方便用户快速开始。您可以从对应地域的BOS路径中下载数据。

```
bos:/aihc-rdw-bj/rdt_ft_data
```

多模态编码器准备

同pretrain

模型权重准备

百舸平台已经在对象存储BOS中预置模型权重，方便用户快速开始。您可以从对应地域的BOS路径中下载数据。

```
bos:/aihc-models-bj/robotics-diffusion-transformer/rdt-1b/ckpt/rdt-1b
```

进行训练

1. 登录[百舸AI计算平台AIHC控制台](https://console.bce.baidu.com/aihc/resources)。
2. 进入 **分布式训练** 列表页面，点击**创建任务**，填写任务配置信息：
   - 基础信息
     - 任务名称：输入您的自定义任务名称
     - 资源池&队列：选择已创建的资源池和队列
     - 优先级：默认 中 优先级，无需更改
     - 训练框架：选择Pytorch
     - 任务创建方式： 选择 基于开源模型训练模版创建
     - 模型：选择对应的模型和训练方式
   - 环境配置
     - 镜像地址：无需更改，平台默认提供
     - 执行命令：系统已经默认训练的启动参数，需要替换`数据集`、`模型权重配置`等配置信息
     - 环境变量：无需更改
   - 资源配置
     - 实例数：这里我们选择的是rdt模型的微调，使用1台H800 8卡训练
     - 加速芯片：这里选择提前准备的GPU，单实例8卡
     - 共享内存：默认10Gi，pretrain训练，需要更新为80Gi
     - RDMA：默认开启，无需更改
   - 设置数据源
     - 存储类型：这里选择我们提前准备的PFS
     - PFS源路径：PFS的挂载路径，默认根目录，按实际情况填写
     - 挂载路径：容器内的挂载路径
3. 点击 **完成**，提交训练任务。

观察训练迭代

任务提交后，您可以通过任务日志观察训练的迭代情况

快速微调openpi模型

OpenPI（Open Physical Intelligence） 是由Physical Intelligence团队开发的开源机器人基础模型项目，专注于**视觉-语言-动作（VLA）**的联合建模，旨在实现通用机器人控制。

目前百舸AI计算平台提供了openpi/π0及π0-fast 模型的训练模版，内置训练代码和镜像，您可以在平台[分布式训练](https://console.bce.baidu.com/aihc/tasks)模块中，快速发起训练。

前置条件

算力和存储资源准备

算力资源

下表给出了不同模型在百舸默认参数和数据集的情况下，训练所依赖的最低配置：

模型名称	训练方式	训练资源配置
openpi/π0	Finetune	H800/A800*1卡
openpi/π0-fast	Finetune	H800/A800*1卡

存储资源

为保证训练的性能，推荐使用[并行文件存储PFS](https://cloud.baidu.com/product/pfs.html)。

数据准备

数据集准备

可使用官方数据集：https://huggingface.co/datasets/openvla/modified_libero_rlds

百舸平台已经在对象存储BOS中预置上述训练数据集。您可以从对应地域的BOS路径中下载数据。请参考[下载百舸平台预置的公共数据到PFS](https://cloud.baidu.com/doc/AIHC/s/Mmfjj3i9f)

```
bos:/aihc-rdw-bj/openpi/pi0-fast/data/modified_libero_rlds
```

数据转换为LeRobot dataset

将自定义的数据集转换为LeRobot dataset。可以使用[官方脚本](https://github.com/Physical-Intelligence/openpi/blob/main/examples/libero/convert_libero_data_to_lerobot.py)进行转换。

百舸平台已经在对象存储BOS中预置转换后数据集，您可以从对应地域的BOS路径中下载数据.

```
bos:/aihc-rdw-bj/openpi/pi0-fast/data/libero
```

准备数据映射文件和训练配置文件（可选）

如需要使用自定义的数据集微调模型，需要重新定义数据处理和训练的配置。以下Libero 数据集作为示例，您可以针对自己的数据集进行修改

- 定义从 Libero 环境到模型的数据映射的配置，[示例参考](https://github.com/Physical-Intelligence/openpi/blob/main/src/openpi/policies/libero_policy.py)，如需要自定义，可以在训练代码中指定`DATA_CONFIG`文件路径替换。

- 定义微调超参数、数据配置和权重加载器配置。如需要自定义，可以在训练代码中指定`TRAINING_CONFIG`文件即可

  config文件参考示例：

  ```
  bos:/aihc-rdw-bj/openpi/pi0-fast/training_config.py
  ```

模型权重& tokenizer准备

百舸平台已经在对象存储BOS中预置模型权重,您可以从对应地域的BOS路径中下载数据。

openpi/π0 模型权重

```
bos:/aihc-models-bj/openpi/pi0/ckpt/pi0_base
```

openpi/π0-fast 模型权重

```
bos:/aihc-models-bj/openpi/pi0-fast/ckpt/pi0_fast_base
```

Tokenizer：

```
bos:/aihc-models-bj/openpi/pi0-fast/tokenizer/paligemma_tokenizer.model
```

进行训练

1. 登录[百舸AI计算平台AIHC控制台](https://console.bce.baidu.com/aihc/resources)。
2. 进入 **分布式训练** 列表页面，点击**创建任务**，填写任务配置信息：
  - 基础信息
    - 任务名称：输入您的自定义任务名称
    - 资源池&队列：选择已创建的资源池和队列
    - 优先级：默认 中 优先级，无需更改
    - 训练框架：选择Pytorch
    - 任务创建方式： 选择 基于开源模型训练模版创建
    - 模型：选择对应的模型和训练方式
  - 环境配置
    - 镜像地址：无需更改，平台默认提供
    - 执行命令：系统已经默认训练的启动参数，需要替换`DATA_PATH`、`TRAINING_CONFIG`等配置信息
      - `DATA_PATH`:数据集的路径
      - `DATA_CONFIG`:如使用上述平台提供的libero数据集，这里则不填写，使用默认的DATA_CONFIG。如自定义数据集，则需要配置此文件
      - `TRAINING_CONFIG`:如使用平台预置的数据集、训练配置等，无需填写，使用默认的TRAINING_CONFIG；如需要自定义更多的训练参数等配置，则需要配置此文件
      - `TRAINING_CONFIG_NAME`:如使用上述平台提供的libero数据集，则无需填写；如自定义数据集，则需要自定义填写
      - `PALIGEMMA_TOKENIZER_PATH`: 模型的tokenizer路径
      - `REPO_ID`:如使用上述平台提供的libero数据集，则无需填写；如自定义数据集，则需要自定义填写
      - `BASE_CHECKPOINT_LOAD_PATH`:模型原始权重的路径
      - `CHECKPOINT_SAVE_BASEDIR`:模型训练后权重的保存路径
      - `CHECKPOINT_SAVE_INTERVAL`:模型训练权重的保存间隔，如1000
    - 环境变量：无需更改
  - 资源配置
    - 实例数：这里我们选择的是openpi模型的微调，使用H800训练
    - 加速芯片：这里选择提前准备的GPU，单实例1/8卡
    - 共享内存：默认10Gi
    - RDMA：默认开启，无需更改
  - 设置数据源
    - 存储类型：这里选择我们提前准备的PFS
    - PFS源路径：PFS的挂载路径，默认根目录，按实际情况填写
    - 挂载路径：容器内的挂载路径
3. 点击 **完成**，提交训练任务。

观察训练迭代

任务提交后，您可以通过任务日志观察训练的迭代情况

百舸监控大盘报表导出

概述
百舸提供资源池、开发、训练、推理等各个模块全方位的可观测能力，本文介绍如何下载监控大盘中的统计报表。

报表导出
操作步骤
1. 登录百舸并找到需要导出的报表，本文以资源池总览-占用卡的工作负载信息报表为例
2. 点击报表名称右侧下拉按钮，选择**Inspect**——**Data**
3. 在右侧弹窗中开启**Data options**中的**Apply panel transformations**和**Formatted data**开关，点击**Download CSV**即可下载对应报表。

快速微调NVIDIA GR00T N1.5模型

NVIDIA GR00T N1.5 是英伟达（NVIDIA）推出的开源通用人形机器人基础模型，相比 N1，N1.5 在 **语言指令理解**、**新环境泛化** 和 **训练效率** 上显著提升。

目前百舸异构计算平台提供了 NVIDIA GR00T N1.5 模型的训练模版，内置训练代码和镜像，您可以在平台[分布式训练](https://console.bce.baidu.com/aihc/tasks)模块中，快速发起训练。

前置条件

算力和存储资源准备

算力资源

下表给出了不同模型在百舸默认参数和数据集的情况下，训练所依赖的最低配置：

模型名称	训练方式	训练资源配置
nvidia/GR00T-N1.5-3B	Finetune	H800/A800*1卡

存储资源

为保证训练的性能，推荐使用[并行文件存储PFS](https://cloud.baidu.com/product/pfs.html)。

数据准备

GR00T-N1.5-3B Finetune

数据集准备

数据集要求：[Huggingface LeRobot](https://github.com/huggingface/lerobot) 格式。

如使用自定义数据，需要将自定义的数据集转换为LeRobot dataset，可参考[Robot Data Conversion Guide](https://github.com/NVIDIA/Isaac-GR00T/blob/aa6441feb4f08233d55cbfd2082753cdc01fa676/getting_started/LeRobot_compatible_data_schema.md)

百舸平台已经在对象存储BOS中，预置NVIDIA 开源的人形机器人数据集 **PhysicalAI-Robotics-GR00T-X-Embodiment-Sim/gr1_arms_only.CanSort**。您可以从对应地域的BOS路径中下载数据。请参考[下载百舸平台预置的公共数据到PFS](https://cloud.baidu.com/doc/AIHC/s/Mmfjj3i9f)

```
bos:/aihc-rdw-bj/nvidia/PhysicalAI-Robotics-GR00T-X-Embodiment-Sim/gr1_arms_only.CanSort
```

模型权重准备

百舸平台已经在对象存储BOS中预置模型权重，方便用户快速开始。您可以从对应地域的BOS路径中下载数据。

```
bos:/aihc-models-bj/nvidia/GR00T-N1.5-3B
```

数据配置文件（DATA_CONFIG_FILE）

若您的数据集完全匹配[data_config.py](https://github.com/NVIDIA/Isaac-GR00T/blob/aa6441feb4f08233d55cbfd2082753cdc01fa676/gr00t/experiment/data_config.py)中 `DATA_CONFIG_MAP` 中的预定义配置，则无需定义数据配置文件，否则需要重新定义此文件，可参考 [data_config.py](https://github.com/NVIDIA/Isaac-GR00T/blob/aa6441feb4f08233d55cbfd2082753cdc01fa676/gr00t/experiment/data_config.py)

进行训练

1. 登录[百舸异构计算平台AIHC控制台](https://console.bce.baidu.com/aihc/resources)。
2. 进入 **分布式训练** 列表页面，点击**创建任务**，填写任务配置信息：
  - 基础信息
    - 任务名称：输入您的自定义任务名称
    - 可见范围：选择任务的可见范围
    - 任务创建方式： 选择 **基于开源模型训练模版创建 **
    - 模型：选择对应的模型和训练方式，这里选择**nvidia/GROOT-N1.5-3B**
  - 环境信息
    - 镜像地址：无需更改，平台默认提供
    - 执行命令：系统已经默认训练的启动参数，需要替换`DATA_PATH`、`DATA_CONFIG`、`DATA_CONFIG_FILE`、`BASE_MODEL_PATH`、`CHECKPOINT_SAVE_PATH`等配置信息
      - `DATA_PATH`：必填，可直接使用平台预置在BOS中的**PhysicalAI-Robotics-GR00T-X-Embodiment-Sim/gr1_arms_only.CanSort**数据集，需要提前转存至资源池所在PFS并行文件存储中
      - `DATA_CONFIG`:必填，如使用上文预置的数据集**PhysicalAI-Robotics-GR00T-X-Embodiment-Sim/gr1_arms_only.CanSort**，则DATA_CONFIG=fourier_gr1_arms_only
      - `DATA_CONFIG_FILE`:若您的数据集完全匹配[data_config.py](https://github.com/NVIDIA/Isaac-GR00T/blob/aa6441feb4f08233d55cbfd2082753cdc01fa676/gr00t/experiment/data_config.py)
`DATA_CONFIG_MAP` 中的预定义配置，则无需填写
      - `BASE_MODEL_PATH`:原始权重加载地址
      - `CHECKPOINT_SAVE_PATH`：训练过程中CKPT保存地址
      - `MAX_STEPS=$MAX_STEPS`：训练的迭代次数，如100000
      - `REPORT_TO`：训练指标支持可视化工具，目前支持wandb、tensorboard。如定义为wandb，需要额外填写启动命令中的wandb的环境变量(请确保训练容器和wandb服务的网络联通性，否则会启动失败):
        - `WANDB_API_KEY`：必填，Wandb 账户的**认证密钥**，用于身份验证和访问权限。**必须设置**才能与 Wandb 服务端通信
        - `WANDB_ENTITY`：非必填，团队或用户名，用于控制实验数据存储的位置（个人账户或团队空间）
        - `WANDB_PROJECT`：非必填，如果不存在，wandb会自动创建
        - `WANDB_MODE`：必填，设置为online
        - `WANDB_NAME`：非必填，自动使用`CHECKPOINT_SAVE_PATH`作为名称
        - `WANDB_RUN_ID`：必填，自定义，如run1
    - 环境变量：无需更改
  - 资源配置
    - 资源池&队列：选择已创建的资源池和队列
    - 优先级：默认 中 优先级，无需更改
    - 训练框架：选择Pytorch
    - 资源配额
      - 实例数：目前支持单机训练
      - GPU类型&卡数：支持A800卡和H800卡，卡数1-8卡
      - CPU/内存：按需填写
    - 共享内存：默认10Gi
    - RDMA：单机训练，无需开启
    - 设置数据源
      - 存储类型：这里选择我们提前准备的PFS
      - PFS源路径：PFS的挂载路径，默认根目录，按实际情况填写
      - 挂载路径：容器内的挂载路径
3. 点击 **完成**，提交训练任务。

观察训练迭代

任务提交后，您可以通过任务日志观察训练的迭代情况

基于百舸平台对节点进行nccl-test

适用场景
为了确保分布式训练等依赖大规模并行计算的任务能高效、稳定运行，在启动任务前，可通过nccl-test来验证多GPU/多节点分布式系统中"集体通信"的性能与可靠性。

前置说明
全托管资源池
为了保证测试质量，建议创建一个专用于nccl-test的空队列，将需要进行nccl-test的节点移入该队列中，再在该队列上创建nccl-test分布式训练任务进行测试。

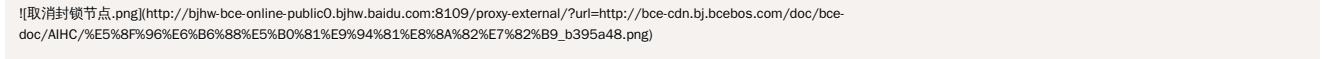


自运维资源池
对于新购入的节点，为了防止被自动归入默认队列被调用，需要在购入节点时开启封锁设置。


创建物理队列用于nccl-test，可指定节点加入该队列。



注意：如果节点设置了封锁，加入该队列后须取消封锁设置。



操作步骤

1. 创建分布式训练任务

您可以在百舸平台[分布式训练](https://console.bce.baidu.com/aihc/tasks)模块中，快速发起训练任务。nccl-test参数配置如下：

1.1 基本信息

* 创建方式：选择“自定义创建”



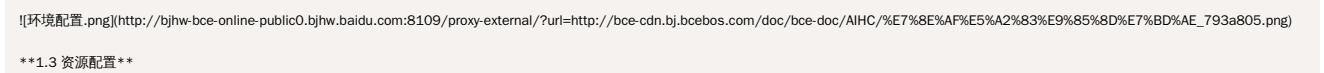
1.2 环境配置

* 镜像地址：registry.baidubce.com/aihc-aiak/aiak-training-llm:ubuntu22.04-cu12.3-torch2.2.0-py310-bccl1.2.7.2_v2.1.5.1_release
* 执行命令为：

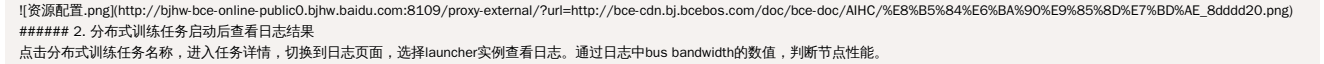
```
export OMPI_ALLOW_RUN_AS_ROOT=1
export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
export NCCL_DEBUG=INFO
export NCCL_IB_DISABLE=0

mpirun \
  -x UCX_NET_DEVICES=mlx5_1:1 \
  -x NCCL_IB_QPS_PER_CONNECTION=8 \
  -x LD_LIBRARY_PATH \
  -x NCCL_NET_PLUGIN=none \
  -x NCCL_IB_DISABLE=0 \
  -x NCCL_DEBUG=INFO \
  -x NCCL_ALGO=Ring \
  /usr/local/bin/all_reduce_perf_mpi -b 32M -e 16G -f 2 -g 1 -n 20
```

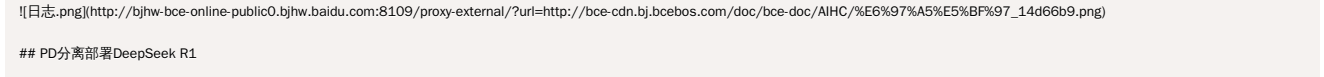
* 环境变量：执行命令中已设置，可跳过该配置



1.3 资源配置
* 资源池类型：全托管资源池
* 队列：选择前面创建的nccl-test队列
* 优先级：按需选择
* 训练框架：选择 MPI
* 资源配额：填写nccl-test队列中全部的资源配额（为了对节点充分测试）
* RDMA：开启



2. 分布式训练任务启动后查看日志结果
点击分布式训练任务名称，进入任务详情，切换到日志页面，选择launcher实例查看日志。通过日志中bus bandwidth的数值，判断节点性能。



PD分离部署DeepSeek R1

本文将为您介绍如何在百舸平台通过多角色服务组功能在8台Hopper架构服务器，使用AIAK推理引擎 PD分离部署 DeepSeek R1。

前置条件
准备资源
1. 创建百舸全托管资源池，购买8台支持RDMA的Hopper架构服务器
2. 开通PFS或CFS存储并与资源池绑定

购买资源需开通白名单，请联系百度售前工程师

准备数据
为了提升部署服务效率，百舸已将常用模型下载到BOS存储中，您可以直接通过BOSCMD工具拉取模型文件到自己的文件系统中，除了模型权重文件，还需要拉取对应模型的NextN文件。

BOSCMD工具拉取模型命令示例：

```
./bcecmd bos sync bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1 /mnt/model/DeepSeek-R1
./bcecmd bos sync bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1 /mnt/model/DeepSeek-R1-NextN
```

模型名称	BOS路径
DeepSeek-R1	bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1-0528
DeepSeek-V3	bos:/aihc-models-bj/deepseek-ai/DeepSeek-V3.1
DeepSeek-R1	bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1
DeepSeek-V3	bos:/aihc-models-bj/deepseek-ai/DeepSeek-V3
DeepSeek-R1-NextN	bos:/aihc-models-bj/deepseek-ai/DeepSeek-R1-NextN
DeepSeek-V3-NextN	bos:/aihc-models-bj/deepseek-ai/DeepSeek-V3-NextN

Tips：
模型路径需要根据您资源所在的Region适当调整，示例路径中的 bj 代表北京 Region，可替换为资源位置首字母缩写。

部署服务
1.登录百舸AI计算平台AIHC控制台，点击左侧导航栏中的 在线服务部署 进入列表页面；

2.切换到 **多角色服务组** 页签，点击 **部署多角色服务组** ；

3.正确填写服务名称及以下参数：
服务配置

* **分别配置Prefill和Decode服务参数**



参数名称	参数说明
资源队列	选择需要部署的算力所属全托管资源池队列，资源配置请选择Hopper架构的算力
资源规格	资源规格推荐Prefill和Decode分别32卡，CPU 100核，内存 1024G
实例数	可以根据业务需求设置Prefill和Decode的期望副本数
单实例Pod数	当前配置推荐单实例Pod数为4，即Prefill和Decode服务分别部署到4台机器
RDMA	当前配置需要多机通讯，建议开启RDMA
Kv Cache感知调度	开启后平台会感知Prefill节点的缓存命中率，将请求调度到命中率高的实例
镜像地址	平台默认推荐AIAK推理引擎，无需修改
端口	按需填写端口，可以为端口设置标签，便于多端口调用时区分用途
存储挂载	挂载模型权重文件

* **环境变量**

以下参数若无特殊需求可以直接使用平台推荐值

参数	说明
ENABLE_REDUNDANT_EXPERTS	冗余专家使能 如需关闭修改false，并添加环境变量 SET_DEEP_DP_NUM_EXPERTS=256
MAX_PREFILL_TOKENS	控制预填充阶段单批处理的最大token数量，推荐值为8192
MAX_RUNNING_REQUESTS	限制并发处理的请求数量，推荐值为256
MAX_TOTAL_TOKENS	设定单个请求的输入和输出token总数上限，推荐值为300000
MEM_FRACTION_STATIC	分配固定比例的GPU内存用于KV Cache等，推荐值为0.95
MODEL_DP_SIZE	指定数据并行的GPU数量，推荐值为8
MODEL_PATH	需填写模型R1权重文件容器挂载的目标路径，以上图为例：/deepseek
MODEL_TP_SIZE	指定张量并行的GPU数量，推荐值为4
MODEL_PATH_NEXTN	需填写模型R1权重文件容器挂载的目标路径，以上图为例：/deepseek-nextn
CUDA_GRAPH_MAX_BS	使用 CUDA Graph 时允许的最大Batch Size，推荐值为64

高级配置
共享内存建议设置为400G

流量接入
选择云原生网关，负载均衡策略设置为轮询，正确填写端口号

调用服务
部署服务成功后，在服务组中查看PD服务的调用信息。

调用示例

```
curl --location '<访问地址>/v1/chat/completions' \
--header 'Content-Type: application/json' \
--header 'Authorization: <TOKEN>' \
--data '{
    "model": "DeepSeek-R1",
    "messages": [{"role": "system", "content": "你是一名天文学家,请回答用户提出的问题。"}, {"role": "user", "content": "人类是否能登上火星?"}, {"role": "assistant", "content": "目前来看,人类登上火星是完全可能的..."}],
    "max_tokens": 1024,
    "temperature": 0
}'
```

使用自定义任务处理数据

在百舸平台中用户可以利用已有资源池队列或创建专门的CPU资源池，使用分布式训练任务模块提交自定义任务方式处理数据，并将处理好的数据写入到BOS、PFS、CFS等存储中供后续业务使用。

此文档提供使用百舸自定义任务处理数据的最佳实践方法和思路，针对不同业务场景，用户可参考该文档实现更加复杂的数据处理任务，亦可以创建可以并发处理的分布式处理任务。

典型场景
* 格式转换
* 数据清洗
* 数据生成

以下以将json格式的数据转换为jsonl格式为例讲解如何进行处理。

假设有批量的json格式的数据文件需要批量转换为jsonl格式，将存储在BOS的test-a存储桶的/json目录中的所有json文件批量转换为同名的jsonl格式并保存在PFS的/jsonl目录。

原始数据格式：

```json
[
{"src": "www.nvidia.com", "text": "The quick brown fox", "type": "Eng", "id": "0", "title": "First Part"},
{"src": "The Internet", "text": "jumps over the lazy dog", "type": "Eng", "id": "42", "title": "Second Part"}
]
```

目标数据格式：

```json
{"src": "www.nvidia.com", "text": "The quick brown fox", "type": "Eng", "id": "0", "title": "First Part"}
{"src": "The Internet", "text": "jumps over the lazy dog", "type": "Eng", "id": "42", "title": "Second Part"}
```
编写数据处理脚本

首先，需要编写数据处理脚本，支持使用任意编程语言编写转换脚本，我们以python编写为例。

创建json2jsonl.py、requirements.txt两个文件并保存在同一目录下。

json2jsonl.py文件：

json2jsonl.py文件：

```python
#### !/usr/bin/env python3
"""
JSON到JSONL批量转换脚本
将JSON数组文件转换为JSONL格式（每行一个JSON对象）
"""

import argparse
import json
import sys
from pathlib import Path


def convert_json_to_jsonl(input_file: Path, output_file: Path) -> bool:
    """转换单个JSON文件为JSONL格式"""
    try:
        with open(input_file, 'r', encoding='utf-8') as f:
            data = json.load(f)

        if not isinstance(data, list):
            print(f"错误: {input_file.name} 不是数组格式")
            return False

        output_file.parent.mkdir(parents=True, exist_ok=True)

        with open(output_file, 'w', encoding='utf-8') as f:
            for item in data:
                if isinstance(item, dict):
                    f.write(json.dumps(item, ensure_ascii=False) + '\n')

        print(f"转换成功: {input_file.name} -> {output_file.name}")
        return True

    except Exception as e:
        print(f"转换失败 {input_file.name}: {e}")
        return False


def main():
    parser = argparse.ArgumentParser(description="JSON到JSONL批量转换")
    parser.add_argument('--input-dir', '-i', required=True, help='输入目录')
    parser.add_argument('--output-dir', '-o', required=True, help='输出目录')
    parser.add_argument('--verbose', '-v', action='store_true', help='详细日志')
    args = parser.parse_args()

    input_path = Path(args.input_dir)
    output_path = Path(args.output_dir)

    if not input_path.exists():
        print(f"错误: 输入目录不存在 {input_path}")
        sys.exit(1)

    json_files = list(input_path.rglob("*.json"))
    if not json_files:
        print(f"警告: 在 {input_path} 中未找到JSON文件")
        return

    print(f"找到 {len(json_files)} 个JSON文件")

    success = 0
    for json_file in json_files:
        relative_path = json_file.relative_to(input_path)
        output_file = output_path / relative_path.with_suffix('.jsonl')

        if convert_json_to_jsonl(json_file, output_file):
            success += 1

    print(f"转换完成: {success}/{len(json_files)} 个文件成功")


if __name__ == "__main__":
    main()
```

requirements.txt文件：

```txt
ujson>=5.0.0
orjson>=3.0.0
```

> 此示例中没有外部依赖库，requirements.txt中的依赖仅用于演示

编写启动命令

```bash
#### !/bin/bash

#### JSON到JSONL批量转换脚本启动器
set -e

cd /mnt/scripts/

#### 默认值
INPUT_DIR="/mnt/data/json"
OUTPUT_DIR="/mnt/data/jsonl"
VERBOSE="${VERBOSE:-false}"
```

```
#### 解析参数
while [[ $# -gt 0 ]]; do
    case $1 in
        -h|--help)
            echo "用法: $0 [-i INPUT_DIR] [-o OUTPUT_DIR] [-v]"
            echo "环境变量: INPUT_DIR, OUTPUT_DIR, VERBOSE"
            exit 0
            ;;
        -v|--verbose) VERBOSE="true"; shift ;;
        -i|--input-dir) INPUT_DIR="$2"; shift 2 ;;
        -o|--output-dir) OUTPUT_DIR="$2"; shift 2 ;;
        *) echo "未知参数: $1"; exit 1 ;;
    esac
done

#### 安装依赖
pip install -r requirements.txt

#### 检查Python
if ! command -v python3 &> /dev/null; then
    echo "错误: Python3 未安装"
    exit 1
fi

#### 检查输入目录
if [ ! -d "$INPUT_DIR" ]; then
    echo "错误: 输入目录不存在: $INPUT_DIR"
    exit 1
fi

#### 创建输出目录
mkdir -p "$OUTPUT_DIR"

#### 构建命令
CMD="python3 json2jsonl.py --input-dir \"$INPUT_DIR\" --output-dir \"$OUTPUT_DIR\""
[ "$VERBOSE" = "true" ] && CMD="$CMD --verbose"

#### 执行转换
echo "转换: $INPUT_DIR -> $OUTPUT_DIR"
eval $CMD
```

创建自定义任务

1. 进入百舸创建任务页面填写任务信息，选择镜像、填写启动命令

2. 上传数据处理脚本代码到test-a存储桶的/scripts中

3. 挂载BOS和PFS存储，选择源数据存储和目标存储，分别设置挂载路径为/mnt/data/json和/mnt/data/jsonl（与启动脚本中的目录保持一致），同时挂载代码目录，并设置挂载路径为/mnt/scripts（与脚本中使用的路径保持一致）



4. 提交任务，等待数据处理完成，在任务详情中可以通过事件、日志查看任务运行情况

高级拓展
1. 业务中常用的周期执行数据处理任务可以使用百舸OpenAPI进行任务提交，也可以集成到自有系统中进行业务调用
2. 常用的处理工具打包成专门的数据处理镜像，可以避免每次上传处理脚本，直接在启动命令中调用
3. 程序处理脚本程序也可以在启动命令中使用git命令从代码库中拉取或者wget从远程服务器下载
4. 可以通过使用环境变量传递密钥、路径等信息进一步保证敏感信息安全

基于模板微调openpi-pi0.5模型

π0.5（Pi0.5）是Physical Intelligence（PI）公司推出的新一代视觉-语言-动作（Vision-Language-Action, VLA）模型，它在π0的基础上进行了显著升级，核心目标是提升机器人在**开放世界环境中的泛化能力**。它能够使机器人在从未见过的家庭环境中执行复杂的多步骤任务，例如清洁厨房或整理卧室

目前百舸异构计算平台提供了π0.5的训练模版，内置训练代码和镜像，您可以在平台[分布式训练](https://console.bce.baidu.com/aihc/tasks)模块中，快速发起训练。

前置条件

算力和存储资源准备

算力资源

下表给出了不同模型在百舸默认参数和数据集的情况下，训练所依赖的最低配置：

模型名称	训练方式	训练资源配置
openpi/π0.5	Finetune	H800/A800*8卡

存储资源

为保证训练的性能，推荐使用[并行文件存储PFS](https://cloud.baidu.com/product/pfs.html)。

数据准备

数据集准备

可使用官方数据集：https://huggingface.co/datasets/openvla/modified_libero_rlds

百舸平台已经在对象存储BOS中预置上述训练数据集。您可以从对应地域的BOS路径中下载数据。请参考[下载百舸平台预置的公共数据到PFS](https://cloud.baidu.com/doc/AIHC/s/Mmfjj3i9f)

```
bos:/aihc-rdw-bj/openpi/pi0-fast/data/modified_libero_rlds
```

数据转换为LeRobot dataset

将自定义的数据集转换为LeRobot dataset。可以使用[官方脚本](https://github.com/Physical-Intelligence/openpi/blob/main/examples/libero/convert_libero_data_to_lerobot.py)进行转换。

百舸平台已经在对象存储BOS中预置转换后数据集，您可以从对应地域的BOS路径中下载数据.

```
bos:/aihc-rdw-bj/openpi/pi0-fast/data/libero
```

准备数据映射文件和训练配置文件（可选）

如需要使用自定义的数据集微调模型，需要重新定义数据处理和训练的配置。以下Libero 数据集作为示例，您可以针对自己的数据集进行修改

- 定义从 Libero 环境到模型的数据映射的配置，[示例参考](https://github.com/Physical-Intelligence/openpi/blob/main/src/openpi/policies/libero_policy.py)，如需要自定义，可以在训练代码中指定`DATA_CONFIG`文件路径替换。

- 定义微调超参数、数据配置和权重加载器配置。[示例参考](https://github.com/Physical-Intelligence/openpi/blob/main/src/openpi/training/config.py)，如需要自定义，可以在训练代码中指定`TRAINING_CONFIG`文件即可

模型权重& tokenizer准备

百舸平台已经在对象存储BOS中预置模型权重,您可以从对应地域的BOS路径中下载数据。

openpi/π0.5 模型权重

```
bos:/aihc-models-bj/openpi/pi0.5/ckpt/pi05_base
```

Tokenizer：

```
bos:/aihc-models-bj/openpi/pi0.5/big_vision/paligemma_tokenizer.model
```

进行训练

1. 登录[百舸异构计算平台AIHC控制台](https://console.bce.baidu.com/aihc/resources)。
2. 进入 **分布式训练** 列表页面，点击**创建任务**，填写任务配置信息：
   - 基础信息
     - 任务名称：输入您的自定义任务名称
     - 资源池&队列：选择已创建的资源池和队列
     - 优先级：默认 中 优先级，无需更改
     - 训练框架：选择Pytorch
     - 任务创建方式： 选择 基于开源模型训练模版创建
     - 模型：选择对应的模型和训练方式
   - 环境配置
     - 镜像地址：无需更改，平台默认提供
     - 执行命令：系统已经默认训练的启动参数，需要替换`DATA_PATH`、`TRAINING_CONFIG`等配置信息
       - `DATA_PATH`：数据集的路径
       - `DATA_CONFIG`：如使用上述平台提供的libero数据集，这里则不填写，使用默认的DATA_CONFIG。如自定义数据集，则需要配置此文件
       - `TRAINING_CONFIG`：如使用平台预置的数据集、训练配置等，无需填写，使用默认的TRAINING_CONFIG；如需要自定义更多的训练参数等配置，则需要配置此文件
       - `TRAINING_CONFIG_NAME`：如使用上述平台提供的libero数据集，则无需填写；如自定义数据集，则需要填写
       - `PALIGEMMA_TOKENIZER_PATH`：模型的tokenizer路径
       - `REPO_ID`：如使用上述平台提供的libero数据集，则无需填写；如自定义数据集，则需要自定义填写
       - `BASE_CHECKPOINT_LOAD_PATH`：模型原始权重的路径
       - `CHECKPOINT_SAVE_BASEDIR`：模型训练后权重的保存路径
       - `EXP_NAME`：会拼接到`CHECKPOINT_SAVE_BASEDIR`后作为子目录存储ckpt；若开启了wandb，则这里默认为实验的名称
       - `WANDB_ENABLE`：是否开启wandb。如开启，需要额外填写启动命令中的wandb的配置
         - `WANDB_API_KEY`：必填，Wandb 账户的**认证密钥**，用于身份验证和访问权限。**必须设置**才能与 Wandb 服务端通信
         - `WANDB_MODE`：必填，设置为online
         - `export http_proxy`：设置代理，可以连接wandb服务器
         - `export https_proxy`：设置代理，可以连接wandb服务器`
     - 环境变量：无需更改
   - 资源配置
     - 实例数：这里选择1，目前仅支持单机训练
     - 加速芯片：使用A800训练，单实例8卡
     - 共享内存：默认10Gi
     - RDMA：默认开启，无需更改
   - 设置数据源
     - 存储类型：这里选择我们提前准备的PFS
     - PFS源路径：PFS的挂载路径，默认根目录，按实际情况填写
     - 挂载路径：容器内的挂载路径
3. 点击 **完成**，提交训练任务。

观察训练迭代

任务提交后，您可以通过任务日志观察训练的迭代情况

基于百舸平台通过P800进行Qwen-2.5模型训练

AIAK-Training-LLM是百度智能云基于百度百舸·AI计算平台，面向大模型训练场景提供的最佳实践解决方案配套AI加速工具，帮助模型开发者高效完成大规模深度学习分布式训练，提升训练效率，相比开源Megatron-LLM性能明显提升。关于AIAK-Training-LLM的使用说明，详见这里[这里](https://cloud.baidu.com/doc/AIHC/s/Alyo476jr)

这里我们以Qwen/Qwen2.5-72B-Instruct模型为示例，介绍如何使用AIAK-Training-LLM在P800上进行训练。

资源准备
算力资源
使用AIAK加速训练Qwen2.5-72B-Instruct，默认使用4台P800*8卡的节点

存储资源
为保证训练的性能，推荐使用并行文件存储PFS，以达到分布式加速训练效果。

开发机环境准备
创建开发机用于下载模型、数据集，并进行模型权重转换、数据集预处理等工作。

开发机创建：
在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【开发机】-【创建实例】开始创建开发机并填写开发机配置信息：

* 基本信息
  * 实例名称：输入您为开发机命名的名称，例如：P800-dev-test
  * 资源池类型/资源池/队列：根据您的资源池类型选择已创建的资源池和队列
  * 资源规格：预处理无需GPU资源，可以选择不使用加速芯片，同时最好保证内存资源不少于16GB以保证预处理脚本正常运行，此处推荐使用128核CPU与256GB以上内存保证预处理效率

* 环境配置
  * 镜像地址：可以从训练任务处，通过模板创建获取，目前镜像地址为：registry.baidubce.com/aihc-aiak/aiak-training-llm:main_p800_v1.2_new
  * 云磁盘：推荐预留100GiB保证开发机正常运行
  * 启动命令：用于填写开发机启动自动运行命令，此处暂时留空
  * 环境变量：一般用于配合启动命令使用，此处暂时留空
  * 存储挂载：用于挂载云存储资源（PFS/BOS/CFS等），此处选择存储类型为PFS的资源，默认带出存储配置区，填写源路径与挂载路径，例如：源路径：/P800/test，挂载路径：/mnt/pfs/P800/test（后续相关存储路径与此高度相关）

* 访问配置&高级配置
  * 可以使用默认配置，此处不做修改

完成后点击确定，即可完成开发机创建。

BOS工具验证：
在开发机中，已经预置了BOS下载工具，输入以下命令以验证下载工具是否安装：

```
bcecmd
```
产生以下输出则为可用：

```
usage: bcecmd [--help] [-configure ['''] [--debug] [--conf-path CONF-PATH] [-multiupload-infos-path MULTIUPLOAD-INFOS-PATH] [-version]
```
反之，执行以下命令：

```
wget https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.9.zip  #下载
unzip linux-bcecmd-0.5.9.zip  #解压
ln linux-bcecmd-0.5.9/bcecmd /usr/sbin/  #设置为全局使用
```
模型&数据集准备
模型准备：
(1) 模型下载
huggingface地址：https://huggingface.co/Qwen/Qwen2.5-72B/tree/main

BOS地址：bos:/aihc-models-bj/Qwen/Qwen2.5-72B/

BOS参考下载指令（此处的第二个参数参考开发机创建-环境配置-存储挂载-挂载路径）：

```
bcecmd bos sync bos:/aihc-models-bj/Qwen/Qwen2.5-72B/ /mnt/pfs/P800/test/Qwen-2.5-72B-model/
```
(2) 模型CKPT预处理
进入开发机/workspace/AIAK-Training-LLM/examples/qwen2.5/checkpoint_convert/convert_qwen2.5_72b_hf_to_mcore.sh目录可以找到模型权重转换脚本，调整模型权重位置与数据集输入输出位置后即可运行：

```
#### ! /bin/bash

AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}
MEGATRON_PATH=${MEGATRON_PATH:-"/workspace/AIAK-Megatron"}
CONVERT_CHECKPOINT_PATH="$AIAK_TRAINING_PATH/tools/convert_checkpoint"

LOAD=/mnt/pfs/P800/test/model/qwen-2.5-72b
SAVE=/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8

python $CONVERT_CHECKPOINT_PATH/model.py \
    --load_platform=huggingface \
    --save_platform=mcore \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2.5-72b.json \
    --tensor_model_parallel_size=4 \
    --pipeline_model_parallel_size=8 \
    --megatron_path=$MEGATRON_PATH \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE \
    --no_load_optim \
    --no_save_optim \
```

```
    --safetensors

```
运行结束后显示INFO:root:Finished convert checkpoint huggingface -> mcore即为转换成功

(3) 模型CKPT结果
输出结果应在SAVE文件夹下应有一个release文件夹，一个latest_checkpointed_iteration.txt文本文档。

数据集准备：
(1) 数据集下载
百舸平台已经在对象存储BOS中预置相关数据集，您可以从对应地域的BOS路径中下载数据，也可以从其他地方导入数据集。

百度公共数据集：

https://aihc-rdw-bj.bj.bcebos.com/huggingface.co/datasets/TigerResearch/tigerbot-alpaca-zh-0.5m/tigerbot-alpaca-zh-0.5m.json

HuggingFace数据集：

https://huggingface.co/datasets/TigerResearch/tigerbot-alpaca-zh-0.5m

数据集样例：

```
{"instruction": "将以下短文中的所有“is”动词改为“was”。", "input": "The cake is delicious and the pie is too. The ice cream is melting in the sun and the lemonade is refreshing.", "output": "The cake was delicious and the pie was too. The ice cream was melting in the sun and the lemonade was refreshing."}
{"instruction": "分类以下三个国家的首都是哪个城市：", "input": "- 美国\n- 法国\n- 中国", "output": "- 美国：华盛顿特区\n- 法国：巴黎\n- 中国：北京"}
{"instruction": "为以下数学公式提供英文翻译：", "input": "f(x) = sin(x) + cos(x)", "output": "f(x) equals sine of x plus cosine of x."}
```
数据集样式（keys：instruction/input/output）

  <img src="https://rte.weiyun.baidu.com/wiki/attach/image/api/imageDownloadAddress?attachId=12fa2f0ba2534e5db726b446e403f4bd&docGuid=naDwU6isS7yelv" alt="数据集截图">

方法1:

开发机中通过wget工具下载：

```
wget https://aihc-rdw-bj.bj.bcebos.com/huggingface.co/datasets/TigerResearch/tigerbot-alpaca-zh-0.5m/tigerbot-alpaca-zh-0.5m.json
```
方法2:

开发机中命令行从HuggingFace下载：

```
#### 安装依赖库
pip install -U huggingface_hub
#### 此处需要复制自己的huggingface网站授权
hf auth login
#### 后边可以自己修改存储地址
hf download TigerResearch/tigerbot-alpaca-zh-0.5m --repo-type=dataset --local-dir /
```
方法3:

本地上传：

先通过web端点击下载符号直接下载相关模型，之后在开发机指定目录进行上传。

(2) 数据集预处理
进入/workspace/AIAK-Training-LLM/examples/qwen2.5/目录可以找到数据集预处理脚本，调整模型权重位置与数据集输入输出位置后即可运行，

```
#### 如果遇到依赖环境报错可尝试执行如下代码切换到最新python&cuda环境
source /root/miniconda/etc/profile.d/conda.sh && conda activate python310_torch25_cuda
```
Pretrain预处理：

```
#### !/bin/bash

MEGATRON_PATH=${MEGATRON_PATH:-"/workspace/AIAK-Megatron"}
AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}

TOKENIZER_PATH=${TOKENIZER_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b"}

input_data=/mnt/pfs/P800/test/datasets/hf-dataset/tigerbot-alpaca-zh-0.5m.json
output_prefix=/mnt/pfs/P800/test/datasets/hf-dataset/output/pretrain

PYTHONPATH=$MEGATRON_PATH:$AIAK_TRAINING_PATH:$PYTHONPATH \
    python ${AIAK_TRAINING_PATH}/tools/data_preprocess/preprocess_pretrain_data.py \
        --input ${input_data} \
        --output-prefix ${output_prefix} \
        --tokenizer-type HFTokenizer \
        --hf-tokenizer-path $TOKENIZER_PATH \
        --json-keys instruction input output \
        --workers 50 \
        --append-eod

```
需注意：请提前将第18行--json-keys后内容改成数据集key值，例如：数据集单条为以下情况

```
{"instruction": "将以下短文中的所有“is”动词改为“was”。", "input": "The cake is delicious and the pie is too. The ice cream is melting in the sun and the lemonade is refreshing.", "output": "The cake was delicious and the pie was too. The ice cream was melting in the sun and the lemonade was refreshing."}
```
则第十八行为：

```
    --json-keys instruction input output \
```
如果出现Killed终止，请适量提升开发机内存量（>= 16GB）

训练
SFT：
在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【分布式训练】-【创建任务】开始创建训练任务并填写训练任务配置信息：

* 基本信息
  * 任务名称：输入您为本次任务命名的名称，例如：P800-SFT-test
  * 可见范围：队列哪可见/仅创建人可见
  * 创建方式：【基于开源模型训练模版创建】
    * 模型：【qwen2.5】-> 【qwen2.5-72b】
    * 芯片类型：昆仑芯
    * 训练模版：AIAK加速版
    * 训练模式：选择【SFT】
    * 训练方法：全量更新

* 资源配置（先配置此项）
  * 资源池类型/资源池/队列：根据您的资源池类型选择P800资源所在的资源池和队列
  * 优先级：默认为高，无需修改
  * 训练框架：PyTorch，无需修改
  * 资源配额：请确保实例数为4，GPU为KUNLUNXIN-P800 * 8，CPU与内存在可配置条件下越大越好。
  * 共享内存：100GiB
  * 数据集挂载：本示例中为空，具体用挂载用法与存储挂载类似
  * 存储挂载：此处与开发机创建配置保持一致：源路径：/P800/test，挂载路径：/mnt/pfs/P800/test

* 环境信息
  * 镜像地址：由基本信息自动带出无需修改
  * 执行命令：需要自行配置数据集路径，Tokenizer路径，Checkpoint加载/保存路径，可通过环境变量注入或直接替换执行命令代码进行修改，例如(替换21-24行)：


```
DATA_PATH=${DATA_PATH:-"/mnt/pfs/P800/test/datasets/hf-dataset/tigerbot-alpaca-zh-0.5m.json"}
TOKENIZER_PATH=${TOKENIZER_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b"}
CHECKPOINT_LOAD_PATH=${CHECKPOINT_LOAD_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8"}
CHECKPOINT_SAVE_PATH=${CHECKPOINT_SAVE_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8_trained"}
```
* 监控信息
  * Tensorboard：打开，并选择PFS，将配置源路径/P800/test加入默认源路径之前，后续此路径会自动注入执行命令中
  * 其余容错与诊断，监控信息可选择性打开。

点击确定，即可开始训练任务，此数据集的SFT任务在4*8卡环境下大约为2小时。

Pretrain：
在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【分布式训练】-【创建任务】开始创建训练任务并填写训练任务配置信息：

* 基本信息
  * 任务名称：输入您为本次任务命名的名称，例如：P800-Pretrain-test
  * 可见范围：队列哪可见/仅创建人可见
  * 创建方式：【基于开源模型训练模版创建】
    * 模型：【qwen2.5】-> 【qwen2.5-72b】
    * 芯片类型：昆仑芯
    * 训练模版：AIAK加速版
    * 训练模式：选择【Pretrain】
    * 训练方法：全量更新

* 资源配置（先配置此项）
  * 资源池类型/资源池/队列：根据您的资源池类型选择P800资源所在的资源池和队列
  * 优先级：默认为高，无需修改
  * 训练框架：PyTorch，无需修改
  * 资源配额：请确保实例数为4，GPU为KUNLUNXIN-P800 * 8，CPU与内存在可配置条件下越大越好。
  * 共享内存：100GiB
  * 数据集挂载：本示例中为空，具体用挂载用法与存储挂载类似
  * 存储挂载：此处与开发机创建配置保持一致：源路径：/P800/test，挂载路径：/mnt/pfs/P800/test

* 环境信息
  * 镜像地址：由基本信息自动带出无需修改
  * 执行命令：需要自行配置数据集路径，Tokenizer路径，Checkpoint加载/保存路径，可通过环境变量注入或直接替换执行命令代码进行修改，例如(替换21-24行)：

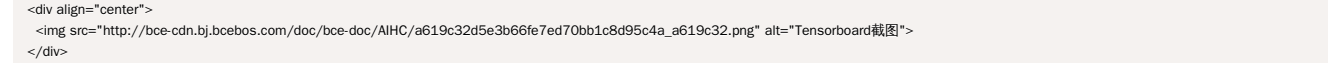

```
DATA_PATH=${DATA_PATH:-"/mnt/pfs/P800/test/datasets/hf-dataset/pretrain/pile-qwen_output_document"}
TOKENIZER_PATH=${TOKENIZER_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b"}
CHECKPOINT_LOAD_PATH=${CHECKPOINT_LOAD_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8"}
CHECKPOINT_SAVE_PATH=${CHECKPOINT_SAVE_PATH:-"/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8/pretrain"}
```
注意：此处的DATA_PATH为预处理后的文件前缀，例如预处理后生成四个文件分别为：

```
/mnt/pfs/P800/test/datasets/hf-dataset/pretrain/pile-qwen_output_document.idx
/mnt/pfs/P800/test/datasets/hf-dataset/pretrain/pile-qwen_output_document.bin
/mnt/pfs/P800/test/datasets/hf-dataset/pretrain/pile-qwen_instruction_document.bin
/mnt/pfs/P800/test/datasets/hf-dataset/pretrain/pile-qwen_input_document.bin
```
则此处DATA_PATH填入地址应为idx文件不包含后缀的地址，为：

```
/mnt/pfs/P800/test/datasets/hf-dataset/pretrain/pile-qwen_output_document
```
* 监控信息
  * Tensorboard：打开，并选择PFS，将配置源路径/P800/test加入默认源路径之前，后续此路径会自动注入执行命令中
  * 其余容错与诊断，监控信息可选择性打开。

点击确定，即可开始训练任务，此数据集的Pretrain任务在4*8卡环境下大约为14小时。

Tensorborad：
在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【分布式训练】，多选需要查看Tensorboard的任务，点击【对比Tensorboard】即可查看对比各任务的Tensorboard，此处为本次SFT与Pretrain任务的对比图：



权重转换
在开发机使用如下脚本进行权重转换，将mcore格式训练结果转换为hf格式用于后续部署工作：

```
#### ! /bin/bash

AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}
MEGATRON_PATH=${MEGATRON_PATH:-"/workspace/AIAK-Megatron"}
CONVERT_CHECKPOINT_PATH="$AIAK_TRAINING_PATH/tools/convert_checkpoint"

LOAD=/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8_trained/iter_0000500
SAVE=/mnt/pfs/P800/test/model/qwen-2.5-72b/Qwen2.5_72B_mcore_tp4pp8_trained/Qwen2.5-72B-Instruct_saved_hf/


python $CONVERT_CHECKPOINT_PATH/model.py \
    --load_platform=mcore \
    --save_platform=huggingface \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2.5-72b.json \
    --tensor_model_parallel_size=4 \
    --pipeline_model_parallel_size=8 \
    --megatron_path=$MEGATRON_PATH \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE \
    --no_load_optim \
    --no_save_optim \
    --safetensors
```
注意：LOAD地址为训练结果最后一轮迭代目录（mp_rank目录的父目录），本示例最终训练轮次为500，因此LOAD文件前缀为iter_0000500

至此，输出的文件即为可直接运行的HuggingFace格式文件，可以进行后续服务部署操作。

Qwen-2.5-VL模型训练

使用百舸平台进行Qwen-2.5-VL模型训练 v2.0

AIAK-Training-LLM是百度智能云基于百度百舸·AI计算平台，面向大模型训练场景提供的最佳实践解决方案配套AI加速工具，帮助模型开发者高效完成大规模深度学习分布式训练，提升训练效率，相比开源Megatron-LLM性能明显提升。关于AIAK-Training-LLM的使用说明，详见[这里](https://cloud.baidu.com/doc/AIHC/s/Alyo476jr)

针对于Qwen2.5vl模型，百度百舸团队基于自研大模型训练加速框架 AIAK-Training-LLM，在 SFT / Pretrain 阶段实现了多模态数据混合训练，并结合离线 Sequence Packing 技术，显著提升训练效率与资源利用率。

这里我们以Qwen2.5-VL-7B-Instruct模型单机训练混训为示例，介绍如何使用AIAK-Training-LLM在A800上进行训练，同时包含如何进行数据集离线预处理（Packing）功能。

资源准备
算力资源
使用AIAK加速训练Qwen2.5-VL-3B/7B/32B/72B-Instruct，请使用A800*8卡/P800*8 卡的节点

存储资源
为保证训练的性能，推荐使用并行文件存储PFS，以达到分布式加速训练效果。

开发机环境准备
创建开发机用于下载模型、数据集，并进行模型权重转换、数据集预处理等工作。

开发机创建：
在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【开发机】-【创建实例】开始创建开发机并填写开发机配置信息：

* 基本信息
    * 实例名称：输入您为开发机命名的名称，例如：VL-dev-test
    * 资源池类型/资源池/队列：根据您的资源池类型选择已创建的资源池和队列
    * 资源规格：预处理无需GPU资源，可以选择不使用加速芯片，同时最好保证内存资源不少于16GB以保证预处理脚本正常运行，此处推荐使用128核CPU与256GB以上内存保证预处理效率

* 环境配置
    * 镜像地址：可以从训练任务处通过模板创建获取最新镜像，当前镜像为：registry.baidubce.com/aihc-aiak/aiak-training-llm:ubuntu24.04-cu12.9-torch2.8.0-py312_v2.2.6.4_release_fixed
    * 云磁盘：推荐预留100GiB保证开发机正常运行
    * 启动命令：用于填写开发机启动自动运行命令，此处暂时留空
    * 环境变量：一般用于配合启动命令使用，此处暂时留空
    * 存储挂载：用于挂载云存储资源（PFS/BOS/CFS等），此处选择存储类型为PFS的资源，默认带出存储配置区，填写源路径与挂载路径，例如：源路径：/vl，挂载路径：/mnt/pfs/（后续相关存储路径与此高度相关）

* 访问配置&高级配置
    * 可以使用默认配置，此处不做修改

完成后点击确定，即可完成开发机创建。

BOS工具验证：
在开发机中，已经预置了BOS下载工具，输入以下命令以验证下载工具是否安装：

```
bcecmd
```
产生以下输出则为可用：

```
usage: bcecmd [--help] [-configure ['']] [--debug] [--conf-path CONF-PATH] [-multiupload-infos-path MULTIUPLOAD-INFOS-PATH] [-version]
```

反之，执行以下命令（请确保在开发机环境下运行）：

```
wget https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.9.zip  #下载
unzip linux-bcecmd-0.5.9.zip  #解压
ln linux-bcecmd-0.5.9/bcecmd /usr/sbin/ #设置为全局使用
```

模型&数据集准备

模型准备：

(1) 模型下载

本次VL 共支持四个模型，均已内置于BOS 中：

模型名称	BOS 位置
Qwen2.5-VL-3B-Instruct	bos:/aihc-models-bj/Qwen/Qwen2.5-VL-3B-Instruct/
Qwen2.5-VL-7B-Instruct	bos:/aihc-models-bj/Qwen/Qwen2.5-VL-7B-Instruct/
Qwen2.5-VL-32B-Instruct	bos:/aihc-models-bj/Qwen/Qwen2.5-VL-32B-Instruct/
Qwen2.5-VL-72B-Instruct	bos:/aihc-models-bj/Qwen/Qwen2.5-VL-72B-Instruct/

BOS参考下载指令（此处的第二个参数参考开发机创建-环境配置-存储挂载-挂载路径）：

```
bcecmd bos sync bos:/aihc-models-bj/Qwen/Qwen2.5-VL-3B-Instruct/ /mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-3B-Instruct/
```

模型 BOS 地址：

```
bcecmd bos sync bos:/aihc-models-bj/Qwen/Qwen2.5-VL-7B-Instruct/ /mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/
```

模型 BOS 地址：

```
bcecmd bos sync bos:/aihc-models-bj/Qwen/Qwen2.5-VL-32B-Instruct/ /mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-32B-Instruct/
```

模型 BOS 地址：

```
bcecmd bos sync bos:/aihc-models-bj/Qwen/Qwen2.5-VL-72B-Instruct/ /mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-72B-Instruct/
```

(2) 模型CKPT预处理

进入开发机/workspace/AIAK-Training-LLM/examples/qwen2_5_vl/目录可以找到模型权重转换脚本，调整模型权重位置与数据集输入输出位置后即可运行（这里以 7B 模型为例）：

```

#### ! /bin/bash

AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}
AIAK_MAGATRON_PATH=${AIAK_MAGATRON_PATH:-"/workspace/AIAK-Magatron"}
CONVERT_CHECKPOINT_PATH="$AIAK_TRAINING_PATH/tools/convert_checkpoint"

LOAD=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct
SAVE=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/qwen2_5-vl-7b-tp1-pp1

SAVE_LANGUAGE_MODEL=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/aiak-training-llm/tmp/language-mcore
SAVE_VISION_MODEL=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/aiak-training-llm/tmp/vision-model-mcore
SAVE_ADAPTER=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/aiak-training-llm/tmp/adapter-mcore
SAVE_PATCH=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/aiak-training-llm/tmp/patch-mcore

TP=1
PP=1

#### qwen2
python $CONVERT_CHECKPOINT_PATH/model.py \
    --load_platform=huggingface \
    --save_platform=mcore \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/qwen2_5.json \
    --tensor_model_parallel_size=$TP \
    --pipeline_model_parallel_size=$PP \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_LANGUAGE_MODEL \
    --safetensors \
    --no_save_optim \
    --no_load_optim

#### vit
python $CONVERT_CHECKPOINT_PATH/model.py \
    --load_platform=huggingface \
    --save_platform=mcore \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/vision-model.json \
    --tensor_model_parallel_size=$TP \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_VISION_MODEL \
    --safetensors \
    --no_save_optim \
    --no_load_optim

#### adapter
python $CONVERT_CHECKPOINT_PATH/custom/qwen2_vl/adapter.py \
    --load_platform=huggingface \
    --save_platform=mcore \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/adapter.json \
    --tensor_model_parallel_size=$TP \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_ADAPTER

#### vision patch in vit
python $CONVERT_CHECKPOINT_PATH/custom/qwen2_vl/vision_patch.py \
    --load_platform=huggingface \
    --save_platform=mcore \
```

```
    --tensor_model_parallel_size=$TP \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/vision-patch.json \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_PATCH

#### merge
python $CONVERT_CHECKPOINT_PATH/custom/qwen2_vl/merge_megatron.py \
    --megatron_path $AIAK_MAGATRON_PATH \
    --language_model_path $SAVE_LANGUAGE_MODEL/release \
    --vision_model_path $SAVE_VISION_MODEL/release \
    --vision_patch $SAVE_PATCH/release \
    --adapter_path $SAVE_ADAPTER/release \
    --save_ckpt_path $SAVE/release \
    --tensor_model_parallel_size $TP \
    --pipeline_model_parallel_size $PP

echo release > $SAVE/latest_checkpointed_iteration.txt
rm -rf $SAVE_LANGUAGE_MODEL
rm -rf $SAVE_VISION_MODEL
rm -rf $SAVE_ADAPTER
rm -rf $SAVE_PATCH
```

运行结束后显示INFO:root:Finished convert checkpoint huggingface -> mcore即为转换成功

需注意：对于72B 模型中有一行--custom_pipeline_layers配置，需要注意将此配置项与训练时一致

```
--custom_pipeline_layers 20,20,20,20
```

(3) 模型CKPT结果
输出结果应在SAVE文件夹下应有一个release文件夹，一个latest_checkpointed_iteration.txt文本文档。

数据集准备：
(1) 预置数据集下载
百舸平台已经在对象存储BOS中预置相关数据集（energon加载器格式），您可以从对应地域的BOS路径中下载数据：

简易版数据集 BOS 地址：bos:/aihc-rdw-bj/dataset/qwen2.5-vl-dataset/demo/

```
bcecmd bos sync bos:/aihc-rdw-bj/dataset/qwen2.5-vl-dataset/demo/ /mnt/pfs/qwen-2-5-vl-dataset
```

数据集格式如下：

```
pretrain-0.tar      pretrain-1.tar.idx  pretrain-3.tar      pretrain-4.tar.idx  pretrain-6.tar
pretrain-0.tar.idx  pretrain-2.tar      pretrain-3.tar.idx  pretrain-5.tar      pretrain-6.tar.idx
pretrain-1.tar      pretrain-2.tar.idx  pretrain-4.tar      pretrain-5.tar.idx
```

(2) 数据集预处理（可选）
用户也可以使用图文对结构的如下格式数据集（包含问答对/图片），需要先将数据集转换为energon 加载器格式：

```
[
  {
    "messages": [
      {
        "content": "<image>Who are they?",
        "role": "user"
      },
      {
        "content": "They're Kane and Gretzka from Bayern Munich.",
        "role": "assistant"
      },
      {
        "content": "What are they doing?<image>",
        "role": "user"
      },
      {
        "content": "They are celebrating on the soccer field.",
        "role": "assistant"
      }
    ],
    "images": [
      "mllm_demo_data/1.jpg",
      "mllm_demo_data/2.jpg"
    ]
  },
  ...
]
```

您可以下载如下数据集进行实验验证：

```
bcecmd bos sync bos:/aihc-rdw-bj/huggingface.co/datasets/LLaVA-Pretrain/ /mnt/pfs/Qwen-2.5-VL-data/datasets/
```

使用/workspace/AIAK-Training-LLM/tools/data_preprocess/convert_to_webdataset.py进行格式转换：

```
python /workspace/AIAK-Training-LLM/tools/data_preprocess/convert_to_webdataset.py \
    --output_dir /mnt/pfs/Qwen-2.5-VL-data/datasets/output \
    --json_file /mnt/pfs/Qwen-2.5-VL-data/datasets/blip_laion_cc_sbu_558k.json \
    --image_dir /mnt/pfs/Qwen-2.5-VL-data/datasets/ \
    --media image \
    --columns_messages messages \
    --maxsize 10000000000
```

--output_dir目录下的输出文件后续可直接作为训练用数据集。

您也可以直接下载已经处理好的energon 加载器格式数据集：

```
bcecmd bos sync bos:/aihc-rdw-bj/huggingface.co/datasets/wds/ /mnt/pfs/Qwen-2.5-VL-data/datasets/output/
```

(3) 数据Packing预处理（可选）

为提升混训效率，百度AIAK-Traing加速团队提供了离线数据集预处理方式，可以进一步提升后续训练效率。

1⃣ 使用/workspace/AIAK-Training-LLM/examples/qwen2_5_vl/finetuning目录下的preprocess_data.sh脚本开始对数据集进行预处理：

```
#### ! /bin/bash
#### The script needs to be run on at least 1 nodes.
#### export WORLD_SIZE=2
#### export RANK=0
#### export MASTER_ADDR=192.168.81.53
#### export MASTER_PORT=6000

MEGATRON_PATH=${MEGATRON_PATH:-"/workspace/AIAK-Megatron"}
AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}

DATA_PATH=${DATA_PATH:-"mnt/pfs/qwen-2-5-vl-dataset/wds"}

TOKENIZER_PATH=${TOKENIZER_PATH:-"/mnt/cluster/Qwen2.5-VL-7B-Instruct/"}

CHECKPOINT_PATH=${CHECKPOINT_PATH:-"/mnt/cluster/aiak-training-llm/qwen2_5-vl/qwen2_5-vl-7b-tp1-pp1"}

TENSORBOARD_PATH=${TENSORBOARD_PATH:-"/mnt/cluster/aiak-training-llm/tensorboard-log/qwen2_5-vl-7b"}

GPUS_PER_NODE=8

#### Change for multinode config
MASTER_ADDR=${MASTER_ADDR:-"localhost"}
MASTER_PORT=${MASTER_PORT:-"6000"}
NNODES=${WORLD_SIZE:-"1"}
NODE_RANK=${RANK:-"0"}

DISTRIBUTED_ARGS=(
    --nproc_per_node $GPUS_PER_NODE
    --nnodes $NNODES
    --node_rank $NODE_RANK
    --master_addr $MASTER_ADDR
    --master_port $MASTER_PORT
)

#### or you can setup qwen2_5-vl-7b by using the following command
MODEL_ARGS=(
    --model-name qwen2_5-vl-7b
)

DATA_ARGS=(
    --tokenizer-type HFTokenizer \
    --hf-tokenizer-path $TOKENIZER_PATH \
    --data-path $DATA_PATH
    --dataloader-type external
    --split 100,0,0
    --num-workers 8
    --chat-template qwen2-vl
    --packing-batch-size 10
    --packing-sft-data
    --packing-pretrain-data
    --preprocess-data-on-cpu
)

TRAINING_ARGS=(
    --training-phase sft
    --trainable-modules language_model adapter
    --seq-length 4096
    --max-position-embeddings 8192
    --micro-batch-size 1
    --global-batch-size 32
    --train-iters 50000
    --initial-loss-scale 65536
    --bf16
    --save-interval 10000000
    --ckpt-format torch
)

MODEL_PARALLEL_ARGS=(
    --attention-backend flash
    --pipeline-model-parallel-size 1
    --tensor-model-parallel-size 1
    --use-distributed-optimizer
    --distributed-backend gloo
    #--sequence-parallel
)

PYTHONPATH=$MEGATRON_PATH:$AIAK_TRAINING_PATH:$PYTHONPATH \
    torchrun ${DISTRIBUTED_ARGS[@]} \
    $AIAK_TRAINING_PATH/tools/data_preprocess/preprocess_megatron_energon_dataset.py \
    ${MODEL_ARGS[@]} \
    ${DATA_ARGS[@]} \
    ${TRAINING_ARGS[@]} \
    ${MODEL_PARALLEL_ARGS[@]} \
    --output-path /mnt/pfs/qwen-2-5-vl-dataset/data
```

脚本中需要注意的点：

1.始能packing功能需要配置如下参数，packing-batch-size可以根据实际情况进行大小调整，48-50 行：

```
```

```
    --packing-batch-size 10
    --packing-sft-data
    --packing-pretrain-data
```
2.如预处理执行在cpu环境需要加上下面参数，51 行

```
--preprocess-data-on-cpu
```
3.预处理tp和pp统一配置成1、1，将预处理数据集的颗粒度最小化，70-71 行

```
    --pipeline-model-parallel-size 1
    --tensor-model-parallel-size 1
```
4.分布式后端使用gloo，73 行

```
--distributed-backend gloo
```
5.指定预处理输出路径，84 行

```
--output-path /mnt/pfs/qwen-2-5-vl-dataset/data
```
执行完成后可在输出路径找到对应数据集输出结果。

2🔲 再执行/workspace/AIAK-Training-LLM/tools/data_preprocess/preprocess_megatron_energon_dataset_stage2.py python文件，对预处理完的数据进行拼接。

注意：此处 world-size 需要能被下面 tp*pp 整除同时需保持与后续训练中配置一致，文件路径为预处理脚本中注入地址。

```
python preprocess_megatron_energon_dataset_stage2.py --data-path /mnt/pfs/qwen-2-5-vl-dataset/data --world-size 8 --tp 1 --pp 1 --clean-temp
```
训练
数据在线Packing场景：
在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【分布式训练】-【创建任务】开始创建训练任务并填写训练任务配置信息：

* 基本信息
    * 任务名称：输入您为本次任务命名的名称，例如：vl-7b-SFT-test
    * 可见范围：队列哪可见/仅创建人可见
    * 创建方式：【基于开源模型训练模版创建】
        * 模型：【qwen2.5】-> 【qwen2.5-vl-7b】
        * 芯片类型：NVIDIA/昆仑芯，此处以 A800 为例
        * 训练模版：AIAK加速版
        * 训练模式：选择【SFT】 / 【Pretrain】，此处以 SFT 为例
        * 训练方法：全量更新

* 资源配置（先配置此项）
    * 资源池类型/资源池/队列：根据您的资源池类型选择A800资源所在的资源池和队列
    * 优先级：默认为高，无需修改
    * 训练框架：PyTorch，无需修改
    * 资源配额：请确保GPU不少于A800 * 4（推荐使用单机 8 卡配置运行以保证效率最大化），CPU与内存在可配置条件下越大越好，在更多资源条件下可以适当提高--global-batch-size等参数
    * 共享内存：100GiB
    * 数据集挂载：本示例中为空，具体用挂载用法与存储挂载类似
    * 存储挂载：此处与开发机创建配置保持一致：源路径：/vl，挂载路径：/mnt/pfs

* 环境信息
    * 镜像地址：由基本信息自动带出无需修改
    * 执行命令：需要自行配置数据集路径，Tokenizer路径，Checkpoint加载/保存路径，可通过环境变量注入或直接替换执行命令代码进行修改，例如(替换15-18行)：
    * MODEL_PARALLEL_ARGS配置：需要注意，此处 --pipeline-model-parallel-size，--tensor-model-parallel-size，--custom-pipeline-layers（仅限 72B 模型），需要保证与 CKPT 转换时同相同精度，参考 84-85 行。


```
#### ! /bin/bash
#### The script needs to be run on at least 1 nodes.

MEGATRON_PATH=${MEGATRON_PATH:-"/workspace/AIAK-Megatron"}
AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}

#### ----------------------------------------------------------------------------------
#### 用户自定义配置:
#### DATA_PATH: 数据集路径
#### TOKENIZER_PATH: Tokenizer路径
#### CHECKPOINT_LOAD_PATH: Checkpoint加载路径
#### CHECKPOINT_SAVE_PATH: Checkpoint保存路径
#### TENSORBOARD_PATH: Tensorboard日志保存路径, 在创建任务时开启Tensorboard后百舸自动注入环境变量
#### ----------------------------------------------------------------------------------
DATA_PATH="/mnt/pfs/qwen-2-5-vl-dataset/wds"
TOKENIZER_PATH="/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct"
CHECKPOINT_LOAD_PATH="/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/qwen2_5-vl-7b-tp1-pp1"
CHECKPOINT_SAVE_PATH="/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/qwen2_5-vl-7b-tp1-pp1/1"
TENSORBOARD_PATH=$AIHC_TENSORBOARD_LOG_PATH

GPUS_PER_NODE=8

#### Change for multinode config
MASTER_ADDR=${MASTER_ADDR:-"localhost"}
MASTER_PORT=${MASTER_PORT:-"6000"}
NNODES=${WORLD_SIZE:-"1"}
NODE_RANK=${RANK:-"0"}

DISTRIBUTED_ARGS=(
    --nproc_per_node $GPUS_PER_NODE
    --nnodes $NNODES
    --node_rank $NODE_RANK
    --master_addr $MASTER_ADDR
```

```
    --master_port $MASTER_PORT
)

#### or you can setup qwen2_5-vl-7b by using the following command
MODEL_ARGS=(
    --model-name qwen2_5-vl-7b
)

DATA_ARGS=(
    --tokenizer-type HFTokenizer \
    --hf-tokenizer-path $TOKENIZER_PATH \
    --data-path $DATA_PATH
    --dataloader-type external
    --split 100,0,0
    --num-workers 16
    --chat-template qwen2-vl
)

TRAINING_ARGS=(
    --training-phase sft
    --trainable-modules language_model adapter
    --seq-length 1024
    --max-position-embeddings 4096
    --init-method-std 0.02
    --micro-batch-size 1
    --global-batch-size 128
    --lr 0.0002
    --min-lr 1.0e-5
    --clip-grad 1.0
    --weight-decay 0.01
    --optimizer adam
    --adam-beta1 0.9
    --adam-beta2 0.95
    --adam-eps 1e-05
    --norm-epsilon 1e-6
    --train-iters 5000
    --lr-decay-iters 5000
    --lr-decay-style cosine
    --lr-warmup-fraction 0.002
    --initial-loss-scale 65536
    --bf16
    --load $CHECKPOINT_LOAD_PATH
    --save $CHECKPOINT_SAVE_PATH
    --save-interval 10000000
    --ckpt-format torch
    --dataloader-save ${CHECKPOINT_SAVE_PATH}/dataloader
)

MODEL_PARALLEL_ARGS=(
    --attention-backend flash
    --pipeline-model-parallel-size 1
    --tensor-model-parallel-size 1
    --use-distributed-optimizer
    --overlap-grad-reduce
    --overlap-param-gather
    --distributed-backend nccl
)

LOGGING_ARGS=(
    --log-interval 1
    --tensorboard-dir ${TENSORBOARD_PATH}
    --log-timers-to-tensorboard
)

if [ -n "${WANDB_API_KEY}" ]; then
    LOGGING_ARGS+=(
        --wandb-project ${WANDB_PROJECT}
        --wandb-exp-name ${WANDB_NAME}
    )
fi

PYTHONPATH=$MEGATRON_PATH:$AIAK_TRAINING_PATH:$PYTHONPATH \
    torchrun ${DISTRIBUTED_ARGS[@]} \
    $AIAK_TRAINING_PATH/aiak_training_llm/train.py \
    ${MODEL_ARGS[@]} \
    ${DATA_ARGS[@]} \
    ${TRAINING_ARGS[@]} \
    ${MODEL_PARALLEL_ARGS[@]} \
    ${LOGGING_ARGS[@]}
```

* 监控信息
    * Tensorboard：打开，并选择PFS，后续此路径会自动注入执行命令中
    * 其余容错与诊断，监控信息可选择性打开。

点击确定，即可开始训练任务，此数据集的SFT任务在1*8卡环境下使用该参数大约为1000iter/小时。

数据离线Packing：
您也可以使用先前离线处理的数据集，来体验更高效的训练过程，在百度百舸.AI计算平台[控制台](https://console.bce.baidu.com/aihc/resources)，点击【分布式训练】-【创建任务】开始创建训练任务并填写训练任务配置信息（在资源处与在线版唯一区别就是把显卡从 1*8 变成了 2*4）：

* 基本信息
    * 任务名称：输入您为本次任务命名的名称，例如：vl-7b-SFT-test-offline
    * 可见范围：队列哪可见/仅创建人可见
    * 创建方式：【基于开源模型训练模版创建】
        * 模型：【qwen2.5】-> 【qwen2.5-vl-7b】
        * 芯片类型：NVIDIA A800
        * 训练模版：AIAK加速版
        * 训练模式：选择【SFT】，此处以 SFT 为例
        * 训练方法：全量更新

* 资源配置（先配置此项）
  * 资源池类型/资源池/队列：根据您的资源池类型选择A800资源所在的资源池和队列
  * 优先级：默认为高，无需修改
  * 训练框架：PyTorch，无需修改
  * 资源配额： 实例数*1，显卡为 A800*8
  * 共享内存：100GiB
  * 数据集挂载：本示例中为空，具体用挂载用法与存储挂载类似
  * 存储挂载：此处与开发机创建配置保持一致：源路径：/vl，挂载路径：/mnt/pfs

* 环境信息
  * 镜像地址：由基本信息自动带出无需修改
  * 执行命令：需要自行配置数据集路径，Tokenizer路径，Checkpoint加载/保存路径，可通过环境变量注入或直接替换执行命令代码进行修改，例如(替换15-18行)：
  * MODEL_PARALLEL_ARGS配置：需要注意，此处 --pipeline-model-parallel-size，--tensor-model-parallel-size，--custom-pipeline-layers（仅限 72B 模型），需要保证与 CKPT 转换时同相同精度，参考 88-89 行。
  * 同时声明使用离线数据集格式进行训练【**--is-tokenized-data**】，参考 50-53 行。

```
#### ! /bin/bash
#### The script needs to be run on at least 1 nodes.

MEGATRON_PATH=${MEGATRON_PATH:-"/workspace/AIAK-Megatron"}
AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}

#### ----------------------------------------------------------------------------------
#### 用户自定义配置:
#### DATA_PATH: 数据集路径
#### TOKENIZER_PATH: Tokenizer路径
#### CHECKPOINT_LOAD_PATH: Checkpoint加载路径
#### CHECKPOINT_SAVE_PATH: Checkpoint保存路径
#### TENSORBOARD_PATH: Tensorboard日志保存路径, 在创建任务时开启Tensorboard后百舸自动注入环境变量
#### ----------------------------------------------------------------------------------
DATA_PATH="/mnt/pfs/qwen-2-5-vl-dataset/data"
TOKENIZER_PATH="/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct"
CHECKPOINT_LOAD_PATH="/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/qwen2_5-vl-7b-tp1-pp1"
CHECKPOINT_SAVE_PATH="/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/qwen2_5-vl-7b-tp1-pp1/1"
TENSORBOARD_PATH=$AIHC_TENSORBOARD_LOG_PATH

GPUS_PER_NODE=8

#### Change for multinode config
MASTER_ADDR=${MASTER_ADDR:-"localhost"}
MASTER_PORT=${MASTER_PORT:-"6000"}
NNODES=${WORLD_SIZE:-"1"}
NODE_RANK=${RANK:-"0"}

DISTRIBUTED_ARGS=(
    --nproc_per_node $GPUS_PER_NODE
    --nnodes $NNODES
    --node_rank $NODE_RANK
    --master_addr $MASTER_ADDR
    --master_port $MASTER_PORT
)

#### or you can setup qwen2_5-vl-7b by using the following command
MODEL_ARGS=(
    --model-name qwen2_5-vl-7b
)

DATA_ARGS=(
    --tokenizer-type HFTokenizer \
    --hf-tokenizer-path $TOKENIZER_PATH \
    --data-path $DATA_PATH
    --dataloader-type external
    --split 100,0,0
    --num-workers 16
    --chat-template qwen2-vl
    --packing-batch-size 10
    --is-tokenized-data
    --packing-pretrain-data
    --packing-sft-data
)

TRAINING_ARGS=(
    --training-phase sft
    --trainable-modules language_model adapter
    --seq-length 1024
    --max-position-embeddings 4096
    --init-method-std 0.02
    --micro-batch-size 1
    --global-batch-size 128
    --lr 0.0002
    --min-lr 1.0e-5
    --clip-grad 1.0
    --weight-decay 0.01
    --optimizer adam
    --adam-beta1 0.9
    --adam-beta2 0.95
    --adam-eps 1e-05
    --norm-epsilon 1e-6
    --train-iters 5000
    --lr-decay-iters 5000
    --lr-decay-style cosine
    --lr-warmup-fraction 0.002
    --initial-loss-scale 65536
    --bf16
    --load $CHECKPOINT_LOAD_PATH
    --save $CHECKPOINT_SAVE_PATH
```

```
    --save-interval 10000000
    --ckpt-format torch
    --dataloader-save ${CHECKPOINT_SAVE_PATH}/dataloader
)

MODEL_PARALLEL_ARGS=(
    --attention-backend flash
    --pipeline-model-parallel-size 1
    --tensor-model-parallel-size 1
    --use-distributed-optimizer
    --overlap-grad-reduce
    --overlap-param-gather
    --distributed-backend nccl
)

LOGGING_ARGS=(
    --log-interval 1
    --tensorboard-dir ${TENSORBOARD_PATH}
    --log-timers-to-tensorboard
)

if [ -n "${WANDB_API_KEY}" ]; then
    LOGGING_ARGS+=(
        --wandb-project ${WANDB_PROJECT}
        --wandb-exp-name ${WANDB_NAME}
    )
fi

PYTHONPATH=$MEGATRON_PATH:$AIAK_TRAINING_PATH:$PYTHONPATH \
    torchrun ${DISTRIBUTED_ARGS[@]} \
    $AIAK_TRAINING_PATH/aiak_training_llm/train.py \
    ${MODEL_ARGS[@]} \
    ${DATA_ARGS[@]} \
    ${TRAINING_ARGS[@]} \
    ${MODEL_PARALLEL_ARGS[@]} \
    ${LOGGING_ARGS[@]}
```

权重转换

在开发机使用如下脚本进行权重转换，将mcore格式训练结果转换为hf格式用于后续部署工作，转换脚本参考位置为/workspace/AIAK-Training-LLM/examples/qwen2_5_vl/checkpoint_convert/convert_qwen2_5_vl_7b_mcore_to_hf.sh：

注意：LOAD 目录地址为结果中最后一次迭代的文件夹名，此处只运行了 50 次迭代因此文件夹名为iter_0000050，具体文件夹信息可见目录中 latest_iteration.txt。

```
#### ! /bin/bash

AIAK_TRAINING_PATH=${AIAK_TRAINING_PATH:-"/workspace/AIAK-Training-LLM"}
AIAK_MAGATRON_PATH=${AIAK_MAGATRON_PATH:-"/workspace/AIAK-Megatron"}
CONVERT_CHECKPOINT_PATH="$AIAK_TRAINING_PATH/tools/convert_checkpoint"

SAVE=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/hf
LOAD=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/qwen2_5-vl-7b-tp1-pp1/1/iter_0000050/

SAVE_LANGUAGE_MODEL=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/tmp/language-expert-hf
SAVE_VISION_MODEL=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/tmp/vision-model-hf
SAVE_ADAPTER=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/tmp/adapter-hf
SAVE_PATCH=/mnt/pfs/qwen-2-5-vl-model/Qwen2.5-VL-7B-Instruct/output/tmp/patch-hf

TP=1
PP=1

#### llama: language expert
python $CONVERT_CHECKPOINT_PATH/model.py \
    --load_platform=mcore \
    --megatron_path $AIAK_MAGATRON_PATH \
    --save_platform=huggingface \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/qwen2_5.json \
    --tensor_model_parallel_size=$TP \
    --pipeline_model_parallel_size=$PP \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_LANGUAGE_MODEL \
    --safetensors \
    --no_save_optim \
    --no_load_optim

#### vit

if [[ $PP -eq 1 ]]; then
    LOAD_PATH=$LOAD
else
    LOAD_PATH=$LOAD/tmp/
    mkdir -p $LOAD_PATH
    for ((i=0;i<$TP;i++)); do
        from=`printf "mp_rank_%02d_000" $i`
        to=`printf "mp_rank_%02d" $i`
        cp -r $LOAD/$from $LOAD_PATH/$to
    done
fi

python $CONVERT_CHECKPOINT_PATH/model.py \
    --load_platform=mcore \
    --save_platform=huggingface \
    --megatron_path $AIAK_MAGATRON_PATH \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/vision-model.json \
    --tensor_model_parallel_size=$TP \
    --pipeline_model_parallel_size=1 \
    --load_ckpt_path=$LOAD_PATH \
    --save_ckpt_path=$SAVE_VISION_MODEL \
    --safetensors \
    --no_save_optim \
```

```
    --no_save_optim \
    --no_load_optim

if [[ $LOAD != $LOAD_PATH ]]; then
    rm -rf $LOAD_PATH
fi

#### adapter
python $CONVERT_CHECKPOINT_PATH/custom/qwen2_vl/adapter.py \
    --load_platform=mcore \
    --save_platform=huggingface \
    --megatron_path $AIAK_MAGATRON_PATH \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/adapter.json \
    --tensor_model_parallel_size=$TP \
    --pipeline_model_parallel_size=$PP \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_ADAPTER

#### vision patch
python $CONVERT_CHECKPOINT_PATH/custom/qwen2_vl/vision_patch.py \
    --load_platform=mcore \
    --save_platform=huggingface \
    --megatron_path $AIAK_MAGATRON_PATH \
    --tensor_model_parallel_size=$TP \
    --pipeline_model_parallel_size=$PP \
    --common_config_path=$CONVERT_CHECKPOINT_PATH/config/qwen2_5-vl-7b/vision-patch.json \
    --load_ckpt_path=$LOAD \
    --save_ckpt_path=$SAVE_PATCH

#### merge
python $CONVERT_CHECKPOINT_PATH/custom/qwen2_vl/merge_huggingface.py \
    --megatron_path $AIAK_MAGATRON_PATH \
    --language_model_path $SAVE_LANGUAGE_MODEL \
    --vision_model_path $SAVE_VISION_MODEL \
    --vision_patch $SAVE_PATCH \
    --adapter_path $SAVE_ADAPTER \
    --save_ckpt_path $SAVE

#### BASE=/mnt/cluster/huggingface.co/Qwen/Qwen2_5-VL-7B-Instruct/
#### find $BASE -type f -not -iname '*safetensors*' -exec cp {} ${SAVE}/ ';'
rm -rf $SAVE_LANGUAGE_MODEL
rm -rf $SAVE_VISION_MODEL
rm -rf $SAVE_ADAPTER
rm -rf $SAVE_PATCH
```

最后输出结果为：Finished convert checkpoint mcore->huggingface即为正确的

Wan视频生成模型部署

Wan是一个开源的AI视频生成大模型，支持T2V（文本生成视频）和I2V（图像生成视频）两种模式。在ComfyUI中，百舸提供了定制化的JSON工作流和API调用方式，帮助您使用Wan模型生成高质量的视频内容。本文将以I2V（图像生成视频）为例，介绍如何通过ComfyUI部署Wan模型，并使用Wan生成视频。

限制条件
仅支持在全托管资源池，单机、单卡部署单实例服务。

部署Wan模型
1. 登录百舸AI计算平台，单击 **快速开始；**
2. 在左侧内容筛选区域的模型系列中，勾选 **通义万相**，搜索 Wan2.1-I2V-14B 模型；
3. 单击 **部署 **按钮，填写资源规格和流量接入方式；
4. 服务部署成功后可以在 **在线服务部署 **中查看服务的 API 调用地址和 WebUI 访问地址。

WebUI使用
服务部署成功后，在服务列表的 **调用信息 **中获取 WebUI地址，在浏览器中打开。

WebUI服务鉴权
首次登录WebUI，需要设置账户密码，本地会生成一个PASSWORD的文件用来保护密码信息。

设置密码后可以在百舸控制台此服务的输出日志中查看Token信息。

取消鉴权
如果不需要服务鉴权，可以更新服务，在启动命令中增加此参数`rm -r /root/workspace/ComfyUI/custom_nodes/ComfyUI-Login`

不支持鉴权的启动命令示例如下：

```
cd /root/workspace/ComfyUI/models/ && rm-r diffusion_models text_encoders vae
ln -s /mnt/aihc_model_dir/diffusion_models /root/workspace/ComfyUI/models/diffusion_models
ln -s /mnt/aihc_model_dir/text_encoders /root/workspace/ComfyUI/models/text_encoders
ln -s /mnt/aihc_model_dir/vae /root/workspace/ComfyUI/models/vae
rm -r /root/workspace/ComfyUI/custom_nodes/ComfyUI-Login
cd /root/workspace/ComfyUI/ && python3 main.py --listen 0.0.0.0 --port 8000
```
输入参数
****LoadImageNode （输入图片）****
方式一：从持久化存储中选择图片

平台默认从服务的`/root/workspace/ComfyUI/input`路径加载图片，您可以将输入的图片存储在持久化存储后挂载到此路径，在ComfyUI中选择该路径下任意一张图片。

方式二：填写URL

可以直接填写图片URL，例如：

方式三：上传本地图片

点击选择要上传的文件会拉起本地资源管理器，上传本地图片即可

PromptNode（填写Prompt）

生成视频

生成的视频默认保存在服务的容器`/root/workspace/ComfyUI/output`路径中，可以更新服务配置挂载具备写权限的PFS或BOS存储到该路径，生成的视频可以持久化保存。

API调用
查看访问地址
服务部署成功后，可以在 **在线服务部署** 的服务列表点击服务的 **调用地址 **获取公网或VPC内网访问地址和Token信息。

获取ComfyUI工作流 API
ComfyUI的API请求体取决于工作流配置。平台会提供基于当前模型的默认工作流导出的API JSON，可以从WebUI中导出API，也可以直接使用文档中的API示例。

T2V
替换请求示例中body里prompt中的内容

```
{
  "1": {
    "inputs": {
      "model_name": "wan2.1_t2v_14B_fp16.safetensors",
      "weight_dtype": "default"
    },
    "class_type": "ModelLoaderNode",
    "_meta": {
      "title": "ModelLoaderNode"
    }
  },
  "3": {
    "inputs": {
      "vae_name": "Wan2.1_VAE.pth"
    },
    "class_type": "VAELoaderNode",
    "_meta": {
      "title": "VAELoaderNode"
    }
  },
  "4": {
    "inputs": {
      "negative_prompt": "low quality, blurry, ugly, poorly drawn hands, deformed face, extra limbs, bad anatomy, low resolution, disfigured, unrealistic, cartoonish, watermark, text, signature, distorted proportions, creepy, glitch, jpeg artifacts"
    },
    "class_type": "NegativePromptNode",
    "_meta": {
      "title": "NegativePromptNode"
    }
  },
  "5": {
    "inputs": {
      "positive_prompt": "A graceful ancient Chinese beauty in flowing white hanfu stands on a moss-covered stone altar within a ruined forest temple under a starlit sky. Glowing magical symbols and celestial energy swirl around her as she gracefully casts a spell, her long black hair cascading like silk, adorned with delicate jade hairpins. She holds an ornate fan with glowing patterns, ethereal light illuminating her serene face. Cinematic lighting and volumetric fog enhance the mystical atmosphere, with her embroidered robe billowing elegantly in the wind. Ultra-detailed, 4K, highly realistic, in the style of greg rutkowski, artgerm, cinematic fantasy. Animation of magical energy swirling, slow motion aura forming, glowing runes pulsing, hanfu sleeves fluttering with every movement."
    },
    "class_type": "PositivePromptNode",
    "_meta": {
      "title": "PositivePromptNode"
    }
  },
  "6": {
    "inputs": {
      "prompt": [
        "5",
        0
      ],
      "negative_prompt": [
        "4",
        0
      ],
      "model": [
        "1",
        0
      ],
      "text_encoder": [
        "8",
        0
      ],
      "vae": [
        "3",
        0
      ],
      "num_inference_steps": 30,
      "seed": 1428,
      "tiled": true,
      "fps": 25,
      "num_frames": 81
    },
    "class_type": "ExVideoNode",
    "_meta": {
      "title": "ExVideoNode"
    }
  },
  "7": {
    "inputs": {
      "video-preview": "",
      "video": [
        "6",
        0
      ]
    },
    "class_type": "PreViewVideo",
    "_meta": {
      "title": "PreViewVideo"
```

```
    }
  },
  "8": {
    "inputs": {
      "text_encoder_name": "models_t5_umt5-xxl-enc-bf16.pth",
      "type": "stable_diffusion",
      "device": "default"
    },
    "class_type": "TextEncoderLoaderNode",
    "_meta": {
      "title": "TextEncoderLoaderNode"
    }
  }
}
```

****I2V ****

替换请求示例中body里prompt中的内容

```
{
  "1": {
    "inputs": {
      "model_name": "wan2.1_i2v_480p_14B_fp16.safetensors",
      "weight_dtype": "default"
    },
    "class_type": "ModelLoaderNode",
    "_meta": {
      "title": "ModelLoaderNode"
    }
  },
  "2": {
    "inputs": {
      "clip_name": "models_clip_open-clip-xlm-roberta-large-vit-huge-14.pth",
      "type": "stable_diffusion",
      "device": "default"
    },
    "class_type": "CLIPLoaderNode",
    "_meta": {
      "title": "CLIPLoaderNode"
    }
  },
  "3": {
    "inputs": {
      "text_encoder_name": "models_t5_umt5-xxl-enc-bf16.pth",
      "type": "stable_diffusion",
      "device": "default"
    },
    "class_type": "TextEncoderLoaderNode",
    "_meta": {
      "title": "TextEncoderLoaderNode"
    }
  },
  "4": {
    "inputs": {
      "vae_name": "Wan2.1_VAE.pth"
    },
    "class_type": "VAELoaderNode",
    "_meta": {
      "title": "VAELoaderNode"
    }
  },
  "5": {
    "inputs": {
      "image": "example.png",
      "url": ""
    },
    "class_type": "LoadImageNode",
    "_meta": {
      "title": "LoadImageNode"
    }
  },
  "6": {
    "inputs": {
      "prompt": [
        "7",
        0
      ],
      "negative_prompt": [
        "8",
        0
      ],
      "model": [
        "1",
        0
      ],
      "text_encoder": [
        "3",
        0
      ],
      "clip": [
        "2",
        0
      ],
      "vae": [
        "4",
        0
      ],
      "num_inference_steps": 30,
      "seed": 430,
      "tiled": true
```

```
      "fps": 25,
      "num_frames": 81,
      "image": [
        "5",
        0
      ]
    },
    "class_type": "ExImageToVideoNode",
    "_meta": {
      "title": "ExImageToVideoNode"
    }
  },
  "7": {
    "inputs": {
      "positive_prompt": "The Little Prince, fox, simple poses, gentle movement, Van Gogh style, starry sky, glowing stars, soft focus, dreamy bokeh, fluffy felt textures, vibrant colors, magical lighting, warm glow, minimalistic action, subtle animation, flowing yellow scarf, peaceful atmosphere, glowing lights, 3D felt figurines, enchanting fantasy, serene, soft swirling vortex"
    },
    "class_type": "PositivePromptNode",
    "_meta": {
      "title": "PositivePromptNode"
    }
  },
  "8": {
    "inputs": {
      "negative_prompt": "Fast movement, chaotic, complex actions, modern, technology, industrial, harsh lighting, dark, dystopian, unrefined, unnatural, flat, lifeless, no star effect"
    },
    "class_type": "NegativePromptNode",
    "_meta": {
      "title": "NegativePromptNode"
    }
  },
  "9": {
    "inputs": {
      "video-preview": "",
      "video": [
        "6",
        0
      ]
    },
    "class_type": "PreViewVideo",
    "_meta": {
      "title": "PreViewVideo"
    }
  }
}
```


发送推理请求

参考发送示例，在示例中输入访问地址、Token、Prompt 发送POST请求启动工作流，会返回此工作流执行的prompt_id。


```
curl --location --request POST '<访问地址>/prompt' \
--header 'Authorization: Bearer <TOKEN>' \
--header 'Content-Type: application/json' \
--data-raw '{
    "prompt":{
      "1": {
        "inputs": {
          "model_name": "wan2.1_t2v_14B_fp16.safetensors",
          "weight_dtype": "default"
        },
        "class_type": "ModelLoaderNode",
        "_meta": {
          "title": "ModelLoaderNode"
        }
      },
      "3": {
        "inputs": {
          "vae_name": "Wan2.1_VAE.pth"
        },
        "class_type": "VAELoaderNode",
        "_meta": {
          "title": "VAELoaderNode"
        }
      },
      "4": {
        "inputs": {
          "negative_prompt": "low quality, blurry, ugly, poorly drawn hands, deformed face, extra limbs, bad anatomy, low resolution, disfigured, unrealistic, cartoonish, watermark, text, signature, distorted proportions, creepy, glitch, jpeg artifacts"
        },
        "class_type": "NegativePromptNode",
        "_meta": {
          "title": "NegativePromptNode"
        }
      },
      "5": {
        "inputs": {
          "positive_prompt": "A graceful ancient Chinese beauty in flowing white hanfu stands on a moss-covered stone altar within a ruined forest temple under a starlit sky. Glowing magical symbols and celestial energy swirl around her as she gracefully casts a spell, her long black hair cascading like silk, adorned with delicate jade hairpins. She holds an ornate fan with glowing patterns, ethereal light illuminating her serene face. Cinematic lighting and volumetric fog enhance the mystical atmosphere, with her embroidered robe billowing elegantly in the wind. Ultra-detailed, 4K, highly realistic, in the style of greg rutkowski, artgerm, cinematic fantasy. Animation of magical energy swirling, slow motion aura forming, glowing runes pulsing, hanfu sleeves fluttering with every movement."
        },
        "class_type": "PositivePromptNode",
        "_meta": {
          "title": "PositivePromptNode"
        }
      },
      "6": {
        "inputs": {
          "prompt": [
            "7",
```

```
            "5",
            0
          ],
          "negative_prompt": [
            "4",
            0
          ],
          "model": [
            "1",
            0
          ],
          "text_encoder": [
            "8",
            0
          ],
          "vae": [
            "3",
            0
          ],
          "num_inference_steps": 30,
          "seed": 1428,
          "tiled": true,
          "fps": 25,
          "num_frames": 81
        },
        "class_type": "ExVideoNode",
        "_meta": {
          "title": "ExVideoNode"
        }
      },
      "7": {
        "inputs": {
          "video-preview": "",
          "video": [
            "6",
            0
          ]
        },
        "class_type": "PreViewVideo",
        "_meta": {
          "title": "PreViewVideo"
        }
      },
      "8": {
        "inputs": {
          "text_encoder_name": "models_t5_umt5-xxl-enc-bf16.pth",
          "type": "stable_diffusion",
          "device": "default"
        },
        "class_type": "TextEncoderLoaderNode",
        "_meta": {
          "title": "TextEncoderLoaderNode"
        }
      }
    }
}'

```
发起推理后可以通过服务日志查看进度。

查询推理结果
推理结束会返回状态、生成视频等信息。结果为空是推理还未结束，请耐心等待。

```
curl --location --request GET '<ENDPOINT>/history/<prompt_id>' \ #
    --header 'Authorization: <TOKEN>'
```

ComfyUI各指定目录说明
路径	**说明**
/ComfyUI	ComfyUI 项目的根目录。
/ComfyUI/output	ComfyUI 生成视频保存目录。
/ComfyUI/input	ComfyUI 输入图片的配置路径
/ComfyUI/custom_nodes	ComfyUI 自定义节点目录。
/ComfyUI/user/default/workflows	ComfyUI 工作流保存目录。
/ComfyUI/models/checkpoints	ComfyUI checkpoints 路径。
/ComfyUI/models/diffusion_models	ComfyUI 扩散模型配置路径
/ComfyUI/models/lora	ComfyUI LoRA 相关插件配置路径。
/ComfyUI/models/vae	ComfyUI VAE 模型配置路径。
/ComfyUI/models/clip	ComfyUI CLIP 模型配置路径。

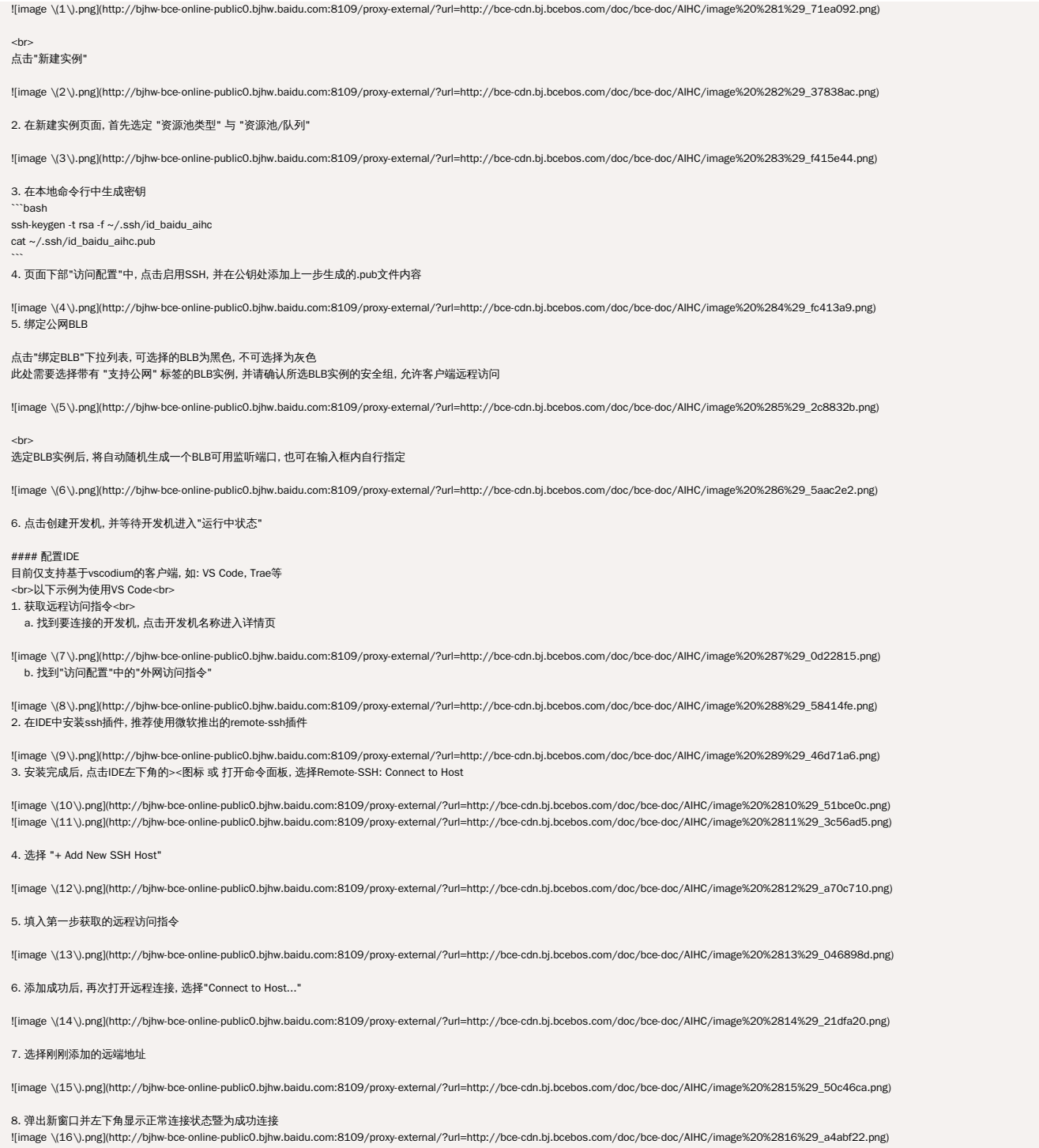
本地IDE连接开发机

为了提供更高效的开发体验，百度百舸允许开发者使用本地熟悉的IDE（如VS Code）远程连接至云端开发机进行代码编写与调试。

通过这种方式，用户既能享受本地IDE丰富的插件与调试功能，又能直接利用云端开发机的高性能算力资源。本文档将详细介绍如何在百舸控制台创建支持公网访问的开发机，并配置本地SSH客户端及IDE插件，实现远程直连开发。

创建开发机
1. 在百舸控制台新建开发机

登录百舸控制台，点击开发机



快速开始搭建 Libero 仿真环境并调用

一、产品简介
LIBERO 是基于百舸平台的**机器人仿真评估与训练工具**。您可以在虚拟环境中训练机器人操作策略，并自动评估其表现——无需真实机器人。

您可以用它来：

* 训练机器人学会抓取、放置、开关柜门等 **130 种**操作任务
* 一键训练模型、一键评测效果，全流程通过脚本完成
* 将服务部署在 GPU 服务器上，团队成员通过浏览器或命令行远程提交评测任务

所有操作均通过一个脚本 `run.sh` 完成，按照本指南操作即可。

二、快速开始（约 30 分钟）
整体流程：**创建开发机 → 登录 → 下载数据 → 训练 → 评测**

2.1 创建开发机
打开百舸控制台 → 进入开发机页面 → 点击【创建实例】。

环境配置

配置项	部署要求	最佳实践
CPU	8核	建议按表单默认值及以上

内存	32G	建议按表单默认值及以上，128G为最佳
GPU	L20等	本示例使用 L20 创建开发机
CDS	按需	推荐100G以上
其它	需挂载bos	建议使用bos中的数据集和模型

创建时需要填写以下配置：

配置项	填写说明
实例名称	自定义命名，例如 `libero-eval`
资源池类型/资源池/队列	根据您已有的资源池类型选择
资源规格	按上方推荐配置选择

其余配置保持默认，点击【确定】即可。

2.2 登录开发机
在开发机列表中等待状态变为「运行中」，点击【登录】进入终端。

> 如果长时间处于「创建中」，请检查资源池是否有可用资源。
2.3 下载数据集
从hugging face下载：

```bash
#### 进入项目目录
cd /root/workspace/LIBERO

#### 下载数据集
bash run.sh download --datasets libero_spatial
```
> 首次使用推荐只下载 `libero_spatial`。从 hugging face 下载可能遇到网络问题
数据集说明：

数据集	大小	任务数	说明
`libero_spatial`	4.2GB	10	同一物体在不同位置
`libero_object`	7.0GB	10	同一操作抓取不同物体
`libero_goal`	6.0GB	10	同一场景执行不同目标
`libero_100`	76GB	100	大规模综合任务集

2.4 训练模型
```bash
#### 开始训练（默认约 5-10 分钟）
bash run.sh train
```
训练完成后，终端会显示模型保存路径（checkpoint），后续评测会自动使用这个模型。

训练参数

```bash
bash run.sh train --benchmark libero_spatial --task_id 4 --n_epochs 10 --batch_size 16
```
2.5 运行评测
```bash
#### 一键评测（自动查找刚训练好的模型）
bash run.sh eval
```
评测完成后会看到类似输出：

```
  Episode 1/10: 失败 (200步)
  Episode 2/10: 成功 (156步)
  Episode 3/10: 失败 (200步)
  ...
  成功率: 30.0%
```
****

三、远程评测（多人共享服务器）
3.1 什么时候需要用？
当您希望：

* 把仿真服务部署在一台 GPU 服务器上，多人共享使用
* 在自己电脑上提交评测任务，不需要安装任何 AI 框架
* 由您的训练机发起，不需要单独占用资源
* 对比多个模型的评测结果

3.2 整体流程
```
您的电脑/笔记本/独立服务器                    GPU 服务器
┌──────────────────────┐    发送评测请求    ┌──────────────────────┐
│                      │ ─────────────────► │                      │
│  发一条命令          │                    │  自动完成：          │
│  等待结果            │ ◄──────────────────│  · 加载模型          │
│                      │    返回评测结果    │  · 运行仿真          │
│  无需安装任何        │                    │  · 计算成功率        │
│  AI 依赖             │                    │                      │
└──────────────────────┘                    └──────────────────────┘
```
3.3 第一步：启动服务（在 GPU 服务器上）
```bash
cd /root/workspace/LIBERO

#### 启动服务
bash run.sh deploy --port 8100
```
看到以下输出表示启动成功：

```
```

[INFO] 仿真服务启动成功 (PID=xxxx)
```
验证服务是否正常运行：

```bash
curl http://localhost:8200/health
#### 应返回 {"status":"ok"}
```
> 如需跨机器访问，请在百舸控制台开启对应自定义端口并配置 BLB。
3.4 第二步：提交评测（在您的电脑上）
客户端**不需要安装 LIBERO**，有两种提交方式：

方式 1：使用脚本（需要 Python）

```bash
bash run.sh remote \
    --server http://<服务器IP>:8200 \
    --ckpt-path /root/workspace/LIBERO/experiments_base/LIBERO_SPATIAL/BCRNNPolicy_seed42/task0_model.pth
```
方式 2：使用 curl（零依赖）

```bash
curl -X POST http://<服务器IP>:8200/eval \
    -H "Content-Type: application/json" \
    -d '{
        "ckpt_path": "/root/workspace/LIBERO/experiments_base/LIBERO_SPATIAL/BCRNNPolicy_seed42/task0_model.pth",
        "benchmark": "libero_spatial",
        "task_id": 0,
        "n_eval": 10
    }'
```
> **关于模型路径**：`--ckpt-path` 填写的是**服务器上**的路径。如果您在服务器上训练了模型，训练完成时终端会输出保存路径，直接复制使用即可。
示例：

```bash
bash run.sh remote \
    --server http://<服务器IP>:8200 \
    --ckpt-path /root/workspace/LIBERO/experiments_base/LIBERO_SPATIAL/BCRNNPolicy_seed42/task0_model.pth \
    --task-id 0 \
    --n-trajs 10
```
> 如果在服务器本机测试，`--server` 填 `http://localhost:8200` 即可。
3.5 查看结果
评测完成后会输出：

```
  任务: pick up the black bowl on the cookie box and place it ...

  Episode 1/10: 失败 (200步, 28.8s)
  Episode 2/10: 成功 (156步, 13.5s)
  Episode 3/10: 失败 (200步, 25.1s)
  ...
============================================================
成功率: 30.0% (3/10)
============================================================
```

四、参数说明
4.1 训练参数（`bash run.sh train`）
参数	说明	默认值	示例
`--benchmark`	任务套件（见第五节）	`libero_spatial`	`libero_object`
`--task_id`	任务编号	`0`	`4`
`--n_epochs`	训练轮数，越大效果越好但越慢	`5`	`10`
`--batch_size`	每批样本数	`8`	`16`
`--device`	训练设备	`cuda`（GPU）	`cpu`

4.2 评测参数（`bash run.sh eval`）
参数	说明	默认值	示例
`--benchmark`	任务套件	`libero_spatial`	`libero_goal`
`--task_id`	任务编号	`0`	`4`
`--ckpt`	模型文件路径（不填则自动查找）	自动	`/path/to/model.pth`
`--n_eval`	评测次数，越多结果越稳定	`10`	`20`
`--device`	推理设备	`cuda`（GPU）	`cpu`

4.3 远程评测参数（`bash run.sh remote`）
参数	必填	说明	默认值	示例
`--server`	是	服务器地址	-	`http://192.168.1.100:8100`
`--ckpt-path`	是	服务器上的模型路径	-	见下方说明
`--benchmark`	否	任务套件	`libero_spatial`	`libero_object`
`--task-id`	否	任务编号	`0`	`4`
`--n-trajs`	否	评测次数	`10`	`20`

> **模型路径在哪？** 训练完成后终端会输出，格式为：
> `/root/workspace/LIBERO/experiments_base/LIBERO_SPATIAL/BCRNNPolicy_seed42/task0_model.pth`
> 其中 `LIBERO_SPATIAL` 对应任务套件，`task0` 对应任务编号。
4.4 服务部署参数（`bash run.sh deploy`）
参数	说明	默认值	示例
`--port`	服务端口号	`8100`	`8100`

五、任务套件一览
套件名称	任务数	说明	推荐度
```

`libero_spatial`	10	同一物体在不同位置（如碗在柜子旁/盘子旁）	强烈推荐（入门）
`libero_object`	10	同一操作抓不同物体（如碗/杯子/罐子）	推荐
`libero_goal`	10	同一场景执行不同目标（如放桌上/放柜里）	推荐
`libero_10`	10	多步骤复合任务（如开柜门→拿碗→放盘子）	进阶
`libero_90`	90	大规模混合任务	进阶

> 通过 `--benchmark` 指定套件，`--task_id` 指定套件内的具体任务编号。

六、命令速查
```bash
#### 验证环境
bash run.sh quickstart

#### 下载数据集
bash run.sh download --datasets libero_spatial

#### 训练
bash run.sh train
bash run.sh train --task_id 4 --n_epochs 10

#### 评测
bash run.sh eval
bash run.sh eval --ckpt /path/to/model.pth

#### 随机策略（用于对比基线）
bash run.sh random

#### 启动远程服务
bash run.sh deploy --port 8200

#### 远程评测
bash run.sh remote \
    --server http://<服务器IP>:8200 \
    --ckpt-path /root/workspace/LIBERO/experiments_base/LIBERO_SPATIAL/BCRNNPolicy_seed42/task0_model.pth

#### 查看帮助
bash run.sh help
```

七、使用建议
1. **第一次用？** 按照「快速开始」走一遍即可：下载 → 训练 → 评测，最快 30 分钟出结果
2. **先跑通再调参**：首次评测用 `--n_eval 3` 减少等待，确认流程没问题后再增加数量
3. **对比模型效果**：用不同的 `--n_epochs` 训练多个模型，在同一任务上评测对比成功率
4. **团队协作**：在服务器上运行 `deploy` 启动服务，团队成员各自用 `curl` 提交评测即可

八、常见问题
Q：运行时出现大量 Warning，是否正常？

正常。这些是底层库的兼容性提示，不影响结果。只要没有红色的 Error 就没问题。

Q：下载数据集超时或断开，怎么办？

重新运行一次即可，脚本会自动断点续传：

```bash
bash run.sh download --datasets libero_spatial
```
Q：评测时提示「Checkpoint 不存在」？

说明还没有训练过模型。请先训练：

```bash
bash run.sh train
bash run.sh eval
```
Q：远程评测连不上服务器？

请依次检查：

1. 服务器上是否已运行 `bash run.sh deploy`（用 `curl http://<IP>:8200/health` 测试）
2. 两台机器是否在同一网络下
3. 是否在百舸控制台开放了对应端口
4. `--ckpt-path` 填的是**服务器上的路径**，不是您本机的路径

Q：可以用自己训练的模型吗？

可以。只要是通过 `bash run.sh train` 训练保存的 `.pth` 文件，用 `--ckpt` 指向该文件即可。

快速开始搭建 Simpler 仿真环境并调用

一、产品简介
Simpler 是一个基于百舸平台的**机器人仿真评估工具**，主要用于对 RT-1 等机器人操作策略模型进行**仿真推理和效果评估**。

核心能力：

* 提供 25+ 种预置仿真任务（抓取、移动、开关抽屉等），无需真实机器人即可验证模型效果
* 支持多种 RT-1 模型版本的一键加载与对比评估
* 支持**本地一体化部署**（仿真+推理同机）和**远程 C/S 架构**（仿真与推理分离），灵活适配不同算力环境

典型使用场景：

* 在训练过程中定期评估模型性能（通过仿真替代真机测试，降低成本）
* 对比不同训练阶段的 checkpoint 表现
* 快速验证新任务/新场景下的策略效果

本指南帮助您快速上手核心功能。

二、快速开始
> **整体流程预览**：创建开发机 → 登录开发机 → 下载模型 → 运行推理，整个过程约 10-15 分钟即可完成首次推理测试。
步骤 1：环境准备与开发机创建
推荐硬件配置
配置项	最低要求	推荐配置
GPU	L20*1	L20*1
显存	24GB	24GB+
内存	16GB	32GB+
存储	50GB 可用空间	100GB+

1.1 创建开发机
打开百舸控制台 → 进入开发机页面 → 点击【创建实例】，按以下信息配置：

基本信息：

配置项	填写说明
实例名称	自定义命名，例如 `simpler-env-server`
资源池类型/资源池/队列	根据您已有的资源池类型选择
资源规格	按上方推荐硬件配置选择
云磁盘	推荐预留 50GiB，保证运行空间
存储挂载	类型选 PFS，源路径如 `/simpler`，挂载路径如 `/mnt/pfs/`（用于存放模型和结果）

访问配置 & 高级配置：

* 保持默认即可，无需修改

填写完成后点击【确定】，等待开发机创建完成。

1.2 登录开发机
创建成功后，在开发机列表中找到刚创建的实例，等待状态变为「运行中」后，点击【登录】进入终端。

> **提示**：如果开发机状态长时间处于「创建中」，请检查资源池是否有可用资源。
步骤 2：下载模型
我们提供了 4 个不同训练阶段的 RT-1 模型 checkpoint，您可以根据需要选择下载：

模型版本	训练阶段	适用场景	推荐度
RT-1-X	多任务联合训练（完全收敛）	正式评估、效果对比	强烈推荐
RT-1-Converged	单任务完整训练（40万步）	正式评估、基线对比	推荐
RT-1-15%	训练中期（约5.8万步）	训练过程分析、中间效果查看	一般
RT-1-Begin	训练初期（约1千步）	对比实验、验证流程可用性	一般

> **建议**：首次使用推荐下载 **RT-1-X**，该模型经过多任务训练，泛化能力最强，在大部分任务上表现最佳。
```bash
#### 登录终端后，先进入工作目录
cd /root/workspace/SimplerEnv

#### 创建模型目录
mkdir -p checkpoints

#### 下载 RT-1-X 模型（推荐，最强泛化）
bcecmd bos cp bos:/aihc-models-bj/RT-1/checkpoints/rt_1_x_tf_trained_for_002272480_step/ checkpoints/rt_1_x_tf_trained_for_002272480_step/ -r

#### 下载 RT-1-Converged 模型（完整训练）
bcecmd bos cp bos:/aihc-models-bj/RT-1/checkpoints/rt_1_tf_trained_for_000400120/ checkpoints/rt_1_tf_trained_for_000400120/ -r

#### 下载 RT-1-15% 模型（训练 15% 的检查点）
bcecmd bos cp bos:/aihc-models-bj/RT-1/checkpoints/rt_1_tf_trained_for_000058240/ checkpoints/rt_1_tf_trained_for_000058240/ -r

#### 下载 RT-1-Begin 模型（训练初始阶段）
bcecmd bos cp bos:/aihc-models-bj/RT-1/checkpoints/rt_1_tf_trained_for_000001120/ checkpoints/rt_1_tf_trained_for_000001120/ -r
```
> **注意**：模型文件较大，下载可能需要几分钟，请耐心等待。下载完成后可通过 `ls checkpoints/` 确认文件完整性。
步骤 3：运行推理
快速测试
使用以下启动命令启动开发机，会自动启动仿真环境，加载 RT-1 模型，在 `google_robot_pick_coke_can`（机器人抓取可乐罐）任务上运行 10 条轨迹的评估。

```bash
#### 1. 激活 conda 环境
eval "$(conda shell.bash hook)"
conda activate simpler_env

#### 2. 设置渲染环境（必须，用于无显示器的 GPU 渲染）
export VK_ICD_FILENAMES=/etc/vulkan/icd.d/nvidia_icd.json
export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:/usr/local/nvidia/lib64:${LD_LIBRARY_PATH:-}

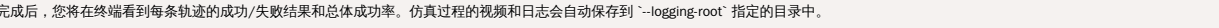
#### 3. 进入项目目录
cd /root/workspace/SimplerEnv

#### 4. 运行推理测试（请将 <ckpt 路径> 替换为实际的模型路径）
xvfb-run python simpler_env/simple_inference_visual_matching_prepackaged_envs.py \
  --policy rt1 \
  --ckpt-path <ckpt 路径> \
  --task google_robot_pick_coke_can \
  --logging-root ./results_simple_eval/ \
  --n-trajs 10
```
示例（使用 RT-1-X 模型），将该代码替换上述#4内容：

```bash
xvfb-run python simpler_env/simple_inference_visual_matching_prepackaged_envs.py \
  --policy rt1 \
  --ckpt-path ./checkpoints/rt_1_x_tf_trained_for_002272480_step \
  --task google_robot_pick_coke_can \
  --logging-root ./results_simple_eval/ \
```

```
    --n-trajs 10
```
> **预期运行时间**：10 条轨迹在 L20 上约需 5-10 分钟。首次运行会自动下载语言编码模型（约 1GB），后续运行将使用缓存。

运行效果
推理完成后，您将在终端看到每条轨迹的成功/失败结果和总体成功率。仿真过程的视频和日志会自动保存到 `--logging-root` 指定的目录中。


【可选】RDP 连接运行
如果您不想用 headless 方式运行，希望有可视化界面也可以通过以下方式启动 RDP 服务：

1. 开启开发机自定义端口
2. 配置 3389 端口（启动脚本使用该端口）
3. 使用以下命令开启开发机 RDP 服务并配置密码

```
bash rdp_setup.sh
#### 使用指令配置密码
passwd root
```
4. 然后在您的电脑上下载 RDP 或者Windows App
5. 添加电脑，输入您的 ip 地址:端口号，并配置账号名和密码
6. 双击图标进入可视化界面，通过其终端输入以下指令即可运行

```
conda activate simpler_env
cd /root/workspace/SimplerEnv/baige
python test_getting_started.py
```

三、进阶功能：远程调用（C/S 架构）
场景说明
在实际使用中，您可能希望将**仿真环境**和**模型推理**部署在不同的机器上，例如：

* 仿真环境需要 GPU + Vulkan 渲染支持（资源要求高）
* 模型推理只需要 TensorFlow + GPU（可复用训练机器）
* 团队中多人共享同一个仿真服务，各自运行不同的模型

此时可以使用远程调用功能，将系统拆分为两台开发机：

```
┌──────────────────────────────┐          HTTP          ┌──────────────────────────────┐
│  推理服务器 (A)              │ ◄──────────────────────► │  仿真服务器 (B)              │
│                              │                        │                              │
│  · RT-1 模型推理             │  /env/create           │  · SAPIEN 仿真引擎           │
│  · 发送动作指令              │  /env/{id}/reset       │  · Vulkan GPU 渲染           │
│  · 接收观测图像              │  /env/{id}/step        │  · 物理模拟计算              │
│                              │  /env/{id}/obs         │                              │
│  需要：TF + GPU              │                        │  需要：GPU + Vulkan          │
└──────────────────────────────┘                        └──────────────────────────────┘
```
创建流程
需要创建**两台开发机**（仿真服务器 + 推理服务器），创建步骤与「二、快速开始」相同，额外注意以下配置。

仿真服务器配置（Server B）
在开发机创建阶段，需要额外操作：

1. **配置自定义端口**：开启端口 8100
2. **配置 BLB**：选择您使用的 BLB（百度负载均衡），配置 BLB 监听 8100 端口
3. **填写启动命令**：将以下代码填入启动命令中

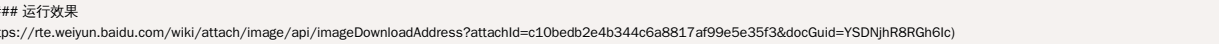
```bash
eval "$(conda shell.bash hook)"
conda activate simpler_env

export VK_ICD_FILENAMES=/etc/vulkan/icd.d/nvidia_icd.json
export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:/usr/local/nvidia/lib64:${LD_LIBRARY_PATH:-}

nohup xvfb-run python /root/workspace/baige/env_server.py --host 0.0.0.0 --port 8100
```
> **创建成功后**，在开发机详情页中会显示该机器的 **IP 地址**，请记录下来，后续推理服务器需要用到。
推理服务器配置（Server A）
在开发机创建阶段，将以下代码填入启动命令中。请将 `<仿真服务器IP>` 和 `<ckpt路径>` 替换为实际值：

```bash
eval "$(conda shell.bash hook)"
conda activate simpler_env

nohup python /root/workspace/baige/rt1_remote_inference.py \
    --server http://<仿真服务器IP>:8100 \
    --ckpt-path <ckpt路径> \
    --task google_robot_pick_coke_can \
    --n-trajs 10
```
示例（假设仿真服务器 IP 为 192.168.1.100）：

```bash
nohup python /root/workspace/baige/rt1_remote_inference.py \
    --server http://192.168.1.100:8100 \
    --ckpt-path /root/workspace/SimplerEnv/checkpoints/rt_1_x_tf_trained_for_002272480_step \
    --task google_robot_pick_coke_can \
    --n-trajs 10
```
运行效果

四、关键参数说明
本地推理参数（步骤 3 使用）
参数	必选	说明	默认值	示例

|-|-|-|
`--policy`	是	策略类型，目前支持 rt1	-	`rt1`
`--ckpt-path`	是	模型 checkpoint 路径	-	`./checkpoints/rt_1_x_tf_trained_for_002272480_step`
`--task`	是	仿真任务名称（见第五节任务列表）	-	`google_robot_pick_coke_can`
`--n-trajs`	否	评估的轨迹数量，越多结果越稳定	`10`	`3`（快速测试用）
`--logging-root`	否	结果文件保存目录	`./results_simple_eval/`	`./my_results/`

远程服务参数（第三节使用）
仿真服务器端（env_server.py）：

参数	必选	说明	默认值	示例
`--host`	否	监听地址	`0.0.0.0`	`0.0.0.0`（监听所有网卡）
`--port`	否	监听端口号	`8100`	`8100`
`--max-envs`	否	最大并发仿真环境数	`5`	`3`（显存不足时可调低）

推理客户端（rt1_remote_inference.py）：

参数	必选	说明	默认值	示例
`--server`	是	仿真服务器地址（含端口）	-	`http://192.168.1.100:8100`
`--ckpt-path`	是	模型 checkpoint 路径	-	`./checkpoints/rt_1_x_tf_trained_for_002272480_step`
`--task`	是	仿真任务名称	-	`google_robot_pick_coke_can`
`--n-trajs`	否	评估的轨迹数量	`10`	`10`
`--tf-memory-limit`	否	TF GPU 显存限制（MB）	`4096`	`2048`（与训练共享 GPU 时可调低）

五、常见任务列表
> 以下是目前支持的所有仿真任务。`--task` 参数需填写「task 名称」列中的值。
Google Robot 任务
抓取类任务：

task 名称	说明
`google_robot_pick_coke_can`	抓可乐罐（默认姿态，推荐入门）
`google_robot_pick_horizontal_coke_can`	可乐罐水平放置（增加难度）
`google_robot_pick_vertical_coke_can`	可乐罐垂直放置
`google_robot_pick_standing_coke_can`	可乐罐站立
`google_robot_pick_object`	抓取随机物体（泛化测试）

移动类任务：

task 名称	说明
`google_robot_move_near`	移动物体到目标附近（v1，推荐）
`google_robot_move_near_v0`	同上 v0 版本
`google_robot_move_near_v1`	同上 v1 版本

抽屉操作任务：

task 名称	说明
`google_robot_open_drawer`	打开抽屉（随机层）
`google_robot_open_top_drawer`	打开顶部抽屉
`google_robot_open_middle_drawer`	打开中间抽屉
`google_robot_open_bottom_drawer`	打开底部抽屉
`google_robot_close_drawer`	关闭抽屉（随机层）
`google_robot_close_top_drawer`	关闭顶部抽屉
`google_robot_close_middle_drawer`	关闭中间抽屉
`google_robot_close_bottom_drawer`	关闭底部抽屉

放置类任务：

task 名称	说明
`google_robot_place_in_closed_drawer`	放入关闭的抽屉（随机层）
`google_robot_place_in_closed_top_drawer`	放入顶部抽屉
`google_robot_place_in_closed_middle_drawer`	放入中间抽屉
`google_robot_place_in_closed_bottom_drawer`	放入底部抽屉
`google_robot_place_apple_in_closed_top_drawer`	放苹果到顶部抽屉

WidowX Bridge 任务
task 名称	说明
`widowx_spoon_on_towel`	勺子放毛巾上
`widowx_carrot_on_plate`	胡萝卜放盘子上
`widowx_stack_cube`	堆叠方块
`widowx_put_eggplant_in_basket`	茄子放篮子里

六、最佳实践建议
1. **首次使用**：推荐从 RT-1-X 模型 + `google_robot_pick_coke_can` 任务开始，这是最经典的评估组合
2. **快速测试**：使用 `--n-trajs 3` 减少测试时间，确认流程跑通后再增加轨迹数
3. **性能优化**：RTX 4090 上仿真服务器建议 `--max-envs` 不超过 5，避免显存溢出
4. **结果保存**：推理结果（视频 + 日志）自动保存在 `--logging-root` 指定的目录中
5. **模型对比**：可以对同一任务分别使用不同 checkpoint 运行评估，对比不同训练阶段的成功率

七、常见问题（FAQ）
Q1：首次运行时出现大量 TF 警告信息（cuDNN、cuFFT、cuBLAS 等），是否正常？

正常。这些是 TensorFlow 初始化 CUDA 库时的日志输出，不影响推理结果。只要最终没有报 Error 即可。

Q2：运行报错 **`OSError: SavedModel file does not exist`，怎么办？**

说明 `--ckpt-path` 指定的路径不正确。请确认：

* 路径指向的目录中包含 `saved_model.pb` 文件
* 可以用 `ls <你的ckpt路径>/` 检查目录内容
* 如果使用 BOS 下载，请确认下载是否完整（`bcecmd` 命令是否成功执行）

Q3：远程调用模式下，推理服务器连不上仿真服务器？

请检查：

* 仿真服务器是否已启动（`curl http://<仿真服务器IP>:8100/tasks` 应返回任务列表）
* 两台开发机是否在同一 VPC 网络下 / 正确配置了 BLB
* BLB 是否正确配置了 8100 端口的监听
* 防火墙是否放行了 8100 端口

Q4：可以用自己训练的 RT-1 模型吗？

可以。只要您的模型是 TensorFlow SavedModel 格式（目录中包含 `saved_model.pb`），将 `--ckpt-path` 指向您的模型目录即可。

Q5：如何查看评估结果？

推理完成后，终端会输出每条轨迹的成功/失败状态和总体成功率。详细的仿真视频和日志保存在 `--logging-root` 指定的目录中，可以直接下载查看，也可以根据输出报文查看。

通过百舸平台快速开始搭建 RoboCasa 仿真环境并调用

一、产品简介
RoboCasa 是斯坦福大学发布的一款**厨房场景机器人仿真工具**，可以让您在电脑上模拟机器人执行各种厨房操作，无需真实机器人即可训练和测试。

核心能力：

* 提供 365 个厨房操作任务（拿取、放置、开关橱柜/冰箱/微波炉等），覆盖常见生活场景
* 基于 MuJoCo 物理引擎，**无需额外 license 文件**，开箱即用
* 内置 2200+ 小时的示教数据集，可直接用于策略训练
* 支持 Diffusion Policy 等主流机器人策略

典型使用场景：

* 在无真实机器人的情况下验证操作策略效果
* 对比不同策略模型在厨房任务中的表现
* 快速迭代和调试机器人控制算法

与 LIBERO 的主要区别：

如果您之前用过 LIBERO 仿真工具，以下是 RoboCasa 的主要不同点：

对比项	LIBERO	RoboCasa
任务规模	130 个桌面操作任务	365 个厨房操作任务
MuJoCo 授权	需要 license 文件	**无需**（免费）
无 GPU 渲染方式	需要 Xvfb	**OSMesa**（更简单）
数据格式	HDF5	lerobot (parquet)

本指南帮助您快速上手核心功能。

二、快速开始
环境准备与开发机创建
推荐硬件配置
配置项	最低要求	推荐配置
GPU	无	L20 或更高规格的加速卡
内存	32GB	64GB+
CPU	8 核	16 核+
存储	按需	100GB+（含数据集）

> **说明**：RoboCasa 的仿真运行可以在无 CPU 机器中运行。但 GPU 能提供更高的仿真效率
1.1 创建开发机
打开百舸控制台 → 进入开发机页面 → 点击【创建实例】，按以下信息配置：

基本信息：

配置项	填写说明
实例名称	自定义命名
资源池类型/资源池/队列	根据您已有的资源池类型选择
资源规格	按上方推荐硬件配置选择
云磁盘	推荐预留 50GiB，保证运行空间

访问配置：

* **配置自定义端口**：开启端口 **8100**（用于后续远程调用仿真服务，如果只做本地验证可先跳过）
* **配置 BLB**：如需远程调用，请配置 BLB

填写完成后点击【确定】，等待开发机创建完成。

1.2 登录开发机
创建成功后，在开发机列表中找到刚创建的实例，等待状态变为「运行中」后，点击【登录】进入终端。

> **提示**：如果开发机状态长时间处于「创建中」，请检查资源池是否有可用资源。
开发机镜像中已经预装了完整的 RoboCasa 环境，目录结构如下：

```
/root/workspace/
├── robocasa/           # RoboCasa 主程序（含厨房场景资源 23GB）
│   └── robocasa-quickstart.ipynb   # 本文件
├── robosuite/          # robosuite 仿真框架
└── baige/              # 百舸预置脚本（包含一键启动、验证等工具）
```

进入`robocasa-quickstart.ipynb`后，选择 RoboCasa 内核环境

下载资产
```python
#### 从百度 BOS 下载
bcecmd bos cp -r bos:/aihc-rdw-bj/robocasa-assets/ /root/workspace/robocasa/robocasa/models/assets/
```
环境可用性检查

```python
import os
import sys

os.environ.setdefault("MUJOCO_GL", "osmesa")
os.environ.setdefault("PYOPENGL_PLATFORM", "osmesa")

import numpy as np
```
```python
"""
RoboCasa 快速验证
测试内容：
  1. 创建环境（PnPCounterToCab 任务）
  2. reset + 随机 step × 5
  3. 打印 obs keys 和 reward
  4. 渲染一帧并保存为 PNG
运行方式：
  MUJOCO_GL=osmesa python baige/robocasa_test_getting_started.py
  或
  sh baige/robocasa_headless_run.sh python baige/robocasa_test_getting_started.py
"""

print("=== RoboCasa 环境验证 ===\n")

#### 1. 导入
try:
    import robosuite
    import robocasa
    print(f"robosuite 版本: {robosuite.__version__}")
    print(f"robocasa 路径: {robocasa.__path__[0]}")
except ImportError as e:
    print(f"[ERROR] 导入失败: {e}")
    sys.exit(1)

import robosuite as suite
from robosuite.controllers import load_composite_controller_config

#### 2. 创建环境
print("\n[1/4] 创建环境 (PnPCounterToCab)...")
try:
    controller_config = load_composite_controller_config(robot="PandaOmron")
    env = suite.make(
        env_name="PickPlaceCounterToCabinet",
        robots="PandaOmron",
        controller_configs=controller_config,
        has_renderer=False,
        has_offscreen_renderer=True,
        use_camera_obs=True,
        camera_names=["robot0_agentview_center"],
        camera_heights=128,
        camera_widths=128,
        render_gpu_device_id=-1,   # -1 = CPU/osmesa
    )
    print("  环境创建成功")
except Exception as e:
    print(f"[ERROR] 创建环境失败: {e}")
    import traceback; traceback.print_exc()
    sys.exit(1)

#### 3. Reset
print("\n[2/4] Reset 环境...")
try:
    obs = env.reset()
    print(f"  obs keys 数量: {len(obs)}")
    print(f"  obs keys 示例: {list(obs.keys())[:8]}")
except Exception as e:
    print(f"[ERROR] Reset 失败: {e}")
    import traceback; traceback.print_exc()
    sys.exit(1)

#### 4. 随机 step
print("\n[3/4] 随机 step × 5...")
try:
    total_reward = 0.0
    low, high = env.action_spec
    for i in range(5):
        action = np.random.uniform(low, high)
        obs, reward, done, info = env.step(action)
        total_reward += reward
        print(f"  step {i+1}: reward={reward:.4f}, done={done}")
    print(f"  累计 reward: {total_reward:.4f}")
except Exception as e:
    print(f"[ERROR] Step 失败: {e}")
    import traceback; traceback.print_exc()
    sys.exit(1)
```

```
#### 5. 渲染并保存
print("\n[4/4] 渲染并保存图像...")
try:
    from PIL import Image
    cam_key = "robot0_agentview_center_image"
    if cam_key in obs:
        img_array = obs[cam_key]
        # robosuite 图像是 (H, W, 3)，值域 [0,255]，需要上下翻转
        img_array = img_array[::-1]
        img = Image.fromarray(img_array.astype(np.uint8))
        img.save("test_frame.png")
        print(f"  图像已保存: test_frame.png  shape={img_array.shape}")
    else:
        print(f"  [WARN] 未找到相机 obs key '{cam_key}'，可用 keys: {[k for k in obs if 'image' in k]}")
except Exception as e:
    print(f"[ERR] {e}")

env.close()
print("\n=== 验证完成 ✓ ===")
```





1.3 基础用法
Gym 封装器
```python
import gymnasium as gym
import robocasa
from robocasa.utils.env_utils import run_random_rollouts

env = gym.make(
    "robocasa/PickPlaceCounterToCabinet",
    split="pretrain", # use 'pretrain' or 'target' kitchen scenes and objects
    seed=0 # seed environment as needed. set seed=None to run unseeded
)

#### run rollouts with random actions and save video
run_random_rollouts(
    env, num_rollouts=3, num_steps=100, video_path="/tmp/test.mp4"
)
```


######

三、远程调用
场景说明
在实际使用中，您可能希望将**仿真环境**和**策略推理**部署在不同的机器上，例如：

* 仿真环境运行在一台服务器上，负责物理模拟和画面渲染
* 策略推理运行在另一台机器上，负责决策计算
* 团队中多人共享同一个仿真服务，各自运行不同的策略

此时可以使用远程调用功能，将系统拆分为两台机器通过网络通信：

创建流程
需要创建**两台开发机**（仿真服务器 + 推理服务器），创建步骤与「二、快速开始」相同，额外注意以下配置。

仿真服务器配置（Server B）
在开发机创建阶段，需要额外操作：

1. **配置自定义端口**：开启端口 **8100**（如在快速开始中已配置，可跳过）
2. **配置 BLB**：选择您使用的 BLB

> **创建成功后**，在开发机详情页中会显示该机器的 **IP 地址**与仿真服务绑定端口
第一步：启动仿真服务
在仿真服务器上登录终端，执行以下命令：

```python
!bash ../baige/robocasa_deploy.sh
```


确认服务是否正常运行：

```python
#### 健康检查
!curl http://localhost:8100/health
#### {"status":"ok"}

#### 查看支持的任务列表
!curl http://localhost:8100/tasks
#### {"common_tasks": [...], "count": 12}
```
第二步：运行推理客户端
如果需要在远程推理服务器（或本地机器）上调用仿真服务，则需要将 `<仿真服务器IP>` 与`<仿真服务实际对外暴露端口>`替换为你实际使用的值

```python
%%bash
python3.11 ../baige/robocasa_random_remote.py \
```

```
    e# --server http://<仿真服务器IP>:<仿真服务实际对外暴露端口> \  # 远程调用
    --server http://localhost:8100 \  # 本地运行推理用于测试
    --task PickPlaceCounterToCabinet \
    --n-trajs 3 \
    --max-steps 100 \
    --save-gif        # 可选：保存仿真动画为 GIF 文件
```



```
=== RoboCasa 远程随机推理 ===
Server : http://192.168.1.100:8100
Task   : PickPlaceCounterToCabinet
Trajs  : 3  MaxSteps: 100

[✓] 服务健康: {'status': 'ok'}
[✓] 可用任务数: 12

Episode 1/3 | env_id=6e90a48f... | action_dim=12
  reset OK | obs_keys=30
  steps=100 | reward=0.0000 | done=False | 72.3s

==================================================
完成轨迹: 3
平均步数: 100.0
平均 reward: 0.0000
成功率: 0.0%  (随机策略预期 0%)
==================================================

[✓] GIF 已保存: baige/robocasa_random_rollout.gif
```

> **说明**：因为使用的是随机策略，所以成功率为 0% 是正常现象。实际使用请替换为训练好的策略模型

四、关键参数说明

仿真服务 API 接口

仿真服务启动后，提供以下 HTTP 接口供推理端调用：

接口	方法	说明
`/health`	GET	健康检查，确认服务是否正常
`/tasks`	GET	列出所有支持的仿真任务
`/env/create`	POST	创建一个新的仿真环境
`/env/{env_id}/reset`	POST	重置环境到初始状态
`/env/{env_id}/step`	POST	执行一个动作，获取结果
`/env/{env_id}`	DELETE	销毁环境，释放资源

创建环境示例：

```python
!curl -s -X POST http://localhost:8100/env/create \
    -H "Content-Type: application/json" \
    -d '{"task_name": "PickPlaceCounterToCabinet", "robot": "PandaOmron"}'
```

推理客户端参数（robocasa_random_remote.py）

参数	必选	说明	默认值
`--server`	是	仿真服务器地址（含端口）	-
`--task`	否	任务名称（见第五节任务列表）	`PickPlaceCounterToCabinet`
`--n-trajs`	否	评测轨迹数量（越多结果越稳定）	`3`
`--max-steps`	否	每条轨迹最大步数	`50`
`--save-gif`	否	是否保存仿真动画为 GIF	否

五、常见任务列表

> 以下是 RoboCasa 中常用的仿真任务。`--task` 参数需填写「任务名称」列中的值。

取放类任务

任务名称	说明
`PickPlaceCounterToCabinet`	从台面取物放入橱柜（推荐入门）
`PickPlaceCabinetToCounter`	从橱柜取物放回台面
`PickPlaceCounterToSink`	从台面取物放入水槽
`PickPlaceCounterToMicrowave`	从台面取物放入微波炉

开关类任务

任务名称	说明
`OpenCabinet`	打开橱柜门
`CloseCabinet`	关闭橱柜门
`OpenMicrowave`	打开微波炉
`CloseMicrowave`	关闭微波炉
`OpenFridge`	打开冰箱
`CloseFridge`	关闭冰箱

> **提示**：RoboCasa 共支持 365 个任务，以上仅列出最常用的。如需查看完整列表，可在 Python 中执行：

```python
import robocasa, robosuite
print(list(robosuite.environments.REGISTERED_ENVS.keys()))
```

六、最佳实践建议

1. **首次使用**：推荐先运行验证脚本（步骤 3），确认环境正常后再进行后续操作
2. **快速测试**：使用 `--n-trajs 3 --max-steps 50` 减少测试时间，确认流程跑通后再增加数量
3. **无 GPU 也能用**：仿真运行不需要 GPU，使用 OSMesa 软件渲染即可，适合在 CPU 开发机上调试
4. **远程调用**：如果需要仿真和推理分离部署，使用内置的 `env_server` + 推理客户端方案
5. **任务名称注意**：RoboCasa v1.0 对任务名做了重命名（如 `PnPCounterToCab` → `PickPlaceCounterToCabinet`），请使用新名称

【可选】RDP 连接运行
如果您希望有可视化界面也可以通过以下方式启动 RDP 服务：

1. 开启开发机自定义端口
2. 配置 3389 端口（启动脚本使用该端口）
3. 使用以下命令开启开发机 RDP 服务并配置密码

```
bash rdp_setup.sh
#### 使用指令配置密码
passwd root
```
4. 然后在您的电脑上下载远程桌面客户端
5. 添加电脑，输入您的 ip 地址:端口号，并配置账号名和密码
6. 双击图标进入可视化界面，通过其终端执行命令即可

七、常见问题（FAQ）
Q1：运行时出现 **`Segmentation fault (core dumped)`，怎么办？**

这是最常见的问题。原因是仿真引擎默认尝试使用图形界面渲染，但服务器没有显示器，导致崩溃。

解决方法： 使用百舸提供的一键启动脚本运行您的程序：

```bash
sh /root/workspace/baige/robocasa_headless_run.sh <您的命令>
```
脚本会自动设置正确的渲染模式。如果您需要手动设置，请确保在运行任何 Python 代码**之前**执行：

```bash
export MUJOCO_GL=osmesa
export PYOPENGL_PLATFORM=osmesa
unset DISPLAY
```
Q2：运行时报错 **`Environment PnPCounterToCab not found`，怎么办？**

这是因为 RoboCasa v1.0 对任务名称做了更新。请使用新名称：

	旧名称（已废弃）	新名称（请使用这个）
`PnPCounterToCab`	`PickPlaceCounterToCabinet`
`PnPCabToCounter`	`PickPlaceCabinetToCounter`

如需查看所有可用任务名，请在 Python 中运行：

```python
import robocasa, robosuite
print(list(robosuite.environments.REGISTERED_ENVS.keys()))
```
Q3：运行时提示缺少系统库（如 **`libGL` 或 **`libosmesa`**），怎么办？**

请执行以下命令安装系统依赖：

```bash
apt-get install -y linux-libc-dev build-essential gcc \
    libgl1 libglib2.0-0 xvfb libosmesa6
```
安装完成后重新运行即可。

Q4：远程调用模式下，推理客户端连不上仿真服务器？

请依次检查：

1. 仿真服务是否已启动：在仿真服务器上执行 `curl http://localhost:8100/health`，应返回 `{"status":"ok"}`
2. 两台开发机是否在同一 VPC 网络下 / 正确配置了 BLB
3. BLB 是否正确配置了 8100 端口的监听
4. 防火墙是否放行了 8100 端口

Q5：RoboCasa 需要 GPU 吗？

* **仅运行仿真**：不需要 GPU，CPU 机器即可，使用 OSMesa 软件渲染
* **策略训练**：需要 GPU（推荐支持 RTX 渲染的显卡）

快速开始搭建 BEHAVIOR-1K 仿真环境并调用

一、产品简介
BEHAVIOR-1K 是斯坦福大学发布的**家庭活动仿真基准**，可以让您在电脑上模拟机器人在家庭场景中执行各种日常活动，无需真实机器人即可训练和测试。

核心能力：

* 提供 51 个家庭场景、1018 个日常活动任务（做饭、清洁、整理等），覆盖真实生活场景
* 基于 OmniGibson + NVIDIA Isaac Sim 仿真后端，提供高质量光线追踪渲染
* 支持 Turtlebot、Fetch、FrankaPanda、R1Pro 等多种机器人
* 内置 BehaviorTask、PointNavigationTask 等任务类型，适用于导航与操作策略研究

典型使用场景：

* 在无真实机器人的情况下验证家庭操作策略效果
* 对比不同策略模型在长程活动任务中的表现
* 利用高保真视觉渲染训练视觉-语言-动作模型

本指南帮助您快速上手核心功能。

````
##### 二、快速开始
###### 环境准备与开发机创建
**硬件配置**
|配置项|最低要求|推荐配置|
|-|-|-|
|GPU|L20|L20|
|NVIDIA 驱动|535.x|535.161.08|
|内存|32GB|64GB+|
|CPU|8 核|16 核+|
|存储|80GB|200GB+（含数据集）|

> BEHAVIOR-1K 使用 NVIDIA Isaac Sim 进行光追渲染，GPU **必须具备 RT Cores**。A100、H800、H100 等纯计算卡**不支持**。请选择 RTX L20 等具备 RT Cores 的 GPU 机型。
> NVIDIA 驱动请在添加资源池节点时进行配置。镜像内已预制 CUDA 12.4。

**1.1 创建开发机**
**访问配置：**

* **配置自定义端口**：开启端口 **8100**（用于后续远程调用仿真服务，如果只做本地验证可先跳过）
* **配置 BLB**：如需远程调用，请配置 BLB

![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/image_31fd80f.png)

**1.2 登录开发机**
开发机镜像中已经预装了完整的 BEHAVIOR-1K 环境，目录结构如下：

```
/root/workspace/
└── BEHAVIOR-1K/        # 项目源码
    ├── datasets/       # 数据集
    ├── baige/          # 百舸预置脚本（包含一键启动、验证等工具）
    └── behavior-1k-quickstart.ipynb   # 本文件
```
Python 环境：全局 Python 3.10（`/usr/bin/python3`），已安装 PyTorch 2.6.0+cu124 和 Isaac Sim 4.5.0。

###### 下载数据集
执行以下命令在开发机中下载数据集：

```bash
#### 从百度 BOS 下载
bcecmd bos cp -r bos:/aibox-private/behavior-1k/ ~/workspace/BEHAVIOR-1K/datasets/
```
数据集包含：omnigibson-robot-assets、behavior-1k-assets、2025-challenge-task-instances 等。

###### 运行环境配置
运行该脚本配置 Behavior-1k 运行所需环境：

```bash
bash ~/workspace/BEHAVIOR-1K/baige/run_env.sh
```
该脚本：

1. 检测内核驱动版本，修复 `libcuda.so.1` 等 symlink
2. 验证 CUDA 可用性
3. 启动 Xvfb 虚拟显示
4. 设置环境变量

###### 环境可用性检查
```python
"""
BEHAVIOR-1K 快速验证
测试内容：
  1. 创建环境（Rs_int 场景 + Turtlebot + DummyTask）
  2. reset + 随机 step × 5
  3. 打印 obs keys 和 reward
运行方式：
  cd ~/workspace/BEHAVIOR-1K/OmniGibson
  bash ../baige/run_env.sh  # 先修复 CUDA
  python3 -u baige/behavior_test.py --/rtx/verifyDriverVersion/enabled=false
"""

import os
import sys

#### 必须在 import omnigibson 之前设置
os.environ.setdefault("OMNIGIBSON_HEADLESS", "True")
os.environ.setdefault("OMNI_KIT_ACCEPT_EULA", "Yes")

import time

print("=== BEHAVIOR-1K 环境验证 ===\n")

#### 1. 导入
try:
    import omnigibson as og
    from omnigibson.macros import gm
    gm.RENDER_VIEWER_CAMERA = False
    print("OmniGibson 导入成功")
except ImportError as e:
    print(f"[ERROR] 导入失败: {e}")
    sys.exit(1)

#### 2. 创建环境
print("\n[1/4] 创建环境 (Rs_int + Turtlebot + DummyTask)...")
try:
    t0 = time.time()
    cfg = {
        "env": {"flatten_action_space": True, "flatten_obs_space": True},
        "scene": {"type": "InteractiveTraversableScene", "scene_model": "Rs_int"},
        "robots": [{"type": "Turtlebot", "obs_modalities": ["rgb", "depth"]}],
````
````

```
        "task": {"type": "DummyTask"},
    }
    env = og.Environment(configs=cfg)
    print(f"  环境创建成功（耗时 {time.time()-t0:.1f}s）")
except Exception as e:
    print(f"[ERROR] 创建环境失败: {e}")
    import traceback; traceback.print_exc()
    sys.exit(1)

#### 3. Reset
print("\n[2/4] Reset 环境...")
try:
    obs, info = env.reset()
    print(f"  obs keys 数量: {len(obs)}")
    print(f"  obs keys 示例: {list(obs.keys())[:5]}")
except Exception as e:
    print(f"[ERROR] Reset 失败: {e}")
    import traceback; traceback.print_exc()
    sys.exit(1)

#### 4. 随机 step
print("\n[3/4] 随机 step × 5...")
try:
    for i in range(5):
        action = env.action_space.sample()
        obs, reward, terminated, truncated, info = env.step(action)
        print(f"  step {i+1}: reward={reward:.4f}, done={terminated}")
except Exception as e:
    print(f"[ERROR] Step 失败: {e}")
    import traceback; traceback.print_exc()
    sys.exit(1)

og.shutdown()
print("\n=== 验证完成 ✓ ===")
```

期望输出：

```
=== BEHAVIOR-1K 环境验证 ===

OmniGibson 导入成功

[1/4] 创建环境 (Rs_int + Turtlebot + DummyTask)...
  环境创建成功（耗时 147.3s）   ← 首次启动较慢

[2/4] Reset 环境...
  obs keys 数量: 3
  obs keys 示例: ['robot_...::rgb', 'robot_...::depth', 'task::low_dim']

[3/4] 随机 step × 5...
  step 1: reward=0.0000, done=False
  step 2: reward=0.0000, done=False
  ...

=== 验证完成 ✓ ===
```

> **首次启动较慢**（~2 分钟），因为 Isaac Sim 需要编译 GPU shader。后续启动约 30-80 秒。

基础用法

```python
import os
os.environ["OMNIGIBSON_HEADLESS"] = "True"
os.environ["OMNI_KIT_ACCEPT_EULA"] = "Yes"

import omnigibson as og
from omnigibson.macros import gm
gm.RENDER_VIEWER_CAMERA = False

#### 创建导航任务环境
cfg = {
    "env": {
        "flatten_action_space": True,
        "flatten_obs_space": True,
        "external_sensors": [{
            "sensor_type": "VisionSensor",
            "name": "external_sensor0",
            "relative_prim_path": "/external_sensor0",
            "modalities": ["rgb"],
            "sensor_kwargs": {"image_height": 128, "image_width": 128},
            "position": [0, 0, 1.0],
            "orientation": [0.707, 0.0, 0.0, 0.707],
            "pose_frame": "parent",
        }],
    },
    "scene": {"type": "InteractiveTraversableScene", "scene_model": "Rs_int"},
    "robots": [{"type": "Turtlebot", "obs_modalities": ["rgb", "depth"]}],
    "task": {"type": "PointNavigationTask", "goal_type": "geolocation"},
}
env = og.Environment(configs=cfg)

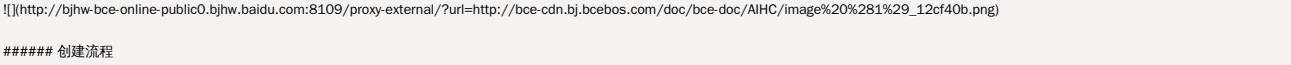
#### 标准交互循环
obs, info = env.reset()
for step in range(100):
    action = env.action_space.sample()  # 替换为您的策略
    obs, reward, terminated, truncated, info = env.step(action)
    if terminated or truncated:
        obs, info = env.reset()

og.shutdown()
```

三、远程调用
场景说明
在实际使用中，您可能希望将**仿真环境**和**策略推理**部署在不同的机器上，例如：

* 仿真环境运行在一台 GPU 服务器上，负责物理模拟和画面渲染（需要 RT Cores GPU）
* 策略推理运行在另一台机器上，负责决策计算
* 团队中多人共享同一个仿真服务，各自运行不同的策略

此时可以使用远程调用功能，将系统拆分为两台机器通过网络通信：



创建流程
需要创建**两台开发机**（仿真服务器 + 推理服务器），创建步骤与「二、快速开始」相同，额外注意以下配置。

仿真服务器配置（Server B）
在开发机创建阶段，需要额外操作：

1. **配置自定义端口**：开启端口 **8100**（如在快速开始中已配置，可跳过）
2. **配置 BLB**：选择您使用的 BLB
3. **GPU 选型**：仿真服务器**必须**选择带 RT Cores 的 GPU

> **创建成功后**，在开发机详情页中会显示该机器的 **IP 地址**与仿真服务绑定端口
第一步：启动仿真服务
在仿真服务器上登录终端，执行以下命令：

```bash
cd ~/workspace/BEHAVIOR-1K
bash baige/behavior_deploy.sh
```
脚本自动执行：修复 CUDA symlink → 启动 Xvfb → 启动 FastAPI 服务 → 健康检查轮询。

启动成功后输出：

```
✓ 服务器就绪！
API 地址: http://<机器IP>:8100
健康检查: curl http://localhost:8100/health
查看日志: tail -f /tmp/behavior_env_server.log
```
确认服务是否正常运行：

```python
#### 健康检查
!curl http://localhost:8100/health
#### {"status":"ok"}

#### 查看支持的场景和机器人
!curl http://localhost:8100/tasks
#### {"scenes": ["Rs_int", ...], "robots": ["A1", "Fetch", ...], "task_types": ["DummyTask", ...]}
```
第二步：运行推理客户端
如果需要在远程推理服务器（或本地机器）上调用仿真服务，则需要将 `<仿真服务器IP>` 与 `<仿真服务实际对外暴露端口>` 替换为你实际使用的值

```python
%%bash
python3 ~/workspace/BEHAVIOR-1K/baige/behavior_test_client.py \
    --server http://<仿真服务器IP>:<仿真服务实际对外暴露端口> \  # 远程调用
    # --server http://localhost:8100 \  # 本地运行推理用于测试
    --scene Rs_int \
    --robot Turtlebot \
    --steps 20 \
    --output behavior_episode.gif
```
输出示例：

```
=== BEHAVIOR-1K 远程随机推理 ===
Server : http://192.168.1.100:8100
Scene  : Rs_int
Robot  : Turtlebot
Steps  : 20

[✓] 服务健康: {'status': 'ok'}
[✓] 可用场景数: 51

创建环境（首次约 2 分钟）...
  env_id=6e90a48f...
  reset OK | obs_keys=3

steps=20 | reward=0.0000 | done=False | 45.2s

[✓] GIF 已保存: behavior_episode.gif
```
> **说明**：因为使用的是随机策略，所以 reward 为 0 是正常现象。实际使用请替换为训练好的策略模型。

四、关键参数说明
仿真服务 API 接口
仿真服务启动后，提供以下 HTTP 接口供推理端调用：

接口	方法	说明
`/health`	GET	健康检查，确认服务是否正常
`/tasks`	GET	列出所有支持的场景、机器人和任务类型
`/env/create`	POST	创建一个新的仿真环境
`/env/{env_id}/reset`	POST	重置环境到初始状态
`/env/{env_id}/step`	POST	执行一个动作，获取结果

|`/env/{env_id}`|DELETE|销毁环境，释放资源|

创建环境示例：

```python
!curl -s -X POST http://localhost:8100/env/create \
  -H "Content-Type: application/json" \
  -d '{"scene_model": "Rs_int", "robot_type": "Turtlebot", "task_type": "PointNavigationTask"}'
```

创建环境参数：

参数	默认值	说明
`scene_model`	`"Rs_int"`	场景名称（51 个可选，见第五节）
`robot_type`	`"Turtlebot"`	机器人类型
`task_type`	`"DummyTask"`	任务类型：DummyTask / PointNavigationTask / BehaviorTask
`activity_name`	null	BEHAVIOR 活动名（task_type 为 BehaviorTask 时必填）
`image_height`	128	图像高度
`image_width`	128	图像宽度

step 请求/响应示例：

请求：

```json
{"action": [0.5, 0.1]}
```

响应：

```json
{
  "reward": -0.001,
  "done": false,
  "truncated": false,
  "info": {...},
  "obs": {"robot_...::rgb": "ndarray(128,128,4)", "task::low_dim": [0.0, ...]},
  "image_base64": "<base64 encoded PNG>",
  "depth_base64": "<base64 encoded PNG>"
}
```

推理客户端参数（behavior_test_client.py）

参数	必选	说明	默认值
`--server`	是	仿真服务器地址（含端口）	-
`--scene`	否	场景名称（见第五节场景列表）	`Rs_int`
`--robot`	否	机器人类型	`Turtlebot`
`--steps`	否	每个轨迹的步数	`20`
`--output`	否	GIF 输出路径	`behavior_episode.gif`

五、常见场景与任务列表

> 以下是 BEHAVIOR-1K 中常用的场景和任务。`scene_model` 参数需填写「场景名称」列中的值。

常用场景

场景名称	说明
`Rs_int`	单层公寓（推荐入门）
`Beechwood_0_int`	Beechwood 别墅底层
`Merom_0_int`	Merom 住宅底层
`Pomaria_0_int`	Pomaria 住宅底层
`Wainscott_0_int`	Wainscott 住宅底层

> **提示**：BEHAVIOR-1K 共支持 51 个场景。如需查看完整列表，可在 Python 中执行：

```python
from omnigibson.utils.asset_utils import get_available_behavior_1k_scenes
print(get_available_behavior_1k_scenes())
```

常用机器人

机器人名称	说明
`Turtlebot`	差速驱动机器人（推荐入门导航任务）
`Fetch`	移动操作机器人（带机械臂）
`FrankaPanda`	固定基座机械臂
`R1Pro`	双臂移动机器人

> 如需查看所有可用机器人，可在 Python 中执行：

```python
import omnigibson as og
print(list(og.REGISTERED_ROBOTS.keys()))
```

任务类型

任务类型	说明
`DummyTask`	空任务（用于环境验证，无目标）
`PointNavigationTask`	导航任务（移动到目标位置）
`BehaviorTask`	行为任务（完成指定日常活动，需配合 activity_name）

常用 BEHAVIOR 活动（需 task_type 为 BehaviorTask）

活动名称	说明
`cleaning_out_fridge`	清理冰箱
`washing_pots_and_pans`	洗锅
`organizing_boxes`	整理箱子
`making_tea`	泡茶
`picking_up_trash`	捡垃圾

> **提示**：BEHAVIOR-1K 共支持 1018 个活动。如需查看完整列表，可在 Python 中执行：

```python
from omnigibson.utils.bddl_utils import BEHAVIOR_ACTIVITIES
print(list(BEHAVIOR_ACTIVITIES)[:20])  # 显示前20个
```

```
print(list(BEHAVIOR_ACTIVITIES)[:20])" # 显示前20个
```

六、最佳实践建议
1. **首次使用**：推荐先运行 CUDA 修复脚本（`bash baige/run_env.sh`）+ 环境验证脚本，确认环境正常后再进行后续操作
2. **快速测试**：使用 `DummyTask` + `Rs_int` 场景进行初步验证，确认流程跑通后再切换到复杂任务
3. **GPU 选型**：务必选择带 RT Cores 的 GPU（RTX 3070/4090/L4/L40），A100/H800/H100 不可用
4. **首次启动慢**：Isaac Sim 首次启动需要编译 shader（~2 分钟），后续启动约 30-80 秒，这是正常现象
5. **远程调用**：如果需要仿真和推理分离部署，使用内置的 `behavior_env_server` + 推理客户端方案
6. **单实例限制**：当前仿真服务仅支持同时运行 1 个环境实例（GPU 内存限制），切换场景/任务需通过 `env/create` 重新创建

七、常见问题（FAQ）
Q1：运行时出现 **`CUDA error 804: forward compatibility was attempted on non supported HW`，怎么办？**

这是百舸容器中 NVIDIA 库 symlink 版本不匹配导致的。运行修复脚本即可：

```bash
bash ~/workspace/BEHAVIOR-1K/baige/run_env.sh
```
如果修复后仍然报错，可手动执行：

```bash
cd /usr/lib/x86_64-linux-gnu
DRIVER_VER=$(cat /proc/driver/nvidia/version | grep -oP 'Module\s+\K[\d.]+')
ln -sf libcuda.so.${DRIVER_VER} libcuda.so.1
ln -sf libnvidia-ptxjitcompiler.so.${DRIVER_VER} libnvidia-ptxjitcompiler.so.1
ln -sf libnvidia-nvvm.so.${DRIVER_VER} libnvidia-nvvm.so.4
```
Q2：运行时出现 **`ERROR_DEVICE_LOST` 或 **`Segmentation fault`**，怎么办？**

最常见的原因是 GPU 不具备 RT Cores。A100、H800、H100 等纯计算卡不支持 Isaac Sim 的光追渲染。

解决方法：更换为具备 RT Cores 的 GPU 机型（RTX 3070/4090/L4/L40 等）。

Q3：运行时出现 **`VK_ERROR_INCOMPATIBLE_DRIVER (-9)`，怎么办？**

这是 NVIDIA 驱动 580.x 与 Isaac Sim 4.5 不兼容导致的。

解决方法：选择提供 535.x 驱动的 GPU 机型。可通过 `nvidia-smi` 查看当前驱动版本。

Q4：环境启动非常慢（>3 分钟），怎么办？

首次启动需要编译 GPU shader（RtPso async compilation），耗时 2-3 分钟是正常的。后续启动会利用 shader 缓存，约 30-80 秒。

如果后续启动仍然很慢，请检查：

1. GPU 驱动是否正确安装（`nvidia-smi` 正常显示）
2. 磁盘 I/O 是否正常（shader 缓存读写受磁盘速度影响）

Q5：远程调用模式下，推理客户端连不上仿真服务器？

请依次检查：

1. 仿真服务是否已启动：在仿真服务器上执行 `curl http://localhost:8100/health`，应返回 `{"status":"ok"}`
2. 两台开发机是否在同一 VPC 网络下 / 正确配置了 BLB
3. BLB 是否正确配置了 8100 端口的监听
4. 防火墙是否放行了 8100 端口

Q6：BEHAVIOR-1K 可以在 CPU 机器上运行吗？

不可以。与 RoboCasa（CPU + OSMesa 可运行）不同，BEHAVIOR-1K 依赖 Isaac Sim 的光追渲染，**必须有带 RT Cores 的 GPU**。

Q7：`OMNIGIBSON_HEADLESS=True`** 设置后仍然报显示相关错误？**

请确保在 `import omnigibson` **之前**设置该环境变量，并且 Xvfb 已启动：

```bash
Xvfb :99 -screen 0 1024x768x24 &
export DISPLAY=:99
export OMNIGIBSON_HEADLESS=True
export OMNI_KIT_ACCEPT_EULA=Yes
```
或直接使用 `bash baige/run_env.sh` 一键配置。

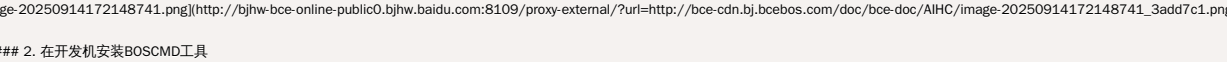
下载平台预置公共数据到用户本地的PFS

这里介绍如何基于开发机，使用BOSCMD工具，将平台公共的数据（模型权重、数据集等）下载到您的并行文件存储PFS

1. 创建开发机

这里我们创建一个百舸开发机，详见：[创建开发机](https://cloud.baidu.com/doc/AIHC/s/Tm6db1z9p)，主要配置：

1. 资源规格：4C8G即可，无需GPU
2. 镜像：推荐选择百舸预置镜像（已经预装BOSCMD工具），如aibox-cuda镜像。
3. 存储挂载：选择我们需要下载数据的PFS，挂载到开发机中。这里作为示例，我们将PFS的根目录(/)挂载到开发机的`mnt/pfs`目录



2. 在开发机安装BOSCMD工具

> 开发机的预置镜像已经内置bcecmd工具，无需执行此步骤

开发机进入运行状态后，登陆开发机

1. 执行以下命令，验证BOSCMD工具是否已经安装。如出现下述提示，则表示已经安装，无需再次安装，直接进行步骤3下载数据

```
```

```
bcecmd
bcecmd: error: command not specified

usage: bcecmd  [--help] [--configure [""]] [--debug] [--conf-path CONF-PATH] [--multiupload-infos-path MULTIUPLOAD-INFOS-PATH] [--version]

Optional flags:
      --help                Show context-sensitive help (also try --help-long and --help-man).
  -c, --configure [""]      configure AK SK Region and Domain for bcecmd and will be written to CONF_PATH(the user's home director by default
  -d, --debug               cli debug
      --conf-path CONF-PATH  config path
      --multiupload-infos-path MULTIUPLOAD-INFOS-PATH
                            multi upload info dir path
  -v, --version             show program's version number and exit

Commands:
  help
    Show help.
  bos
    bos service
  bosapi
    BOS API command.
  bosprobe
    bos probe
  completion
    Generate completion script
```

2. 如未安装，则执行以下的命令安装BOSCMD工具。更多版本详见[安装BOSCMD](https://cloud.baidu.com/doc/BOS/s/qjwvyqegc)。

这里以BOSCMD客户端0.5.9版本为例，在开发机上执行以下命令安装BOSCMD工具

```
wget https://doc.bce.baidu.com/bos-optimization/linux-bcecmd-0.5.9.zip   #下载
unzip linux-bcecmd-0.5.9.zip   #解压
ln linux-bcecmd-0.5.9/bcecmd /usr/sbin/  #设置为全局使用
```

3 . 下载数据

BOSCMD工具配置完成后，数据的上传下载等命令详见：[BCECMD-Object管理](https://cloud.baidu.com/doc/BOS/s/kmcn3zrup)。

这里我们下载openpi/pi0.5的模型权重为例：

```
bos:/aihc-models-bj/openpi/pi0.5/ckpt/pi05_base  #pi0.5权重的地址，无需执行
bcecmd bos sync bos:/aihc-models-bj/openpi/pi0.5/ckpt/pi05_base /mnt/pfs/pi0.5/ckpt/pi05_base  #下载pi0.5权重到开发机机/mnt/pfs/pi0.5/ckpt/pi05_base目录
```

更多示例

- **同步下载bos:/mybucket/pre/下文件到当前目录下的temp目录**

```
bcecmd bos sync bos:/mybucket/pre/ ./temp/
```

- **下载Object**

```
bcecmd bos cp bos:/mybucket/test.txt text.txt
```

使用 OpenPI-0.5 加速镜像进行训练

概述
本文介绍如何在百舸训练环境中，使用 `openpi-0.5` 加速镜像快速启动 `pi05_libero` 训练任务。该镜像已预置训练依赖，并内置了部分 OpenPI 基础资源，适合用于快速验证 OpenPI 0.5 在 LIBERO 数据集上的训练流程与性能表现。

功能介绍
* **预置训练环境**：镜像内依赖已配置完成，启动任务后可直接使用镜像内 Python 环境运行训练。
* **内置基础资源**：镜像内已缓存 OpenPI 相关资源，默认位置为 `/root/workspace/openpi/.cache_openpi`。
* **支持 LIBERO 训练**：默认示例面向 `pi05_libero` 任务与 `physical-intelligence/libero` 数据集。
* **训练性能优化**：原始测试中，优化后 BS256 配置 step time 为 1.290s，samples/s 为 198.4，相比 baseline 提升约 156%。

计费说明
本文仅说明训练启动方式。实际费用以百舸训练任务运行时消耗的算力、存储等资源为准，请结合所在项目的资源与计费规则确认。

说明
开始前请确认具备以下权限或资源访问能力：

* 百舸项目中有可用 GPU 训练资源。
* 本快速开始说明为在双机八卡情况下进行训练的案例，您可以根据您的实际情况变更启动命令。

物料准备
数据集与权重已内置于镜像内，您可以直接使用，也可以参考使用格式替换为您自己的数据。

类型	说明
数据集	* huggingface 地址：https://huggingface.co/datasets/physical-intelligence/libero * bos 下载地址：bos:/aihc-rdw-bj/huggingface.co/datasets/physical-intelligence/libero/
权重	* huggingface 地址：https://huggingface.co/lerobot/pi05_base * bos 下载地址：bos:/aihc-models-bj/lerobot/pi05_base
镜像内置资源	`/root/workspace/openpi/.cache_openpi`中： 包含 `gs://openpi-assets/checkpoints/pi05_base`、 `gs://big_vision/paligemma_tokenizer.model`、

`assets/pi05_libero/physical-intelligence/libero/norm_stats.json`|

快速开始
1. 检查或生成归一化统计文件
镜像内已经对 `pi05_libero` 任务和 `physical-intelligence/libero` 数据集完成归一化，默认路径为：

```
assets/pi05_libero/physical-intelligence/libero/norm_stats.json
```
如果需要重新生成，可执行：

```shell
/root/workspace/openpi/.venv/bin/python3 scripts/compute_norm_stats.py --config-name pi05_libero
```
2. 启动训练
您可以按照实际情况修改以下参数：

```
#### 模型权重
export CHECKPOINT_PATH=

#### 数据集
export DATA_PATH=
```

通过以下代码启动训练：
```shell
JAX_PLATFORMS=cpu \
BATCH_SIZE=256 \
GRAD_ACC=2 \
NUM_TRAIN_STEPS=50 \ CHECKPOINT_PATH=/mnt/pfs/models/pi05_base \
DATA_PATH=/mnt/pfs/datasets bash \
bash /root/workspace/run_train.sh
```

镜像内预置的脚本 `/root/workspace/run_train.sh` 内容：

```shell
#### !/usr/bin/env bash
set -o pipefail
set -x

WORKSPACE_DIR=${WORKSPACE_DIR:-"/root/workspace/openpi"}
cd "$WORKSPACE_DIR"

PY=${PY:-"$WORKSPACE_DIR/.venv/bin/python"}

CHECKPOINT_PATH=${CHECKPOINT_PATH:-"/root/workspace/checkpoints/pi05_base"}
DATA_PATH=${DATA_PATH:-"/root/workspace/datasets/libero"}
ASSETS_BASE_DIR=${ASSETS_BASE_DIR:-"$WORKSPACE_DIR/assets"}
OPENPI_DATA_HOME=${OPENPI_DATA_HOME:-"$WORKSPACE_DIR/.cache_openpi"}
OUTPUT_DIR=${OUTPUT_DIR:-"/root/workspace/outputs"}
TENSORBOARD_PATH=${TENSORBOARD_PATH:-"/root/workspace/tensorboard-log"}
TORCHINDUCTOR_CACHE_DIR=${TORCHINDUCTOR_CACHE_DIR:-"$WORKSPACE_DIR/torchinductor_cache"}

mkdir -p "$OUTPUT_DIR" "$TENSORBOARD_PATH"

BATCH_SIZE=${BATCH_SIZE:-384}
GRAD_ACC=${GRAD_ACC:-2}
GRAD_ACCUM_MODE=${GRAD_ACCUM_MODE:-always_sync}
NUM_TRAIN_STEPS=${NUM_TRAIN_STEPS:-12}
EXP_NAME=${EXP_NAME:-openpi_pi05_libero_gbs${BATCH_SIZE}_acc${GRAD_ACC}}

COMPILE_SCOPE=${COMPILE_SCOPE:-multi_group}
GEMMA_MODE=${GEMMA_MODE:-max-autotune-no-cudagraphs}
GEMMA_GC_EVERY=${GEMMA_GC_EVERY:-0}
BUCKET_MB=${BUCKET_MB:-200}
SAVE_INTERVAL=${SAVE_INTERVAL:-999999}

export JAX_PLATFORMS=cpu
export HF_HUB_OFFLINE=1 TRANSFORMERS_OFFLINE=1
export WANDB_MODE=offline WANDB_DISABLED=true
export HF_LEROBOT_HOME="$DATA_PATH"
export OPENPI_DATA_HOME
export TORCHINDUCTOR_CACHE_DIR
export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True
export TORCHINDUCTOR_MIX_ORDER_REDUCTION=0
export OMP_NUM_THREADS=${OMP_NUM_THREADS:-8}
export TORCH_DYNAMO_RECOMPILE_LIMIT=64
export TORCHDYNAMO_CACHE_SIZE_LIMIT=256

NNODES=${NNODES:-${WORLD_SIZE:-1}}
NODE_RANK=${NODE_RANK:-${RANK:-0}}
MASTER_ADDR=${MASTER_ADDR:-127.0.0.1}
MASTER_PORT=${MASTER_PORT:-23456}
NPROC=${NPROC:-${NPROC_PER_NODE:-8}}
unset RANK WORLD_SIZE LOCAL_RANK
$PY -m torch.distributed.run --nnodes=$NNODES --node_rank=$NODE_RANK --master_addr=$MASTER_ADDR --master_port=$MASTER_PORT --nproc_per_node=$NPROC \
  scripts/train_pytorch.py pi05_libero \
  --exp_name "$EXP_NAME" \
  --ddp-grad-accum-mode "$GRAD_ACCUM_MODE" \
  --batch_size "$BATCH_SIZE" --gradient_accumulation_steps "$GRAD_ACC" \
  --num_train_steps "$NUM_TRAIN_STEPS" --log_interval 1 --save_interval "$SAVE_INTERVAL" --no-wandb_enabled \
  --no-ddp-find-unused-parameters --ddp-bucket-cap-mb "$BUCKET_MB" \
  --model.pytorch-compile-scope "$COMPILE_SCOPE" \
  --model.pytorch-compile-gemma-mode "$GEMMA_MODE" \
  --gemma-gc-every-n-layers "$GEMMA_GC_EVERY" \
  --pytorch_weight_path "$CHECKPOINT_PATH" \
  --assets_base_dir "$ASSETS_BASE_DIR" \
  --checkpoint_base_dir "$OUTPUT_DIR" \
```

```
  "$@"
```

性能参考
配置	step time	samples/s	vs baseline
默认 baseline	3.300s	77.6	—
优化后（BS256）	1.290s	198.4	+156%
优化后（BS384，grad_acc 2, max throughput）	1.779s	215.9	+178%

注意事项
1. **不要默认使用 **`uv`** 运行训练**：镜像内依赖已配置完成，训练应优先使用镜像内 `.venv` 环境。
2. **按实际资源调整并行度与 batch size**：示例中 `NPROC_PER_NODE=8`、`BATCH_SIZE=256`，请根据百舸任务实际 GPU 数量与显存调整。

使用 Cosmos3-Nano-Policy-DROID 加速镜像训练机器人策略模型

概述
Cosmos3-Nano-Policy-DROID 加速镜像面向机器人策略模型训练场景，内置 Cosmos Framework 训练环境，适用于基于 DROID 数据集进行 Cosmos3-Nano-Policy 模型训练与调优。

通过百舸平台，您可以直接选择已适配的训练镜像、GPU 资源规格、数据集和模型权重，快速启动分布式训练任务，无需手动拉起 Docker 容器或自行维护底层运行环境。

适用场景
* 使用 DROID 数据集进行机器人策略模型训练。
* 基于 Cosmos3-Nano 模型进行行动策略微调。
* 在百舸平台上验证单机或多机 GPU 训练性能。
* 需要使用预置训练镜像快速复现实验流程。

环境与资源要求
项目	推荐配置
GPU 规格	推荐使用有图形加速的高显存规格显卡
单机训练	单机 8 卡
多机训练	双机 16 卡，可按需扩展
存储空间	如果您需要下载资源测试，建议不少于 600G
CUDA / 驱动	建议使用满足平台镜像要求的默认驱动环境
工作目录	`/root/workspace/cosmos-framework`
数据集目录	`/root/workspace/datasets/droid_lerobot_v30`
模型目录	`/root/workspace/models`
输出目录	`/root/workspace/outputs`

> 说明：本文中的路径均以百舸训练任务内的 `/root/workspace` 为例。实际使用时，可根据任务挂载目录调整。

准备工作
如果您所在网络环境访问 HuggingFace 较慢，建议提前配置 HuggingFace 镜像源，或配置[代理加速](https://cloud.baidu.com/doc/BCM/s/3mmswg2oh)。

huggingface 镜像配置：

```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
```
百舸平台已提供数据集和模型权重的 BOS 地址，您也可以直接从 BOS 下载到训练任务的工作目录，避免依赖外部网络。

步骤二：准备数据集
方式一：通过 HuggingFace 下载
如果您选择从 HuggingFace 下载数据集，请先配置镜像源，然后执行下载命令。

```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1

mkdir -p /root/workspace/datasets/droid_lerobot_v30
huggingface-cli download lerobot/droid_1.0.1 \
  --repo-type dataset \
  --local-dir /root/workspace/datasets/droid_lerobot_v30
```
方式二：通过 BOS 下载
如果已提供 DROID 数据集的 BOS 地址，建议优先通过 BOS 下载。

```bash
mkdir -p /root/workspace/datasets/droid_lerobot_v30
bcecmd bos cp -r bos:/aihc-datasets/huggingface.co/datasets/lerobot/droid_1.0.1/v3.0/ /root/workspace/datasets/droid_lerobot_v30/
```
可选：规范化 tasks.parquet
部分 LeRobot v3 格式的 DROID 数据集可能会将任务文本写入 `__index_level_0__` 列。当前训练代码已兼容 `task` 与 `__index_level_0__` 两种 schema，因此该步骤不是必需项，仅用于将数据集元信息统一为更清晰的结构。

```bash
cd /root/workspace/cosmos-framework
python -m cosmos_framework.scripts.normalize_droid_tasks_parquet \
  /root/workspace/datasets/droid_lerobot_v30 \
  --json
```
如果数据集已符合规范，脚本会自动跳过处理。

验证数据集
下载完成后，可以检查数据集关键文件是否存在。

```bash
python3 - <<'PY'
from pathlib import Path
root = Path('/root/workspace/datasets/droid_lerobot_v30')
assert (root / 'meta/info.json').exists(), 'meta/info.json 不存在，请检查数据集路径'
data_files = list((root / 'data').glob('chunk-*/file-*.parquet'))
video_files = list((root / 'videos').glob('**/*.mp4'))
print(f'data parquets: {len(data_files)}')
print(f'video mp4s: {len(video_files)}')
PY
```

步骤三：准备模型权重
训练需要准备 Cosmos3-Nano 权重和 Wan2.2-TI2V-5B 相关权重。

方式一：通过 HuggingFace 下载并转换
```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
export HF_HOME=/root/workspace/models/huggingface
export HF_HUB_CACHE=$HF_HOME/hub
export TRANSFORMERS_CACHE=$HF_HOME/transformers
export HF_DATASETS_CACHE=$HF_HOME/datasets

cd /root/workspace/cosmos-framework
python -m cosmos_framework.scripts.convert_model_to_dcp \
  --checkpoint-path Cosmos3-Nano \
  -o /root/workspace/models/Cosmos3-Nano-DCP
```
方式二：通过 BOS 下载
如果已提供转换后的模型权重 BOS 地址，建议直接下载到 `/workspace/models` 下。

```bash
mkdir -p /root/workspace/models/Cosmos3-Nano/
bcecmd bos cp -r bos:/aihc-models-bj/nvidia/Cosmos3-Nano/ /root/workspace/models/Cosmos3-Nano/

mkdir -p /root/workspace/models/Wan2.2-TI2V-5B
bcecmd bos cp -r bos:/aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B/ /root/workspace/models/Wan2.2-TI2V-5B/

mkdir -p /root/workspace/models/Qwen3-VL-8B-Instruct
bcecmd bos cp -r bos:/aihc-models-bj/Qwen/Qwen3-VL-8B-Instruct/ /root/workspace/models/Qwen3-VL-8B-Instruct/
```
因为原权重转换脚本会从 huggingface 在线下载权重，此处可以通过以下代码生成本地读取脚本进行替换：

```bash
cat > /root/workspace/cosmos-framework/convert_local.py << CONVERT_SCRIPT
import os
os.environ["COSMOS_DEVICE"] = "cpu"
os.environ["HF_HUB_OFFLINE"] = "1"
os.environ["HF_HOME"] = "/root/workspace/models/huggingface"

from cosmos_framework.inference.common.init import init_script
init_script(env={"COSMOS_DEVICE": "cpu"})

import cosmos_framework.utils.checkpoint_db as ckpt_db
_original_hf_download = ckpt_db._hf_download

LOCAL_FILE_MAP = {
    ("Wan-AI/Wan2.2-TI2V-5B", "Wan2.2_VAE.pth"): "/root/workspace/models/Wan2.2-TI2V-5B/Wan2.2_VAE.pth",
}
LOCAL_DIR_MAP = {
    "Qwen/Qwen3-VL-8B-Instruct": "/root/workspace/models/Qwen3-VL-8B-Instruct",
    "nvidia/Cosmos3-Nano": "/root/workspace/models/Cosmos3-Nano",
}

def _patched_hf_download(cmd_args):
    repo_id = cmd_args[0] if cmd_args else None
    filename = None
    for arg in cmd_args[1:]:
        if not arg.startswith("--") and "." in arg:
            filename = arg
            break
    if (repo_id, filename) in LOCAL_FILE_MAP:
        local = LOCAL_FILE_MAP[(repo_id, filename)]
        print(f"[PATCH] Redirecting file {repo_id}/{filename} -> {local}")
        return local
    if repo_id in LOCAL_DIR_MAP:
        local = LOCAL_DIR_MAP[repo_id]
        for i, arg in enumerate(cmd_args):
            if arg == "--include" and i + 1 < len(cmd_args):
                subdir = cmd_args[i + 1].split("/")[0] if "/" in cmd_args[i + 1] else ""
                if subdir and os.path.isdir(os.path.join(local, subdir)):
                    local = os.path.join(local, subdir)
                    break
        print(f"[PATCH] Redirecting dir {repo_id} -> {local}")
        return local
    print(f"[WARN] No local mapping for {repo_id}, attempting original download...")
    return _original_hf_download(cmd_args)

ckpt_db._hf_download = _patched_hf_download

import cosmos_framework.configs.base.defaults.vlm as vlm_mod
_original_download_tokenizer = vlm_mod.download_tokenizer_files

def _patched_download_tokenizer(model_name, config_variant):
    if model_name in LOCAL_DIR_MAP:
        print(f"[PATCH] Using local tokenizer: {LOCAL_DIR_MAP[model_name]}")
        return LOCAL_DIR_MAP[model_name]
    return _original_download_tokenizer(model_name, config_variant)

vlm_mod.download_tokenizer_files = _patched_download_tokenizer

from cosmos_framework.scripts.convert_model_to_dcp import main
main()
CONVERT_SCRIPT
```
执行以下脚本即可实现本地权重转换：

```bash
cd /root/workspace/cosmos-framework && source .venv/bin/activate
PYTHONPATH=. python convert_local.py \
```

```
  --checkpoint-path /root/workspace/models/Cosmos3-Nano \
  -o /root/workspace/models/Cosmos3-Nano-DCP
```
步骤四：配置训练环境变量
在百舸训练任务启动命令中，配置以下环境变量。路径可根据实际挂载位置调整。

```bash
export WORKSPACE_DIR=/root/workspace/cosmos-framework
export DROID_ROOT=/root/workspace/datasets/droid_lerobot_v30
export BASE_CHECKPOINT_PATH=/root/workspace/models/Cosmos3-Nano-DCP
export WAN_VAE_PATH=/root/workspace/models/Wan2.2-TI2V-5B/Wan2.2_VAE.pth
export IMAGINAIRE_OUTPUT_ROOT=/root/workspace/outputs

export HF_HOME=/root/workspace/models/huggingface
export HF_HUB_OFFLINE=1

#### 避免部分 Inductor/Triton 编译场景下出现资源不足问题
export TORCHINDUCTOR_MIX_ORDER_REDUCTION=0
export COSMOS_GPU_COLOR_JITTER=1
export COSMOS_AC_LAYER_POLICY=full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,full,none,none,none,none,none,none,none,none,none,none,none,none

export GPUS_PER_NODE=8
```
多机训练时，百舸平台通常会注入或提供分布式训练所需的节点信息。请根据平台任务环境设置以下变量。

```bash
export MASTER_ADDR=${MASTER_ADDR:-"127.0.0.1"}
export MASTER_PORT=${MASTER_PORT:-"6100"}
export NNODES=${WORLD_SIZE:-"1"}
export NODE_RANK=${RANK:-"0"}
```
步骤五：启动训练
在百舸训练任务的启动命令中使用以下命令。

```bash
cd /root/workspace/cosmos-framework
source .venv/bin/activate

PYTHONPATH=. torchrun \
  --nnodes=$NNODES \
  --node_rank=$NODE_RANK \
  --master_addr=$MASTER_ADDR \
  --master_port=$MASTER_PORT \
  --nproc_per_node=$GPUS_PER_NODE \
  -m cosmos_framework.scripts.train \
  --sft-toml examples/toml/sft_config/action_policy_droid_repro.toml \
  -- trainer.max_iter=10000 \
     model.config.parallelism.data_parallel_replicate_degree=$NNODES \
     trainer.logging_iter=50 \
     checkpoint.save_iter=1000 \
     dataloader_train.max_samples_per_batch=32 \
     ++dataloader_train.dataloader.num_workers=8 \
     ++dataloader_train.dataloader.prefetch_factor=16 \
     ++dataloader_train.dataloader.pin_memory=true \
     ++dataloader_train.dataloader.persistent_workers=true \
     model.config.compile.enabled=true \
     model.config.compile.compiled_region=language \
     model.config.compile.compile_dynamic=true \
     model.config.compile.use_cuda_graphs=false \
     model.config.activation_checkpointing.mode=full \
     trainer.callbacks.compile_tokenizer.enabled=true \
     trainer.callbacks.compile_tokenizer.compile_after_iterations=3 \
     'trainer.callbacks.compile_tokenizer.warmup_resolutions=["480"]' \
     'trainer.callbacks.compile_tokenizer.aspect_ratio=3\,4' \
     trainer.callbacks.compile_tokenizer.mode=null \
     job.wandb_mode=disabled
```
性能参考
在单机 8 卡测试环境下，参考性能如下。

指标	参考值
Iter Time	11.89 s
SPS / GPU	2.69
Global SPS	21.5
MFU	约 42%
显存占用	约 84.7 GiB
数据集初始化	约 25 s

> 以上数据为特定测试环境下的参考值。实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

常见问题
训练过程中出现 Inductor 或 Triton 资源不足
如果训练过程中出现与 Inductor 或 Triton 编译相关的资源不足问题，可以优先确认是否已设置以下环境变量。

```bash
export TORCHINDUCTOR_MIX_ORDER_REDUCTION=0
```
如果仍然出现显存不足，可以尝试适当降低单批次样本数。

```bash
dataloader_train.max_samples_per_batch=16
```
后续操作
训练任务启动后，可以在百舸平台查看任务日志、资源利用率和输出目录。训练产物默认输出到：

```
/root/workspace/outputs
```
如需保留训练结果，可在任务结束后将输出目录同步到 BOS 或其他持久化存储。

开发机 WebIDE 或本地 IDE 卡顿：避免扫描共享存储目录

适用场景
当开发机中挂载了共享存储（PFS、CFS、BOS、RapidFS等）目录时，建议不要直接用 WebIDE 打开这些目录或其上级目录。

最佳实践：IDE中只打开代码目录，共享存储目录通过命令行或业务程序访问。

为什么需要这样配置
WebIDE 会对当前工作区做文件监听和 Git 仓库探测。如果工作区包含共享存储目录，IDE 可能会递归扫描这些目录，导致大量文件元数据并发访问。

共享存储目录的文件元数据访问需要经过网络请求，递归扫描会放大访问量，可能导致 WebIDE 文件树、Git 面板、终端操作或开发机访问体验变慢。

不建议被 IDE 扫描的目录类型
以下类型属于远端 / 网络挂载目录，不建议放入 IDE 工作区扫描范围：

类型	说明
PFS	并行文件系统
CFS	云文件存储 / NAS 类文件系统
BOS	对象存储通过文件系统方式挂载后的目录
RapidFS	远端 / 网络挂载文件系统

CDS 云磁盘、CFS持久盘、本地系统盘、本地数据盘等本地块存储目录通常不受此问题影响。

Notebook 是否也需要注意
Notebook 通常不会像 IDE Git 扩展一样主动递归扫描整个工作区，也不会默认持续探测 `.git` 仓库。因此，单纯打开 Notebook 一般不等同于 IDE 的持续扫描行为。

但在以下情况，Notebook 仍可能触发远端存储的大量文件访问：

* Notebook 的启动目录设置在 `/root`、`/mnt`、`/data` 等较大的父目录下，文件浏览器会展示包含远端存储的目录树。
* 用户在 Notebook 中执行了递归遍历、批量读取、`glob`、`os.walk`、`find`、`du` 等操作。
* 安装的 JupyterLab 扩展、文件搜索插件或自动索引功能访问了远端存储目录。
* Notebook 中的训练、数据处理代码把远端存储目录作为数据集根目录进行全量扫描。

建议：Notebook 的启动目录也尽量设置为具体项目目录或 Notebook 目录，不要设置为包含共享存储挂载点的上级目录。需要访问共享存储时，在代码中按需读取具体路径，避免全目录递归扫描。

推荐配置方式
1.IDE 工作区只打开没有【存储挂载】的目录
调整工作区，只显示没有【存储挂载】的目录，或者直接设置为一个空目录地址（例如/opt）

设置方式： 在 IDE 中：

1. 点击 `File > Close Folder`，关闭当前工作区。
2. 点击 `File > Open Folder`，只选择代码目录，例如：

```bash
/root/workspace/project/your-project
```
不要直接打开包含远端存储挂载点的父目录，例如：

```bash
/root
/mnt
/data
```
2.配置 WebIDE 用户设置
如 WebIDE 只是卡顿，依然能运行，则以本章节方式调整 WebIDE 用户设置，如已经无法运行，请参考后面的SSH/Notebook方式

2.1.打开配置文件
通过WebIDE打开配置页面（本地 IDE 设置方法类似）


右上角红圈中的按钮，就可以把配置文件打开


2.2.打开配置文件并修改配置
在找到的该文件中添加或合并以下配置，通常一旦修改文件，配置立即生效。请根据实际挂载路径调整 `/mnt/pfs`、`/mnt/cfs` 等路径：

```json
{
  ... # 这里是其他内容，下面是要添加或合并的内容
  "files.watcherExclude": {
    "/mnt/pfs/**": true,
    "/mnt/cfs/**": true,
    "/mnt/bos/**": true,
    "/mnt/rapidfs/**": true,
    "/mnt/nfs/**": true
  },
  "files.exclude": {
    "/mnt/pfs": true,
    "/mnt/cfs": true,
    "/mnt/bos": true,
    "/mnt/rapidfs": true,
    "/mnt/nfs": true
  },
  "git.autoRepositoryDetection": "openEditors",
  "git.ignoredRepositories": [
    "/mnt/pfs",
    "/mnt/cfs",
    "/mnt/bos",
    "/mnt/rapidfs",
    "/mnt/nfs"
```

```
    "/mnt/nfs"
  ]
}
```
配置说明：

* `files.watcherExclude`：避免 WebIDE 持续监听共享存储目录。
* `files.exclude`：在 WebIDE 文件树中隐藏远端存储目录，减少误操作。
* `git.autoRepositoryDetection`：只对左侧目录内打开的文件进行Git扫描
* `git.ignoredRepositories`：显式忽略共享存储目录下的 Git 仓库扫描

如何确认配置是否生效
1. WebIDE 文件树中不再展示共享存储目录。
2. Git 面板不再长时间处于扫描或刷新状态。
3. 终端命令、文件浏览、代码编辑响应恢复正常。

如果远端存储目录仍然出现在文件树中，请确认配置中的路径是否和实际挂载路径一致。可以通过以下命令查看挂载路径：

```bash
mount | grep -Ei 'pfs|cfs|bos|rapidfs|nfs'
df -h | grep -Ei 'pfs|cfs|bos|rapidfs|nfs'
```
3.SSH/Notebook 配置 WebIDE 用户设置
如果WebIDE已经无法运行，请使用【关闭WebIDE后用SSH连接】或【从WebIDE切换到Notebook】方式进行设置

通过上面的方式，对WebIDE用户设置文件 `/root/.local/share/code-server/User/settings.json` 中添加或合并以下配置

修改内容和WebIDE直接修改一样，请根据实际挂载路径调整 `/mnt/pfs`、`/mnt/cfs` 等路径：

```json
{
  ... # 这里是其他内容，下面是要添加或合并的内容
  "files.watcherExclude": {
    "/mnt/pfs/**": true,
    "/mnt/cfs/**": true,
    "/mnt/bos/**": true,
    "/mnt/rapidfs/**": true,
    "/mnt/nfs/**": true
  },
  "files.exclude": {
    "/mnt/pfs": true,
    "/mnt/cfs": true,
    "/mnt/bos": true,
    "/mnt/rapidfs": true,
    "/mnt/nfs": true
  },
  "git.autoRepositoryDetection": "openEditors",
  "git.ignoredRepositories": [
    "/mnt/pfs",
    "/mnt/cfs",
    "/mnt/bos",
    "/mnt/rapidfs",
    "/mnt/nfs"
  ]
}
```
配置说明：

* `files.watcherExclude`：避免 WebIDE 持续监听共享存储目录。
* `files.exclude`：在 WebIDE 文件树中隐藏共享存储目录，减少误操作。
* `git.autoRepositoryDetection`：只对左侧目录内打开的文件进行Git扫描
* `git.ignoredRepositories`：显式忽略共享存储目录下的 Git 仓库扫描

使用 GR00T N1.7 加速镜像训练机器人操作策略模型

概述

GR00T N1.7 加速镜像面向机器人操作策略模型的 embodiment finetune 场景，内置 NVIDIA Isaac GR00T 训练环境与预置执行脚本，适用于基于 GR00T-N1.7-3B 基座模型，在 DROID、BOWL 等数据集上进行操作策略模型的训练与调优。

通过百舸平台，您可以直接选择已适配的 RPBZZZ6（sm_120 Blackwell）加速镜像、GPU 资源规格、数据集与模型权重，通过开发机 SSH 连接快速启动单机 8 卡分布式训练任务，无需手动拉起 Docker 容器或自行维护底层运行环境。

适用场景

- 使用 DROID 或 BOWL 数据集进行机器人操作策略模型训练。
- 基于 GR00T-N1.7-3B 基座模型进行 embodiment finetune。
- 在百舸平台上验证单机 8 卡 GPU 训练性能。
- 需要使用预置训练脚本和执行环境快速复现实验流程。

环境与资源要求

项目	推荐配置
GPU 规格	sm_120 Blackwell（RPBZZZ6 96G 或同等），单机 8 卡
存储空间	建议可用不少于 100G，推荐使用PFS `/mnt/pfs`，
CUDA / 驱动	13.2
PyTorch	2.12.0
工作目录	`/root/workspace/Isaac-GR00T`（训练代码）
启动脚本	`/root/workspace/run_train.sh`（env 覆盖式）
数据集目录	内置 bowl：`/root/workspace/Isaac-GR00T/demo_data/cube_to_bowl_5`
模型目录	建议外部存储，如 PFS：`/mnt/pfs/models/`
输出目录	建议外部存储，如 PFS：`/mnt/pfs/output/gr00t/`

说明：本文中的代码路径均以百舸训练任务内的 `/root/workspace` 为例，模型与输出等持久化路径以外部存储 `/mnt/pfs` 为例。实际使用时可根据任务挂载目录调整；所有下载资源必须落外部存储（如 PFS 路径 `/mnt/pfs/...`），禁止写入根盘。

````
##### 准备工作

百舸平台已提供数据集和模型权重的 BOS 地址，您可以直接从 BOS 下载到训练任务的外部存储目录，避免依赖外部网络。本文涉及的 BOS 地址均为公共读，无需配置 AK/SK。

##### 步骤一：准备数据集

###### 方式一：使用内置 BOWL 数据集（推荐 smoke 验证）

BOWL 数据集已内置在代码仓库中，无需额外下载，可直接用于快速验证训练链路是否打通：

```bash
ls /root/workspace/Isaac-GR00T/demo_data/cube_to_bowl_5/
```

##### 步骤二：准备模型权重

训练需要准备 GR00T-N1.7-3B 基座模型和 Cosmos VLM 权重。建议优先通过 BOS 下载到外部存储目录。

```bash
#### 基座模型 GR00T-N1.7-3B（约 6.5G）
mkdir -p /mnt/pfs/models/GR00T-N1.7-3B
cd /mnt/pfs/models/GR00T-N1.7-3B
bcecmd bos sync bos:/aihc-models-bj/nvidia/GR00T-N1.7-3B/ ./ --concurrency 16 --yes

#### Cosmos VLM（约 4.6G）
mkdir -p /mnt/pfs/models/Cosmos-Reason2-2B
cd /mnt/pfs/models/Cosmos-Reason2-2B
bcecmd bos sync bos:/aihc-models-bj/nvidia/Cosmos-Reason2-2B/ ./ --concurrency 16 --yes
```

###### 验证下载

下载完成后，可以检查模型关键文件是否存在：

```bash
#### 基座模型：分片 safetensors
ls /mnt/pfs/models/GR00T-N1.7-3B/model.safetensors.index.json

#### Cosmos VLM
ls /mnt/pfs/models/Cosmos-Reason2-2B/model.safetensors
```

##### 步骤三：配置训练参数

启动脚本 `/root/workspace/run_train.sh` 全量支持环境变量覆盖，您只需传入想修改的变量，其余走默认值即可。

###### 默认配置（GBS=3072，bowl/SO100，smoke 验证）

无需额外配置，直接启动即可。

###### 自定义配置示例

```bash
export PFS_ROOT=/mnt/pfs
export HF_HOME=/mnt/pfs/models
export BASE_MODEL=/mnt/pfs/models/GR00T-N1.7-3B
export COSMOS_MODEL_PATH=/mnt/pfs/models/Cosmos-Reason2-2B
export GBS=2048
export MAX_STEPS=500
```

###### 关键环境变量说明

|变量|默认值|说明|
|-|-|-|
|`GBS`|`3072`|Global Batch Size|
|`MAX_STEPS`|`20`|最大训练步数|
|`NUM_GPUS`|`8`|GPU 数量|
|`PFS_ROOT`|—|PFS 用户根目录，设为 `/mnt/pfs`|
|`HF_HOME`|—|HuggingFace 缓存目录，设为 `/mnt/pfs/models`|
|`BASE_MODEL`|—|基座模型路径|
|`COSMOS_MODEL_PATH`|—|Cosmos VLM snapshot 目录（含 model.safetensors）|
|`GR00T_BACKBONE_COMPILE`|`1`|backbone torch.compile 开关，OOM 时设为 `0`|
|`GR00T_STEP_TIME`|`0`|设为 `1` 打印每步 GPU 耗时|
|`HF_HUB_OFFLINE`|`1`|强制离线模式|
|`GR00T_GPU_PATCHIFY`|`1`|GPU patchify，消除 collator CPU 瓶颈|

##### 步骤四：启动训练

```bash
cd /root/workspace/Isaac-GR00T

export PFS_ROOT=/mnt/pfs
export HF_HOME=/mnt/pfs/models
export BASE_MODEL=/mnt/pfs/models/GR00T-N1.7-3B
export COSMOS_MODEL_PATH=/mnt/pfs/models/Cosmos-Reason2-2B
export GBS=3072
export MAX_STEPS=20
export GR00T_BACKBONE_COMPILE=1
export GR00T_STEP_TIME=1

bash /root/workspace/run_train.sh
```

> 建议使用 tmux 后台运行，避免 SSH 断开导致训练中断。

##### 性能参考

在单机 8 卡 RPBZZZ6 测试环境下，参考性能基线如下：
````

配置	Runtime	Samples/s	Step Time	Loss	峰值显存
GBS=64, bowl	84s	15.2	~336ms	1.098	~9G
GBS=2048, bowl	128s	319.8	~1416ms	1.083	~56G
GBS=3072, bowl（默认）	140s	438.1	~2060ms	1.083	~84G

> 镜像已预置 `/tmp/torchinductor_root`（1.4G）编译缓存，包含 GBS=64/2048/3072 三种 shape 的 triton kernel，相同 shape 训练可免编译。以上数据为特定测试环境下的参考值，实际性能会受资源规格、数据读取方式、任务参数等因素影响。

常见问题

编译期 OOM（最常见）

现象：step1 崩溃，报错含 `InductorError: CUDA driver error: out of memory`。

根因：`GR00T_BACKBONE_COMPILE=1` 首步编译 triton kernel 时打爆 CUDA driver module 内存，并非显存不足。

解决方案：

方案	做法	说明
烘焙缓存（推荐）	镜像已预置缓存	同 shape 免编译，不 OOM
关闭 backbone compile	`export GR00T_BACKBONE_COMPILE=0`	慢约 4%，任意 shape 兜底
连续冷启动 warm	跑 2 次冷启动	第 1 次 OOM 跑一半入缓存，第 2 次跑通

GPU 架构要求

必须使用 sm_120 Blackwell 架构 GPU，inductor 缓存 cubin 不可跨架构移植。

后续操作

训练任务启动后，可以在百舸平台查看任务日志、资源利用率和输出目录。训练产物默认输出到 `/mnt/pfs/output/gr00t/`。如需长期保留训练结果，可在任务结束后将输出目录同步到 BOS 或其他持久化存储。

在百舸上使用 RLinf加速镜像训练 OpenPI

概述

RLinf OpenPI Torch 加速镜像面向具身策略模型监督微调（SFT）场景，在社区官方镜像的OpenPI Torch版本对齐JAX版精度的基础之上，进行了性能优化，并将官方镜像基础环境进行了 aihc.lgn7ib 实例架构适配。

通过百舸平台，您可以直接选择已适配的训练镜像、GPU 资源规格、数据集和模型权重，使用通用环境变量启动单机或多机分布式训练任务，无需手动拉起 Docker 容器或自行维护底层运行环境。

适用场景

- 使用 LIBERO 数据集训练 OpenPI π0.5 策略模型。
- 基于 `pi05_base` 基座权重进行 Full SFT 或 LoRA SFT。
- 在百舸平台上验证单机 / 多机 GPU 训练性能。
- 需要 torch 2.13 + CUDA 13.2 + cuBLAS 13.5 加速运行时的场景。

环境与资源要求

项目	推荐配置
GPU 规格	aihc.lgn7ib ，单卡约 96-97 GiB
单机训练	单机 8 卡
多机训练	双机 16 卡，可按需扩展
工作目录	`/root/workspace/RLinf`
启动脚本	`/root/workspace/run_libero_openpi_rlinf.sh`
训练入口	`/root/workspace/RLinf/examples/sft/train_vla_sft.py`
默认配置	`libero_sft_openpi_rlinf`
数据集目录	`/root/workspace/asset/datasets/physical-intelligence/libero`
模型权重目录	`/root/workspace/asset/models/pi05_base_new`
Python 环境	`/opt/venv/openpi`

说明：本文中的路径均以百舸训练任务内的 /root/workspace 为例。实际使用时，可根据任务挂载目录调整。

加速镜像软件栈

本镜像与官方 `agentic-rlinf0.3-maniskill_libero`（torch 2.11.0+cu130）的核心差异在 openpi 运行环境，实测版本如下：

组件	版本
Python	3.12.13
torch	2.13.0+cu132
torch CUDA runtime	13.2
nvidia-cublas	13.5.1.27
nvidia-cudnn	9.20.0.48（92000）
NCCL	2.29.7
triton	3.7.1
flash-attn	2.8.3.post1
jax / jaxlib	0.11.0
numpy	2.5.2
torchvision	0.28.0+cu132
torchcodec	0.15.0+cu130
lerobot	0.1.0
ray	2.56.0
openpi-client	0.1.2
rlinf-openpi	0.1.1
rlinf-transformer-openpi	4.53.2

准备工作

建议先配置 HuggingFace 镜像源（如需从 HuggingFace 拉取资源）：

```bash
```

```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
```

百舸平台已提供数据集和模型权重的 BOS 地址，建议直接从 BOS 下载或使用平台挂载，避免依赖外部网络。

步骤一：连接并预检资源

连接开发机后，先确认 GPU、显存、数据集、权重、镜像环境是否就绪：

```bash
nvidia-smi
nvidia-smi --query-gpu=index,name,memory.total,memory.used --format=csv
free -h
ls -ld /root/workspace/RLinf
ls /root/workspace/run_libero_openpi_rlinf.sh
ls -ld /root/workspace/asset/datasets/physical-intelligence/libero
ls -l /root/workspace/asset/models/pi05_base_new/model.safetensors
```

确认 openpi 运行环境：

```bash
/opt/venv/openpi/bin/python -c "import torch; print(torch.__version__, torch.version.cuda, torch.cuda.get_arch_list())"
```

预期输出 `torch 2.13.0+cu132`、`13.2`。

步骤二：准备数据集

OpenPI SFT 要求数据集为 LeRobot format，目录必须包含 `meta/info.json`。

```bash
mkdir -p /root/workspace/asset/datasets/physical-intelligence/libero
bcecmd bos cp -r bos:/aihc-rdw-bj/huggingface.co/datasets/physical-intelligence/libero/ \
  /root/workspace/asset/datasets/physical-intelligence/libero/ --yes
```

验证数据集结构：

```bash
python3 - <<'PY'
from pathlib import Path
root = Path('/root/workspace/asset/datasets/physical-intelligence/libero')
assert (root / 'meta/info.json').exists(), 'meta/info.json 不存在'
print('parquet:', len(list(root.glob('data/**/*.parquet'))))
print('videos:', len(list(root.glob('videos/**/*'))))
PY
```

步骤三：准备模型权重与转换

训练时 `BASE_MODEL_PATH` 必须指向包含 `model.safetensors` 的 PyTorch 原生（new 格式）checkpoint 目录（本镜像默认使用 `pi05_base_new`）。BOS 上没有 new 格式，只有 old 格式的 `pi05_base`，需要本地执行 old→new 转换。

涉及三个目录：

目录	格式	用途
`pi05_base`	old 格式	转换输入，仅需 `model.safetensors` + `config.json` 两个文件
`pi05_base_new`	new 格式	训练的 `model_path`（`BASE_MODEL_PATH` 默认值）
`RLinf-Pi05-LIBERO-SFT`	assets	存放 norm stats

3.1 下载 old 格式基座与 norm stats

```bash
#!/usr/bin/env bash
set -eo pipefail

PFS_ROOT=/root/workspace/asset
BASE_DIR=${PFS_ROOT}/models/pi05_base          # old 格式，转换的输入
MODEL_DIR=${PFS_ROOT}/models/pi05_base_new     # new 格式，训练的 model_path
ASSETS_DIR=${PFS_ROOT}/models/rlinf/RLinf-Pi05-LIBERO-SFT
NORM_STATS=${ASSETS_DIR}/physical-intelligence/libero/norm_stats.json

mkdir -p "${BASE_DIR}" "${MODEL_DIR}" "${ASSETS_DIR}/physical-intelligence/libero"

# pi0.5 基座权重，old 格式。只需两个文件
bcecmd bos cp "bos:/aihc-models-bj/lerobot/pi05_base/model.safetensors" "${BASE_DIR}/model.safetensors"
bcecmd bos cp "bos:/aihc-models-bj/lerobot/pi05_base/config.json"       "${BASE_DIR}/config.json"

# norm stats：整仓 7.5 G，只取这一个 4507 B 的文件
bcecmd bos cp \
  "bos:/aihc-models-bj/RLinf/RLinf-Pi05-LIBERO-SFT/physical-intelligence/libero/norm_stats.json" \
  "${NORM_STATS}"
```

3.2 old → new 格式转换

```bash
#!/usr/bin/env bash
set -eo pipefail

PFS_ROOT=/root/workspace/asset
DATASET_DIR=${PFS_ROOT}/datasets/physical-intelligence/libero
BASE_DIR=${PFS_ROOT}/models/pi05_base          # old 格式，转换的输入
MODEL_DIR=${PFS_ROOT}/models/pi05_base_new     # new 格式，训练的 model_path
ASSETS_DIR=${PFS_ROOT}/models/rlinf/RLinf-Pi05-LIBERO-SFT
```

```
NORM_STATS=${ASSETS_DIR}/physical-intelligence/libero/norm_stats.json

#### 4) old -> new 转换（BOS 无 new 格式，必做）
#### 纯 CPU，峰值约 30 G 内存；丢弃 action-expert 的 lm_head，
#### 812 张量/3.6168B -> 667 张量/3.3534B，不改 dtype（仍 fp32）
cd "${REPO_PATH:-/root/workspace/RLinf}"
python -m rlinf.utils.ckpt_convertor.openpi.convert old2new \
  --input-model       "${BASE_DIR}/model.safetensors" \
  --input-norm-stats  "${NORM_STATS}" \
  --output-model      "${MODEL_DIR}" \
  --output-norm-stats "${MODEL_DIR}/physical-intelligence/libero/norm_stats.json"

#### 5) 校验（只读）。期望 model.safetensors 13413809816 B，两份 norm_stats sha256 相同
ls -l "${MODEL_DIR}/model.safetensors" "${NORM_STATS}"
sha256sum "${NORM_STATS}" "${MODEL_DIR}/physical-intelligence/libero/norm_stats.json"
du -sh "${DATASET_DIR}" "${BASE_DIR}" "${MODEL_DIR}"
#### 转换后 BASE_DIR（13.5 GiB）可删，训练只读 MODEL_DIR
```

转换说明：

- 纯 CPU 运行，峰值约 30 G 内存。
- 丢弃 action-expert 的 `lm_head`，张量数 812 → 667（3.6168B → 3.3534B）。
- 不改 dtype，仍为 fp32。

3.3 校验

```bash
#### 期望 model.safetensors 为 13413809816 B，两份 norm_stats sha256 相同
ls -l "${MODEL_DIR}/model.safetensors" "${NORM_STATS}"
sha256sum "${NORM_STATS}" "${MODEL_DIR}/physical-intelligence/libero/norm_stats.json"
du -sh "${PFS_ROOT}/datasets/physical-intelligence/libero" "${BASE_DIR}" "${MODEL_DIR}"
```

转换完成后 `BASE_DIR`（约 13.5 GiB）可删除，训练只读 `MODEL_DIR`。转换产物 `${MODEL_DIR}/physical-intelligence/libero/norm_stats.json` 即训练时 openpi 读取的 norm stats（与 `ASSETS_PATH` 指向同一目录 `/mnt/pfs/models/pi05_base_new`）。

步骤四：配置训练环境变量

启动脚本 `/root/workspace/run_libero_openpi_rlinf.sh` 已集中定义全部可覆盖变量，路径均可通过通用环境变量调整：

变量	默认值	含义
`WORKSPACE_DIR`	`/root/workspace/RLinf`	仓库根
`DATA_PATH`	`/root/workspace/asset/datasets/physical-intelligence/libero`	LIBERO 数据集根
`BASE_MODEL_PATH`	`/root/workspace/asset/models/pi05_base_new`	pi0.5 基座权重
`ASSETS_PATH`	`/root/workspace/asset/models/pi05_base_new`	openpi assets
`PROFILE`	`opt_all`	`baseline` / `mbs32` / `opt_all`
`NUM_NODES`	1	节点数（多机时设 >1）
`GBS`	自动 = mbs×8×NUM_NODES	全局 batch size
`RAY_PORT`	6379	Ray GCS 端口
`RLINF_RENDEZVOUS_DIR`	`/mnt/pfs/tmp/rlinf`	多机 rendezvous 共享目录,需要用共享存储

脚本内部会自动把通用变量映射回下游 config 硬依赖的旧变量名（`LIBERO_LEROBOT_ROOT` / `PI05_BASE_PYTORCH_CKPT` / `OPENPI_ASSETS_DIR`），训练逻辑零改动，无需手工 export。

步骤五：启动训练

单机 8 卡

```bash
bash /root/workspace/run_libero_openpi_rlinf.sh
```

多机（PyTorchJob，所有 pod 同一行，靠 `RANK`/`WORLD_SIZE` 自动分工）

```bash
bash /root/workspace/run_libero_openpi_rlinf.sh actor.micro_batch_size=32
```

通用变量覆盖示例

```bash
WORKSPACE_DIR=/root/workspace/RLinf \
DATA_PATH=/root/workspace/asset/datasets/physical-intelligence/libero \
BASE_MODEL_PATH=/root/workspace/asset/models/pi05_base_new \
ASSETS_PATH=/root/workspace/asset/models/pi05_base_new \
PROFILE=opt_all \
bash /root/workspace/run_libero_openpi_rlinf.sh
```

`PROFILE=opt_all` 会启用 FSDP 分片、`torch.compile`、fused AdamW、数据预取、图像管线等 12 项优化（`+opt/...` 组）。训练日志默认输出到 `${WORKSPACE_DIR}/logs/<timestamp>-libero_sft_openpi_rlinf/`。

步骤六：监控训练

查看日志：

```bash
tail -f /root/workspace/RLinf/logs/<timestamp>-libero_sft_openpi_rlinf/run_embodiment.log
```

查看 GPU：

```bash
watch -n 2 nvidia-smi
```

查看 Ray 集群：

```bash
ray status
```

性能参考

在 aihc.lgn7ib 实例单机 8 卡、`PROFILE=opt_all`、`actor.global_batch_size=32` 测试环境下，参考性能如下：

指标	参考值
Step Time	1399.2 ms
Samples/s	182.99

以上数据为特定测试环境下的参考值，实际性能会受到资源规格、数据读取方式、网络环境、任务参数等因素影响。

常见问题

1. 数据集路径不存在

默认数据集为 `/mnt/pfs/datasets/physical-intelligence/libero`。若挂载位置不同，通过 `DATA_PATH` 覆盖即可：

```bash
DATA_PATH=/your/libero/dataset bash /root/workspace/run_libero_openpi_rlinf.sh
```

2. 模型权重缺少 `model.safetensors`

`BASE_MODEL_PATH` 必须指向包含 `model.safetensors` 的 PyTorch 原生 checkpoint 目录。确认：

```bash
ls -lh /root/workspace/asset/models/pi05_base_new/model.safetensors
```

3. torch 2.13 首次训练崩溃（`Cannot access data pointer of Tensor`）

torch 2.13 的 inductor 缓存不可删除，否则首次训练在 Triton GEMM autotune 冷缓存下可能崩溃。制作镜像时已把预热缓存一并打入，请勿清空 `torchinductor_cache_213`。

4. 多机 worker 未接入 Ray

确认 `RLINF_RENDEZVOUS_DIR` 共享盘可读写，master 已写入 head IP；worker 会轮询等待 head 地址（默认超时 1800s）。可临时调大 `RAY_WAIT_TIMEOUT`：

```bash
RAY_WAIT_TIMEOUT=3600 NUM_NODES=2 bash /root/workspace/run_libero_openpi_rlinf.sh
```

RoboDojo 仿真评测快速开始

产品简介
RoboDojo 是面向机器人操作策略（Manipulation Policy）的仿真评测框架，基于 Isaac Sim 物理仿真环境驱动机器人执行真实操作任务，自动输出成功率、得分与多路相机视频，无需实体机器人即可完成策略评估。

本文档以 **FastWAM** 作为示例，**FastWAM** 是 RoboDojo 已集成的策略之一，基于 Wan 系列视频生成基座构建，官方公布 54 配置均分为 3.14。

核心能力
能力	**说明**
54 个标准任务	覆盖泛化、记忆、精度、长程、开放五个维度
42 个策略适配器	托管 30 个已训练完成的 checkpoint
自动输出结果	每个 episode 输出成绩单 `_result.json` 与三路相机视频（头部、左/右手腕，640×480@25FPS）
C/S 架构	推理进程与仿真进程可同机部署，也可分置两台机器

典型使用场景
* 评估自研 VLA（视觉-语言-动作）模型在标准任务集上的表现
* 与榜单已收录策略横向对比
* 在无实体机器人条件下验证策略的泛化与长程能力

部署架构
单机部署下，一台开发机同时运行两个进程：

	推理进程	仿真进程
职责	加载 FastWAM 模型，接收观测并返回动作	构建仿真场景、驱动机器人、录制视频、计算得分
启动命令	`rd_server.sh`	`rd_client.sh`
Python 环境	conda `fastwam`	conda `RoboDojo`
显存占用	约 24.3 GB	约 15 GB（单 episode 峰值）

两进程通过 `127.0.0.1:6000` 本机回环通信，无需对外开放端口。同卡运行时**单卡显存建议不低于 48 GB**。

快速开始
步骤 1：创建开发机
在百舸控制台 → 开发机页面，单击**创建实例**，推荐配置如下：

配置项	**最低要求**	**推荐配置**
单卡显存	48 GB（推理与仿真同卡）	80 GB 及以上
内存	64 GB	128 GB
云盘	150 GiB	200 GiB
操作系统	Ubuntu 24.04 x64	Ubuntu 24.04 x64

实例运行后，登录开发机并确认 GPU 可见：

```bash
nvidia-smi --query-gpu=name,memory.total,driver_version --format=csv,noheader
```
步骤 2：下载权重与资产

评测需要三份内容（互不依赖，可并行下载）：

内容	**体积**
仿真资产	约 38.4 GiB
FastWAM checkpoint	约 11.2 GiB
Wan 基座（VAE + T5 + tokenizer）	约 11.9 GiB

一条命令备齐全部内容：

```bash
nohup bash /root/workspace/dl_base_bos.sh > /root/rd_dl.log 2>&1 &
tail -f /root/rd_dl.log
```
下载完成后日志末尾输出 `ALL_READY stages=assets,ckpt,base policy=FastWAM`。

步骤 3：启动推理服务
```bash
bash /root/workspace/rd_server.sh
```
服务就绪后输出：

```
CKPT /root/fastwam-ckpt/RoboDojo-sim-arx_x5-joint-0
STARTED pid=12345 port=6000 log=/root/logs/rd_server.log
SERVER_READY 0100007F:1770 after 96s
```
权重加载约需 1～2 分钟。服务启动后保持运行，可服务多次评测。

步骤 4：运行评测
另开终端，执行单任务评测（以叠碗任务为例）：

```bash
bash /root/workspace/rd_client.sh stack_bowls 1
```
评测结束时输出视频路径、成功/失败计数与退出码。

步骤 5：查看结果
结果文件落盘位置：

```
eval_result/RoboDojo/<任务名>/FastWAM/arx_x5/<种子及参数>/<时间戳>/
├──── _result.json              # 成功率、得分、每个 layout 明细
├──── episode_0000000_cam_head_*.mp4
├──── episode_0000000_cam_left_wrist_*.mp4
└──── episode_0000000_cam_right_wrist_*.mp4
```
结果验收四项检查（全部通过才算部署成功）：

```bash
cd /root/workspace/RoboDojo
grep -c "CLIENT EXIT=0" /root/logs/rd_client_stack_bowls.log   # 预期 1
grep -ci "[Error]" /root/logs/rd_client_stack_bowls.log        # 预期 0
find eval_result/ -name "*.mp4" | wc -l                        # 预期 3
find eval_result/ -name "_result.json" -exec cat {} \;
```
> **注意**：单个 episode 的 `success_rate` 不代表部署成功与否，验收请依据上述四项产物检查。

批量评测
串行遍历多个任务
```bash
for T in stack_bowls build_tower insert_key; do
  EVAL_NUM=3 bash /root/workspace/rd_client.sh "$T" || echo "FAILED $T"
done

#### 生成汇总表
source /root/miniconda3/bin/activate RoboDojo
cd /root/workspace/RoboDojo && bash scripts/robodojo.sh summarize
```
多 worker 并行（单机）
使用 `smoke_all_tasks.sh` 一键铺满多卡，无需手动启动推理服务：

```bash
source /root/miniconda3/bin/activate RoboDojo
export OMNI_KIT_ACCEPT_EULA=YES
unset http_proxy https_proxy HTTP_PROXY HTTPS_PROXY
cd /root/workspace/RoboDojo

bash scripts/internal/smoke_all_tasks.sh \
  --policy-dir XPolicyLab/policy/FastWAM --ckpt sim --policy-env fastwam \
  --action-type joint --eval-num native \
  --gpu-ids 0,1,2,3 --seed 0 --run-id fastwam_s0 --resume
```
`--gpu-ids` 元素个数即 worker 数，同一张卡可重复出现（如 `0,0,1,1` 表示每卡 2 个 worker）。

全量评测参考指标（8 卡 × 2 worker，54 任务，seed 0）：

指标	实测值
任务数	54/54 全部 PASS
总 episode 数	2100
视频产物	6300 个 mp4
墙钟耗时	约 6.87 小时
等效加速比	14.1×

任务列表
框架提供 54 个可运行任务配置，分属五个维度：

框架提供 34 个可运行任务配置，分属五个维度：

维度	任务数	代表任务
泛化（generalization）	24（含 random 变体）	`stack_bowls`（推荐入门）、`fold_clothes`、`hang_mugs`
记忆（memory）	6	`match_and_pick_from_conveyor`、`imitate_sorting_sequence`
精度（precision）	8	`insert_key`、`build_tower`、`fasten_screws`
长程（long-horizon）	8	`fill_egg_holder`、`organize_table`、`put_bottles_into_dustbin`
开放（open）	8	`general_pickup`、`classify_objects_by_language`

查看完整任务列表：

```bash
cd /root/workspace/RoboDojo
bash scripts/robodojo.sh tasks          # 列出全部任务名
bash scripts/robodojo.sh dimensions     # 按维度列出
```

常见问题
Q：仿真进程长时间无进展、GPU 占用 0%？

检查 `/usr/lib/x86_64-linux-gnu/libGLU.so.1` 是否存在，缺失时执行：

```bash
unset http_proxy https_proxy HTTP_PROXY HTTPS_PROXY
apt-get install -y libglu1-mesa
```
Q：仿真进程连不上推理服务？

最常见原因是代理变量拦截了本机回环地址，执行以下命令后重试：

```bash
unset http_proxy https_proxy HTTP_PROXY HTTPS_PROXY
```
同时确认推理服务仍在运行：`bash /root/workspace/rd_server.sh status`

Q：`--eval-num`** 未传入，退出码为 2？**

该参数为必选项，请传入具体数字，例如 `--eval-num 1`；全量对榜时使用 `native`（仅 `smoke_all_tasks.sh` 支持）。

Q：本地分数比官方低，是部署有问题吗？

先确认对比口径相同。实测 seed 0 全量 sweep（2100 集）54 配置均值为 3.09，官方同口径为 3.14，相差 0.05 分，在统计误差内属于正常。验收标准应依据链路健康度（`[Error]` 计数为 0、mp4 数量正确、`_result.json` 完整），而非 score 数值。

Q：日志中出现 **`Invalid PhysX transform` 是故障吗？**

多数情况不是。框架会自动丢弃不稳定的 layout 并继续运行。只要日志中 `Error code 700` 计数为 0、产物完整，即属正常。

计费说明
RoboDojo 评测框架本身免费使用，计费项为运行评测所使用的**百舸开发机实例**，按实例规格与使用时长计费。请在评测结束后及时释放实例以控制成本。

权限说明
使用 RoboDojo 评测框架需具备以下权限：

* **百舸开发机创建权限**：在百舸控制台创建并管理开发机实例
* **BOS 访问权限**：下载评测所需权重与资产（镜像内已预置 `bcecmd`，无需额外配置）

如遇权限不足，请联系所在团队的百舸管理员申请。

在百舸上使用加速镜像训练 DM05 模型

在百舸上使用加速镜像训练 DM05 模型

一、概述
本文档介绍如何在百舸平台上，使用百舸提供的 DM0.5 训练加速镜像，对 Dexmal 原力灵机 DM05 具身基础模型的预训练权重执行监督微调（SFT）。

该镜像已预装训练所需 Python 环境与依赖（开箱即用，无需额外挂载虚拟环境），并内置一组训练加速优化项（FusedAdamW、DDP Bucket 优化、异步 Loss AllReduce、torch.compile、CUDA Graph 等）。在保证训练效果与 loss 收敛趋势一致的前提下，可显著缩短单轮训练迭代耗时、提升训练吞吐，支持单机多卡与多机多卡的分布式扩展。

二、模型简介
DM05 是 Dexmal（原力灵机）推出的面向开放世界具身智能的通用基础模型，基于 15 万小时数据训练，参数量约 4B，支持导航、抓取、全身控制、长时记忆及跨形态部署。其架构为 **Gemma3 VLM + Action Expert** 双分支 VLA（Vision-Language-Action），共享 attention/FFN 结构，通过时间条件 modulator 注入 action chunk 时序信号。

参考资料：

* [dm05_libero 训练说明（opendm 仓库）](https://github.com/dexmal/opendm/blob/main/docs/zh/dm05_libero.md)
* [DM05 模型页（HuggingFace）](https://huggingface.co/Dexmal/DM05)

三、环境与资源要求
项目	**说明**
推荐 GPU	lgn7ib，96GB 显存，单机 8 卡；支持多机扩展
镜像内工作目录	`/root/workspace/dm05`
训练框架	torchrun
建议镜像数据盘大小	300GB，或者使用PFS

四、准备数据集
训练使用 `libero_pi0_all` 数据集（Libero 数据集的子集，用于 DM05 微调）。

4.1 数据集目录组织
数据集在容器内需按以下结构组织，如数据目录 `/root/workspace/dm05/data/libero`：

```
dm05
```

```
├── data
│   └── libero
│       └── libero_pi0_all
│           ├── image
│           └── jsonl
└── index_cache.json
```
4.2 从 BOS 下载（推荐）
```bash
**下载到本地临时目录 ####################**
export DATA_ROOT=/root/workspace/dm05/data/libero
export DATA_DOWNLOAD_ROOT=/root/workspace/dm05/data_download/libero

#### 创建数据集目录
mkdir -p $DATA_ROOT $DATA_DOWNLOAD_ROOT

#### 从 BOS 下载 libero 数据集
bcecmd bos cp -r bos://aihc-rdw-bj/huggingface.co/datasets/Dexmal/libero/ $DATA_DOWNLOAD_ROOT

**下载到PFS目录 ####################**
export DATA_ROOT=/mnt/pfs/workspace/dm05/data/libero
export DATA_DOWNLOAD_ROOT=/mnt/pfs/datasets/Dexmal/libero

#### 创建数据集目录
mkdir -p $DATA_ROOT $DATA_DOWNLOAD_ROOT

#### 从 BOS 下载 libero 数据集
bcecmd bos cp -r bos://aihc-rdw-bj/huggingface.co/datasets/Dexmal/libero/ $DATA_DOWNLOAD_ROOT
```
4.3 从 HuggingFace 下载
```bash
#### 设置国内镜像加速下载
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
export HF_HOME=/workspace/models/huggingface
export HF_HUB_CACHE=$HF_HOME/hub
export HF_DATASETS_CACHE=$HF_HOME/datasets
export DATA_DOWNLOAD_ROOT=/root/workspace/dm05/data_download/libero
#### 或者PFS路径
#### export DATA_DOWNLOAD_ROOT=/mnt/pfs/datasets/Dexmal/libero

#### 下载完整 libero 数据集
huggingface-cli download --repo-type dataset Dexmal/libero --local-dir $DATA_DOWNLOAD_ROOT
```
> 说明：从 HuggingFace 下载的原始数据集需要按上述目录结构进行预处理（生成 `index_cache.json` 等训练索引文件）后使用，具体处理方式可参考 [opendm 的 dm05_libero.md](https://github.com/dexmal/opendm/blob/main/docs/zh/dm05_libero.md)。
4.4 数据集预处理（HF格式转opendm解析）
> 参考官方文档：https://github.com/dexmal/opendm/blob/main/docs/zh/dm05_libero.md（数据准备章节）
```bash
**本地临时目录 ####################**
export WORKSPACE_DIR=/root/workspace/dm05
export DATA_ROOT=/root/workspace/dm05/data/libero
export DATA_DOWNLOAD_ROOT=/root/workspace/dm05/data_download/libero

#### 下载并整理 LIBERO 数据
$WORKSPACE_DIR/src/opendm/script/libero_runner.sh dataset --data-root $DATA_ROOT --dataset-download-dir $DATA_DOWNLOAD_ROOT


**PFS目录 ####################**
#### 或者pfs路径
export WORKSPACE_DIR=/root/workspace/dm05
export DATA_ROOT=/mnt/pfs/workspace/dm05/data/libero
export DATA_DOWNLOAD_ROOT=/mnt/pfs/datasets/Dexmal/libero
#### 下载并整理 LIBERO 数据
$WORKSPACE_DIR/src/opendm/script/libero_runner.sh dataset --data-root $DATA_ROOT --dataset-download-dir $DATA_DOWNLOAD_ROOT
#### PFS场景下，设置软链
mkdir -p /root/workspace/dm05/data/libero && rm -rf /root/workspace/dm05/data/libero
ln -sf /mnt/pfs/workspace/dm05/data/libero /root/workspace/dm05/data/libero
```
* `script/libero_runner.sh dataset` 会从 `Dexmal/libero` 下载数据到 `$WORKSPACE/data/.hf_downloads/libero`，自动处理 `libero.tar.part-*` 分片、解压并整理到 `$WORKSPACE/data/libero`。
* `script/libero_runner.sh model` 默认会从 `Dexmal/DM05` 下载基础模型，并保存到 `$WORKSPACE/checkpoints/DM05`。如果需要使用其他模型仓库或保存目录，可通过 `--model-repo` 和 `--model-dir` 覆盖。

五、准备模型权重
5.1 权重目录组织
模型权重在容器内需按以下结构组织：

```
dm05
├── checkpoints
│   └── DM05
│       ├── model.safetensors
│       └── tokenizer.json
```
5.2 从 BOS 下载（推荐）
```bash
**下载到本地临时目录 ####################**
export CHECKPOINT_PATH=/root/workspace/dm05/checkpoints/DM05/

#### 创建权重目录
mkdir -p $CHECKPOINT_PATH

#### 从 BOS 下载 DM05 完整模型权重
bcecmd bos cp -r bos://aihc-models-bj/Dexmal/DM05/ $CHECKPOINT_PATH

**下载到PFS目录 ####################**
export CHECKPOINT_PATH=/mnt/pfs/models/Dexmal/DM05
#### 创建权重目录
mkdir -p $CHECKPOINT_PATH
```

从 BOS 下载 DM05 完整模型权重
bcecmd bos cp -r bos://aihc-models-bj/Dexmal/DM05/ $CHECKPOINT_PATH
```
###### 5.3 从 HuggingFace 下载
```bash
#### 设置国内镜像加速下载
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
export HF_HOME=/root/workspace/models/huggingface
export HF_HUB_CACHE=$HF_HOME/hub
export CHECKPOINT_PATH=/root/workspace/dm05/checkpoints/DM05/
#### 或者PFS路径
#### export CHECKPOINT_PATH=/mnt/pfs/models/Dexmal/DM05

#### 下载 DM05 开源权重
huggingface-cli download Dexmal/DM05 --local-dir $CHECKPOINT_PATH
```


六、在百舸平台创建训练任务
1. 登录百舸平台，进入「训练任务」创建页。
2. 选择推荐 GPU 资源规格：**lgn7ib 96GB × 8**（单机或按需多机）。
3. 挂载数据集目录至 `/root/workspace/dm05/data/libero/libero_pi0_all`。
4. 挂载模型权重目录至 `/root/workspace/dm05/checkpoints/DM05`。
5. 设置工作目录为 `/root/workspace/dm05`。
6. 通过启动脚本配置启动命令（见下文）并提交任务。

> 镜像内已包含训练代码与全部 Python 依赖，无需额外安装或挂载 .venv / conda 环境。
七、关键训练参数说明
参数	**默认值**	**说明**
`--data-config.dataset-name`	`libero_pi0_all`	数据集名称
`--data-config.norm-stats-root`	`/root/workspace/dm05/patches/norm_stats/`	action normalization 统计文件目录，需与数据集和 action chunk 配置匹配
`--model-config.model-name-or-path`	`/root/workspace/dm05/checkpoints/DM05`	模型权重（checkpoint）路径
`--model-config.chunk-size`	`10`	每个样本的 action chunk 长度
`--optimizer-config.optim`	`adamw`	优化器类型
`--optimizer-config.base-lr`	`2.5e-5`	基础学习率
`--optimizer-config.warmup-steps`	`1000`	Warmup 步数
`--trainer-config.per-device-train-batch-size`	`10`	每张 GPU 的 micro-batch 大小（可通过 `PER_DEVICE_BATCH` 环境变量覆盖）
`--trainer-config.gradient-accumulation-steps`	`1`	梯度累积步数
`--trainer-config.num-train-steps`	`300`	训练总步数（可通过 `TRAIN_STEPS` 环境变量覆盖）
`--trainer-config.save-strategy`	`no`	训练过程中不保存 checkpoint；正式训练可改为 `steps`
`--trainer-config.dataloader-num-workers`	`4`	每个训练进程的 DataLoader worker 数

全局 Batch Size（GBS）计算方式：

```
GBS = NNODES × GPUS_PER_NODE × per_device_train_batch_size × gradient_accumulation_steps
```
八、启动训练
镜像内已提供启动脚本，单机与多机场景使用同一脚本，通过环境变量区分拓扑。脚本默认使用 `/root/workspace/dm05` 作为工作目录，并兼容平台注入的 `RANK` / `WORLD_SIZE` 等变量。

8.1 启动脚本
优化脚本开启内置加速优化项，推荐**同时开启 R1+R5+R7+R8+R10+R13+R14+R16+R18** 组合（脚本已默认配置）。

```bash
export WORKSPACE_DIR="/root/workspace/dm05"
export DATA_PATH="$WORKSPACE_DIR/data/libero/libero_pi0_all"

**本地临时目录可设置 ####################**
export CHECKPOINT_PATH="$WORKSPACE_DIR/checkpoints/DM05"

**PFS目录可设置 ####################**
#### export CHECKPOINT_PATH="/mnt/pfs/models/Dexmal/DM05"

#### export TRITON_PTXAS_PATH="/usr/local/cuda/bin/ptxas"
#### export TRITON_NVDISASM_PATH="/usr/local/cuda/bin/nvdisasm"
#### export TRITON_CUOBJDUMP_PATH="/usr/local/cuda/bin/cuobjdump"

#### 默认配置（单卡 bsz 为 4）
bash /root/workspace/dm05/scripts/run_full_sft_distributed.sh

#### 修改配置：设置单卡 bsz 为 8
PER_DEVICE_BATCH=8 bash /root/workspace/dm05/scripts/run_full_sft_distributed.sh

#### 修改配置：设置训练步数为 2000
TRAIN_STEPS=2000 bash /root/workspace/dm05/scripts/run_full_sft_distributed.sh
```
核心优化开关（可通过环境变量开启/关闭）：

```bash
export DM05_PERF_ENABLE=1                    # 启用加速优化（总开关）
export DM05_PERF_LOG=1

export DM05_PERF_R1_FUSED_ADAMW=1            # R1: FusedAdamW 优化
export DM05_PERF_R5_DDP_BUCKET=1             # R5: DDP bucket 优化
export DM05_DDP_BUCKET_CAP_MB="$BUCKET_MB"
export DM05_PERF_R7_ASYNC_LOSS_AR=1          # R7: 异步 loss allreduce
export DM05_PERF_R8_COMPILE_SUFFIX_LOOP=1    # R8: torch.compile 后缀循环
export DM05_PERF_R10_TF32_MATMUL=1           # R10: TF32 矩阵运算
export DM05_PERF_R10_MATMUL_PRECISION=high
export DM05_PERF_R10_CUDNN_BENCHMARK=1       # R10: cuDNN benchmark
export DM05_PERF_R13_GEMMA3_FORWARD_SYNC=1   # R13: Gemma3 forward sync
export DM05_PERF_R14_COMPILE_SIGLIP=1        # R14: SIGLIP torch.compile
export DM05_PERF_R16_MODULE_REORDER=1        # R16: 模型结构重排
export DM05_PERF_R18_CUDA_GRAPH=1            # R18: CUDA Graph

#### 其他配套优化
export DM05_MUON_BATCHED_GATHER=1            # muon batched gather
export DM05_DDP_STATIC_GRAPH=1               # DDP 静态图
```
```

```
export DM05_DDP_STATIC_GRAPH=1                # DDP 静态图
```
> 优化项均通过 `DM05_PERF_ENABLE=1` 总开关控制，单项优化可独立开关。若训练出现异常，可先关闭部分优化项（如 R18 CUDA Graph）进行排查。
8.3 多机训练说明
* 多机时需通过 `MASTER_ADDR` 设置 rank 0 节点地址，并确保所有节点 `MASTER_PORT`、`NNODES`、`NPROC_PER_NODE` 配置一致。
* 单机场景 `DDP_BUCKET_MB` 默认为 1000，多机场景默认为 512。

九、性能参考
以下数据为特定测试环境（lgn7ib 96GB × 8）下的参考值，step time 采集自训练稳态（第 100 步），实际性能受资源规格、数据读取方式、网络环境、任务参数等因素影响。

9.1 单机（1 节点 × 8 卡）
配置	**Step time (s)**	**Samples/s**	**加速比**
基线（BS=8）	1.06	60.38	—
优化后（BS=8）	0.93	69.12	1.140x
基线（BS=4）	0.7813	40.96	—
优化后（BS=4）	0.6667	48.00	1.172x

9.2 双机（2 节点 × 8 卡）
配置	**Step time (s)**	**Samples/s**	**加速比**
基线（BS=8）	1.11	115.32	—
优化后（BS=8）	0.96	133.12	1.156x
基线（BS=4）	0.85	75.72	—
优化后（BS=4）	0.77	83.20	1.104x

9.3 Loss 对齐验证
优化前后的 loss 收敛曲线一致，加速优化不影响模型训练效果与收敛趋势。

十、常见问题
10.1 显存不足（OOM）
双机场景下单卡 batch size 较大时（如 BS=10）可能出现显存不足。可尝试：

* 降低 `PER_DEVICE_BATCH`（如从 10 降至 8 或 4）；
* 将 `--trainer-config.gradient-accumulation-steps` 设为 `2`，将全局 batch 拆分为多个 micro-batch，降低单次显存压力；
* 如仍无法解决，可关闭部分显存敏感优化项（如 `DM05_PERF_R18_CUDA_GRAPH=0`）。

10.2 需要保存训练权重（Checkpoint）
默认启动脚本不保存 checkpoint。如需按步保存，将 `--trainer-config.save-strategy` 修改为 `steps` 并配置保存间隔：

```bash
--trainer-config.save-strategy steps \
--trainer-config.save_steps 100 \
--trainer-config.output-dir "/root/workspace/outputs"  # ckpt 默认保存路径
```
10.3 断点续训
通过修改 `--model-config.model-name-or-path` 指向待加载的权重目录即可，与加载开源权重使用同一配置接口：

```bash
--model-config.model-name-or-path "/root/workspace/dm05/checkpoints/DM05"
```

Wan2.2-T2V-A14B 文生视频模型加速训练

一、概述
本文档介绍如何在百舸（AIHC）开发机中，使用百舸提供的 `diffsynth` 训练加速镜像，对 Wan2.2-T2V-A14B 文生视频 MoE 扩散模型执行全参微调。

镜像已预装训练代码与全部 Python 依赖，开箱即用，无需额外安装虚拟环境；内置的启动脚本自适应单机与多机拓扑，进程数与通信参数全部由平台注入的环境变量推导，使用者无需修改。

二、适用场景
* 需要在自有视频数据上微调 Wan2.2-T2V-A14B（high_noise 专家）的文生视频任务。
* 需要单机多卡或多机多卡分布式训练。
* 希望直接复用已调优的启动脚本与并行配置，减少环境搭建与参数调试成本。

三、模型简介
Wan2.2-T2V-A14B 是 Wan2.2 系列的文生视频 MoE 扩散模型，按 timestep 分成 `high_noise` 与 `low_noise` 两个专家。本文的训练流程只微调 `high_noise` 专家的 DiT（`--max_timestep_boundary 0.417`、`--trainable_models "dit"`），`low_noise` 不加载，因此权重只需取 `high_noise_model/*`。

训练框架为 DiffSynth-Studio，使用 `accelerate` + DeepSpeed ZeRO-2，配合 USP（unified sequence parallel，本文配置 `--sp_size 1`，即不切序列、纯数据并行）。文本编码器使用 UMT5-XXL，VAE 使用 Wan2.1_VAE（TI2V-5B 走 Wan2.2_VAE，同仓库但不可混用）。

四、环境与资源要求
项目	**说明**
推荐加速卡	lgn7ib，单机 8 卡；支持多机扩展
镜像内代码目录	`/root/workspace/DiffSynth-Studio`
运行时环境	无 venv / conda / uv，依赖直接安装在系统 Python（`/usr/bin/python3`，3.12.3）
软件栈	torch 2.8.0（cu12.9）、NCCL 2.27.3、DeepSpeed 0.18.6、accelerate 1.10.1、flash-attn 2.7.4.post1、modelscope 1.33.0
内置资产	`TestVidData/video_1sampe`（592 KB 单样本冒烟数据）；`models/` 为空目录，作为权重挂载点

镜像内目录结构：

```
/root/workspace/DiffSynth-Studio/
├── diffsynth/                    # 源码（未安装进 site-packages）
├── examples/
│   └── wanvideo/
│       └── model_training/
│           ├── train.py
│           └── full/             # 启动脚本 + accelerate/deepspeed 配置
├── models/                       # 权重挂载点（镜像内为空）
│   └── train/                    # 训练输出
└── TestVidData/
    └── video_1sampe/             # 592 KB 单样本冒烟数据
```
> `diffsynth` 未安装到 site-packages，而是从源码目录导入，因此启动脚本必须设置 `PYTHONPATH`（内置脚本已处理，无需手动设置）。
五、准备工作：创建百舸开发机
本文的训练流程包含准备权重、校验、按需调参重跑等交互式操作，建议在百舸开发机中执行：

1. 登录百舸平台，进入「开发机」创建页。
2. 选择带有加速卡资源规格（单机 8 卡即可满足本文档；多机训练时按需申请多节点）

2. 选择推荐加速卡资源规格：**1gn7i-8x**（单机；多机训练时按需申请多节点）。
3. 挂载权重存储目录，挂载点与后文 `MODEL_BASE_PATH` 保持一致（默认 `/root/workspace/DiffSynth-Studio/models`）。
4. 如需使用自有数据集，挂载数据集存储目录。
5. 开发机启动后，打开终端（或 WebIDE 终端），进入代码目录：

```bash
cd /root/workspace/DiffSynth-Studio
```
后续「准备模型权重」「准备数据集」「启动训练」等命令均在该终端中按顺序执行。

六、准备模型权重
镜像不含权重。权重目录约定为 `${MODEL_BASE_PATH}/<org>/<repo>/...`，涉及两个 org：`Wan-AI` 与 `DiffSynth-Studio`。

`MODEL_BASE_PATH` 默认为 `${WORKSPACE_DIR}/models`，须与开发机的权重挂载点一致。

6.1 权重清单
A14B 训练需要下面 4 个仓库，共约 69 GB：

仓库	**取用文件**	**体积**	**落地路径**
Wan-AI/Wan2.2-T2V-A14B	`high_noise_model/*`	57.2 GB	`${MODEL_BASE_PATH}/Wan-AI/Wan2.2-T2V-A14B/`
DiffSynth-Studio/Wan-Series-Converted-Safetensors	`models_t5_umt5-xxl-enc-bf16.safetensors`、`Wan2.1_VAE.safetensors`	11.6 GB	`${MODEL_BASE_PATH}/DiffSynth-Studio/Wan-Series-Converted-Safetensors/`
Wan-AI/Wan2.1-T2V-1.3B	`google/umt5-xxl/`（tokenizer）	21 MB	`${MODEL_BASE_PATH}/Wan-AI/Wan2.1-T2V-1.3B/`
Wan-AI/Wan2.2-S2V-14B	`wav2vec2-large-xlsr-53-english/` 下 4 个 json	2.2 KB	`${MODEL_BASE_PATH}/Wan-AI/Wan2.2-S2V-14B/`

> 仓库 2、3、4 虽未出现在启动命令的 `--model_id_with_origin_paths` 中，但训练过程会强制加载，缺失时报错信息不会直接指出缺哪个文件，请务必备齐。详见「常见问题」。
6.2 从 BOS 下载
```bash
#### 仓库 1：Wan2.2-T2V-A14B 主权重
bosutil cp -r bos:/aihc-models-bj/Wan-AI/Wan2.2-T2V-A14B/ ${MODEL_BASE_PATH}/Wan-AI/Wan2.2-T2V-A14B/

#### 仓库 2：Wan-Series-Converted-Safetensors（T5 / VAE）
bosutil cp -r bos:/aihc-models-bj/noodlepop/Wan-Series-Converted-Safetensors/ ${MODEL_BASE_PATH}/DiffSynth-Studio/Wan-Series-Converted-Safetensors/

#### 仓库 3：Wan2.1-T2V-1.3B（tokenizer）
bosutil cp -r bos:/aihc-models-bj/Wan-AI/Wan2.1-T2V-1.3B/ ${MODEL_BASE_PATH}/Wan-AI/Wan2.1-T2V-1.3B/

#### 仓库 4：Wan2.2-S2V-14B（audio processor 配置）
bosutil cp -r bos:/aihc-models-bj/Wan-AI/Wan2.2-S2V-14B/ ${MODEL_BASE_PATH}/Wan-AI/Wan2.2-S2V-14B/
```
6.3 校验
权重落地后先确认 glob 能匹配到文件，否则会在加载阶段报难以定位的错误：

```bash
M=${MODEL_BASE_PATH:-/workspace/DiffSynth-Studio/models}
ls $M/Wan-AI/Wan2.2-T2V-A14B/high_noise_model/diffusion_pytorch_model*.safetensors | wc -l   # 应 > 0
ls $M/DiffSynth-Studio/Wan-Series-Converted-Safetensors/models_t5_umt5-xxl-enc-bf16.safetensors
ls $M/DiffSynth-Studio/Wan-Series-Converted-Safetensors/Wan2.1_VAE.safetensors
ls $M/Wan-AI/Wan2.1-T2V-1.3B/google/umt5-xxl/
ls $M/Wan-AI/Wan2.2-S2V-14B/wav2vec2-large-xlsr-53-english/
```
七、准备数据集
镜像内置单样本冒烟数据集 `TestVidData/video_1sampe`（592 KB），可直接用于跑通验证，无需额外下载。

如需使用完整数据集，从 BOS 下载：

```bash
mkdir -p /root/workspace/datasets/VidData
bosutil cp -r bos:/<your-bucket>/<path-to-dataset>/ /workspace/datasets/VidData/
```
> 完整数据集尚未上传至百舸 BOS，请联系百舸相关人员协助上传后获取实际路径。
自定义数据集需包含视频文件与 `metadata.csv`，目录结构参考镜像内置的 `TestVidData/video_1sampe`。下载完成后通过 `DATA_PATH` 指向数据目录即可。

八、配置训练环境变量
启动脚本已为下列变量设置默认值，仅在需要覆盖时显式传入。

变量	默认值	说明
`WORKSPACE_DIR`	`/root/workspace/DiffSynth-Studio`	代码根目录，覆盖时必须指向 DiffSynth-Studio 仓库根
`MODEL_BASE_PATH`	`${WORKSPACE_DIR}/models`	权重根目录，须与挂载点一致
`DATA_PATH`	`${WORKSPACE_DIR}/TestVidData/video_1sampe`	数据目录
`OUTPUT_DIR`	`${WORKSPACE_DIR}/models/train/Wan2.2-T2V-A14B_high_noise_full`	checkpoint 输出目录
`DIFFSYNTH_MODEL_BASE_PATH`	同 `MODEL_BASE_PATH`	传给框架用于定位权重
`DIFFSYNTH_SKIP_DOWNLOAD`	`true`	关闭联网下载
`TARGET_STEPS`	`0`（不启用）	目标步数。大于 0 时按当前总卡数反推 `DATASET_REPEAT`，单机与多机步数一致；优先级高于 `DATASET_REPEAT`
`NUM_EPOCHS`	`2`	epoch 数，同时作为 `TARGET_STEPS` 换算的分母，传 `1` 可消除取整误差
`DATASET_REPEAT`	`1000`	单样本重复次数，`TARGET_STEPS > 0` 时被覆盖

九、关键训练参数说明
启动脚本自适应单机 / 多机，进程数与拓扑全部由平台注入的环境变量推导，**使用者无需修改拓扑相关参数**。

参数	值	说明
`--num_processes`	由脚本自动计算	`$(( NNODES * GPUS_PER_NODE ))`，取自注入的 `WORLD_SIZE`（节点数）与 `NPROC_PER_NODE`（每节点卡数）
`--num_machines` / `--machine_rank`	由脚本自动计算	取自注入的 `WORLD_SIZE` / `RANK`（`RANK` 是节点序号，非进程序号）
`--main_process_ip` / `--main_process_port`	由脚本自动计算	取自注入的 `MASTER_ADDR` / `MASTER_PORT`
`--deepspeed_multinode_launcher`	`standard`	多机必需，详见「多机训练说明」
`--config_file`	`.../full/accelerate_config_zero2_usp.yaml`	DeepSpeed ZeRO-2 + USP，路径为仓库根的相对路径
`--height` / `--width` / `--num_frames`	`480` / `832` / `49`	分辨率与帧数，影响显存与算力
`--dataset_num_workers`	`16`	视频解码开销大，建议保持
`--dataset_repeat`	`${DATASET_REPEAT}`	步数 = `ceil(DATASET_REPEAT / 总卡数) × NUM_EPOCHS`。默认 1000：单机 8 卡 → 250 step，双机 16 卡 → 126 step
`--trainable_models`	`dit`	只训练 DiT
`--max_timestep_boundary`	`0.417`	只训练 high_noise 专家，low_noise 不加载
`--min_timestep_boundary`	`0`	同上
`--sp_size`	`1`	sequence parallel 关闭，纯数据并行
`--gradient_checkpointing_interval`	`1`	每层都重计算，节省显存
`--remove_prefix_in_ckpt`	`pipe.dit.`	存盘时去掉前缀
`--total_flops`	`2600.933`	用于 TFLOPS 换算，修改分辨率 / 帧数后需重算
`--learning_rate` / `--num_epochs`	`1e-5` / `${NUM_EPOCHS}`	`NUM_EPOCHS` 默认 2

> `--model_id_with_origin_paths` 中写的是 `.pth` 原始文件名，框架会自动改写成 Converted-Safetensors 仓库中的 `.safetensors`，**请勿手工改成 **`.safetensors`。
十、启动训练
在开发机终端中执行内置启动脚本，单机与多机使用同一脚本：

```bash
S=/root/workspace/DiffSynth-Studio/examples/wanvideo/model_training/full/Wan2.2-T2V-A14B.sh

bash "$S"                    # 默认 DATASET_REPEAT=1000、2 epoch，单机 8 卡约 250 step
TARGET_STEPS=70 bash "$S"    # 只跑 70 step，单机 8 卡与双机 16 卡都是 70 step
DATASET_REPEAT=5000 bash "$S"  # 长稳测试，按样本数控制
```
如需切换数据目录或输出目录：

```bash
DATA_PATH=/root/workspace/datasets/VidData bash "$S"
OUTPUT_DIR=/root/workspace/outputs bash "$S"
```
10.1 启动脚本内容
脚本位于 `/root/workspace/DiffSynth-Studio/examples/wanvideo/model_training/full/Wan2.2-T2V-A14B.sh`，单机 / 多机合并为一份，靠平台注入的变量自动判断（百舸 PyTorch 任务默认注入 `MASTER_ADDR` / `MASTER_PORT` / `RANK` / `WORLD_SIZE` / `NPROC_PER_NODE`，其中 `RANK` 是**节点**序号、`WORLD_SIZE` 是**节点总数**）：

```bash
#### !/bin/bash
set -o pipefail

WORKSPACE_DIR=${WORKSPACE_DIR:-/workspace/DiffSynth-Studio}
MODEL_BASE_PATH=${MODEL_BASE_PATH:-${WORKSPACE_DIR}/models}
DATA_PATH=${DATA_PATH:-${WORKSPACE_DIR}/TestVidData/video_1sampe}
OUTPUT_DIR=${OUTPUT_DIR:-${WORKSPACE_DIR}/models/train/Wan2.2-T2V-A14B_high_noise_full}

export PYTHONPATH="${WORKSPACE_DIR}:/usr/local/lib/python3.12/dist-packages:$PYTHONPATH"
export DIFFSYNTH_MODEL_BASE_PATH=${MODEL_BASE_PATH}
export DIFFSYNTH_SKIP_DOWNLOAD=true

#### 多机参数（单机时 WORLD_SIZE 缺省为 1）
GPUS_PER_NODE=${NPROC_PER_NODE:-8}
NNODES=${WORLD_SIZE:-1}
NODE_RANK=${RANK:-0}

#### 步数控制：TARGET_STEPS 优先（按卡数反推 repeat），未设时用 DATASET_REPEAT
NUM_EPOCHS=${NUM_EPOCHS:-2}
TARGET_STEPS=${TARGET_STEPS:-0}
DATASET_REPEAT=${DATASET_REPEAT:-1000}
if [ "${TARGET_STEPS}" -gt 0 ]; then DATASET_REPEAT=$(( TARGET_STEPS * NNODES * GPUS_PER_NODE / NUM_EPOCHS )); fi

cd ${WORKSPACE_DIR}
accelerate launch
  --deepspeed_multinode_launcher standard
  --num_machines ${NNODES}
  --machine_rank ${NODE_RANK}
  --main_process_ip ${MASTER_ADDR:-127.0.0.1}
  --main_process_port ${MASTER_PORT:-29500}
  --num_processes $(( NNODES * GPUS_PER_NODE ))
  --config_file examples/wanvideo/model_training/full/accelerate_config_zero2_usp.yaml
  examples/wanvideo/model_training/train.py
  --dataset_base_path ${DATA_PATH}
  --dataset_metadata_path ${DATA_PATH}/metadata.csv
  --height 480 --width 832 --num_frames 49
  --dataset_num_workers 16
  --dataset_repeat ${DATASET_REPEAT}
  --model_id_with_origin_paths "Wan-AI/Wan2.2-T2V-A14B:high_noise_model/diffusion_pytorch_model*.safetensors,Wan-AI/Wan2.2-T2V-A14B:models_t5_umt5-xxl-enc-bf16.pth,Wan-AI/Wan2.2-T2V-A14B:Wan2.1_VAE.pth"
  --learning_rate 1e-5
  --num_epochs ${NUM_EPOCHS}
  --remove_prefix_in_ckpt "pipe.dit."
  --output_path ${OUTPUT_DIR}
  --trainable_models "dit"
  --max_timestep_boundary 0.417
  --min_timestep_boundary 0
  --sp_size 1
  --total_flops 2600.933
  --gradient_checkpointing_interval 1
```
10.2 步数控制说明
默认行为为 `DATASET_REPEAT=1000`、2 epoch。冒烟或复测只想跑固定步数时传 `TARGET_STEPS`，脚本按当前卡数反推 `DATASET_REPEAT`，单机与多机得到相同步数——`TARGET_STEPS=70` 在单机 8 卡和双机 16 卡都是 70 步，无需自己按卡数换算。整数除法在 `TARGET_STEPS × 总卡数` 不被 `NUM_EPOCHS` 整除时会有 ±1 步误差，需要精确时同时传 `NUM_EPOCHS=1`。

10.3 多机训练说明
* 脚本自动读取平台注入的 `MASTER_ADDR`、`MASTER_PORT`、`RANK`、`WORLD_SIZE`、`NPROC_PER_NODE`，无需手动设置。
* `--deepspeed_multinode_launcher standard` 在百舸这类无 pdsh / SSH 免密的环境做多机训练时必需：accelerate 在 DeepSpeed 模式下默认走 pdsh launcher 枚举节点，该环境下不会报错，而是退化成每节点各自跑单机。该参数在单机场景下无副作用。
* 脚本内不设置 NCCL 变量，交由平台注入。

10.4 查看日志与产物
在开发机终端中查看训练日志的 `[Step N]` 指标行（仅 rank 0 打印，一行一步，步号跨 epoch 累加）：

```
[Step   126] | Rank: 0 | loss: 0.00296 | lr: 1.00000e-05 | step/t: 13.856s | fwd/t: 0.000s | bwd/t: 0.000s | opt/t: 0.000s | TFLOPS: 187.714
```
* `step/t` 为单步瞬时墙钟时间（含前向 / 反向 / 优化器），是性能观测的主要口径。
* 进度条上的 `13.82s/it` 是 tqdm 自算的平均迭代耗时，含首步与数据加载摊薄，与 `step/t` 不是同一量，不建议用于性能评估。
* 首步包含编译预热，耗时明显偏高，不计入稳态性能。
* `fwd/t`、`bwd/t`、`opt/t` 恒为 `0.000`，为当前版本未开启细分计时，不是异常。

checkpoint 输出在 `OUTPUT_DIR`（默认 `${WORKSPACE_DIR}/models/train/Wan2.2-T2V-A14B_high_noise_full`），存盘时已去掉 `pipe.dit.` 前缀，为 bf16 精度。

十一、性能参考
以下数据为特定测试环境下的参考值，`step time` 取自训练稳态末 10 步的区间，TFLOPS 由 `--total_flops 2600.933` ÷ step time 换算，与 step time 同源。实际性能会受资源规格、数据读取方式、网络环境、任务参数等因素影响。

硬件	镜像	并行	step time	吞吐（sample/s）	TFLOPS	末步 loss	加速比
lgn7ib × 8	优化版	单机 8 卡，250 step	13.134–13.141 s	0.6089	197.9–198.0	0.00073	1.004×
lgn7ib × 8	baseline	单机 8 卡，250 step	13.184–13.194 s	0.6066	197.1–197.3	0.00057	—
lgn7ib × 16	优化版	双机 16 卡，126 step	13.812–13.856 s	1.1566	187.7–188.3	0.00296	—

双机口径说明：global batch 由 8 翻到 16，因此 step 数减半为 63/epoch × 2 = 126 step，与单机不可直接比较总耗时与 loss。按每卡计算量不变折算，双机吞吐为单机的约 1.90 倍，弱扩展效率约 95%，差值即跨机通信开销。

> 测试扩展比时无需修改 batch size 等配置，直接使用脚本默认值即可；当前配置 micro batch size 仅支持 1。

十二、常见问题

12.1 权重加载报错 `Cannot detect the model type`

报错示例：

```
ValueError: Cannot detect the model type. File: []. Model hash: d41d8cd98f00b204e9800998ecf8427e
```

该 hash 是空内容的 MD5，说明 glob 没有匹配到任何文件。

原因：框架加载时默认开启 `redirect_common_files`，会把 T5 / VAE 的 `.pth` 静默改写到 Converted-Safetensors 仓库中的 `.safetensors`。若该仓库未下载或路径不对，就会找不到文件。日志中出现 `... is redirected to ('DiffSynth-Studio/Wan-Series-Converted-Safetensors', ...)` 即为该逻辑生效。

解决：确认权重清单中的仓库 2 已下载至 `${MODEL_BASE_PATH}/DiffSynth-Studio/Wan-Series-Converted-Safetensors/`。

12.2 tokenizer 加载失败

未显式传 `--tokenizer_path` 时，框架会从 `Wan-AI/Wan2.1-T2V-1.3B:google/umt5-xxl/` 加载 tokenizer。

解决：补齐权重清单中的仓库 3；或显式传 `--tokenizer_path`，指向 A14B 仓库自带的 `google/umt5-xxl/`。

12.3 报错 `Can't load feature extractor`

未显式传 `--audio_processor_path` 时，框架会加载 `Wan-AI/Wan2.2-S2V-14B:wav2vec2-large-xlsr-53-english/`。T2V 任务用不到该模块，但 pipeline 构造阶段仍会加载。

解决：补齐权重清单中的仓库 4 下的 4 个 json 文件（仅 2.2 KB）。

12.4 多机训练退化为多个独立单机

日志中出现 `world_info={"localhost": [0..7]}`、`--master_addr=127.0.0.1` 或 `Unable to find hostfile`，说明多机通信退化成每节点各自跑单机，此时的 step time 不能作为多机性能数据。

解决：确认启动命令中包含 `--deepspeed_multinode_launcher standard`；确认平台注入的 `MASTER_ADDR`、`WORLD_SIZE` 等环境变量正确。可通过以下方式核对实际并行规模：

```bash
#### NCCL 通信域、DeepSpeed 数据并行度均应等于总卡数（双机 16 卡场景为 16）
grep -m1 -oE 'nranks [0-9]+' <训练日志>
grep -m1 -oE 'self.dp_world_size=[0-9]+' <训练日志>
```

12.5 工作目录约束

脚本内部会 `cd ${WORKSPACE_DIR}`，因此从任意目录调用均可。但若显式覆盖 `WORKSPACE_DIR`，必须指向 DiffSynth-Studio 仓库根，否则 `--config_file` 与 `train.py` 的相对路径会找不到。

12.6 断点续训

当前训练框架版本（DiffSynth-Studio 2026.1 main branch）不支持断点续训，加速镜像也未提供该能力。如有续训需求，请联系百舸相关人员评估。

十三、后续操作

* 训练产出的 checkpoint 位于 `OUTPUT_DIR`，可用于后续推理或继续微调。
* 需要调整分辨率或帧数时，同步重算 `--total_flops`，否则 TFLOPS 指标不准确。
* 需要长稳压测时，直接加大 `DATASET_REPEAT`；需要对齐固定步数时使用 `TARGET_STEPS`。

通过 WebRTC 方式可视化连接 Isaac Sim

案例介绍

NVIDIA Isaac Sim 是基于 NVIDIA Omniverse 构建的机器人仿真平台，本文是在开发机上基于 Isaac Sim 实现可视化能力的实践教程。

NVIDIA Isaac Sim 支持从 URDF、MJCF 和 CAD 等常见格式导入机器人系统。通过 GPU 加速物理引擎来实现精确仿真，并支持大规模、多传感器 RTX 渲染。集成数据生成、强化学习、ROS 集成和数字孪生模拟等端到端工作流程。其优势包括：

* **资产导入 & 导出**：支持从非 USD 格式导入机器人和环境，或将其导出为非 USD 格式；
* **机器人微调**：优化机器人的物理精度、计算效率或真实感；
* **机器人仿真**：提供全流程运动控制工具链，如控制器、运动生成和运动学求解器以及策略集成接口；
* **传感器系统**：支持 RTX 和物理学传感器。

* **官方地址**：https://docs.isaacsim.omniverse.nvidia.com/latest/index.html

部署方案

本文采用 **WebRTC 串流**方式实现 Isaac Sim 的可视化：由开发机上的 Isaac Sim 直接对外推流，本地安装 NVIDIA 官方的 Isaac Sim WebRTC Streaming Client 进行连接，画面由 GPU 的 NVENC 硬件编码器实时压缩，交互性能较好。

该方式**不支持身份鉴权**，推荐配合云的安全组使用（详见 2.7 访问控制）。

部署环境要求

组件	部署要求	最佳实践
CPU	建议 8 核及以上	按需
内存	建议 64G 及以上	按需
GPU	Isaac Sim：支持 RT Core 的 NVIDIA GPU 卡，1 卡及以上	按需
CDS	按需	按需
其它	无	无

使用需知：

* Isaac Sim 的仿真渲染依赖 DLSS（超采样技术，Deep Learning Super Sampling），因此推荐使用 GeForce RTX 系列显卡。
* 本案例使用 RT Core 的显卡，且使用的 BLB 网络带宽为 50Mbps 时，无噪点、无卡顿，表现更为符合期望。
* 最新版本的 Isaac Sim 需要较高的驱动版本，但 NVIDIA 官方在较高版本驱动显卡兼容性存在一些未解决的问题，如果出现不能运行的情况，请联系售后支持同学调整。
* 官方未对 CUDA 和 cuDNN 有限制，不同的 CUDA 版本和 cuDNN 版本是否对可视化界面的表现有影响未验证。
* 由于该工具的较多场景需要访问外网，可以通过设置服务网卡，并设置 HTTPS 网络代理支持访问，参考[服务网卡的配置](https://cloud.baidu.com/doc/VPC/s/Plztid5es#2-%E5%88%9B%E5%BB%BA%E6%9C%8D%E5%8A%A1%E7%BD%91%E5%8D%A1)。
* 建议在距离您最近的地域创建开发机实例使用当前方案。如果画面不流畅，建议在客户端降低 Resolution。

使用说明

本文会先介绍如何【创建与登录开发机】、如何【通过 WebRTC 方式可视化连接 Isaac Sim】，然后以【执行官方四足和人形可操作演示】、【宇树 GR1 机器人简易操作演示】和【Replicator 数据生成】几个例子，简单介绍 Isaac Sim 的使用。

例子，简单介绍 Isaac Sim 的使用。

在试用案例之前，请确保已挂载了案例所需的数据集，否则脚本无法正常运行，可在脚本内更改数据集地址配置。

1. 创建与登录开发机
1.1 从快速开始创建一个【Isaac Sim 可视化】开发机实例
点击【在开发机中打开】创建开发机，主要配置如下：

* **基本信息**
  * 版本选择：Isaac Sim 6.0

* **资源配置**
  * GPU 选择：支持 RT Core 的 NVIDIA GPU 卡
  * CPU/内存：推荐 8 核 64G 及以上，保证运行所需的基础资源配置

* **访问配置**
  * 启动自定义端口，选择支持**公网访问**的 BLB（建议选择网络带宽 ≥ 50Mbps 的 BLB，带宽越大越好，越少共享复用越好）
  * 新增访问端口：WebRTC 串流所需的两条端口映射见 [2.1 端口配置要求](#21-端口配置要求)

1.2 登录开发机
填写完表单，提交后，在开发机列表可见创建的开发机，等待资源调度并部署成功后，点击【登录】，即可进入开发机 webIDE。

主要路径介绍：

* Isaac Sim USD 资产挂载路径：`/mnt/bos/isaac-sim`，该目录下存有 Isaac Sim 预置的部分 USD 资产。

1.3 验证 GPU 硬件和显卡驱动版本
根据部署环境要求成功创建开发机后，点击登录开发机，进入开发机 webIDE，打开 Terminal 界面，输入以下命令确认 NVIDIA 的 GPU 硬件和显卡驱动版本：

```bash
nvidia-smi
```
执行结果中可查看显卡驱动版本和 GPU 硬件型号，**如显卡获取不到出现报错，建议联系售后支持同学重装节点驱动。**

2. 通过 WebRTC 方式可视化连接 Isaac Sim
2.1 端口配置要求
创建开发机时需开启自定义端口，并配置以下两条映射：

服务名称	开发机监听端口	BLB 监听端口	传输协议	用途
`webrtc-signal`	49100	49100	TCP	信令握手
`webrtc-media`	47988	47988	**UDP**	音视频数据

> **注意**
> 1. **两个端口的「开发机监听端口」与「BLB 监听端口」必须填相同的数字。** Isaac Sim 会把自己监听的端口号写进 SDP 告知客户端，BLB 的端口转换不会被改写。
> 2. **媒体端口必须是 UDP。**
>
> 49100/47988 只是本文示例值，可改为其它未占用端口，但必须两侧一致，后文命令中的端口请一律替换为您实际配置的值。

2.2 预检 GPU 编码器（几秒钟，请勿跳过）
本方式由 GPU 的 **NVENC 硬件编码器**实时压缩画面，但当前 NVIDIA 官方针对部分显卡支持存在问题，请先运行以下检查脚本并确认输出结果。

```bash
/root/workspace/quick_start/check_gpu.sh
```
通过时输出：

```
== GPU ==
  NVIDIA GeForce RTX 4090, 580.159.04
== CUDA ==
  [ OK ] CUDA 可用, 显存 21269/24080 MiB 空闲
== NVENC 编码器(决定 WebRTC 能否出画面)==
  [ OK ] 成功打开编码会话 (NVENC API 13.0)
== NVDEC 解码器(参考项)==
  [ OK ] h264 解码支持, 1 个 NVDEC, 最大 4096x4096

结论: 通过 -- 本机可用于 WebRTC 串流
```
不通过时输出（示例为编码器不可用的典型错误码）：

```
== NVENC 编码器(决定 WebRTC 能否出画面)==
  [失败] 无法打开编码会话: 2 (UNSUPPORTED_DEVICE)

结论: 失败(1 项) -- 本机的 GPU 编码器在容器内不可用
      WebRTC 方式必定黑屏。请换一台开发机,或提工单请平台在宿主机侧修复。
```
结论为失败时，请提交工单联系官方为您在宿主机侧解决问题。

2.3 启动串流服务
把 `<EIP>` 替换为开发机详情页的**公网访问地址（EIP）**，同时将端口号依次改为您配置的 TCP 与 UDP 端口：

```bash
/root/workspace/quick_start/launch_stream.sh -b <EIP> 49100 47988
```
输出示例：

```
客户端请填 ->  Server 114.111.22.233   Signal 49100/tcp   Stream 47988/udp
pid 614   日志 /root/webrtc-test/stream.log
首次启动约需 310 秒(编译 shader),之后约 20 秒。就绪后执行 check_stream.sh 自检。
```
`-b` 表示后台运行。若希望前台运行（Ctrl-C 即停），去掉 `-b`。端口使用默认值 49100/47988 时可省略，即 `/root/workspace/quick_start/launch_stream.sh -b <EIP>`。

不确定参数是否填对时，可用 `-n` 干跑，只打印将要执行的命令而不启动：

```bash
/root/workspace/quick_start/launch_stream.sh -n <EIP> 49100 47988
```

2.4 等待就绪并自检
百舸官方最新版镜像中已预置了 shader 缓存，大约可节省 2～5 分钟的编译时间：

情形	耗时（实测）
镜像已预热缓存、同型号 GPU（常见情形）	约 **17 秒**
缓存未命中，需完整编译 shader	约 **5 分钟**（306～308 秒）
换了不同型号 GPU，缓存部分命中	约 **3 分钟**（178 秒）

```bash
/root/workspace/quick_start/check_stream.sh
```
全部通过时输出：

```
== 1. 进程 ==
  [ OK ] isaacsim 正在运行, pid 614
  预期  publicIp=114.111.22.233  信令=49100/tcp  媒体=47988/udp
== 2. app ready ==
  [ OK ] [16.750s] app ready
  [ -- ] 日志中的 ROS2 Bridge 报错 -- 无害,可忽略
  [ -- ] glfw 插件警告 -- 无显示器时的正常现象,可忽略
== 3. 信令端口监听 ==
  [ OK ] TCP 49100 处于 LISTEN
== 4. SDP ICE candidate(决定性)==
  candidate:1 1 udp 2122260223 114.111.22.233 47988 typ host
  [ OK ] SDP 通告 114.111.22.233:47988/udp -- 与启动参数一致
  [ -- ] 仍需确认 BLB 已将 47988 映射为 47988/UDP;此处不一致时表现为纯黑屏且无任何报错

结论: 通过
客户端请填 ->  Server 114.111.22.233   Signal 49100   Stream 47988
```
若第 2 项显示「尚未就绪」，说明还在启动中，等待后重试，或用 `tail -f /root/webrtc-test/stream.log` 观察进度。

停止服务

```bash
pkill -f "livestream.app/primaryStream"
```

2.5 在本地安装并连接客户端
第 1 步：下载客户端

从 NVIDIA 官方下载 Isaac Sim WebRTC Streaming Client，选择与开发机 Isaac Sim 版本匹配的客户端版本：https://docs.isaacsim.omniverse.nvidia.com/latest/installation/download.html

> **注意**：官方仅提供 Windows x86_64、Linux x86_64 和 macOS x86_64 版本。Apple Silicon（M 系列）Mac 需通过 Rosetta 运行 x86_64 版本。
第 2 步：填写连接参数

打开客户端，按 2.4 节最后输出的三个值填写：

字段	填写内容	示例
Server	开发机 EIP	EIP
Signal Port	信令端口	`49100`
Stream Port	媒体端口	`47988`
Resolution	分辨率	`1280x720 HD`

点击 Connect。正常情况下几秒内即出现 Isaac Sim 界面，右上角性能面板可见帧率（实测 1280x720 下 60.02 FPS，帧时间 16.66 ms）。

> **第一次 Connect 黑屏时，先原地重连一次再排查。** 一个串流服务同时只服务**一个客户端**，`check_stream.sh` 的第 4 项检查本身会短暂占用这个名额，服务端回收有数秒延迟。若刚跑完自检就立刻 Connect，可能撞上名额未释放，等几秒重连即可。
> **注意**：第二个客户端连接会得到黑屏；重启串流服务会强制接管会话并丢失未保存的工作。

2.6 常见问题
Q1：客户端连上了，但画面一直全黑

这是本方式最典型的故障。**先做两件不花钱的事**：等几秒**重连一次**（排除会话名额未释放），然后看日志区分原因：

```bash
grep -aE "Got stop event|Failed to create video stream" /root/webrtc-test/stream.log
```
日志中出现	原因	去看
`Got stop event while waiting for client connection`	媒体通路不通（端口/IP 配置）	下面的 A
`Failed to create video stream 0` + `NVST_R_INVALID_STATE`	开发机 GPU 编码器不可用	Q9
两条都没有	客户端还没真正连上，或仍在启动中	先跑 `/root/workspace/quick_start/check_stream.sh`

> **注意**：在客户端**连接之前**，日志里不会有任何相关报错——端口填错不会在启动阶段暴露，`launch_stream.sh` 与 `check_stream.sh` 都会显示正常。只有在客户端点了 Connect 之后，日志里才会留下上表的痕迹。
A. 媒体通路不通的排查顺序

1. 在开发机执行 `/root/check_stream.sh`。它的第 4 项会打印 Isaac Sim 实际告知客户端的地址：

```
candidate:1 1 udp 2122260223 114.111.22.233 47988 typ host
                             ^^^^^^^^^^^^^^ ^^^^^
                             必须是 EIP     必须等于 BLB 的 UDP 端口
```
2. 若这里的 IP 是内网地址 → 2.3 节的 `<EIP>` 填错了。
3. 若这里的端口与开发机详情页 BLB 配置的 UDP 端口不一致 → 修正 2.1 节的端口映射，或用一致的端口重启服务。**端口数字写错一位（如 47988/47998）是最常见的原因。**
4. 若 IP 和端口都对，则检查 BLB 的 `webrtc-media` 一条是否配成了 **UDP** 而非 TCP。

5. **以上全部核对无误仍然黑屏时，不要相信控制台的端口表，去实测一次**——我们遇到过控制台已显示 `47988 → 47988` 但转发规则实际没有生效的情况（改配置后 BLB 内侧监听已刷新、EIP 外侧的端口转换仍是旧值）。实测方法见 Q9。

**Q2：日志里有 **`[Error] ... ROS2 Bridge startup failed`

无害，可忽略。串流不依赖 ROS2 Bridge。

**Q3：界面底部状态栏有黄色警告 **`Failed to startup plugin carb.windowing-glfw.plugin`

无害，可忽略。串流模式下开发机没有本地显示器，该插件本就无法启动；Isaac Sim 状态栏会一直显示最后一条警告。

Q4：日志里找不到 **`Started primary stream server on signal port ...` 这一行**

不代表失败。该日志行在部分环境不输出。**请以 **`/root/check_stream.sh`** 的结果为准**，不要以这行日志判断是否启动成功。

Q5：`netstat`** / **`ss`** 看不到媒体端口（47988/UDP）在监听**

正常现象。该 UDP 端口在客户端接入前不会出现在监听列表中，不是故障。

Q6：连接成功后想确认是否真的在用 GPU 硬件编码

客户端连上后，在开发机执行：

```bash
nvidia-smi --query-gpu=encoder.stats.sessionCount,encoder.stats.averageFps --format=csv
```
正常输出 `1, 59` 左右。**注意此命令仅在客户端已连接时有效**，连接前 `sessionCount` 恒为 0，不能用于连接前的健康检查。

Q7：`check_stream.sh`** 第 4 项偶尔报「没有 ICE candidate」或「peerRemoved」**

因为串流会话名额同时只有一个，而这项检查本身要占用它，服务端回收有延迟，实测约每 3 次尝试才有 1 次能拿到 SDP。脚本内部已做 6 次重试，正常情况下不会暴露该现象。若您手工写检查逻辑，**务必重试**，否则会得到随机的假失败。

Q8：画面卡顿

参考前文「使用需知」：确认 BLB 带宽不低于 50 Mbps，开发机创建在离使用者最近的地域，并在客户端降低 Resolution。

Q9：控制台端口表看起来是对的，如何实测转发真的生效？

在开发机起一个临时 UDP 接收端（把 47988 换成您的媒体端口）：

```bash
python3 - <<'EOF'
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("0.0.0.0", 47988)); s.settimeout(60)
print("listening 47988/udp")
for _ in range(5):
    d, a = s.recvfrom(2048); print("收到", d, "来自", a, flush=True)
EOF
```
在本地电脑（客户端所在机器）另开一个终端，把 `<EIP>` 换成实际地址：

```bash
python3 -c "import socket;s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM);[s.sendto(b'ping%d'%i,('<EIP>',47988)) for i in range(5)]"
```
开发机侧打印出 5 行「收到」即转发生效。若收不到但控制台显示映射正确，说明 BLB 转发规则未真正生效，需重建端口映射或提工单。

> 此测试需在**串流服务未运行**时做，否则端口已被 Isaac Sim 占用，`bind` 会失败。

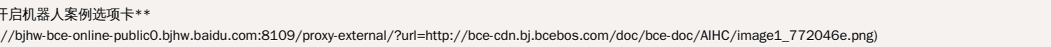
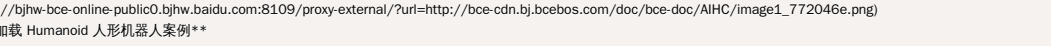
2.7 访问控制（必读）
WebRTC 串流服务**没有任何身份认证**。BLB 上开放的 49100/TCP 与 47988/UDP 对全网可达，任何人都可以连入并完整操作您的 Isaac Sim。

请在 BLB 上配置安全组，仅放通您自己的出口 IP：

* 参见 [BLB 安全组配置文档](https://cloud.baidu.com/doc/BLB/index.html)
* 需同时限制 **TCP 49100 和 UDP 47988** 两条规则，只限制其中一条无效
* 不再使用时请及时停止串流服务并关闭端口映射

---

3. 案例演示
以通过 WebRTC 客户端连入 Isaac Sim 为例：

3.1 执行官方四足和人形可操作演示
3.1.1 步骤一：开启机器人案例选项卡

3.1.2 步骤二：加载 Humanoid 人形机器人案例

点击 Load 按钮加载，Reset 按钮重置案例（或选择右侧的 Quadruped 四足机器人案例，同样点击 Load 按钮加载、Reset 按钮重置）。

一般来说，对一个模型或者脚本点击 Load 按钮，客户端窗口会停止响应一些时间进行网络加载。但在开发机刚新建时，这里的加载时间过长，也有几率是客户端卡死。

如果客户端卡死，可以关闭客户端后重新 Connect；必要时按 2.4 节的方式停止串流服务，再按 2.3 节重新启动。一般一到两次之后，一旦有一次 Load 成功，之后客户端就不会再卡死。

3.1.3 步骤三：操作机器人
窗口缩放操作：

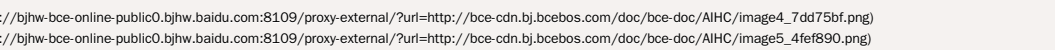
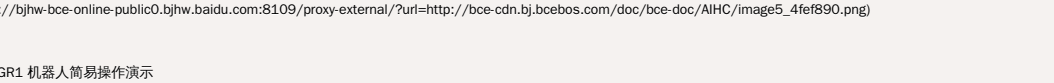
* 在窗口内使用鼠标滚轮

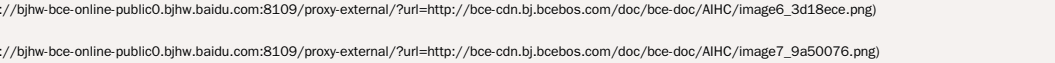
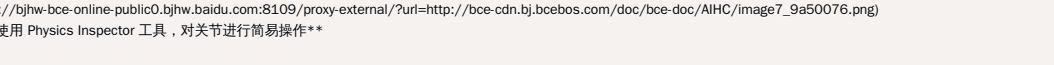
Humanoid 人形机器人操作：

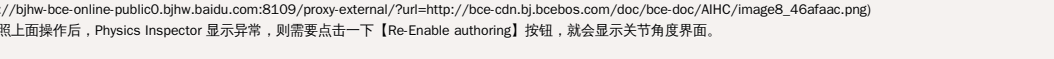
* 前进：向上箭头 / 数字键 8
* 左转：左箭头 / 数字 4
* 右转：右箭头 / 数字 6

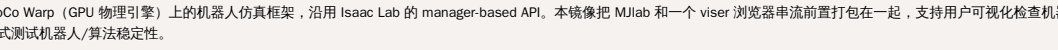
Quadruped 四足机器人操作：

* 前进：向上箭头 / 数字键 8
* 后退：向下箭头 / 数字 2
* 向左移动：左箭头 / 数字 4
* 向右移动：右箭头 / 数字键 6
* 左转：N / NUM 7
* 右转：M / NUM 9

3.1.4 运行效果



3.2 宇树 GR1 机器人简易操作演示
3.2.1 步骤一：在 Assets 中找到 GR1 机器人，并加载到工作空间中

操作结果：

3.2.2 步骤二：使用 Physics Inspector 工具，对关节进行简易操作
操作步骤：


额外步骤：如果按照上面操作后，Physics Inspector 显示异常，则需要点击一下【Re-Enable authoring】按钮，就会显示关节角度界面。



MJlab 可视化预览

MJlab 是跑在 MuJoCo Warp（GPU 物理引擎）上的机器人仿真框架，沿用 Isaac Lab 的 manager-based API。本镜像把 MJlab 和一个 viser 浏览器串流前置打包在一起，支持用户可视化检查机器人运行情况，同时可以通过编程方式测试机器人/算法稳定性。

镜像里已经预置：

内容	说明
mjlab venv	约 6.5 GB，含 torch / mujoco / mujoco-warp / viser
warp JIT 缓存	把冷启动从分钟级压到十几秒
走路人形预训练资产	demo_ckpt.pt（策略）+ motion 参考序列，开箱即可演示"会跳舞的人形机器人"

部署环境要求
部署要求	最佳实践
GPU	NVIDIA 显卡，支持 CUDA（warp 为 GPU 物理引擎）
CPU / 内存	按需
磁盘	按需（镜像含约 6.5 GB venv 与预训练资产）
网络	1 条对外 TCP 端口（默认 8435，页面与 wss 同口）+ 1 个公网 / 浮动 IP
客户端	支持 WebGL / three.js 的现代浏览器

使用需知
* 本实践需要开启 BLB 公网端口，虽然会使用自动生成 token 进行鉴权，但请谨慎使用。

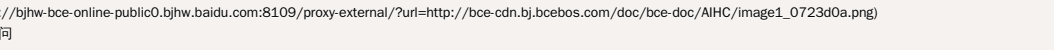
使用说明
1. 启动服务（一条命令起站）
在开发机 / 服务器上执行：

```bash
cd /root/workspace
WEB_PORT=<你的 BLB 端口> EIP=<你的公网IP> bash mjlab-stream-deploy.sh --yes
```
参数说明：

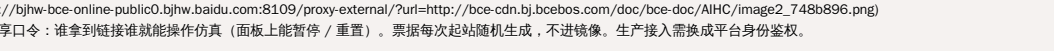
* WEB_PORT：对外 HTTPS 端口
* EIP：公网 / 浮动 IP，会写进自签证书
* --yes：非交互模式，其余全部使用镜像内默认值

约 15 秒后打印带token的访问链接。默认起一个走路人形实例（Mjlab-Tracking-Flat-Unitree-G1，4 个环境，官方预训练策略），开箱就是会跳舞的人形机器人。

启动结果：


2. 打开访问
访问链接形如：

```
https://<EIP>:<WEB_PORT>/?token=<自动生成的票据>
```
自签证书，浏览器首次会拦一下，点「继续访问」信任一次即可。打开后是实例列表，点进去就是实时画面。页面上的面板可以暂停 / 重置 / 调速 / 切视角。


> ⚠ 票据是一层共享口令：谁拿到链接谁就能操作仿真（面板上能暂停 / 重置）。票据每次起站随机生成，不进镜像。生产接入需换成平台身份鉴权。

3. 实例变更
MJ_INSTANCES 是唯一的"跑什么"来源，格式 名字=任务:环境数，逗号分隔。名字会进 URL 路径：

```bash
#### 单实例、16 个环境(只涨码率,几乎不涨显存)
MJ_INSTANCES='walk=Mjlab-Tracking-Flat-Unitree-G1:16' \ WEB_PORT=8435 EIP=<公网IP> bash mjlab-stream-deploy.sh --yes# 起多个实例,打开是列表,各点各的
MJ_INSTANCES='walk=Mjlab-Tracking-Flat-Unitree-G1:4,go1=Mjlab-Velocity-Flat-Unitree-Go1:8' \ MJ_AGENT=zero \ WEB_PORT=8435 EIP=<公网IP> bash mjlab-stream-deploy.sh --yes
```
查看有哪些任务

查看有哪些任务：

```bash
bash mjlab-stream-deploy.sh --list-tasks # 共 12 个
```
> ⚠ --agent zero 下 29 自由度的 G1 会倒、Go1 站得住。要"会走路的人形"就用默认的 trained + 走路 checkpoint。

4. 热加载（不换 URL、不换票据）
--yes 会重新生成票据，且对已经在跑的实例是"跳过"。要改运行中的站点用 --only，它不动票据、不动 URL：

```bash
#### 把 walk 从 4 环境热改成 16,URL / 票据不变
MJ_INSTANCES='walk=Mjlab-Tracking-Flat-Unitree-G1:16' \ bash mjlab-stream-deploy.sh --only walk# 热加一个新实例(两步:先起实例,再让前置把它收进 URL)
MJ_INSTANCES='walk=...:4,walk2=Mjlab-Tracking-Flat-Unitree-G1:4' \ bash mjlab-stream-deploy.sh --only walk2MJ_INSTANCES='walk=...:4,walk2=Mjlab-Tracking-Flat-Unitree-G1:4' \ bash mjlab-stream-deploy.sh --only web
```
5. 换策略 checkpoint
用 MJ_EXTRA 把参数透传给底层 play（-- 之后的参数）：

```bash
MJ_EXTRA='--checkpoint-file /你的/xxx.pt --motion-file /你的/yyy.npz' \ bash mjlab-stream-deploy.sh --only walk
```
也可以直接在面板的 Checkpoints 下拉里热换（同一 run 目录下的多个 model_*.pt，进程不重启）。

> ⚠ 当前 MJ_EXTRA 是所有实例共用同一个 ckpt。要不同实例跑不同策略，需要给脚本加每实例配置。

运维命令参考
```bash
bash mjlab-stream-deploy.sh --check # 六项就绪门禁(前置/端口/回环/安全绑定/握手/收帧)
bash mjlab-stream-deploy.sh --status # 端口/进程/实例占用,不改状态
bash mjlab-stream-deploy.sh --url # 只打印带票据的访问链接
bash mjlab-stream-deploy.sh --gpu # GPU 预检(驱动 / torch CUDA / warp)
bash mjlab-stream-deploy.sh --teardown # 停所有进程(前置 + 实例)
```
常见问题
现象	原因 / 处理
画面里有个绿色半透明人形	那是 motion 参考影(ghost)，不是第二个机器人，默认只画在选中的那个环境上；它没有物理、不参与仿真
一堆人形里某个动作不一样	tracking 各环境的参考相位起点不同（同一策略、同一支舞的不同小节），或它刚被重置恢复。面板 Select env + Track camera 可单独盯它确认
打开后画面永远"连接中"	中间套了会改请求头的反代(nginx / 平台 /proxy/<port>/)，子协议被吃掉。用本镜像自带前置，别再套一层
换端口后门禁说"前置没在跑"	门禁探的是默认口；看 conf/ports.env，或删掉它回默认
--only <名字> 找不到实例	名字要和 MJ_INSTANCES 里的一致
对外口被别的进程占着	换一个口：WEB_PORT=8439 …
curl … | bash 直接报错退出	自解压脚本要读自身文件，管道进来读不到载荷。先落地成 .sh 再 bash

运行效果

通过网页方式可视化连接 Isaac Sim

案例介绍

Isaac Sim 是 NVIDIA 基于 Omniverse 构建的机器人仿真平台，支持 URDF/MJCF/CAD 资产导入、GPU 加速物理仿真和渲染，可用于合成数据生成、强化学习和数字孪生等场景。本文介绍如何在百舸开发机上用一条命令启动 Isaac Sim，并在本地浏览器中直接查看和操作仿真画面。

部署环境要求

组件	部署要求	最佳实践
CPU	建议 8 核及以上	按需
内存	建议 64G 及以上	按需
GPU	支持图形渲染的 NVIDIA GPU，1 卡及以上	按需
浏览器	最新版 Chrome 或 Edge	按需

使用需知：
* 新版 Isaac Sim 对驱动版本要求较高，无法运行时请联系售后。

1. 创建与登录开发机

1.1 创建开发机

* 资源配置：选择支持图形渲染的 NVIDIA GPU，推荐 8 核 64G 及以上。

1.2 登录开发机

在开发机列表点击【登录】进入 webIDE，并点击页面顶端的【Notebook】

2. 进入 Isaac Sim 可视化界面

2.1 进入仿真界面
在页面 Notebook 中点击 Isaac Sim 图标，会自动跳转至可视化界面

2.2 在浏览器中打开画面
在此界面中您可以快速搭建您的仿真环境，同时百舸预置了一些简单的延迟测试/全屏/重启功能，帮助您选择更适合的分辨率。

2.3 页面功能

功能	说明
键鼠操作	点击画面后，鼠标、滚轮和键盘直接操作 Isaac Sim 的整个界面，包括菜单、面板和 3D 视口。Mac 上的 Cmd 组合键留给浏览器使用
全屏	点击【全屏】、双击画面或按 F11，画面按比例铺满屏幕。Chrome / Edge 中短按 Esc 为 Isaac Sim 中的 Esc 按钮，长按 Esc 退出全屏

只看视口	按 F7（Isaac Sim 自带功能）隐藏所有面板，只保留 3D 视口；再按一次恢复
统计	显示帧率、码率、网络延迟和解码延迟
测延迟 ×10	点击【统计】后出现。自动测 10 次从操作到画面变化的延迟
重启与分辨率调整	点击【重启】，选择分辨率（1280×720、1440×900、1920×1080、2560×1440、3840×2160）后点击【重启 Isaac】。只重启 Isaac Sim，网页不断开，约 15–20 秒出新画面；所选分辨率会保存，下次开机沿用。

2.4 命令与配置

以下命令在 JupyterLab 终端中执行。

命令	说明
`isaac-stream status`	查看状态、告警和访问地址
`isaac-stream start`	启动；已在运行则只打印访问地址
`isaac-stream stop`	停止 Isaac Sim 和网页服务
`isaac-stream restart`	重启；修改配置文件后执行
`isaac-stream check --wait 900`	等待就绪后自检：端口只在本机、帧率、键鼠通道、访问鉴权
`isaac-stream logs [kit|bridge|supervisor] [-f]`	查看日志，默认是 Isaac Sim 的日志，`-f` 持续输出

配置文件为 `/opt/isaac-stream/etc/isaac-stream.conf`，常用配置项如下，其余项保持默认即可：

配置项	默认值	说明
AUTOSTART	1	开机是否自动启动，`0` 为不启动
SCENE	空	启动时打开的 USD 文件，填绝对路径；为空时打开空场景
CAMERA	空	视口使用的相机 prim 路径，例如 `/World/Camera`；为空时使用默认透视相机
AUTOPLAY	0	打开场景后是否自动播放仿真，`1` 为自动播放
RESOLUTION	1440x900	画面分辨率；页面上【重启】时选择的分辨率会写入这里
READY_TIMEOUT	1200	启动后多少秒内没出画面就自动重启 Isaac Sim

例如，让 Isaac Sim 启动时打开自己的场景并自动播放：

```bash
CONF=/opt/isaac-stream/etc/isaac-stream.conf
sed -i 's|^SCENE=.*|SCENE=/root/workspace/scenes/my_scene.usd|; s|^AUTOPLAY=.*|AUTOPLAY=1|' $CONF
isaac-stream restart
```

将示例中的路径替换为实际的 USD 文件路径即可。修改配置后，执行 `isaac-stream restart` 或在页面上点击【重启】生效。

2.5 常见问题

Q1：页面显示「连不上网页服务」？
Isaac Sim 没有在运行。在终端执行 `bash /root/workspace/start_isaac.sh`，页面会自动显示启动进度，无需刷新。

Q2：页面一直显示「Isaac 启动中」或「Isaac 加载中」？
首次启动时，需要编译着色器，约 5 分钟，页面会显示已等待时间。如果超过 10 分钟仍未出画面，执行 `isaac-stream logs kit` 查看日志末尾；若日志中有 NVENC 或编码相关的报错，请提工单，由平台在宿主机侧修复。启动后 20 分钟仍无画面时，系统会自动重启 Isaac Sim。

Q3：页面显示「Isaac 没有起来」？
Isaac Sim 连续 3 次启动失败后会停止重试，页面上会显示失败原因。执行 `isaac-stream logs kit` 查看日志，处理后点击【重启】或执行 `isaac-stream restart`。若提示配置文件有错，按提示改好配置文件后再点击【重启】。

Q4：页面显示「这个浏览器放不了画面」？
当前浏览器不支持 WebCodecs 视频解码。请换用最新版 Chrome 或 Edge。

Q5：键盘输入没有反应？
先点击一下画面，让画面获得键盘焦点。切换过标签页或窗口后，也需要重新点击一下。

Q6：画面卡顿或延迟大？
点击【统计】查看帧率和网络延迟。可在【重启】中调低分辨率，并只保留一个画面标签页，每个标签页都单独占用一份带宽。

Q7：页面提示「画面已 N 秒没有更新，Isaac 可能正忙或卡住了」？
Isaac Sim 可能正在加载大场景或执行耗时的脚本，等待即可。长时间不恢复时，点击【重启】。

Q8：执行 `isaac-stream stop` 后，点击【Isaac Sim】图标打不开画面？
启动器图标每次开机后只执行一次启动，之前点过就不会再执行。请在终端执行 `bash /root/workspace/start_isaac.sh`，然后打开它打印的访问地址。

RLinf v0.4 基于RL的仿真-真机协同训练

RLinf v0.4 基于RL的仿真-真机协同训练

概述

[RLinf](https://rlinf.readthedocs.io/zh-cn/latest/) 是一个面向 **Embodied AI（具身智能）** 和 **Agentic AI（智能体 AI）** 的开源强化学习基础框架。

仿真-真机协同训练（RL-Co）把两种训练信号放进同一个优化循环：在仿真环境中用 PPO 做强化学习，同时在真机轨迹上做 SFT。这样可以提升仿真任务的成功率，同时保留真机数据带来的先验，避免策略只在仿真里过拟合、削弱 sim-to-real 迁移效果。技术细节可参考论文 [Beyond Imitation: RL-Based Sim-Real Co-Training for VLA Models](https://rlinf.readthedocs.io/zh-cn/latest/rst_source/resources/publications/rlinf_co.html)。

通过百舸快速开始，您可以使用开发机调试部署 RLinf v0.4，基于 ManiSkill 数字孪生环境和 50 条真机轨迹，对 π0.5 模型进行两阶段协同训练，并通过 TensorBoard、Ray Dashboard、训练日志和 GPU 指标监控训练进展。

![仿真-真机协同训练总览](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/AIHC/overview_4e5ec2b.png)

项目	**说明**
算法	PPO + SFT（RL-Co）
模型	π0.5
环境 / 数据	ManiSkill 数字孪生 `PandaPutOnPlateInScene25DigitalTwin-v1` + 50 条真机轨迹
训练流程	阶段一 SFT 预训练 → 阶段二 仿真-真机协同 RL 训练
配置文件	`examples/embodiment/config/maniskill_ppo_co_training_openpi_pi05.yaml`

适用场景

- 使用少量真机数据和对齐的仿真数字孪生，训练抓取放置类机器人策略模型。
- 在 PPO 强化学习中引入真机 SFT 约束，缓解纯仿真 RL 带来的 sim-to-real 退化。
- 复现官方 `maniskill_ppo_co_training_openpi_pi05` 示例，或迁移到自己的机器人与仿真场景。
- 在单机开发机上排查调试 ManiSkill 资产、OpenPI 数据加载、norm stats、Ray 和 TensorBoard 相关训练问题。
- 观察 `train/ppo_loss`、`train/sft_loss` 与 `train/loss_ratio`，调整 `sft_loss_weight` 平衡 RL 与 SFT。

任务说明

本示例仅提供一个演示任务。用于自己的机器人时，需要采集真机数据并构建与之对齐的仿真场景。

部分	**说明**
任务	抓取放置：将桌上的物体放入碗中
真机	Franka Emika Panda + RealSense；第三人称 RGB（640×480）；7 自由度动作（x, y, z, roll, pitch, yaw, 夹爪）
仿真	基于 ManiSkill3 的数字孪生，在布局、相机视角、任务逻辑、语言与动作空间上与真机对齐；动力学经过调校以逼近真实物理
真机数据	LeRobot v2.1 格式，50 条轨迹、3937 帧、fps=5，`state` / `actions` 均为 7 维

训练规格参考 & 环境配置

推荐训练配置如下：

项目	**推荐配置**
GPU	单机 8 卡，建议单卡 80GiB 以上显存；
CPU	建议 64 核以上，ManiSkill GPU 仿真与 Ray worker 会占用较多 CPU
容器内存	建议 512GiB 以上
共享内存	建议 `--shm-size` 不小于 20G（与官方 Docker 启动参数一致）
存储空间	模型、数据与资产约 26G（混合数据集 15.5G、SFT 权重 7.5G、真机数据 2.5G）；每个 checkpoint 实测约 16G，100 步存 2 个、1000 步存 20 个，完整训练建议预留 400G 以上

项目目录配置如下表所示：

项目	**推荐配置**
Python 环境	镜像内 `source switch_env openpi`；自定义环境为 `/root/workspace/RLinf/.venv`
工作目录	`/root/workspace/RLinf`
训练入口	`/root/workspace/RLinf/examples/embodiment/train_embodied_agent.py`
训练配置	`/root/workspace/RLinf/examples/embodiment/config/maniskill_ppo_co_training_openpi_pi05.yaml`
ManiSkill 资产	`/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets`
模型目录	`/root/workspace/models`
数据集目录	`/root/workspace/datasets`
日志目录	`/root/workspace/logs`

预期效果参考

以下结果来自 RLinf 官方示例：

阶段	仿真成功率（`env/success_once`）
加载阶段一 SFT 权重，零样本	约 35%
阶段二协同训练 100 步后	约 50%
效果提升	约 +15%

真机部署效果和消融实验见论文 [Beyond Imitation: Reinforcement Learning-Based Sim-Real Co-Training for VLA Models](https://rlinf.readthedocs.io/zh-cn/latest/rst_source/resources/publications/rlinf_co.html)。

百舸实测（8×B30Z，官方复现 100 步）

环境：单机 8 卡 B30Z（275G），镜像内 `openpi` 环境，RLinf commit `91a302e`，配置 `maniskill_ppo_co_training_openpi_pi05`，仅追加 `runner.max_steps=100` 与三个路径 override。训练正常退出，无报错、无 OOM。

项	实测值
总耗时	100 步约 4.71 小时
step 0	277s（含首次加载）
普通步	平均约 160s（rollout 约 137s + actor 训练约 22s + 权重同步约 2s）
评测步（每 10 步）	约 235s，评测本身约 76s
评测 + 存盘步（第 50、100 步）	约 275–280s，每个 checkpoint 约 16G
显存	约 31G/卡
训练 `env/success_once`	前 10 步均值 0.357 → 最后 10 步均值 0.516
评测 `eval/success_once`（64 条轨迹）	第 10 步 0.50 → 第 100 步 0.75
评测 `eval/success_at_end`	第 10 步 0.344 → 第 100 步 0.578

各评测点 `eval/success_once`：0.50、0.50、0.5625、0.5625、0.609、0.6875、0.6875、0.6875、0.719、0.75。训练侧成功率从约 35% 升到约 52%，与官方示例一致。`train/loss_ratio` 常态 0.1–2.7，偶有单步尖峰（实测 18.8、4.0），下一步即回落，不影响训练。

准备工作

如果您所在网络环境访问 HuggingFace 或 GitHub 较慢，建议提前配置 HuggingFace 镜像源，或配置[代理加速](https://cloud.baidu.com/doc/BCM/s/3mmswg2oh)。

huggingface 镜像配置：

```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
```

百舸平台已提供 ManiSkill 基础资产（`carrot/`）、SFT 权重和真机数据的 BOS 副本，可直接从 BOS 下载到训练任务的工作目录；只有约 4.5MB 的本示例专用资产 `custom_assets/` 需从 HuggingFace 下载。实测 BOS 拉取 7.5G 权重约 30 秒，HuggingFace 拉取 2.5G 真机数据约 7 分钟。

资源	BOS 路径（目录必须带结尾 `/`）	实测规模
SFT 权重	`bos:/aihc-models-bj/RLinf/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT/`	4 个对象，`model.safetensors` 7,473,091,464 字节
真机数据	`bos:/aihc-rdw-bj/huggingface.co/datasets/RLinf/RLCo-Example-Real-Data/`	156 个对象，约 2.5G
ManiSkill 基础资产	`bos:/aihc-rdw-bj/RLinf/maniskill_assets/carrot/`	132 个对象，约 84M

环境说明

> 说明：本文中的路径均以百舸训练任务内的 `/root/workspace` 为例。实际使用时，可根据任务挂载目录调整。

本文使用以下目录：

本文使用以下目录：

```text
/root/workspace/RLinf
/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets
/root/workspace/models
/root/workspace/datasets
/root/workspace/logs
```

步骤一：准备 ManiSkill 资产

本示例需要两部分资产：ManiSkill 基础资产 `carrot/`（胡萝卜、盘子、桌面三类模型）和本示例专用资产 `custom_assets/bridge_table_long.glb`。环境默认从 `rlinf/envs/sim/maniskill/assets` 加载资产，也可以通过环境变量 `MANISKILL_ASSET_DIR` 指向其他目录。

> 说明：`partnet_mobility/` 在本示例中不被引用（`rlinf/` 代码内零引用），无需下载；ManiSkill 自带的内置数据位于 `~/.maniskill`，镜像已预置。

最终目录结构应为：

```text
/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets/
├── carrot/
│   ├── more_carrot/   # 含 model_db.json
│   ├── more_plate/    # 含 model_db.json
│   └── more_table/    # 含 model_db.json 与 imgs/
└── custom_assets/
    └── bridge_table_long.glb
```

方式一：通过 BOS 下载基础资产（推荐）

只需拉取 `carrot/` 子目录（实测 132 个对象、约 84M）：

```bash
ASSET_DIR=/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets
mkdir -p "$ASSET_DIR/carrot"
bcecmd bos cp -r bos:/aihc-rdw-bj/RLinf/maniskill_assets/carrot/ "$ASSET_DIR/carrot/"
```

如果 `bcecmd` 报 `Object don't exist`，先确认真实目录和结尾 `/`（必须带 `-a`，否则只列前 1000 条且不报错）：

```bash
bcecmd bos ls -a bos:/aihc-rdw-bj/RLinf/maniskill_assets/
```

方式二：通过 HuggingFace 下载基础资产

先测试 HuggingFace 连通性，直连可用时不必设置镜像源：

```bash
curl -sS -o /dev/null -w '%{http_code}\n' --max-time 10 https://huggingface.co   # 200 即可直连
#### 不通时再执行：export HF_ENDPOINT=https://hf-mirror.com
cd /root/workspace
hf download --repo-type dataset RLinf/maniskill_assets --include "carrot/*" --local-dir ./maniskill_assets
cp -r ./maniskill_assets/carrot /root/workspace/RLinf/rlinf/envs/sim/maniskill/assets/
```

下载本示例专用资产

`RLinf/RLCo-maniskill-assets` 仓库只包含 `custom_assets/bridge_table_long.glb`（4,471,172 字节），需从 HuggingFace 下载：

```bash
cd /root/workspace/RLinf/rlinf/envs/sim/maniskill/assets
hf download --repo-type dataset RLinf/RLCo-maniskill-assets --include "custom_assets/*" --local-dir .
```

验证资产目录：

```bash
ASSET_DIR=/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets
for d in more_carrot more_plate more_table; do
  ls -l "$ASSET_DIR/carrot/$d/model_db.json" || echo "MISSING carrot/$d/model_db.json"
done
ls -l "$ASSET_DIR/custom_assets/bridge_table_long.glb"   # 预期 4471172 字节
```

步骤二：准备模型权重与数据

协同训练分为两个阶段：

阶段	内容	所需资源
阶段一：SFT 预训练	用真机 + 仿真混合数据做监督学习，快速注入真机与仿真知识	混合数据集 `RLinf/RLCo-Example-Mix-Data`（50 条真机 + 1499 条仿真轨迹，约 15.5G）
阶段二：协同 RL 训练	在 PPO 训练循环中加入真机 SFT 损失	阶段一 SFT 权重 + 真机数据 `RLinf/RLCo-Example-Real-Data`（50 条轨迹，约 2.5G）

阶段一可以自行训练，也可以直接使用官方 SFT 权重。快速复现建议直接下载 SFT 权重，跳过阶段一。

方式一：从 BOS 下载 SFT 权重与真机数据（推荐）

建议放入 tmux 执行，下载完成后逐文件对账大小（`bcecmd` 打印的 `[N] objects downloaded` 不能证明文件完整）：

```bash
mkdir -p /root/workspace/models /root/workspace/datasets /root/workspace/logs
cat > /tmp/bos_download_rlco.sh <<'EOF'
#### !/usr/bin/env bash
set -euo pipefail
M=/root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
D=/root/workspace/datasets/RLCo-Example-Real-Data
MB=bos:/aihc-models-bj/RLinf/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT/
DB=bos:/aihc-rdw-bj/huggingface.co/datasets/RLinf/RLCo-Example-Real-Data/
```

```
#### 逐文件对账：BOS 列表中的每个对象都必须在本地存在且大小一致
rec() { bcecmd bos ls -a -r "$1" | awk '{print $3, $NF}' | while read sz key; do
  f="$2/$key"; [ -f "$f" ] && [ "$(stat -c %s "$f")" = "$sz" ] || echo "BAD $key bos=$sz"; done; }
{
  date
  mkdir -p "$M" "$D"
  bcecmd bos cp -r "$MB" "$M/"
  bcecmd bos cp -r "$DB" "$D/"
  echo "reconcile model:"; rec "$MB" "$M"
  echo "reconcile data:";  rec "$DB" "$D"
  echo "download done"; date
} 2>&1 | tee -a /root/workspace/logs/bos_download_rlco.log
EOF
chmod +x /tmp/bos_download_rlco.sh
tmux new-session -d -s bos_download_rlco /tmp/bos_download_rlco.sh
```

查看进度与对账结果（`reconcile` 段之后没有 `BAD` 行即为完整）：

```bash
tail -20 /root/workspace/logs/bos_download_rlco.log
du -sh /root/workspace/models/* /root/workspace/datasets/* 2>/dev/null || true
```

> 说明：对账函数中 `awk '{print $3, $NF}'` 依赖 `bcecmd bos ls -a -r` 的输出列（第 3 列为大小、最后一列为相对 key）。若 BOS 列表只看到少量小文件，可能是上传尚未完成，以 `ls -a -r` 的完整列表为准。`bcecmd` 不支持 `bos du`，统计体积请用 `ls -a -r` 累加。

方式二：从 HuggingFace 下载 SFT 权重与真机数据

```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
mkdir -p /root/workspace/models /root/workspace/datasets /root/workspace/logs

hf download RLinf/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT \
  --local-dir /root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT

hf download --repo-type dataset RLinf/RLCo-Example-Real-Data \
  --local-dir /root/workspace/datasets/RLCo-Example-Real-Data
```

建议放入 tmux 并持续检查结果：

```bash
SESSION=hf_download_rlco
cat > /tmp/download_rlco.sh <<'EOF'
#### !/usr/bin/env bash
set -euo pipefail
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
{
  date
  echo "download sft model start"
  hf download RLinf/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT \
    --local-dir /root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
  echo "download real data start"
  hf download --repo-type dataset RLinf/RLCo-Example-Real-Data \
    --local-dir /root/workspace/datasets/RLCo-Example-Real-Data
  echo "download done"
  date
} 2>&1 | tee -a /root/workspace/logs/hf_download_rlco.log
EOF
chmod +x /tmp/download_rlco.sh
tmux new-session -d -s "$SESSION" /tmp/download_rlco.sh
```

查看下载进度：

```bash
tmux ls 2>/dev/null | grep hf_download_rlco || true
tail -50 /root/workspace/logs/hf_download_rlco.log
du -sh /root/workspace/models/* /root/workspace/datasets/* 2>/dev/null || true
```

方式三：自行训练阶段一 SFT

下载混合数据集：

```bash
export HF_ENDPOINT=https://hf-mirror.com
hf download --repo-type dataset RLinf/RLCo-Example-Mix-Data \
  --local-dir /root/workspace/datasets/RLCo-Example-Mix-Data
```

训练方法请参考 [OpenPI 官方代码](https://github.com/Physical-Intelligence/openpi) 或 RLinf 文档的 [监督训练微调](https://rlinf.readthedocs.io/zh-cn/latest/rst_source/examples/embodied/sft_openpi.html) 章节。阶段一与阶段二使用同一个 OpenPI 配置名 `pi05_maniskill_sim_real_co_training`（定义在 `rlinf/models/embodiment/openpi/dataconfig/__init__.py`），需确保模型结构与 normalization 统计量在两个阶段保持一致。自训权重目录需与官方 SFT 权重保持相同结构（见下文验证）。

验证模型与数据目录

SFT 权重目录中除 `model.safetensors` 外，还必须包含 norm stats；阶段二会按原始 `asset_id`（`physical-intelligence/pick_and_place_real`）从模型目录加载它：

```bash
MODEL=/root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
ls -l "$MODEL"/model.safetensors "$MODEL"/metadata.pt   # model.safetensors 预期 7473091464 字节
ls -lh "$MODEL"/physical-intelligence/pick_and_place_real/norm_stats.json
```

真机数据下载后，LeRobot 数据集根目录（包含 `meta/info.json` 的目录）位于仓库内的子目录：

```bash
SFT_DATA=/root/workspace/datasets/RLCo-Example-Real-Data/physical-intelligence/pick_and_place_real
ls "$SFT_DATA"
ls "$SFT_DATA"/meta
ls "$SFT_DATA"/data/chunk-000 | wc -l     # 预期 50
python -c "import json; d=json.load(open('$SFT_DATA/meta/info.json')); print(d['total_episodes'], d['total_frames'], d['fps'])"
#### 预期输出：50 3937 5
```

步骤三：确认训练配置

以 `maniskill_ppo_co_training_openpi_pi05` 为例，核心配置如下：

```text
cluster:
  num_nodes: 1
  component_placement:
    actor,env,rollout: all

runner:
  max_epochs: 1000
  val_check_interval: 10
  save_interval: 50

algorithm:
  adv_type: gae
  loss_type: actor_critic
  reward_type: chunk_level
  update_epoch: 2
  entropy_bonus: 0.005
  clip_ratio_high: 0.2
  clip_ratio_low: 0.2
  gamma: 0.99
  gae_lambda: 0.95

env:
  train:
    rollout_epoch: 2
    total_num_envs: 64
    max_episode_steps: 80
  eval:
    total_num_envs: 64

actor:
  micro_batch_size: 40
  global_batch_size: 2560
  enable_sft_co_train: True
  sft_loss_weight: 0.2
  sft_data_path: "/path/to/RLCo-Example-Real-Data"
  model:
    model_path: "/path/to/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT"
    add_value_head: True
    num_action_chunks: 4
    openpi:
      config_name: "pi05_maniskill_sim_real_co_training"
      noise_method: "flow_noise"
      action_horizon: 8
  optim:
    lr: 4e-6
    value_lr: 2e-4

rollout:
  model:
    model_path: "/path/to/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT"
```

关键参数说明：

参数	说明
`actor.enable_sft_co_train`	设为 `True` 开启协同训练；设为 `False` 则退化为纯 PPO 训练
`actor.sft_loss_weight`	SFT 损失权重 β，总损失为 `L_total = L_PPO + β · L_SFT`，默认 `0.2`
`actor.sft_data_path`	真机 LeRobot 数据集根目录，必须是包含 `meta/info.json` 的那一层目录
`actor.model.openpi.config_name`	OpenPI 数据配置名 `pi05_maniskill_sim_real_co_training`，决定数据变换与 norm stats 的 `asset_id`
`actor.model.model_path` / `rollout.model.model_path`	都指向阶段一 SFT 权重目录，norm stats 也从该目录加载
`actor.global_batch_size` / `actor.micro_batch_size`	每次参数更新的 PPO 样本数 / 每卡单次前向样本数；8 卡下梯度累积步数为 2560 / (40 × 8) = 8
`runner.max_epochs`	每个 epoch 对应 1 个 global step，默认 1000；复现官方 100 步结果可用 `runner.max_steps=100`
`runner.val_check_interval` / `runner.save_interval`	每 10 步评测、每 50 步保存；最后一步无论是否整除都会评测并保存；`save_interval` 必须能被 `val_check_interval` 整除

PPO 样本数与 batch 的对应关系：`64 个并行环境 × 2 个 rollout_epoch × 每个 episode 20 个 action chunk（80 步 / 4）= 2560`，正好是一个 `global_batch_size`。如果修改 `total_num_envs`、`rollout_epoch` 或卡数，需要同步调整 `global_batch_size`，并保证它能被 `micro_batch_size × GPU 数` 整除。

SFT 分支每个 micro step 从真机数据中取一个 batch：`pi05_maniskill_sim_real_co_training` 的 `batch_size=16`，按 8 卡均分后每卡每次 2 条样本；数据迭代完一轮后会自动进入下一个 epoch。

原始 YAML 中模型路径和数据路径都是占位符：

```text
/path/to/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
/path/to/RLCo-Example-Real-Data
```

实际启动时请通过 Hydra CLI override 注入真实路径。注意 `sft_data_path` 要指向仓库内的子目录：

```bash
rollout.model.model_path=/root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
actor.model.model_path=/root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
actor.sft_data_path=/root/workspace/datasets/RLCo-Example-Real-Data/physical-intelligence/pick_and_place_real
```

官方 `run_embodiment.sh` 只透传 `STEPS`、`SAVE_INTER`、`NODES` 三个环境变量，不支持追加任意 Hydra override，因此下文使用直接 Python 入口启动。如果您更习惯用官方脚本，需要先把上述三个路径

官方 run_embodiment.sh 只读取 `CUDA_VISIBLE_DEVICES`、`ROBOT_PLATFORM` 三个环境变量，不支持通过命令行 override 配置，且直接用裸 `python` 启动。如果想复现官方脚本，需要先把这些路径改写进 YAML。

推荐参数分档

档位	参数	用途
冒烟验证	`runner.max_steps=2 runner.val_check_interval=-1 runner.save_interval=-1`	首次运行，确认资产、数据、norm stats、Ray 全链路可用；关闭评测与存盘（否则最后一步会强制评测并保存）
官方复现	`runner.max_steps=100`	对齐官方示例，观察 `env/success_once` 从约 35% 上升到约 50%；8×B30Z 实测约 4.7 小时
完整训练	不追加 override（`max_epochs=1000`）	长时间训练，按 `save_interval=50` 保存 checkpoint；8×B30Z 按实测外推约 47 小时，checkpoint 合计约 320G
纯 PPO 对照	`actor.enable_sft_co_train=False`	消融对照，观察去掉真机 SFT 约束后的表现
调整 SFT 权重	`actor.sft_loss_weight=0.1` 或 `0.5`	`train/loss_ratio` 持续过大时降低，真机先验保持不足时适当提高

步骤四：启动训练

建议使用 tmux 启动训练，避免 SSH 断开导致任务中止。

以下命令使用官方复现档（100 步）启动。首次运行建议先把 `EXTRA` 改为冒烟验证档，确认链路正常后再正式启动。

```bash
cd /root/workspace/RLinf

SESSION=rlco_train
CONFIG=maniskill_ppo_co_training_openpi_pi05
MODEL=/root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
SFT_DATA=/root/workspace/datasets/RLCo-Example-Real-Data/physical-intelligence/pick_and_place_real
EXTRA="runner.max_steps=100"
#### 冒烟验证：EXTRA="runner.max_steps=2 runner.val_check_interval=-1 runner.save_interval=-1"
START_SCRIPT=/tmp/start_rlco_train.sh

LOG_DIR=/root/workspace/logs/$(date +%Y%m%d-%H:%M:%S)-${CONFIG}
LOG_FILE=${LOG_DIR}/run_embodiment.log
mkdir -p "$LOG_DIR"

cat > "$START_SCRIPT" <<EOF
#### !/usr/bin/env bash
set -e
cd /root/workspace/RLinf
#### 使用官方镜像时激活 openpi 环境；自定义环境改为 source /root/workspace/RLinf/.venv/bin/activate
source switch_env openpi
export EMBODIED_PATH=/root/workspace/RLinf/examples/embodiment
export REPO_PATH=/root/workspace/RLinf
export PYTHONPATH=/root/workspace/RLinf:\$PYTHONPATH
export MUJOCO_GL=egl
export PYOPENGL_PLATFORM=egl
export MANISKILL_ASSET_DIR=/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets
export RAY_memory_usage_threshold=0.98
export TOKENIZERS_PARALLELISM=false
CMD="python /root/workspace/RLinf/examples/embodiment/train_embodied_agent.py --config-path /root/workspace/RLinf/examples/embodiment/config/ --config-name ${CONFIG}
runner.logger.log_path=${LOG_DIR} rollout.model.model_path=${MODEL} actor.model.model_path=${MODEL} actor.sft_data_path=${SFT_DATA} ${EXTRA}"
echo "\$CMD" > ${LOG_FILE}
echo "LOG_DIR=${LOG_DIR}" >> ${LOG_FILE}
echo "MODEL=${MODEL}" >> ${LOG_FILE}
echo "SFT_DATA=${SFT_DATA}" >> ${LOG_FILE}
\$CMD 2>&1 | tee -a ${LOG_FILE}
EOF

chmod +x "$START_SCRIPT"

if tmux has-session -t "$SESSION" 2>/dev/null; then
  tmux kill-session -t "$SESSION"
fi

tmux new-session -d -s "$SESSION" "$START_SCRIPT"
printf "session=%s\nlog=%s\n" "$SESSION" "$LOG_FILE"
```

启动前建议先做一次路径预检，避免训练拉起 Ray 后才报错：

```bash
MODEL=/root/workspace/models/RLinf-Pi05-RLCo-PandaPutOnPlateInScene25DigitalTwin-V1-SFT
SFT_DATA=/root/workspace/datasets/RLCo-Example-Real-Data/physical-intelligence/pick_and_place_real
ASSET_DIR=/root/workspace/RLinf/rlinf/envs/sim/maniskill/assets
for f in \
  "$MODEL/model.safetensors" \
  "$MODEL/physical-intelligence/pick_and_place_real/norm_stats.json" \
  "$SFT_DATA/meta/info.json" \
  "$ASSET_DIR/carrot/more_carrot/model_db.json" \
  "$ASSET_DIR/carrot/more_plate/model_db.json" \
  "$ASSET_DIR/carrot/more_table/model_db.json" \
  "$ASSET_DIR/custom_assets/bridge_table_long.glb"
do
  [ -f "$f" ] && echo "OK   $f" || echo "MISS $f"
done
df -h /root/workspace   # 100 步约需 32G checkpoint 空间，完整训练约 320G
```

启动后验证点：

- `tmux ls` 中存在 `rlco_train`。
- 日志中 `--config-name maniskill_ppo_co_training_openpi_pi05`，且三个路径 override 均为真实路径。
- Hydra 配置中 `enable_sft_co_train: true`、`config_name: pi05_maniskill_sim_real_co_training`。
- Ray 启动并识别 1 node / 8 accelerators。
- 无 `Traceback`、`FileNotFoundError`、`RayOutOfMemory`、`worker died`。
- 第一个 global step 完成后（实测约 277s），TensorBoard 中出现 `sft_loss`、`ppo_loss`、`loss_ratio` 指标；实测 step 0 `env/success_once` 为 0.359，与官方零样本约 35% 吻合。

训练启动后建议立即打开 TensorBoard 和 Ray Dashboard：

```bash
ssh -p <port> -L 6006:127.0.0.1:6006 -L 18265:127.0.0.1:8265 root@<host>
```

重点查看：

- TensorBoard：`env/success_once`、`env/return`、`eval/success_once`、`train/ppo_loss`、`train/sft_loss`、`train/loss_ratio`、`train/actor/total_loss`
- Ray Dashboard：actor/rollout/env worker 状态、CPU/GPU 使用、异常 worker
- 主日志：rollout 进度、`global_step`、`SFT/PPO loss imbalance detected` 告警、异常栈

步骤五：监控训练

查看主日志：

```bash
tail -f /root/workspace/logs/<run-dir>/run_embodiment.log
```

提取关键进展：

```bash
grep -nE 'Generating Rollout Epochs|global_step|success_once|sft_loss|ppo_loss|loss imbalance|Traceback|Exception|FileNotFoundError|RayOutOfMemory|worker died' \
  /root/workspace/logs/<run-dir>/run_embodiment.log | tail -100
```

查看 GPU：

```bash
watch -n 2 'nvidia-smi --query-gpu=index,utilization.gpu,memory.used,memory.total --format=csv,noheader,nounits'
```

查看 cgroup 内存和 OOM：

```bash
watch -n 2 'cat /sys/fs/cgroup/memory.current; cat /sys/fs/cgroup/memory.max; cat /sys/fs/cgroup/memory.events'
```

查看 Ray worker：

```bash
ps -eo pid,stat,pcpu,pmem,rss,args --sort=-pcpu | \
  grep -E 'train_embodied_agent.py|ray::EmbodiedFSDPActor|ray::MultiStepRolloutWorker|ray::EnvWorker' | grep -v grep | head -30
```

查看评测视频（`env.eval.video_cfg.save_video=True`，每次评测后按 seed 分目录写入；实测 100 步共 8 个 `seed_*` 目录、约 63M）：

```bash
ls -lt /root/workspace/logs/<run-dir>/video/eval | head
```

训练结束判据（三项同时满足）：tmux 会话已退出、日志无 `Traceback`、checkpoint 目录出现最后一步。实测目录结构：

```text
/root/workspace/logs/<run-dir>/
├── run_embodiment.log
├── metrics.log
├── tensorboard/
├── video/eval/seed_*/
└── maniskill_ppo_co_training_openpi_pi05/checkpoints/
    ├── global_step_50/actor/{dcp_checkpoint,model_state_dict}  # 约 16G
    └── global_step_100/                                        # 约 16G
```

步骤六：启动 TensorBoard 和 Ray Dashboard（可选）

启动 TensorBoard：

```bash
cd /root/workspace/RLinf
tmux new-session -d -s tensorboard_rlinf \
  'source switch_env openpi && tensorboard --logdir /root/workspace/logs --port 6006 --host 127.0.0.1'
```

本地访问：

```bash
ssh -p <port> -L 6006:127.0.0.1:6006 root@<host>
```

浏览器打开：

```text
http://127.0.0.1:6006
```

Ray Dashboard 远端默认监听：

```text
127.0.0.1:8265
```

本地访问：

```bash
ssh -p <port> -L 18265:127.0.0.1:8265 root@<host>
```

浏览器打开：

```text
http://127.0.0.1:18265
```

远端验证 Ray Dashboard：

```bash
curl -s http://127.0.0.1:8265/api/version
```

步骤七：解析 TensorBoard 指标

使用脚本读取 TensorBoard scalar：

```bash
cd /root/workspace/RLinf
source switch_env openpi
export TF_CPP_MIN_LOG_LEVEL=3

python - <<'PY'
from pathlib import Path
from tensorboard.backend.event_processing.event_accumulator import EventAccumulator
run = Path('/root/workspace/logs/<run-dir>')
for event in sorted((run / 'tensorboard').glob('events.out.tfevents.*')):
    print(f'EVENT {event.name} size={event.stat().st_size}')
    ea = EventAccumulator(str(event), size_guidance={'scalars': 0})
    ea.Reload()
    for tag in sorted(ea.Tags().get('scalars', [])):
        vals = ea.Scalars(tag)
        if vals:
            v = vals[-1]
            print(f'{tag}\tcount={len(vals)}\tstep={v.step}\tvalue={v.value}')
PY
```

重点关注：

- `env/success_once`：训练 rollout 的仿真成功率，官方示例从约 0.35 上升到约 0.50（百舸实测 0.359 → 0.523）
- `eval/success_once` / `eval/success_at_end`：每 10 步评测的仿真成功率（百舸实测 0.50 → 0.75 / 0.344 → 0.578）
- `env/return`
- `train/ppo_loss`：PPO（RL）损失
- `train/sft_loss`：真机数据上的 SFT 损失
- `train/loss_ratio`：SFT 损失与 PPO 损失的量级比
- `train/actor/total_loss`：`L_PPO + β · L_SFT`（已除以梯度累积步数）
- `train/actor/approx_kl`
- `train/actor/grad_norm`
- `time/actor/run_training`

协同训练相关指标的判读：

指标	正常表现	异常信号与处理
`train/sft_loss`	在 SFT 权重基础上维持较低水平，随训练小幅波动	持续上升说明策略在偏离真机分布，可适当提高 `sft_loss_weight`
`train/ppo_loss`	绝对值较小，随 rollout 波动	长期接近 0 会使 `loss_ratio` 剧烈放大
`train/loss_ratio`	量级稳定（实测常态 0.1–2.7，偶发单步尖峰后即回落）	持续多步高位，或大于 `1e5` 时日志打印 `SFT/PPO loss imbalance detected`，应降低 `sft_loss_weight`
`env/success_once`	前 100 步呈上升趋势	长期不升可做 `enable_sft_co_train=False` 对照，排除 SFT 约束过强

> 说明：当前代码中 `train/loss_ratio` 按 `|L_SFT| / |L_PPO|` 计算，未乘 β；官方文档公式为 `β·|L_SFT| / |L_PPO|`。判读量级时以实际日志为准。

百舸实测关键指标（repro100，step 99 终值）

指标	值
`eval/success_once`	0.75（第 10 步 0.50）
`eval/success_at_end`	0.5781（第 10 步 0.3438）
`env/success_once`	0.5234（step 0 为 0.3594；最后 10 步均值 0.516，最高 0.5703）
`env/success_at_end`	0.375（step 0 为 0.2656）
`env/return`	0.6188（step 0 为 0.4359）
`env/num_trajectories` / `env/episode_len`	128 / 80
`train/sft_loss`	0.01245（step 0 为 0.0070，最高 0.0156）
`train/ppo_loss`	0.0224（step 0 为 0.7557）
`train/loss_ratio`	0.287（常态 0.1–2.7；step 69 单步 18.8、step 85 单步 4.0）
`train/actor/approx_kl`	0.0067（全程 0.005–0.011）
`train/actor/clip_fraction`	0.0018
`train/actor/grad_norm`	2.735（step 0 为 49.9，前 10 步均值 11.6，最后 10 步均值 3.8）
`train/critic/explained_variance`	0.636（step 0 为 -8.5）
`train/critic/value_loss`	0.0204（step 0 为 0.758）
`SFT/PPO loss imbalance detected` 告警	0 次

各评测点（每次 64 条轨迹）：

步数	10	20	30	40	50	60	70	80	90	100
`eval/success_once`	0.500	0.500	0.563	0.563	0.609	0.688	0.688	0.688	0.719	0.750
`eval/success_at_end`	0.344	0.422	0.438	0.391	0.469	0.469	0.531	0.516	0.578	0.578

`success_once` 与 `success_at_end` 的区别：本任务配置 `ignore_terminations: True`、`auto_reset: False`，每条轨迹固定跑满 80 步。`success_once` 统计 80 步内任一时刻满足成功条件（胡萝卜 xy 落在盘子范围内、与盘子接触且接触力超过阈值），`success_at_end` 只看最后一步是否仍满足。两者差值约 0.16–0.17，即 64 条中约 10–11 条曾放上盘子但随后滑落或被碰开。`success_at_end` 更接近真机部署关心的"放稳"效果。

性能参考（8×B30Z，repro100）

项	实测值
总墙钟	4.71 小时（00:00:34 – 04:46:05）
step 0	277s（含首次加载）
普通步	平均 159.9s（155.7–164.6s）：`generate_rollouts` 136.5s + `actor_training` 21.7s + `sync_weights` 1.7s
评测步（第 10/20/…/90 步）	233.6–240s，其中评测 75.6s
评测 + 存盘步（第 50、100 步）	280s / 274.5s
显存	约 31G/卡
checkpoint	每个约 16G，100 步共约 32G
完整训练外推	1000 步约 47 小时，20 个 checkpoint 约 320G

rollout（ManiSkill GPU 仿真 + π0.5 推理）占普通步约 85%，actor 训练只占约 14%；换机型或调整 `total_num_envs` 时优先关注 rollout 耗时。

常见问题处理

问题	**解决方案**
模型或数据路径仍是 `/path/to/...`	启动命令必须追加 `rollout.model.model_path`、`actor.model.model_path`、`actor.sft_data_path` 三个 override
`actor.sft_data_path must be set ...`	`enable_sft_co_train=True` 时必须设置 `actor.sft_data_path`
加载真机数据时报 `meta/info.json` 不存在，或尝试从 HuggingFace 拉取 `meta/`	`sft_data_path` 指到了下载目录顶层，应改为 `.../RLCo-Example-Real-Data/physical-intelligence/pick_and_place_real`
加载 norm stats 失败	检查 SFT 权重目录下是否存在 `physical-intelligence/pick_and_place_real/norm_stats.json`；自训阶段一权重需按同样结构放置
找不到 `bridge_table_long.glb` 或 `carrot` 资产	确认资产位于 `rlinf/envs/sim/maniskill/assets`，或设置 `MANISKILL_ASSET_DIR` 指向资产目录；`custom_assets` 需单独从 `RLinf/RLCo-maniskill-assets` 下载（该仓库不含 `carrot`，`carrot` 从 BOS 或 `RLinf/maniskill_assets` 获取）
`xxx is not divisible by yyy` 断言失败	修改了 `total_num_envs`、`rollout_epoch` 或卡数后，需同步调整 `global_batch_size`，并保证能被 `micro_batch_size × GPU 数` 整除
`save_interval must be divisible by val_check_interval`	两者需整除，或将 `val_check_interval` 设为 `-1` 关闭评测
日志出现 `SFT/PPO loss imbalance detected`	PPO 损失过小或 SFT 损失过大，降低 `sft_loss_weight`（如 `0.1`），并观察 `train/loss_ratio` 是否回落
Ray worker 被 memory killer 杀	检查 `/sys/fs/cgroup/memory.max` 和 `/sys/fs/cgroup/memory.events`，调大容器内存，或降低 `env.train.total_num_envs` 并同步调整 batch
rollout 阶段显存 OOM	可开启 `actor.enable_offload=True rollout.enable_offload=True`，或降低 `actor.micro_batch_size`（需保持整除关系）
主日志长时间无输出	先看 TensorBoard、GPU、worker CPU 与 Ray Dashboard，不要只凭主日志判断停止；必要时抓栈保留现场
BOS 目录只列出少量小文件	上传可能尚未完成，或未带 `-a` 被截断在 1000 条；用 `bcecmd bos ls -a -r <目录/>` 查看完整列表后再下载
`bcecmd bos du` 报不支持	`bcecmd` 无 `du` 子命令，用 `bcecmd bos ls -a -r` 输出的大小列累加
`bcecmd` 显示 `[N] objects downloaded` 但训练加载失败	该提示不能证明文件完整，按步骤二的 `rec` 函数逐文件对账大小，有 `BAD` 行则重新下载对应文件
资产加载报 `model_db.json` 不存在	`carrot/` 下 `more_carrot`、`more_plate`、`more_table` 三个子目录都需要各自的 `model_db.json`，只拷贝部分子目录会失败
每 10 步出现一次耗时约 235s 的长步	正常，为评测步（评测约 76s）；第 50、100 步再叠加约 40s 存盘，约 275–280s
`train/loss_ratio` 单步跳到 10 以上	实测 step 69 为 18.8、step 85 为 4.0，均在下一步回落；只有持续多步高位或出现 imbalance 告警才需降低 `sft_loss_weight`
`eval/success_at_end` 明显低于 `eval/success_once`	正常，前者要求最后一步仍放稳，实测差值约 0.16–0.17；两者同步上升即说明训练有效

RLinf v0.4 在Libero上进行 Cosmos3 监督微调

概述

[Rlinf](https://rlinf.readthedocs.io/zh-cn/latest/) 是一个面向 **Embodied AI（具身智能）** 和 **Agentic AI（智能体 AI）** 的开源强化学习基础框架。

[Cosmos3-Nano](https://huggingface.co/nvidia/Cosmos3-Nano) 是 NVIDIA 发布的 OmniMoT 视频世界模型。RLinf v0.4 支持以 Cosmos3-Nano 为基座，按照 LIBERO 的动作空间做监督微调（SFT），得到一个可以直接在 LIBERO 仿真器中评测的 Cosmos3 策略模型。训练从基座热启动，冻结理解塔，只训练生成分支与动作头。

通过百舸快速开始，您可以基于预置镜像创建开发机，完成「确认环境 → 准备基座模型与数据 → 启动 SFT 训练 → 转换 checkpoint」的完整流程，并通过 TensorBoard、训练日志和 GPU 指标监控训练进展。转换得到的 `model_diffusers` 目录可直接用于 Cosmos3 SGLang 评测。

适用场景

- 以 Cosmos3-Nano 为基座，在 LIBERO 全部 task suite（`libero_all`）或单独 `libero_10` 上做动作头 SFT。
- 把 diffusers 格式的 Cosmos3-Nano 转成训练所需的 DCP 格式基座权重。
- 把 SFT 产出的 FSDP2 checkpoint 转成 diffusers 组件目录，交给 SGLang 做 LIBERO 仿真评测。
- 在单机开发机上排查初始化 OOM、动作头 shape 不匹配、HuggingFace 拉取卡死、动作归一化错位等问题。
- 不适合：Cosmos3 的强化学习训练、非 LIBERO 数据集。当前 RLinf 中 Cosmos3 的数据加载只支持 LIBERO。

训练规格参考 & 环境配置

推荐训练配置如下：

项目	**推荐配置**
GPU	单机 8 卡，官方验证为 8 × A800 80GB；实测 8 × 183GB 卡上默认配置每卡约 76–79GB
CPU	建议 64 核以上（数据加载默认每 rank 4 个 worker）
容器内存	建议 1TiB 以上。初始化时模型先留在 CPU（`net` bf16 约 27GB + `net_ema` fp32 约 54GB），再由 FSDP2 分片上卡，8 个 rank 同时构建时内存峰值较高
存储空间	训练前资产约 90GB（HF 缓存 ~22GB、Cosmos3-Nano 原始权重 ~33GB、DCP 基座 ~29GB、LIBERO 数据 ~4GB）；每个 checkpoint 约 196GB，默认 5000 步存 10 份约 2TB；每次 checkpoint 转换的产物约 230GB

项目目录配置如下表所示：

项目	**推荐配置**
CUDA / 驱动	使用镜像自带的 CUDA（torch 2.11 + CUDA 13.0）与平台默认驱动，不要单独升级
工作目录	`/root/workspace/RLinf`（镜像预置）
Python 环境	`/root/workspace/RLinf/.venv`（镜像预置，cosmos3 + libero 依赖）
cosmos-framework	`/root/workspace/cosmos-framework`（镜像预置，commit `0460be81`，已 editable 装入上面的 venv）
训练入口	`/root/workspace/RLinf/examples/sft/train_vla_sft.py`
训练配置	`/root/workspace/RLinf/examples/sft/config/libero_sft_cosmos3.yaml`
模型目录	`/root/workspace/models`
数据目录	`/root/workspace/datasets/LIBERO_LeRobot_v3`
HF 缓存	`/root/workspace/hf-cache`
日志目录	`/root/workspace/logs`

训练产出与结果口径

训练完成后会留下以下产物：

产物	**路径**
主训练日志	`<LOG_DIR>/run_embodiment.log`
TensorBoard event	`<LOG_DIR>/tensorboard/`
SFT checkpoint（续训）	`<LOG_DIR>/libero_sft_cosmos3/checkpoints/global_step_<N>/actor/dcp_checkpoint/`（约 111GB）
SFT checkpoint（转换输入）	`<LOG_DIR>/libero_sft_cosmos3/checkpoints/global_step_<N>/actor/model_state_dict/full_weights.pt`（约 85GB）
评测用模型	转换后的 `<OUT>/model_diffusers`（约 30GB）

其中 `<LOG_DIR>` 在启动命令里通过 `runner.logger.log_path` 指定，`libero_sft_cosmos3` 是 `runner.logger.experiment_name`。checkpoint 每 `runner.save_interval` 步（默认 500）保存一次，每份约 196GB。

官方文档未给出 SFT 后的 LIBERO 成功率参考值，本文也不预置。成功率需要把 checkpoint 转换后，按 Cosmos3 SGLang 评测流程跑出来，以评测日志中的 `eval/success_once` 为准。

准备工作

如果您所在网络环境访问 HuggingFace 较慢，建议提前配置 HuggingFace 镜像源，或配置[代理加速](https://cloud.baidu.com/doc/BCM/s/3mmswg2oh)。

huggingface 镜像配置：

```bash
export HF_ENDPOINT=https://hf-mirror.com
```

```
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
```

hf-mirror 只适用于模型仓库：对 datasets 接口会返回 308 重定向，下载 LIBERO 数据集时需要 `unset HF_ENDPOINT` 直连 huggingface.co（见步骤四）。

百舸平台已提供以下模型的 BOS 地址（目录路径必须带结尾 `/`），可以直接从 BOS 拉取，避免依赖外部网络。LIBERO LeRobot 数据集目前没有 BOS 副本。

```text
bos:/aihc-models-bj/nvidia/Cosmos3-Nano/
bos:/aihc-models-bj/Qwen/Qwen3-VL-8B-Instruct/
bos:/aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B/
bos:/aihc-models-bj/Wan-AI/Wan2.2-TI2V-5B-Diffusers/
```

环境说明

> 说明：本文中的路径均以百舸开发机内的 `/root/workspace` 为例。`/root/workspace` 常是 100GB 左右的盘，放不下 checkpoint；建议用 `df -h` 找一块大容量挂载（如 `/mnt/pfs`），把下文的 `models/`、`datasets/`、`hf-cache/`、`logs/` 统一换到该目录下，`RLinf` 代码与 `.venv` 可以留在原处。

当前使用以下目录：

```text
/root/workspace/RLinf
/root/workspace/RLinf/.venv
/root/workspace/cosmos-framework
/root/workspace/hf-cli-venv
/root/workspace/models/Cosmos3-Nano
/root/workspace/models/Cosmos3-Nano-DCP
/root/workspace/datasets/LIBERO_LeRobot_v3
/root/workspace/hf-cache
/root/workspace/logs
```

步骤一：确认预置环境

创建开发机时选择推荐镜像 `ccr-registry.baidubce.com/aihc/rlinf:agentic-rlinf0.4-maniskill_libero_cosmos_torch211_cuda130_20261021`。镜像已预置 RLinf 代码、cosmos3 + libero 的 venv 和 cosmos-framework，**无需执行 `git clone` 或 `install.sh`**，本步骤只做检查。检查失败说明镜像或 shell 环境不对，不要在 venv 里自行 `pip install` 升降级。

> 说明：镜像登录 shell 默认激活的是 `/opt/venv/openvla`，不是本文所用的环境。后续每个 shell 和启动脚本都要先 `source /root/workspace/RLinf/.venv/bin/activate`。

```bash
unset PYTHONPATH
source /root/workspace/RLinf/.venv/bin/activate
cd /root/workspace/RLinf
#### 先确认 LIBERO 配置存在，否则下面 import libero 会交互询问路径
test -f /root/.libero/config.yaml && echo "libero config OK" || echo "libero config MISSING"

which python; python --version
python - <<'PY'
import importlib, typing
for name in ["torch", "transformers", "diffusers", "lerobot", "ray", "rlinf", "cosmos_framework", "libero"]:
    try:
        m = importlib.import_module(name)
        print(f"OK   {name:18s} {getattr(m, '__version__', '-')}")
    except Exception as e:
        print(f"FAIL {name:18s} {type(e).__name__}: {e}")
import torch
print("cuda", torch.cuda.is_available(), "devices", torch.cuda.device_count())
print("typing.override", hasattr(typing, "override"))
from diffusers import Cosmos3OmniTransformer  # noqa: F401
print("Cosmos3OmniTransformer OK")
PY
python -c 'import cosmos_framework, os; print(os.path.dirname(os.path.dirname(cosmos_framework.__file__)))'
git -C /root/workspace/cosmos-framework rev-parse HEAD
command -v uv hf
nvidia-smi --query-gpu=index,name,driver_version,memory.total --format=csv,noheader
```

判定标准：

检查项	**期望**	**不满足时**
各包 import	全部 `OK`	当前 shell 不是 `/root/workspace/RLinf/.venv`，重新 `source`；仍失败说明镜像不对
CUDA	`cuda True devices 8`	检查开发机 GPU 规格与驱动
`typing.override`	`True`	见下文 `sitecustomize.py` 说明
`Cosmos3OmniTransformer`	`OK`	训练不受影响；步骤八转换需要它
cosmos-framework 位置与 commit	`/root/workspace/cosmos-framework`，`0460be81f16883aa380e716dc6f58c1189481172`	commit 不一致时，训练用的 experiment 与统计文件可能不同，以 RLinf 验证过的版本为准
`uv`、`hf`	均有路径	缺 `uv` 时 cosmos `hf_cli` 无法启动；缺 `hf` 只影响 HF 下载，可改走 BOS
`/root/.libero/config.yaml`	存在	首次 import LIBERO 会交互询问路径，在 tmux 中会卡住

venv 是 Python 3.11，镜像在 site-packages 里放了一个 `sitecustomize.py`，补齐 3.12 才有的 `typing.override`，否则 `cosmos_framework` 无法导入。cosmos-framework 仓库根目录也有一个同名的 `sitecustomize.py`，把该目录加进 `PYTHONPATH` 会遮蔽 venv 里的那个，导致 `cannot import name 'override' from 'typing'`。**不要把 cosmos-framework 根目录加进 `PYTHONPATH`**，它已经以 editable 方式装进 venv，可以直接导入。

cosmos-framework 的 `hf_cli` 会以 `uv run --locked` 启动一个独立环境去读 HF 资产。百舸开发机全局设置了 `UV_DEFAULT_INDEX`（内部 PyPI 源），它与 cosmos 自带 `uv.lock` 的 registry 不一致，会报 `uv.lock needs to be updated, but --locked`。在后续所有步骤的 shell 中执行：

```bash
unset UV_DEFAULT_INDEX
#### 复用一个持久的 hf_cli 环境；首次使用时由 uv 自动创建（约 29MB），之后直接复用
export IMAGINAIRE_HF_CLI_ENVIRONMENT=/root/workspace/hf-cli-venv
```

步骤二：离线缓存 Qwen3-VL-8B-Instruct 和 Wan2.2 模型

SFT 与基座转换会从 HuggingFace 拉取三个模型资产。建议先全部下进 HF 本地缓存，再开离线模式（`HF_HUB_OFFLINE=1`），让 `from_pretrained` / `hf download` 直接读缓存，否则 worker 可能在网络重试处长时间卡住：

模型	**用途**
`Qwen/Qwen3-VL-8B-Instruct`（revision `0c351dd01ed87e9c1b53cbc748cba10e6187ff3b`）	tokenizer / VLM 配置；SFT 启动与 `convert_model_to_dcp` 重建模型时都会用到
`Wan-AI/Wan2.2-TI2V-5B`（`Wan2.2_VAE.pth`，revision `921dbaf3f1674a56f47e83fb80a34bac8a8f203e`）	Wan2.2 VAE；`convert_model_to_dcp` 重建模型时读取，SFT 通过 `wan_vae_path` 引用
`Wan-AI/Wan2.2-TI2V-5B-Diffusers`（`vae/*`）	diffusers 格式 VAE；基座转换与 checkpoint 转换时使用

cosmos 的 `hf_cli` 按上表的**固定 revision** 解析 Qwen 与 Wan VAE。离线时缓存里必须存在以该 commit 命名的 snapshot 目录，`refs/main` 指向其他 commit 也找不到。

方式一：通过 HuggingFace 下载

直接下载进 HF 缓存（缓存布局自动正确），建议放入 tmux：

```bash
mkdir -p /root/workspace/hf-cache /root/workspace/logs
SESSION=hf_cache_download
LOG=/root/workspace/logs/hf_cache_download.log

cat > /tmp/download_hf_cache.sh <<'EOF'
#### !/usr/bin/env bash
set -euo pipefail
source /root/workspace/RLinf/.venv/bin/activate
export HF_HOME=/root/workspace/hf-cache
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
{
  date
  hf download Qwen/Qwen3-VL-8B-Instruct \
    --revision 0c351dd01ed87e9c1b53cbc748cba10e6187ff3b
  hf download Wan-AI/Wan2.2-TI2V-5B Wan2.2_VAE.pth \
    --revision 921dbaf3f1674a56f47e83fb80a34bac8a8f203e
  hf download Wan-AI/Wan2.2-TI2V-5B-Diffusers --include "vae/*"
  echo "download done"
  date
} 2>&1 | tee -a /root/workspace/logs/hf_cache_download.log
EOF
chmod +x /tmp/download_hf_cache.sh

tmux new-session -d -s "$SESSION" /tmp/download_hf_cache.sh
```

`hf download` 的文件名参数必须紧跟仓库名，`--revision` 放在后面；顺序写反时会被当成另一个仓库，报找不到。

方式二：通过 BOS 下载

从 BOS 拷贝时，必须手工摆成 HF 缓存布局，否则离线解析不到：Qwen 与 Wan VAE 放进固定 revision 的 snapshot 目录；Wan Diffusers 只需 `model_index.json` 与 `vae/`，snapshot 用占位 commit；每个仓库都写 `refs/main`。

```bash
mkdir -p /root/workspace/logs
cat > /tmp/download_hf_cache_bos.sh <<'EOF'
#### !/usr/bin/env bash
set -euo pipefail
HUB=/root/workspace/hf-cache/hub
QWEN_REV=0c351dd01ed87e9c1b53cbc748cba10e6187ff3b
WAN_REV=921dbaf3f1674a56f47e83fb80a34bac8a8f203e
FAKE_REV=0000000000000000000000000000000000000000
QWEN_SNAP=$HUB/models--Qwen--Qwen3-VL-8B-Instruct/snapshots/$QWEN_REV
WAN_SNAP=$HUB/models--Wan-AI--Wan2.2-TI2V-5B/snapshots/$WAN_REV
WAND_SNAP=$HUB/models--Wan-AI--Wan2.2-TI2V-5B-Diffusers/snapshots/$FAKE_REV
mkdir -p "$QWEN_SNAP" "$WAN_SNAP" "$WAND_SNAP/vae"
for d in Qwen--Qwen3-VL-8B-Instruct:$QWEN_REV Wan-AI--Wan2.2-TI2V-5B:$WAN_REV Wan-AI--Wan2.2-TI2V-5B-Diffusers:$FAKE_REV; do
  mkdir -p "$HUB/models--${d%%:*}/refs"
  echo -n "${d##*:}" > "$HUB/models--${d%%:*}/refs/main"
done
B=bos:/aihc-models-bj
{
  date
  bcecmd bos cp -r $B/Qwen/Qwen3-VL-8B-Instruct/ "$QWEN_SNAP/"
  bcecmd bos cp $B/Wan-AI/Wan2.2-TI2V-5B/Wan2.2_VAE.pth "$WAN_SNAP/Wan2.2_VAE.pth"
  bcecmd bos cp $B/Wan-AI/Wan2.2-TI2V-5B-Diffusers/model_index.json "$WAND_SNAP/model_index.json"
  bcecmd bos cp -r $B/Wan-AI/Wan2.2-TI2V-5B-Diffusers/vae/ "$WAND_SNAP/vae/"
  echo "download done"
  date
} 2>&1 | tee -a /root/workspace/logs/hf_cache_download.log
EOF
chmod +x /tmp/download_hf_cache_bos.sh

tmux new-session -d -s hf_cache_download /tmp/download_hf_cache_bos.sh
```

如果 `bcecmd` 报 `Object don't exist`，先 `bcecmd bos ls bos:/aihc-models-bj/<上级目录>/` 确认真实路径与结尾 `/`。

查看下载进度：

```bash
tail -50 /root/workspace/logs/hf_cache_download.log
du -sh /root/workspace/hf-cache/hub/* 2>/dev/null || true
```

验证缓存

下载结束（日志出现 `download done`）后，在离线模式下验证三项资产能被解析：

```bash
source /root/workspace/RLinf/.venv/bin/activate
export HF_HOME=/root/workspace/hf-cache
export HF_HUB_OFFLINE=1
hf download Qwen/Qwen3-VL-8B-Instruct --revision 0c351dd01ed87e9c1b53cbc748cba10e6187ff3b >/dev/null && echo "OK Qwen"
hf download Wan-AI/Wan2.2-TI2V-5B Wan2.2_VAE.pth --revision 921dbaf3f1674a56f47e83fb80a34bac8a8f203e >/dev/null && echo "OK Wan VAE"
hf download Wan-AI/Wan2.2-TI2V-5B-Diffusers --include "vae/*" >/dev/null && echo "OK Wan Diffusers"
```

```

三行都输出 `OK` 再继续；缺哪一项，只对该项改用另一种方式补齐。

`Wan2.2_VAE.pth` 固定了 revision，因此它在缓存中的路径是确定的，后面训练配置里的 `wan_vae_path` 直接指向这个文件：

```bash
WAN_VAE=/root/workspace/hf-cache/hub/models--Wan-AI--Wan2.2-TI2V-5B/snapshots/921dbaf3f1674a56f47e83fb80a34bac8a8f203e/Wan2.2_VAE.pth
ls -lh "$WAN_VAE"
```

如果 `ls` 找不到文件，用 `find /root/workspace/hf-cache -name Wan2.2_VAE.pth` 确认实际路径。

步骤三：准备基座 DCP 权重

Cosmos3 SFT **从基座模型 Cosmos3-Nano 热启动，只训练动作头及生成分支**。`actor.model.model_path` 必须指向一个 **DCP 格式** 的基座权重目录。

3.1 下载 Cosmos3-Nano（diffusers 格式）

HuggingFace 上的 `nvidia/Cosmos3-Nano` 是 **diffusers/safetensors 格式**（`model_index.json` + `transformer/*.safetensors` 分片 + `vae/` + `text_tokenizer/` 等），不是 DCP，SFT 不能直接加载。基座下载到普通目录即可：

```bash
source /root/workspace/RLinf/.venv/bin/activate
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
mkdir -p /root/workspace/models

hf download nvidia/Cosmos3-Nano \
  --local-dir /root/workspace/models/Cosmos3-Nano
```

也可以从 BOS 拉取（约 33GB）：

```bash
bcecmd bos cp -r bos:/aihc-models-bj/nvidia/Cosmos3-Nano/ /root/workspace/models/Cosmos3-Nano/
```

验证目录：

```bash
ls /root/workspace/models/Cosmos3-Nano
ls /root/workspace/models/Cosmos3-Nano/transformer | head
```

3.2 转成 DCP

为什么必须转换：cosmos3 的基座加载器 `_load_base_weights` 通过 `_is_safetensors_checkpoint` 判断格式，它只检查路径**顶层**是否有 `*.safetensors` 文件（非递归）。diffusers 目录把分片放在 `transformer/` 子目录，顶层没有，于是被当成 DCP 去加载，而它又不是 DCP，加载失败。

本步会重建整个 OmniMoTModel，期间经 cosmos 的 `hf_cli` 读取 Qwen3-VL-8B-Instruct（tokenizer）和 Wan2.2 VAE。步骤二已经把它们下进缓存，这里开离线模式运行：

```bash
source /root/workspace/RLinf/.venv/bin/activate   # 必须是训练用的同一个 venv
export HF_HOME=/root/workspace/hf-cache
export HF_HUB_OFFLINE=1
export TRANSFORMERS_OFFLINE=1
unset UV_DEFAULT_INDEX
export IMAGINAIRE_HF_CLI_ENVIRONMENT=/root/workspace/hf-cli-venv

python -m cosmos_framework.scripts.convert_model_to_dcp \
  --checkpoint-path /root/workspace/models/Cosmos3-Nano \
  --no-use-ema-weights \
  -o /root/workspace/models/Cosmos3-Nano-DCP 2>&1 | tee /root/workspace/logs/convert_base_dcp.log
```

注意事项：

- `--checkpoint-path` 既接受模型名（触发 HF 下载），也接受本地 diffusers 目录。离线机器请指向本地路径。
- DCP 的 `.metadata` 是 pickle 序列化的，**与保存时的 Python 版本绑定**。`convert_model_to_dcp` 必须在与训练**相同的 venv** 里运行，产出的 DCP 才能在训练时加载。可以在目录名上带 Python 版本后缀（如 `Cosmos3-Nano-DCP-py311`）以免混用。
- 报 `uv.lock needs to be updated, but --locked` 时，确认已 `unset UV_DEFAULT_INDEX`（见步骤一）。
- 报 `LocalEntryNotFoundError`、`OfflineModelsEnabled` 或 tokenizer `vocab_file` 为 None，说明 HF 缓存缺资产或 snapshot 名不是固定 revision，回到步骤二的「验证缓存」。

验证 DCP 目录：

```bash
ls -la /root/workspace/models/Cosmos3-Nano-DCP | head
find /root/workspace/models/Cosmos3-Nano-DCP -maxdepth 2 -name .metadata
du -sh /root/workspace/models/Cosmos3-Nano-DCP
```

能找到 `.metadata` 且目录下有 `*.distcp` 分片（实测约 29GB），说明转换成功。如果 `.metadata` 不在顶层而在某个子目录中，`model_path` 应指向包含 `.metadata` 的那一级目录。

步骤四：准备 LIBERO 数据

训练数据使用**原始** LeRobot v3 布局（含 `meta/`、`data/`）的 LIBERO 数据集，通过 `data.train_data_paths` 指定。可从 HuggingFace 数据集 `nvidia/LIBERO_LeRobot_v3`（20 FPS，与 cosmos-framework 自带的归一化统计对应）下载。hf-mirror 对 datasets 接口返回 308，下载时需要直连 huggingface.co：

```bash
source /root/workspace/RLinf/.venv/bin/activate
unset HF_ENDPOINT
mkdir -p /root/workspace/datasets/LIBERO_LeRobot_v3
tmux new-session -d -s libero_data \
  'hf download nvidia/LIBERO_LeRobot_v3 --repo-type dataset \
     --local-dir /root/workspace/datasets/LIBERO_LeRobot_v3 \
     2>&1 | tee -a /root/workspace/logs/libero_data_download.log'
```

数据集还包含 `libero_90`、`libero_all`，训练只用 spatial / object / goal / 10 四个 suite；四个目录齐全后即可开始训练。`libero_90` 可以继续在后台下载，连接中断时重跑同一条命令，已完成的文件会跳过
```

数据集建包含 `libero_90`、`libero_all` 等级别的 spatial / object / goal / 10 四个 suite，四个目录齐全后即可开始训练；`libero_90` 可以通过在后台下载。建议下载完成后跑一遍命令，已完成的文件会跳过。

`data.data_type` 决定使用哪个 cosmos-framework experiment：

data_type	**cosmos experiment**	**说明**
`libero_all`	`action_policy_libero_all_nano`	LIBERO 全部 task suite，默认配置
`libero_10`	`action_policy_libero_nano`	只训练 `libero_10`

`data_type` 不填时会按目录结构自动判断：`train_data_paths` 下同时存在 `libero_spatial`、`libero_object`、`libero_goal`、`libero_10` 四个子目录时识别为 `libero_all`，否则按 `libero_10` 处理。建议在配置中显式写明，避免目录不全时被静默降级成 `libero_10`。

`libero_all` 的推荐目录布局：

```text
/root/workspace/datasets/LIBERO_LeRobot_v3
├── libero_spatial/   # 每个子目录都是 LeRobot v3 布局
│   ├── meta/
│   └── data/
├── libero_object/
├── libero_goal/
└── libero_10/
```

检查数据目录：

```bash
DATA=/root/workspace/datasets/LIBERO_LeRobot_v3
for SUITE in libero_spatial libero_object libero_goal libero_10; do
  echo "===== $SUITE ====="
  ls "$DATA/$SUITE" 2>/dev/null || echo "MISSING: $DATA/$SUITE"
done
```

动作转换**不需要预处理数据集**。`rlinf/data/datasets/cosmos3/dataloader.py` 调用 cosmos-framework 的 experiment，加载时在线把原始 7 维动作按 `frame_wise_relative` + `rot6d` 转成 10 维，并做 `quantile_rot` 归一化。dataloader 会把 `LIBERO_ROOT` 环境变量设为 `train_data_paths`，训练时无需手动导出。

归一化统计文件 `libero_native_frame_wise_relative_rot6d.json` 由 cosmos-framework 预先算好，评测侧通过 `action_stats_path` 引用。**训练与评测必须使用同一套 recipe 和同一个 stats 文件**，否则去归一化会错位，表现为动作混乱。

步骤五：确认训练配置

`examples/sft/config/libero_sft_cosmos3.yaml` 的核心配置如下：

```yaml
defaults:
  - hybrid_engines/fsdp@actor.fsdp_config
  - model/cosmos3@actor.model        # 引入 examples/sft/config/model/cosmos3.yaml

cluster:
  num_nodes: 1
  component_placement:
    actor: all

runner:
  task_type: sft
  max_steps: 5000
  save_interval: 500
  log_interval: 50
  resume_dir: null

data:
  train_data_paths: path/to/LIBERO_LeRobot_v3
  data_type: "libero_all"
  num_workers: 4
  prefetch_factor: 4
  val_ratio: 0.01

actor:
  training_backend: "fsdp"
  micro_batch_size: 32
  global_batch_size: 2048
  model:
    precision: bf16
    load_to_device: false
    ema_enabled: true
    model_path: path/to/Cosmos3-Nano-DCP
    wan_vae_path: path/to/Wan2.2-TI2V-5B/Wan2.2_VAE.pth
    max_samples_per_batch: 32
  optim:
    lr: 5.0e-05
    lr_scheduler: "lambda_linear"
    lr_warmup_steps: 500
    clip_grad: 1.0
    weight_decay: 0.05
  fsdp_config:
    strategy: "fsdp2"
    gradient_checkpointing: False
    save_full_model_weights: True
```

原始 YAML 中三个路径仍是占位符：

```text
data.train_data_paths: path/to/LIBERO_LeRobot_v3
actor.model.model_path: path/to/Cosmos3-Nano-DCP
actor.model.wan_vae_path: path/to/Wan2.2-TI2V-5B/Wan2.2_VAE.pth
```

实际启动时通过 Hydra CLI override 注入真实路径：

```bash
data.train_data_paths=/root/workspace/datasets/LIBERO_LeRobot_v3
actor.model.model_path=/root/workspace/models/Cosmos3-Nano-DCP
actor.model.wan_vae_path=${WAN_VAE}
```

官方入口 `examples/sft/run_vla_sft.sh` 只接收配置名一个参数，不透传额外的 Hydra override，因此本文使用直接 Python 入口启动。如果坚持用 `run_vla_sft.sh`，需要先把上面三个路径改进 YAML。

关键字段说明

大多数字段照抄 `libero_sft_cosmos3.yaml` 即可，真正需要关注的如下：

字段	**说明**
`actor.model.model_path`	Cosmos3-Nano 基座权重目录（DCP）
`actor.model.wan_vae_path`	Wan2.2 VAE 权重文件（`Wan2.2_VAE.pth`）
`actor.model.load_to_device`	**必须为 `false`**。`net`（bf16 约 27GB）+ `net_ema`（fp32 约 54GB）≈ 81GB，若先 `model.to(device)`，单张 80GB 卡在 FSDP2 `fully_shard` 分片之前就会 OOM
`actor.model.ema_enabled`	Cosmos3 power-EMA（rate=0.1），每个优化步更新 `net_ema`，占一份 fp32 网络显存；仍 OOM 时可临时设 `false`
`actor.model.max_samples_per_batch`	cosmos PackingDataLoader 每个 packed batch 的样本数（recipe 默认 128，RLinf 配置为 32）
`actor.micro_batch_size` / `global_batch_size`	每卡微批与全局批，默认 32 / 2048
`actor.optim.lr` / `lr_warmup_steps`	学习率与 warmup，默认 5e-5 / 500。动作头另有 5 倍 LR 倍率（`model/cosmos3.yaml` 的 `lr_multipliers`）
`runner.max_steps` / `save_interval`	训练步数与存档间隔，默认 5000 / 500
`runner.resume_dir`	断点续训时指向某个 `global_step_<N>` 目录

`model/cosmos3.yaml` 中与热启动相关的默认值（一般无需修改）：

- `keys_to_skip_loading`：跳过 `net_ema.` 和基座自带的 DROID 8 维动作头（`action2llm`、`llm2action`、`action_modality_embed`、`action_pos_embed`），这些参数重新初始化，以适配 LIBERO 的 10 维动作。
- `keys_to_select`：只训练 `moe_gen`、`time_embedder`、`vae2llm`、`llm2vae` 和动作头，理解塔冻结。
- `disable_cosmos_parallelism: true`：关闭 cosmos 内部的 FSDP/CP/CFGP，由 RLinf 的 FSDP2 统一分片。

步骤六：启动训练

建议使用 tmux 启动训练，避免 SSH 断开导致任务中止。

```bash
cd /root/workspace/RLinf

SESSION=cosmos3_sft
CONFIG=libero_sft_cosmos3
MODEL=/root/workspace/models/Cosmos3-Nano-DCP
DATA=/root/workspace/datasets/LIBERO_LeRobot_v3
WAN_VAE=/root/workspace/hf-cache/hub/models--Wan-AI--Wan2.2-TI2V-5B/snapshots/921dbaf3f1674a56f47e83fb80a34bac8a8f203e/Wan2.2_VAE.pth
START_SCRIPT=/tmp/start_cosmos3_sft.sh

LOG_DIR=/root/workspace/logs/$(date +%Y%m%d-%H:%M:%S)-${CONFIG}
LOG_FILE=${LOG_DIR}/run_embodiment.log
mkdir -p "$LOG_DIR"

cat > "$START_SCRIPT" <<EOF
#### !/usr/bin/env bash
set -e
cd /root/workspace/RLinf
source /root/workspace/RLinf/.venv/bin/activate
export EMBODIED_PATH=/root/workspace/RLinf/examples/sft
export REPO_PATH=/root/workspace/RLinf
export PYTHONPATH=/root/workspace/RLinf
export MUJOCO_GL=egl
export PYOPENGL_PLATFORM=egl
export HF_HOME=/root/workspace/hf-cache
export HF_HUB_OFFLINE=1
export TRANSFORMERS_OFFLINE=1
export TOKENIZERS_PARALLELISM=false
unset UV_DEFAULT_INDEX
export IMAGINAIRE_HF_CLI_ENVIRONMENT=/root/workspace/hf-cli-venv
CMD="python /root/workspace/RLinf/examples/sft/train_vla_sft.py --config-path /root/workspace/RLinf/examples/sft/config/ --config-name ${CONFIG} runner.logger.log_path=${LOG_DIR}
data.train_data_paths=${DATA} actor.model.model_path=${MODEL} actor.model.wan_vae_path=${WAN_VAE}"
echo "\$CMD" > ${LOG_FILE}
echo "LOG_DIR=${LOG_DIR}" >> ${LOG_FILE}
echo "MODEL=${MODEL}" >> ${LOG_FILE}
\$CMD 2>&1 | tee -a ${LOG_FILE}
EOF

chmod +x "$START_SCRIPT"

if tmux has-session -t "$SESSION" 2>/dev/null; then
  tmux kill-session -t "$SESSION"
fi

tmux new-session -d -s "$SESSION" "$START_SCRIPT"
printf "session=%s\nlog=%s\n" "$SESSION" "$LOG_FILE"
```

训练流程：读取 `libero_sft_cosmos3.yaml`，在每张 GPU 上用 FSDP2 分片 Cosmos3 模型并训练；checkpoint 每 `save_interval` 步存入 `${LOG_DIR}/libero_sft_cosmos3/checkpoints/global_step_<N>/`。

如需只训练 `libero_10`，在 `CMD` 末尾追加 `data.data_type=libero_10`，并把 `DATA` 指向 libero_10 的 LeRobot 数据目录。

首次运行建议先做一次冒烟：在 `CMD` 末尾追加 `runner.max_steps=10 runner.save_interval=10 runner.log_interval=1`，约 15 分钟即可走完「加载基座 → 训练 → 存档」全链路，并得到一份可用于验证转换流程的 `global_step_10` checkpoint。

启动后约 8 分钟验证（基座 DCP 加载在启动后约 6 分钟才出现，此前是 Ray 启动和 8 个 rank 在 CPU 上建模型）：

- `tmux ls` 中存在 `cosmos3_sft`。
- 日志第一行是完整启动命令，Hydra 打印的配置中三个路径已替换为真实路径。
- Ray 启动并识别 1 node / 8 accelerators。
- 日志中出现 `Cosmos3: loading DCP base into cosmos3_model from <MODEL>`。
- 日志中出现 `Cosmos3 SFT dataloader (cosmos PackingDataLoader): data_type='libero_all' -> experiment='action_policy_libero_all_nano'`。各 rank 打印的 dataloader 长度不同属正常。
- 出现 `Global Step` 进度条。

- 无 `Traceback`、`CUDA out of memory`、`worker died`，也没有长时间停在 HuggingFace 请求上。

断点续训

设置 `runner.resume_dir` 指向某个 `global_step_<N>` 目录后重跑，`global_step` 会从目录名解析：

```bash
runner.resume_dir=/root/workspace/logs/<run-dir>/libero_sft_cosmos3/checkpoints/global_step_1000
```

停止训练

优先通过 tmux 停止，再确认残留进程：

```bash
if tmux has-session -t cosmos3_sft 2>/dev/null; then
  tmux kill-session -t cosmos3_sft
fi
sleep 8

ps -eo pid,stat,pcpu,pmem,rss,args --sort=-pcpu | \
  grep -E 'train_vla_sft.py|ray::FSDPVlaSftWorker' | grep -v grep || true
```

如果仍有残留的 Ray worker，在确认本机没有其他用户的 Ray 任务（`ray status`）后再执行 `ray stop`。

步骤七：监控训练

查看主日志：

```bash
tail -f /root/workspace/logs/<run-dir>/run_embodiment.log
```

提取关键进展：

```bash
grep -nE 'Cosmos3: loading|PackingDataLoader|Global Step|Traceback|Exception|out of memory|worker died' \
  /root/workspace/logs/<run-dir>/run_embodiment.log | tail -100
```

`Global Step` 是 tqdm 进度条，postfix 中带 `time/step` 与 `train/*` 指标。tqdm 用 `\r` 刷新，单行可能很长，只看最后几次刷新：

```bash
tr '\r' '\n' < /root/workspace/logs/<run-dir>/run_embodiment.log | grep 'Global Step' | tail -3
```

查看 GPU：

```bash
watch -n 2 'nvidia-smi --query-gpu=index,utilization.gpu,memory.used,memory.total --format=csv,noheader,nounits'
```

初始化阶段模型留在 CPU，显存几乎不增长、容器内存快速上涨属于正常现象；FSDP2 分片上卡后各卡显存会同步上升。可以同时观察 cgroup 内存：

```bash
watch -n 2 'cat /sys/fs/cgroup/memory.current; cat /sys/fs/cgroup/memory.max; cat /sys/fs/cgroup/memory.events'
```

查看已保存的 checkpoint：

```bash
ls /root/workspace/logs/<run-dir>/libero_sft_cosmos3/checkpoints/
du -sh /root/workspace/logs/<run-dir>/libero_sft_cosmos3/checkpoints/* 2>/dev/null
```

每个 `global_step_<N>` 约 196GB：`actor/dcp_checkpoint/`（约 111GB，含优化器状态，用于续训）与 `actor/model_state_dict/full_weights.pt`（约 85GB，用于转换评测）。存档前用 `df -h` 确认剩余空间，空间紧张时调大 `runner.save_interval`。

步骤八：转换 Checkpoint

SFT 产出的 `full_weights.pt` 是 FSDP2 格式，**不能直接评测**。Cosmos3 评测默认使用 diffusers 组件目录 `model_diffusers`（含 `model_index.json` / `transformer/` / `vae/` / `text_tokenizer` / `scheduler/`）。`toolkits/cosmos3/convert_checkpoint.py` 依次完成四步转换：

步骤	**输入 → 输出**	**说明**	**实测占用**
1	`full_weights.pt` → `model_safetensors`	去掉 `omni.` 前缀	~85GB
2	`model_safetensors` → `model_dcp`	存成 DCP（`torch.distributed.checkpoint`）	~85GB
3	`model_dcp` → `model_hf`	调用 `cosmos_framework.scripts.export_model`	~30GB
4	`model_hf` → `model_diffusers`	调用 `cosmos_framework.scripts.convert_model_to_diffusers`，拆成 diffusers 组件并修正 `_class_name`	~30GB

四步产物合计约 230GB，`$OUT` 所在盘需预留足够空间；脚本会把整份权重读进内存。实测整个转换约 8 分钟（步骤 1–2 约 1.5 分钟，步骤 3–4 约 4 分钟，其余为加载）。

转换须在 **cosmos-framework 环境**下进行：`transformers 4.57.x` + `diffusers 0.39.0`（含 `Cosmos3OmniTransformer`）。更高版本的 transformers 会在 `save_pretrained` 处崩溃，且其配套 diffusers 缺少 cosmos3 相关类。先确认当前环境版本：

```bash
python -c 'import transformers, diffusers; print(transformers.__version__, diffusers.__version__)'
python -c 'from diffusers import Cosmos3OmniTransformer; print("Cosmos3OmniTransformer ok")'
```

版本不符时，请单独准备一个满足上述版本的 cosmos-framework 环境执行转换，不要直接在训练 venv 中升降级。使用推荐镜像时，训练 venv 已满足要求，实测可直接在其中完成转换。

执行转换。脚本默认的 `--config-file cosmos_framework/configs/base/config.py` 是相对 cosmos-framework 根目录的路径，因此要在该目录下运行；`PYTHONPATH` 只放 RLinf，不放 cosmos-framework 根目录（原因见步骤一的 `sitecustomize.py` 说明）：

```bash
source /root/workspace/RLinf/.venv/bin/activate
COSMOS_ROOT=$(python -c 'import cosmos_framework, pathlib; print(pathlib.Path(cosmos_framework.__file__).resolve().parent.parent)')
```

```
#### SFT 产出的 RLinf checkpoint（omni.net.* 前缀）
export SRC=/root/workspace/logs/<run-dir>/libero_sft_cosmos3/checkpoints/global_step_5000/actor/model_state_dict/full_weights.pt
#### 转换工作目录，四步的中间产物都写在这里
export OUT=/root/workspace/models/cosmos3-libero-sft-step5000
#### export_model 需要读取 LIBERO 数据集路径
export LIBERO_ROOT=/root/workspace/datasets/LIBERO_LeRobot_v3
export HF_HOME=/root/workspace/hf-cache
export HF_HUB_OFFLINE=1
export TRANSFORMERS_OFFLINE=1
export PYTHONPATH=/root/workspace/RLinf
unset UV_DEFAULT_INDEX
mkdir -p "$OUT"

cd "$COSMOS_ROOT"
python /root/workspace/RLinf/toolkits/cosmos3/convert_checkpoint.py --src "$SRC" --out "$OUT" \
  2>&1 | tee /root/workspace/logs/convert_cosmos3_ckpt.log
```

参数说明：

- `--use-ema-weights`：保留 EMA 权重，默认丢弃。
- `--experiment`：默认 `action_policy_libero_all_nano`，对应 `data_type=libero_all`。如果训练时用的是 `libero_10`，应改为 `--experiment action_policy_libero_nano`，与训练 recipe 保持一致。
- 完整参数见 `python toolkits/cosmos3/convert_checkpoint.py --help`。

脚本结束时会打印 `Done. model_diffusers -> <OUT>/model_diffusers`。验证产物：

```bash
ls "$OUT"
ls "$OUT/model_diffusers"
cat "$OUT/model_diffusers/model_index.json"
```

`model_diffusers` 下应有 `model_index.json`、`transformer/`、`vae/`、`text_tokenizer`、`scheduler/`，`model_index.json` 的 `_class_name` 为 `Cosmos3OmniDiffusersPipeline`。

步骤 3 或 4 失败、而 `$OUT/model_dcp/model/.metadata` 已经存在时，修好原因后无需重读 85GB 的 `full_weights.pt`，可以只续跑后两步。先删除失败留下的 `$OUT/model_hf` / `$OUT/model_diffusers`（如有），然后在与上面相同的环境变量和 `cd "$COSMOS_ROOT"` 下执行：

```bash
python - <<'PY'
import os
import sys

sys.path.insert(0, "/root/workspace/RLinf/toolkits/cosmos3")
import convert_checkpoint as cc

out = os.environ["OUT"]
dcp_dir = os.path.join(out, "model_dcp", "model")
assert os.path.exists(os.path.join(dcp_dir, ".metadata")), f"expected DCP metadata under {dcp_dir}"
#### 最后一个参数 True 表示丢弃 EMA，与默认行为一致；libero_10 训练改用 "action_policy_libero_nano"
hf_dir = cc.step3_export_to_hf(dcp_dir, out, cc.DEFAULT_CONFIG, cc.DEFAULT_EXPERIMENT, True)
print(f"Done. model_diffusers -> {cc.step4_convert_to_diffusers(hf_dir, out)}")
PY
```

转换成功后，`model_safetensors`、`model_dcp`、`model_hf` 三个中间目录（约 200GB）可以按需删除，只保留 `model_diffusers`。

转换得到的 `model_diffusers` 即 [Cosmos3 SGLang 评测](https://rlinf.readthedocs.io/zh-cn/latest/rst_source/evaluations/guides/cosmos3_sglang.html) 中的 `rollout.model.model_path`。评测侧的 `action_stats_path` 必须指向与训练相同的 `libero_native_frame_wise_relative_rot6d.json`。

步骤九：启动 TensorBoard（可选）

启动 TensorBoard：

```bash
cd /root/workspace/RLinf
source /root/workspace/RLinf/.venv/bin/activate
tmux new-session -d -s tensorboard_rlinf \
  'cd /root/workspace/RLinf && tensorboard --logdir /root/workspace/logs --port 6006 --host 127.0.0.1'
```

本地访问：

```bash
ssh -p <port> -L 6006:127.0.0.1:6006 -L 18265:127.0.0.1:8265 root@<host>
```

浏览器打开：

```text
http://127.0.0.1:6006
```

Ray Dashboard 远端默认监听 `127.0.0.1:8265`，按上面的端口转发后在本地打开 `http://127.0.0.1:18265`，可查看 SFT worker 状态与 CPU/GPU 使用情况。远端验证：

```bash
curl -s http://127.0.0.1:8265/api/version
```

步骤十：解析 TensorBoard 指标

使用脚本读取 TensorBoard scalar：

```bash
cd /root/workspace/RLinf
source /root/workspace/RLinf/.venv/bin/activate
export TF_CPP_MIN_LOG_LEVEL=3

python - <<'PY'
from pathlib import Path
from tensorboard.backend.event_processing.event_accumulator import EventAccumulator
```

```
from tensorboard.backend.event_processing.event_accumulator import EventAccumulator
run = Path('/root/workspace/logs/<run-dir>')
for event in sorted((run / 'tensorboard').rglob('events.out.tfevents.*')):
    print(f'EVENT {event.name} size={event.stat().st_size}')
    ea = EventAccumulator(str(event), size_guidance={'scalars': 0})
    ea.Reload()
    for tag in sorted(ea.Tags().get('scalars', [])):
        vals = ea.Scalars(tag)
        if vals:
            v = vals[-1]
            print(f'{tag}\tcount={len(vals)}\tstep={v.step}\tvalue={v.value}')
PY
```

重点关注：

指标	**含义**
`train/loss`	SFT 训练损失，应随步数稳定下降
`train/grad_norm`	梯度范数（`clip_grad=1.0`），持续偏大或出现 NaN 需检查数据与学习率
`train/learning_rate`	当前学习率，前 500 步 warmup 后按 `lambda_linear` 变化
`time/training`	每个 step 的训练耗时

官方文档未给出 loss 曲线和耗时的参考值。下表是 8 × 183GB GPU、`libero_all`、默认批大小下 10 步冒烟的实测值，只用于判断量级是否正常，不代表收敛结果：

指标	**实测值**
`train/loss`	15.7 → 15.5
`train/flow_matching_loss_action`	1.42 → 1.41
`train/flow_matching_loss_vision`	约 0.144
`train/grad_norm`	约 14.8
`train/learning_rate`	约 1e-6（warmup 初期）
`time/step`	首步约 110 秒，稳态约 28.5 秒
显存	每卡 76–79GB
启动到首步	约 7 分钟（基座 DCP 加载约 6 分钟）

按稳态 28.5 秒/步估算，默认 5000 步约需 40 小时（不含存档）。日志中参数组学习率打印为 `[5e-05, 0.00025]` 属正常。指标含义可参考 [训练指标](https://rlinf.readthedocs.io/zh-cn/latest/rst_source/reference/metrics.html)。

特殊说明：action 转换 rot6d 10 维 vs axis-angle 7 维

Cosmos3 内部用 **10 维 rot6d** 表示动作，LIBERO 环境用 **7 维 axis-angle**，这是理解整条训练与评测链路的关键：

表示	**维度**	**组成**
`raw_action_dim`（模型侧）	10	3 平移 + 6 rot6d + 1 夹爪
`action_dim`（环境侧）	7	3 平移 + 3 axis-angle + 1 夹爪

使用 rot6d 是因为 axis-angle 在 ±π 处不连续，扩散模型难以回归；rot6d（旋转矩阵前两列）连续，更容易学习。

训练时 cosmos 的数据加载器在线把 LIBERO 的 7 维动作转成 10 维 rot6d（见步骤四）；推理时再把模型输出的 10 维 rot6d 转回 7 维交给环境执行。基座 Cosmos3-Nano 自带的是 DROID 8 维动作头，因此训练时通过 `keys_to_skip_loading` 跳过这部分参数并重新初始化。

常见问题处理

问题	**解决方案**
路径仍是 `path/to/...`	启动命令必须追加 `data.train_data_paths=... actor.model.model_path=... actor.model.wan_vae_path=...`；`run_vla_sft.sh` 不透传 override
初始化 OOM（单卡 80GB 也爆）	确认 `actor.model.load_to_device: false`；仍 OOM 时可临时设 `actor.model.ema_enabled=false`
初始化阶段容器内存被打满	8 个 rank 同时在 CPU 上构建约 81GB 的模型，检查 `/sys/fs/cgroup/memory.max` 与 `memory.events`，优先调大容器内存
基座加载失败 / 格式判断错误	`model_path` 指向的是 diffusers 目录而非 DCP；按步骤三转换后再使用
DCP 加载报 pickle / 反序列化错误	DCP 在另一个 Python 版本的环境中生成；在训练 venv 中重新执行 `convert_model_to_dcp`
加载报动作头 shape 不匹配	确认 `keys_to_skip_loading` 跳过了基座 DROID 8 维动作头（`model/cosmos3.yaml` 默认已配置）
HuggingFace 拉取卡住	离线 + 本地缓存：`HF_HUB_OFFLINE=1`、`TRANSFORMERS_OFFLINE=1`，`HF_HOME` 指向已下载好的缓存目录
`cannot import name 'override' from 'typing'`	多半是 `PYTHONPATH` 含 cosmos-framework 根目录，其自带的 `sitecustomize.py` 遮蔽了 venv 里的补丁；也可能是当前 shell 没激活 `/root/workspace/RLinf/.venv`。先把 `PYTHONPATH` 只设为 RLinf 目录重试。checkpoint 转换在步骤 3 报此错时，按步骤八只续跑后两步
`uv.lock needs to be updated, but --locked`	全局 `UV_DEFAULT_INDEX` 与 cosmos `hf_cli` 的 lock 不一致；`unset UV_DEFAULT_INDEX` 后重跑
离线模式下找不到 Qwen3-VL / Wan VAE，或 tokenizer `vocab_file` 为 None	缓存 snapshot 目录名不是固定 revision（常见于 BOS 拷贝成普通目录）；按步骤二方式二的布局重放，并用「验证缓存」确认
`hf download` 探测单个文件报找不到	参数顺序写反；文件名紧跟仓库名，`--revision` 放后面
下载 LIBERO 数据集报 308	hf-mirror 不支持 datasets 接口；`unset HF_ENDPOINT` 后直连 huggingface.co
存档时 `No space left on device`	每个 checkpoint 约 196GB；调大 `runner.save_interval`、清理无用的旧 step，或把日志目录放到大容量盘
数据被识别为 `libero_10`	`train_data_paths` 下缺少某个 suite 子目录；补齐数据或显式设置 `data.data_type`
转换在 `save_pretrained` 处崩溃	transformers 版本过高；切到 `transformers 4.57.x` + `diffusers 0.39.0` 的 cosmos-framework 环境
评测动作混乱 / loss 异常	核对训练与评测两侧 `action_normalization`（`quantile_rot`）与统计文件同源，转换时的 `--experiment` 与训练 `data_type` 一致

RLinf v0.4 基于ApxInf 在Libero上评测PI 0.5

概述

[Rlinf](https://rlinf.readthedocs.io/zh-cn/latest/) 是一个面向 **Embodied AI（具身智能）** 和 **Agentic AI（智能体 AI）** 的开源强化学习基础框架。

[ApxInf](https://github.com/infinigence/ApxInf) 是一个进程内的 Rust/CUDA 推理引擎。RLinf 通过 [APXinf-robo](https://github.com/RLinf/APXinf-robo) 提供的 L1 bare-model 接口（`load_bare_model` 与 `Model.infer_rgb`）把它接入成 rollout 后端：环境交互、观测 batch、OpenPI 预处理与后处理、动作块执行仍由 RLinf 负责，GPU 上的模型推理交给 ApxInf。

通过百舸快速开始，您可以直接在开发机镜像内启动 LIBERO-10 上的 $\pi_{0.5}$ 评测任务，并读取 `eval/success_once` 成功率、评测日志与 TensorBoard 指标。当前接入只支持**在 LIBERO 上评测 $\pi_{0.5}$**，不支持训练、解耦 rollout 或非具身任务。

适用场景

- 在 LIBERO-10 上评测 $\pi_{0.5}$（`lerobot/pi05_libero_base`）基线成功率。
- 用进程内推理后端替代独立的 rollout 服务，观察同机评测的启动时间与显存占用。
- 排查 checkpoint 文件、`norm_stats.json`、CUDA binding、EGL 渲染与轨迹预算相关的问题。
- 不适合：训练任务、解耦 rollout、$\pi_{0.5}$ 以外的模型。

评测规格参考 & 环境配置

推荐评测配置如下：

项目	**推荐配置**

GPU	单卡即可；环境与 rollout 共址，显存按一份 bf16 权重加一个并行环境估算，建议单卡 ≥ 24GiB
CPU	建议 16 核以上
容器内存	建议 64GiB 以上
存储空间	建议不少于 100G（checkpoint、LIBERO 资产与日志）

项目目录配置如下表所示：

项目	**推荐配置**
CUDA / 驱动	使用镜像自带的驱动与 CUDA，不要单独升级
工作目录	`/root/workspace/RLinf`
Python 环境	`/root/workspace/RLinf/.venv`（openpi + libero 依赖）
ApxInf 运行环境	镜像内预装 `apxinf_py`（CUDA binding）与 `apxinf_robo`（L1 封装）
评测入口	`/root/workspace/RLinf/evaluations/run_eval.sh`
评测配置	`/root/workspace/RLinf/evaluations/libero/libero_10_apxinf_pi05_eval.yaml`
模型目录	`/root/workspace/models/pi05_libero_base`
日志目录	`/root/workspace/RLinf/logs`

配置把 env 与 rollout 都放在全部 GPU 上，每个 rollout worker 在自己的 GPU 上常驻一份 bf16 权重；单卡节点就是一份拷贝、一组并行环境。

评测产出与结果口径

单卡默认配置的评测协议是：LIBERO-10 的 10 个任务，每个任务 10 个 reset state，共 100 条轨迹（10 个并行环境 × 10 个 rollout epoch）。终端打印的 `eval/success_once` 就是这批轨迹的成功率。多卡节点上实际并发的环境数会变，见「轨迹预算」。

评测完成后会留下以下产物：

产物	**路径**
主评测日志	`<LOG_DIR>/eval_embodiment.log`
指标表	`<LOG_DIR>/metrics.log`
TensorBoard event	`<LOG_DIR>/tensorboard/`
启动期完整 stdout	启动脚本 tee 到 `/root/apxinf-eval.out`

其中 `<LOG_DIR>` 由 `run_eval.sh` 生成，形如 `/root/workspace/RLinf/logs/20261020-15:30:00-libero_10_apxinf_pi05_eval`。配置文件里的 `logger.log_path: "../results"` 会被这个值覆盖，不需要也不能在配置里指定日志目录。

成功率取决于 checkpoint 权重与实际并行环境数。8 卡节点按 10 epoch × 80 环境跑满 800 条轨迹时，实测 `eval/success_once=0.9525`、`eval/success_at_end=0.82125`、`eval/return=4.7625`、`eval/reward=0.01978`，整轮评测约 12 分钟；单卡或改变环境数后数值会有差异，以本次运行写入 `metrics.log` 的结果为准。

准备工作

如果您所在网络环境访问 HuggingFace 较慢，建议提前配置 HuggingFace 镜像源，或配置[代理加速](https://cloud.baidu.com/doc/BCM/s/3mmswg2oh)。

huggingface 镜像配置：

```bash
export HF_ENDPOINT=https://hf-mirror.com
export HF_HUB_ENABLE_HF_TRANSFER=1
```

如果所在环境已提供该 checkpoint 的 BOS 地址，直接从 BOS 拉取到工作目录，避免依赖外部网络；本文的步骤二把 BOS 列为推荐方式。

环境说明

> 说明：本文中的路径均以百舸开发机内的 `/root/workspace` 为例。实际使用时，可根据任务挂载目录调整。

当前使用的目录如下：

```text
/root/workspace/RLinf
/root/workspace/RLinf/.venv
/root/workspace/models/pi05_libero_base
/root/workspace/RLinf/logs
```

APXinf-robo 与 `apxinf_py` 已预装在镜像内，无需执行 `git clone`、`maturin build` 或 `pip install`。步骤一仅做导入验证；如果导入失败，说明当前 shell 激活的不是镜像自带的那个 `.venv`，而不是需要重新编译。

步骤一：确认镜像内的 ApxInf 运行环境

激活镜像自带的 Python 环境，验证 binding 可以导入：

```bash
source /root/workspace/RLinf/.venv/bin/activate
cd /root/workspace/RLinf

python -c 'import apxinf_py; print("apxinf_py", apxinf_py.__version__)'
python -c 'import apxinf_robo; print("apxinf_robo", apxinf_robo.__version__)'
python -c 'from apxinf_robo import load_bare_model; print("load_bare_model ok")'
```

三行都能正常输出，说明 ApxInf 的 CUDA binding 与 L1 封装在该环境内可用。

再确认 GPU 与驱动：

```bash
nvidia-smi --query-gpu=index,name,driver_version,memory.total --format=csv,noheader
```

镜像预装的 binding 面向镜像发布时的目标 GPU 架构（例如 `sm_87`、`sm_89`、`sm_101`、`sm_110`）。换到其他架构的机器上，`import` 仍可能成功，但加载权重时会因为找不到对应 kernel 而失败；这种情况需要换用为目标架构构建的镜像（`APXINF_CUDA_ARCH` 是该 binding 的构建参数），而不是修改本文的评测命令。运行时按设备 SM 精确匹配编译目标，所以一份 binding 要覆盖多种卡，必须在构建时就把架构列全。本文实测的 binding 以 `APXINF_CUDA_ARCH=sm_100,sm_103,sm_120` 构建，`cuobjdump --list-elf` 能看到这三个 `sm_` 段；B30Z（compute 10.3）命中 `sm_103`。

渲染后端不需要手动配置：`run_eval.sh` 在 `MUJOCO_GL` 未设置时会自动设为 `egl`，同时设置 `PYOPENGL_PLATFORM=egl`。

步骤二：准备 $\pi_{0.5}$ LIBERO Checkpoint

评测需要一个包含 `config.json`、`model.safetensors` 和 `norm_stats.json` 的 checkpoint 目录。`tokenizer.model` 不是必需文件：ApxInf 用 checkpoint 自带的 tokenizer 配置加载，实测缺少该文件也能正常推理。

方式一：通过 BOS 下载（推荐）

百舸环境已提供该 checkpoint 与归一化统计的 BOS 地址，内网直连，不受 HuggingFace 出口影响。目录路径必须带结尾 `/`：

```text
bos:/aihc-models-bj/lerobot/pi05_libero_base/
bos:/aihc-models-bj/openpi/pi05_libero/assets/physical-intelligence/libero/norm_stats.json
```

下载前可以先确认前缀存在：

```bash
bcecmd bos ls bos:/aihc-models-bj/lerobot/pi05_libero_base/
```

`bcecmd bos cp` 支持一次拷贝整个前缀，权重与 `norm_stats.json` 分两条命令拉取；`norm_stats.json` 必须落在 checkpoint 根目录：

```bash
mkdir -p /root/workspace/models/pi05_libero_base

bcecmd bos cp -r \
  bos:/aihc-models-bj/lerobot/pi05_libero_base/ \
  /root/workspace/models/pi05_libero_base/

bcecmd bos cp \
  bos:/aihc-models-bj/openpi/pi05_libero/assets/physical-intelligence/libero/norm_stats.json \
  /root/workspace/models/pi05_libero_base/norm_stats.json
```

同机实测这份 BOS 权重约 17G，通常一分钟内完成。下载量本身不大，但仍建议放进 tmux，避免 SSH 断开导致重来。

方式二：通过 HuggingFace 下载模型权重

HF 下载放在 tmux 里执行：

```bash
cat > /tmp/download_pi05_libero.sh <<'EOF'
#### !/usr/bin/env bash
set -euo pipefail
export HF_ENDPOINT=${HF_ENDPOINT:-https://hf-mirror.com}
export HF_HUB_ENABLE_HF_TRANSFER=1
source /root/workspace/RLinf/.venv/bin/activate
pip install "huggingface_hub<1.0"
mkdir -p /root/workspace/models
{
  date
  hf download lerobot/pi05_libero_base \
    --local-dir /root/workspace/models/pi05_libero_base
  date
  echo "download done"
} 2>&1 | tee -a /root/workspace/models/download_pi05_libero.log
EOF
chmod +x /tmp/download_pi05_libero.sh

tmux new-session -d -s apxinf_download /tmp/download_pi05_libero.sh
```

> 不要用 `pip install -U huggingface_hub`：它会把版本升到 1.x，与镜像内 `transformers` 的依赖约束冲突，必须固定在 `<1.0`。

查看下载进度：

```bash
tmux ls 2>/dev/null | grep apxinf_download || true
tail -20 /root/workspace/models/download_pi05_libero.log
du -sh /root/workspace/models/pi05_libero_base 2>/dev/null || true
```

`lerobot/pi05_libero_base` 仓库不包含复现参考评测行为所需的归一化统计，`norm_stats.json` 必须单独下载到 checkpoint 根目录：

```bash
curl -fL \
  https://storage.googleapis.com/openpi-assets/checkpoints/pi05_libero/assets/physical-intelligence/libero/norm_stats.json \
  -o /root/workspace/models/pi05_libero_base/norm_stats.json
```

GCS 不通时可改用方式一里的 BOS 单文件地址。

验证 checkpoint

> 说明：`APXINF_PI05_MODEL_DIR` 是评测配置中 `model_path` 的唯一来源（`${oc.env:APXINF_PI05_MODEL_DIR}`），必须先导出，否则 Hydra 解析配置时直接失败。

```bash
export APXINF_PI05_MODEL_DIR=/root/workspace/models/pi05_libero_base

for f in config.json model.safetensors norm_stats.json; do
  if [ -f "${APXINF_PI05_MODEL_DIR}/${f}" ]; then
    echo "OK   ${f}"
  else
    echo "MISS ${f}"
  fi
done
du -sh "${APXINF_PI05_MODEL_DIR}"
```

三行全部为 `OK` 再继续。

步骤三：确认评测配置

评测配置是 `evaluations/libero/libero_10_apxinf_pi05_eval.yaml`，核心字段如下：

评测配置是 `evaluations/libero/libero_10_apxinf_pi05_eval.yaml`，核心字段如下：

```yaml
runner:
  task_type: embodied_eval
  only_eval: true
  logger:
    logger_backends: ["tensorboard"]

env:
  eval:
    rollout_epoch: 10
    total_num_envs: 10
    auto_reset: false
    max_episode_steps: 520
    max_steps_per_rollout_epoch: 520

rollout:
  rollout_backend: "apxinf"
  generation_backend: "apxinf"
  pipeline_stage_num: 1
  model:
    model_path: ${oc.env:APXINF_PI05_MODEL_DIR}
    num_action_chunks: 5
    action_dim: 7
    openpi:
      config_name: "pi05_libero"
      num_images_in_input: 2
      num_steps: 5
      noise_method: "flow_sde"
    apxinf:
      precision: "bf16"
      action_horizon: 10
      num_flow_steps: ${..openpi.num_steps}
      noise_source: "apxinf"
```

这些字段在代码里有硬性校验，启动前逐条核对：

约束	**位置**	**不满足时**
`runner.only_eval` 必须为 `true`	`runner`	`ApxInfRolloutWorker` 初始化断言失败
`runner.enable_decoupled_mode` 必须为 `false`	`runner`	同上，ApxInf 不支持解耦 rollout
`openpi.config_name` 必须以 `pi05_` 开头	`rollout.model.openpi`	报只支持 PI0.5 的 OpenPI 配置
`apxinf.num_flow_steps` 必须等于 `openpi.num_steps`	两处	报两者必须一致（数值对齐要求）
`apxinf.action_horizon` 必须 ≥ `num_action_chunks`	`rollout.model`	报 `action_horizon` 小于动作块长度
`openpi.noise_method` 只能是 `flow_ode` / `flow_sde`	`rollout.model.openpi`	报不支持；评测模式下 `flow_sde` 退化为 ODE
`apxinf.noise_source` 只能是 `apxinf` / `torch` / `observation`	`rollout.model.apxinf`	报非法取值
`model_path` 由 `APXINF_PI05_MODEL_DIR` 注入	环境变量	Hydra 解析配置时失败

`noise_source: observation` 只用于成对数值对齐测试，需要显式传入 `[B, 10, 32]` 的 noise 张量；常规评测保持 `apxinf`，由 ApxInf 采样初始高斯噪声。

多 epoch 评测必须关掉 auto_reset

`env.eval.auto_reset` 为 `true` 时，`evaluate()` 只在第 0 个 epoch 做一次 reset，之后依赖 `chunk_step` 自动重置；但 `chunk_step` 调用的是 `step(..., auto_reset=False)`，实际不会重置。环境状态会跨 epoch 一直累积，累计步数超过 robosuite 内部 horizon（1000）后，下一次 `step` 抛 `ValueError: executing action in terminated episode`，MuJoCo 子进程以退出码 1 结束，父进程 `recv()` 收到 `EOFError`，随后整组 worker 被 `ray.kill`，评测以 `EXIT=255` 中断。

单 epoch（520 步 < 1000）不会触发这个问题，所以它只在多 epoch 的完整评测里出现。把 `env.eval.auto_reset` 设为 `false` 后，每个 epoch 开始时统一 reset，单条轨迹固定 520 步，10 个 epoch × 80 环境正好 800 条轨迹。

仓库里的配置默认是 `true`，启动时用 Hydra override 覆盖即可：`env.eval.auto_reset=false`。

轨迹预算

并行环境数必须能被 env worker 数整除。`rlinf/config.py` 里的 `validate_embodied_cfg` 断言 `total_num_envs % env_world_size == 0`，不满足时启动阶段直接 `AssertionError` 退出，不会静默截断。整除之后，每个 env worker 分到 `total_num_envs // worker 数 // pipeline_stage_num` 个环境。

配置里 `component_placement` 为 `env,rollout: all`，所以在 8 卡节点上 env 组有 8 个 worker，`total_num_envs` 必须是 8 的倍数。默认值 10 只适用于单卡（10 环境 × 10 epoch = 100 条轨迹），在 8 卡节点上会直接启动失败。

按默认协议跑满时，8 卡节点上设 `env.eval.total_num_envs=80`，即每个 worker 10 个环境，10 个 epoch 共 800 条轨迹。

LIBERO 的评测 reset state 池按任务交错排列后做 round-robin 分配，改变环境数会改变每个任务实际覆盖的 reset state 数量。

步骤四：启动评测

建议用 tmux 启动，避免 SSH 断开导致任务中断。启动脚本落地到 `/root`，评测命令只写一次：

```bash
cd /root/workspace/RLinf

cat > /root/run_apxinf_eval.sh <<'EOS'
#!/usr/bin/env bash
set -uo pipefail
source /root/workspace/RLinf/.venv/bin/activate
export APXINF_PI05_MODEL_DIR=/root/workspace/models/pi05_libero_base
export MUJOCO_GL=egl
export PYOPENGL_PLATFORM=egl
cd /root/workspace/RLinf
bash evaluations/run_eval.sh libero libero_10_apxinf_pi05_eval
rc=$?
echo "EXIT=${rc}"
EOS
chmod +x /root/run_apxinf_eval.sh

if tmux has-session -t apxinf_eval 2>/dev/null; then
  tmux kill-session -t apxinf_eval
fi
```

```
tmux new-session -d -s apxinf_eval \
  'bash /root/run_apxinf_eval.sh 2>&1 | tee /root/apxinf-eval.out'
```

需要放大环境数并跑满 10 个 epoch 时，把脚本里的评测命令改成带 Hydra override 的形式。`run_eval.sh` 从第三个参数起把内容原样透传：

```bash
bash evaluations/run_eval.sh libero libero_10_apxinf_pi05_eval \
  env.eval.total_num_envs=80 \
  env.eval.auto_reset=false
```

只跑单 epoch 做冒烟时，前一节提到的 `auto_reset` 问题不会出现，但统一带上 `env.eval.auto_reset=false` 可以保证多轮结果口径一致。

日志目录由 `run_eval.sh` 生成，不要自己指定：

```bash
ls -dt /root/workspace/RLinf/logs/*libero_10_apxinf_pi05_eval | head -1
```

启动后验证点：

- `tmux ls` 中存在 `apxinf_eval`。
- `/root/apxinf-eval.out` 中出现 `Evaluation Mode: Standard LIBERO` 和 `Using benchmark=libero, config=libero_10_apxinf_pi05_eval`。
- 日志中出现 `Initializing a new cluster with all connected nodes.`，Ray 启动并占用预期数量的 GPU。
- 日志中出现 `ApxInf policy loaded: device=cuda:N, metadata={...}`，每个 rollout worker 一行，`metadata` 含 `action_horizon`、`model_action_dim`、`num_views`、`image_size`。
- 出现 `Evaluating Rollout Epochs` 进度条。
- 无 `Traceback`、`RayOutOfMemory`、`worker died`。

`Using Python at ...` 只写 stdout，不会进 `eval_embodiment.log`；查这一行要看 `/root/apxinf-eval.out` 或 `tmux capture-pane`。

单条轨迹最长 520 个环境步，每 `num_action_chunks=5` 步调用一次推理，因此单条轨迹最多 104 次 `infer_rgb` 调用，可以据此估算总耗时。

步骤五：监控评测

查看主日志：

```bash
tail -f /root/workspace/RLinf/logs/<run-dir>/eval_embodiment.log
```

提取关键进展：

```bash
grep -nE 'ApxInf policy loaded|Evaluating Rollout Epochs|eval/success_once|Traceback|Error|worker died' \
  /root/workspace/RLinf/logs/<run-dir>/eval_embodiment.log | tail -50
```

查看 tmux 里的实时输出（含只在 stdout 的行）：

```bash
tmux capture-pane -p -t apxinf_eval | tail -40
```

查看 GPU：

```bash
watch -n 2 'nvidia-smi --query-gpu=index,utilization.gpu,memory.used,memory.total --format=csv,noheader,nounits'
```

查看 Ray worker：

```bash
ps -eo pid,stat,pcpu,pmem,rss,args --sort=-pcpu | \
  grep -E 'eval_embodied_agent.py|ApxInfRolloutWorker|ray::EnvWorker|ray::ApxInf' | head -20
```

步骤六：查看评测结果

评测结束（`EXIT=0`）后，先读指标表：

```bash
RUN_DIR=$(ls -dt /root/workspace/RLinf/logs/*libero_10_apxinf_pi05_eval | head -1)
cat "${RUN_DIR}/metrics.log"
```

指标表中每一项都带 `eval/` 前缀：

指标	**含义**
`eval/success_once`	轨迹成功率：episode 内任意时刻达成任务即计成功，是本次评测的核心结论
`eval/reward`	每条轨迹的 return 除以成功步数（未成功则除以当前步数）
`eval/return`	每条轨迹累计回报
`eval/episode_len`	每条轨迹的步数
`eval/num_trajectories`	参与统计的轨迹总数，用于确认实际跑了多少条

`eval/num_trajectories` 应当等于「实际并发环境数 × `rollout_epoch`」。如果它小于预期，先回到「轨迹预算」核对 `total_num_envs` 与 worker 数。

TensorBoard（可选）

评测指标只在 `step=0` 写一个点，TensorBoard 里是一条单点曲线，适合与历史 run 并列对比。启动 TensorBoard：

```bash
cd /root/workspace/RLinf
source .venv/bin/activate
tmux new-session -d -s tensorboard_eval \
  'cd /root/workspace/RLinf && tensorboard --logdir /root/workspace/RLinf/logs --port 6006 --host 127.0.0.1'
```

本地访问：

```bash
ssh -p <port> -L 6006:127.0.0.1:6006 root@<host>
```

浏览器打开：

```text
http://127.0.0.1:6006
```

如果不想起 TensorBoard，也可以直接读 event 文件：

```bash
cd /root/workspace/RLinf
source .venv/bin/activate
export TF_CPP_MIN_LOG_LEVEL=3

python - <<'PY'
from pathlib import Path
from tensorboard.backend.event_processing.event_accumulator import EventAccumulator

run = Path('/root/workspace/RLinf/logs/<run-dir>')
for event in sorted((run / 'tensorboard').glob('events.out.tfevents.*')):
    print(f'EVENT {event.name} size={event.stat().st_size}')
    ea = EventAccumulator(str(event), size_guidance={'scalars': 0})
    ea.Reload()
    for tag in sorted(ea.Tags().get('scalars', [])):
        vals = ea.Scalars(tag)
        if vals:
            v = vals[-1]
            print(f'{tag}\tcount={len(vals)}\tstep={v.step}\tvalue={v.value}')
PY
```

常见问题处理

问题	**解决方案**
Hydra 报 `APXINF_PI05_MODEL_DIR` 无法解析	在启动脚本里 `export APXINF_PI05_MODEL_DIR=...` 后再调 `run_eval.sh`；不要只写进当前交互 shell
报 `apxinf_robo is not importable in the rollout worker`	激活的不是镜像自带的 venv，执行 `source /root/workspace/RLinf/.venv/bin/activate`；不要重新 `pip install`
加载权重时报找不到 kernel 或设备非法	镜像内的 binding 与当前 GPU 架构不匹配；换用按该架构 `APXINF_CUDA_ARCH` 构建的镜像
报 `active camera views ... but the ApxInf checkpoint expects ...`	checkpoint 的视野数与 `openpi.num_images_in_input` / `apxinf.num_views` 不一致，按 checkpoint 调整
报 norm stats 相关错误	`norm_stats.json` 缺失，或没放在 checkpoint 根目录；按步骤二单独下载
升级 `huggingface_hub` 后 `transformers` 报依赖冲突	执行 `pip install "huggingface_hub<1.0"` 回退；镜像内 `transformers` 要求该版本上界
报 `num_flow_steps must match OpenPI num_steps`	修改 `openpi.num_steps` 时必须同步 `apxinf.num_flow_steps`（配置里用 `${..openpi.num_steps}` 引用即可）
报 `action_horizon must be at least num_action_chunks`	提高 `apxinf.action_horizon`，或降低 `num_action_chunks`
启动报 `total_num_envs ... divisible by the number of environment processes`	`total_num_envs` 不是 env worker 数（8 卡为 8）的倍数；改成 8 的倍数，例如 `env.eval.total_num_envs=80`
多 epoch 评测中途 `EXIT=255`，日志里是 `EOFError` 或 `ValueError: executing action in terminated episode`	`env.eval.auto_reset` 为 `true` 导致环境跨 epoch 不重置；加上 `env.eval.auto_reset=false` 重跑
日志里找不到 `Using Python at`	该行只写 stdout，看 `/root/apxinf-eval.out` 或 `tmux capture-pane -p -t apxinf_eval`
任务无输出或进程消失	先 `tmux ls`、`tmux capture-pane`，再 grep `Traceback` / `RayOutOfMemory` / `worker died`，不要只看 `eval_embodiment.log` 尾部
Ray 启动报端口或资源冲突	同机可能还有旧集群；先 `ray status` 确认，再停掉旧集群重试
EGL 初始化失败	确认驱动与 EGL 库可用；无 EGL 的平台改为 `export MUJOCO_GL=osmesa PYOPENGL_PLATFORM=osmesa`
指标表里没有 `eval/success_once`	确认 `runner.only_eval: true`，且日志里出现过 `Evaluating Rollout Epochs` 与 `ApxInf policy loaded`

百舸工作流快速体验：以视觉清洗管线为例

本文以视觉清洗管线为例，介绍如何使用百舸工作流编排一条多步骤数据管线：从预置模板创建工作流，配置队列和存储参数，运行并查看各节点的状态、日志与监控，最后查看输出结果。平台公共 BOS 上自带样例数据，您无需准备任何数据即可跑通全流程。文末介绍如何将样例数据替换为您自己的数据。

背景信息

什么是百舸工作流

百舸新一代工作流是百舸平台上用于编排端到端数据管线的模块，开箱即用，帮助算法工程师高效地组织、管理和执行复杂的机器学习工作流，覆盖从数据准备、模型开发到模型部署的全生命周期。

工作流将数据处理各环节中分散的步骤组织为一条有向无环图（DAG）流水线：每个节点是一个独立的任务，节点之间的连线定义执行顺序和依赖关系。工作流定义一次即可反复运行，也支持定时运行和失败后从断点续跑。

为什么使用工作流

在具身智能数据生产中，一条管线通常包含解码切分、规则过滤、模型推理、格式导出等十几个步骤，CPU 步骤和 GPU 步骤交替出现，数据量动辄几十 TB。如果将所有步骤放在单个作业中运行，会遇到以下三个问题，工作流分别提供了对应的能力：

* **资源按节点分配**：单个作业中所有步骤共用同一批资源，CPU 步骤运行时 GPU 处于闲置。工作流中每个节点单独配置资源，只有需要 GPU 的节点才申请 GPU。
* **失败后从断点续跑**：单个作业中一步失败，往往需要从头重跑。工作流可以从失败的节点继续运行。
* **按节点排查问题**：单个作业中步骤边界不清，排查困难。工作流中每个节点都有独立的运行状态、日志、监控和中间文件。

本文示例：视觉清洗管线

本文使用[视觉清洗管线](https://cloud.baidu.com/doc/AIHC/s/2mruca7ig)作为示例。视觉清洗管线是百舸工作流具身系列数据管线的入口，对应预置工作流模板 **人类演示数据视觉预处理与质量清洗**。

在视频进入手部重建、VLM 打标等成本较高的 GPU 环节之前，该管线先用 CPU 剔除模糊、静止、过短的片段。输出的 `clips.jsonl` 可直接作为下游管线 「人类演示视频转 LeRobot v3.0」和 「多模态 Caption 标注」的输入。

前提条件

在控制台操作之前，请逐项确认以下准备工作。

* 已开通[百舸工作流](https://cloud.baidu.com/doc/AIHC/s/Gmn0f3jdf)服务。
* 已创建全托管资源池和资源队列，详情请参见[创建全托管资源池](https://cloud.baidu.com/doc/AIHC/s/am794vnny)。
* 已创建 PFS 实例并绑定至上述资源池，用于存放中间结果和最终输出，详情请参见[PFS 存储配置](https://cloud.baidu.com/doc/AIHC/s/Om7995fxw#pfs-%E5%AD%98%E5%82%A8%E9%85%8D%E7%BD%AE)。
* （可选）如需查看结果文件，需准备一台挂载**同一个 PFS 实例**的开发机，详情请参见[开发机存储挂载与数据集挂载](https://cloud.baidu.com/doc/AIHC/s/dmojisum4)。
* （可选）如需使用自己的 BOS 数据，请参见下文“使用自有数据运行管线”中对应方式的准备事项。

工作原理

运行时，平台按 DAG 顺序将每个节点（在控制台中称为“子任务”）提交为一个独立的分布式任务。在视觉清洗管线中，每个节点是一个 RayJob：由 1 个 Head 和 N 个 Worker 组成一个临时 Ray 集群，节点结束后集群即释放，下游节点再拉起自己的集群。

节点之间不直接传递数据，而是通过共享存储上的文件交接：上游节点将结果写入约定路径，下游节点从该路径读取。

核心处理流程

视觉清洗管线包含 6 个节点，各节点的功能、默认参数和输出文件如下表所示。

节点	**功能**	**节点参数（默认值）**	**输出文件**
step1 去畸变	按相机内参和畸变系数做镜头校正	`camera_model=pinhole`（可选 `fisheye`）	`s1_undistort.jsonl`
step2 场景切分	检测画面突变点，将长视频切分为独立片段，每个片段一行	`threshold=27.0`、`min_scene_len=15`	`s2_split_scene.jsonl`、`clips/*.mp4`
step3 元数据提取	使用 ffprobe 读取时长、分辨率、帧率等	无	`s3_metadata.jsonl`
step4 清晰度评分	计算采样帧的 Laplacian 方差，值越大越清晰	`method=laplacian`、`fps=2.0`、`max_frames=60`	`s4_sharpness.jsonl`
step5 运动评分	计算 Farneback 光流运动量，并标准化到 0~1	`optical_flow_algorithm=farneback`、`use_cuda=false`	`s5_motion.jsonl`
step6 质量过滤	清晰度、运动量、时长三项同时达标的片段才保留	`clarity_min=20.0`、`motion_min=0.05`、`min_duration=0.5`	最终产物，写入 `output_file`

> **说明**
>
> * step1–step5 的输出文件写在 `tmp_dir` 下；step6 的最终产物写入 `output_file` 指定的路径。
> * 全局参数 `no_undistort=true` 时，step1 直接跳过。
> * 每个节点读取上一节点写出的 JSONL，在原有列的基础上追加新列后写出，因此任意一个中间文件都可以单独打开排查。
>

操作流程

步骤一：从模板创建工作流

1. 登录百舸控制台，在左侧导航栏单击 **工作流**。页面包含 **我的工作流**、**运行记录**、**工作流模板**、**子任务模板** 四个页签。
2. 切换到 **工作流模板** 页签，找到 **人类演示数据视觉预处理与质量清洗** 模板。您可以单击卡片查看模板详情。

3. 单击卡片右下角的 **创建工作流**，在弹窗中配置以下参数。

   * **工作流名称**：仅支持大小写字母、数字、`-`、`_`、`.`，例如 `p1-visual-clean-demo`。
   * **资源池**：选择前提条件中准备的全托管资源池。选择后，控制台会自动校验 `aihc-argo-workflows` 组件；如提示组件状态异常，请先联系管理员处理。
   * **定时运行**：保持关闭。
   * **描述**：选填。

4. 单击 **确定**。页面提示“创建工作流成功”后，自动进入工作流编辑器。

步骤二：获取队列 ID 和 PFS ID

1. 在工作流编辑器中，单击 **查看队列信息**。
2. 在弹窗中复制 PFS ID，以及要使用的 CPU 队列的队列 ID（格式为 `aihcq-xxxx`）。弹窗中会显示每个队列的 CPU 和内存空闲量。请确认空闲资源足够运行一个节点（默认 Head 4 核 / 8 GB + Worker 6 核 / 12 GB）。

步骤三：修改全局参数

在工作流编辑器中修改以下全局参数。

参数	**填写内容**	**配置示例**
`queue_id`	上一步复制的队列 ID	`aihcq-xxxxxxxx`
`pfs_id`	上一步复制的 PFS ID	`pfs-xxxxxx`
`tmp_dir`	PFS 上的中间数据目录，须以 `/mnt/pfs/` 开头	`/mnt/pfs/<YOUR-PATH-NAME>/p1/demo-001/tmp`
`output_file`	PFS 上的最终输出文件	`/mnt/pfs/<YOUR-PATH-NAME>/p1/demo-001/clips.jsonl`

配置示例中的 `<YOUR-PATH-NAME>` 请替换为您在 PFS 上的目录。

在当前线上版本中，您可以任选以下一种方式修改：

* **可视化模式**：在配置面板的 **总览信息 > 全局输入参数** 表单中，填入队列 ID（`queue_id`）、PFS ID（`pfs_id`）等参数。

* **Yaml 模式**：直接修改顶部 `inputs` 中对应参数的 `defaultValue`。

> **说明**
>
> 工作流参数分为全局参数和节点参数：
> * **全局参数**：被多个节点共同引用，修改一处即全部生效。例如本例中的 `tmp_dir` 被 6 个节点共同引用。
> * **节点参数**：仅作用于所在节点，例如本例中 step6 的 `clarity_min`。在可视化模式下，单击节点后在 **子任务信息 > 输入参数** 中修改；在 Yaml 模式下，修改 `tasks` 中该节点的 `inputs.value`。
>

本例的 6 个节点使用同一套资源参数。在实际管线中，如果某个节点需要 GPU（例如接在 P1 之后的 HaWoR 手部重建），只需在该节点的资源配置中添加卡型和卡数，其他节点不受影响。数据量较大时，调大 `worker_replicas` 即可横向扩展。

步骤四：保存并运行

1. 单击右上角的 **保存**。
2. 单击 **运行**（运行前会自动保存一次）。页面提示“运行成功”后，将跳转到该工作流的运行记录。

步骤五：查看运行状态、日志和监控

1. 在运行记录中单击本次运行，查看 6 个子任务的运行状态、持续时间和结束时间。子任务按 DAG 顺序执行，上一个子任务成功后，下一个才会开始。

2. 在子任务所在行的 **操作** 列，可以打开该子任务的 **监控** 和 **日志**。

3. 6 个子任务全部成功，即表示整条管线已跑通。

步骤六：查看结果

使用挂载**同一个 PFS 实例**的开发机查看输出结果。开发机的 PFS 挂载配置要求如下，详细操作请参见[开发机存储挂载与数据集挂载](https://cloud.baidu.com/doc/AIHC/s/dmojisum4)。

* PFS ID：与工作流全局参数 `pfs_id` 一致。
* 源路径：`/`。
* 目标路径：`/mnt/pfs`。

1. 在开发机终端中执行以下命令，查看输出目录和结果文件。

```bash
ls /mnt/pfs/<YOUR-PATH-NAME>/p1/demo-001/           # clips.jsonl 和 tmp/
cat /mnt/pfs/<YOUR-PATH-NAME>/p1/demo-001/clips.jsonl
```
> **说明**
>
> 开发机 WebIDE 左侧默认只显示 `/root/workspace` 目录，看不到 `/mnt/pfs`。您可以通过菜单 **File > Open Folder** 打开 `/mnt/pfs/<你的目录>/p1`，或在终端执行 `ln -s /mnt/pfs/<你的目录>/p1 /root/workspace/p1` 创建软链接。



2. 查看 `clips.jsonl` 的内容。文件中每行是一个通过过滤的片段，示例如下：

```json
  {"clip": "/mnt/pfs/zhangsan/p1/demo-001/tmp/clips/clip_000.mp4", "duration": 2.0, "clarity": 835.5, "motion": 0.4000000059604645}
```
各字段说明如下。

字段	**说明**
`clip`	片段视频的路径，下游 P2、P3 直接读取该文件
`duration`	片段时长，单位为秒
`clarity`	采样帧 Laplacian 方差的平均值
`motion`	运动量的标准化分数，取值范围 0~1

* `clarity` 值越大画面越清晰，没有上限，仅适合在同一批数据内部比较。
* `motion` 越接近 0，画面越接近静止。

中间文件保存在 `tmp/` 目录下。完整的统计值（如 `clarity` 的 median/max/min、`motion` 的 `motion_pattern` 等）保存在 `s5_motion.jsonl` 中。

使用自有数据运行管线

完成样例数据的运行后，您可以将输入替换为自己的视频数据。

准备 manifest 文件

manifest 文件描述待处理的视频清单，每条记录对应一个视频，包含以下字段。

字段	**是否必填**	**说明**
`video`	是	视频路径，支持 PFS 路径或 BOS 路径
`intrinsics`	去畸变时必填	3×3 相机内参矩阵
`distortion`	否	畸变系数

manifest 支持 JSON 数组格式，也支持每行一个对象的 JSONL 格式。JSONL 格式示例如下：

```echarts-json
{"video":"/mnt/pfs/<YOUR-PATH-NAME>/p1/input/videos/clip_000.mp4","intrinsics":[[640.0,0.0,320.0],[0.0,640.0,240.0],[0.0,0.0,1.0]],"distortion":[0.0,0.0,0.0,0.0,0.0]}
```
选择数据存放方式

根据数据存放位置，从以下两种方式中选择一种。

方式	**数据存放位置**	**访问方式**	**需要修改的内容**	**适用场景**
方式一：数据放在 PFS	PFS	挂载，路径为 `/mnt/pfs/...`	仅参数	推荐，性能好，改动最小
方式二：数据放在 BOS	自己的 BOS bucket	通过 BOSFS 挂载为目录	参数和 `datasources`	没有 PFS、数据量不大

> **注意**
>
> 方式二通过 FUSE 将 BOS 映射为目录，使用最简单，但在小文件多、随机读写多的场景下，性能明显低于 PFS。此外，BOSFS 写文件时在 close 后才上传，且不支持回退写；写 MP4 这类需要 seek 的文件时，需先写到本地 `/tmp`，再拷贝到挂载目录。

方式一：数据放在 PFS

1. 将视频和 manifest 文件上传到 PFS，例如 `/mnt/pfs/<YOUR-PATH-NAME>/p1/input/`。您可以通过挂载同一 PFS 实例的开发机上传。
2. 修改工作流参数：

    * 将 `input_file` 修改为 `/mnt/pfs/<YOUR-PATH-NAME>/p1/input/manifest.json`。
    * 为 `tmp_dir`、`output_file` 换一个新的批次号。
    * 如需去畸变，将 `no_undistort` 修改为 `"false"`，并在 step1 的节点参数中将 `camera_model` 设置为 `pinhole` 或 `fisheye`。

3. 先用小样本验证。建议先用 5~20 条样本运行一遍，查看 `clips.jsonl` 中 `clarity`、`motion` 的分布，再调整 step6 的三个阈值。确认效果后再处理全量数据；数据量较大时，调大 `worker_replicas`。

方式二：数据放在 BOS

将您自己的 bucket 挂载到容器中，输入、中间结果和输出都放在 BOS 上，不再使用 PFS。

1. 确认资源池已支持 BOS 挂载（需安装 BOSFS），详情请参见[在训练任务中使用云存储](https://cloud.baidu.com/doc/AIHC/s/bmol5rql0)。
2. 在 Yaml 模式中，将**每个节点**（6 个 `taskTemplates`）的 `datasources` 修改为以下写法：保留平台公共 BOS（其中存放管线脚本），将 PFS 替换为您自己的 bucket。

```yaml
    datasources:
```

```
      - type: bos                    # 平台公共 BOS，脚本和样例数据，保持不变
        name: ""
        sourcePath: '{{pipeline.parameters.bos_source_path}}'
        mountPath: '{{pipeline.parameters.bos_mount_path}}'
      - type: bos                    # 自己的 bucket，替换原来的 PFS 挂载
        name: ""
        sourcePath: <your-bucket>/<prefix>
        mountPath: /mnt/bos
```

3. 修改工作流参数：将 `input_file`、`tmp_dir`、`output_file` 都修改为 `/mnt/bos/...` 下的路径。此时 `pfs_id` 等 PFS 相关参数已不再被引用，可以保留或删除。

搭建 Genesis World 仿真环境并生成数据

一、简介

Genesis World 是一个面向机器人与具身智能的**通用物理仿真平台**。可**进行 GPU 物理仿真、相机渲染和仿真数据生成**。

核心能力：

* 多物理求解器：刚体、MPM、PBD（布料/弹性体）、FEM、SPH，并支持相互耦合
* 大规模并行：单卡可同时仿真上万个环境
* 无头 GPU 渲染：基于 EGL 离屏输出 RGB / 深度 / 分割图与 mp4 视频，无需桌面或 RDP
* 开箱即用：预置 Franka 抓取数据生成示例、v1.4.5 官方示例全集和一键回归脚本

典型使用场景：

* 批量生成机器人操作数据（关节状态、动作、相机图像），用于模仿学习或 VLA 模型训练
* 大规模并行强化学习训练
* 软体、布料、流体等可变形物体的交互仿真

二、快速开始

步骤 1：环境准备与开发机创建

推荐硬件配置

配置项	要求
GPU	NVIDIA GPU，驱动 ≥ 570（需支持 CUDA 12.8）
操作系统	镜像内置，无需选择
存储	系统盘 ≥ 50 GB（镜像内环境约 10 GB）

1.1 创建开发机

登录百舸控制台，进入 开发机 → 【创建开发机】。

基本信息：

配置项	填写说明
实例名称	自定义，如 `genesis-dev`
资源池类型/资源池/队列	选择有 GPU 配额的资源池和队列
资源规格	参考上表，至少 1 张 GPU
云磁盘	建议 ≥ 50 GB

访问配置 & 高级配置：

* 保持默认即可

点击【确定】，等待创建完成。

1.2 登录开发机

在开发机列表中找到刚创建的实例，状态变为「运行中」后点击【登录】，进入终端。默认工作目录为 `/root/workspace`。

步骤 2：激活环境

每打开一个新终端，执行一次：

```bash
#### 1. 激活 Genesis 运行环境（Python 虚拟环境 + 无头 EGL 渲染配置）
source /opt/genesis/env.sh

#### 2. 进入交付目录
cd /root/workspace/genesis
```

> **注意**：`/opt/genesis` 下的 Python 环境路径是写死的，请勿移动或重命名该目录。

步骤 3：运行抓取数据生成示例

该示例在 GPU 上仿真 Franka 机械臂抓起方块，同时用相机录制视频，并逐步记录关节状态、动作和图像。

```bash
python gs_smoke.py
```

示例（自定义输出目录、分辨率与帧率），将上述命令替换为：

```bash
python gs_smoke.py --out ./out/run01 --res 1280 720 --fps 50
```

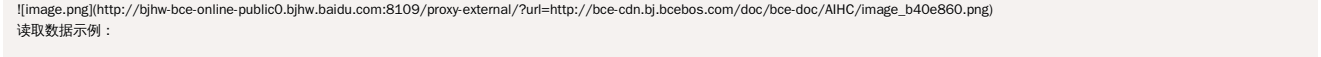
> **预期运行时间**：约 15 秒。如果内核编译缓存被删除，首次运行需要额外约 2 分钟重新编译，之后会恢复正常。

运行效果

终端最后一行打印 `SMOKE_RESULT {...}`，其中 `"lifted": true` 表示方块已被成功抬起。产出保存在 `out/smoke/`：

文件	内容

`episode.mp4`	640×480、25 fps 的第三人称相机视频
`episode.npz`	每步的 `qpos` `qvel` `cube_pos` `action`，以及相机的 `rgb` / `depth`（深度 4 倍降采样）
`meta.json`	运行摘要：步数、帧数、耗时、方块最终高度、`lifted` 是否成功等


读取数据示例：

```python
import numpy as np
d = np.load("out/smoke/episode.npz")
print({k: d[k].shape for k in d.files})  # qpos: (410, 9)，rgb: (103, 480, 640, 3) ……
```
步骤 4：运行官方示例

`examples/` 目录是 Genesis World v1.4.5 的官方示例原版，未做任何改动。部分官方示例写死了打开可视化窗口（`show_viewer=True`），开发机没有显示器，需要先开启 pyglet 的无头开关，窗口会改为在 GPU 上离屏创建。

```bash
#### 1. 激活环境（新终端需执行）
source /opt/genesis/env.sh

#### 2. 开启无头模式（无显示器时必加，否则会报 No display detected）
export PYGLET_HEADLESS=1

#### 3. 进入官方示例目录（示例产出会写到 examples/out/）
cd /root/workspace/genesis/examples

#### 4. 运行官方快速开始示例
python tutorials/hello_genesis.py
```
示例（GPU 运行 Franka 并录制视频），将上述 #4 替换为：

```bash
python rigid/single_franka_envs.py -g -r
```
> **注意**：带 `-g/--gpu` 参数的示例默认走 CPU，请务必加上 `-g`。
运行效果

终端会持续打印仿真帧率，如 `Running at 795.57 FPS`，示例结束后自动退出。录像类示例的视频保存在 `examples/out/` 下。

> **提示**：开启可视化窗口的示例会按实时速率运行（`dt=0.01` 时约 100 FPS），这是正常现象，不是性能问题。

三、进阶功能

3.1 并行环境批量生成数据

构建场景时指定环境数量，即可在一张 GPU 上同时仿真多个环境，所有状态张量的第一维为环境编号：

```python
source /opt/genesis/env.sh
mkdir -p /root/workspace/genesis/out && cd /root/workspace/genesis

cat > parallel_demo.py <<'EOF'
import time

import genesis as gs

gs.init(backend=gs.gpu, logging_level="warning")
scene = gs.Scene(show_viewer=False)
scene.add_entity(gs.morphs.Plane())
franka = scene.add_entity(gs.morphs.MJCF(file="xml/franka_emika_panda/panda.xml"))
scene.build(n_envs=256)  # 256 个并行环境

t0 = time.time()
for _ in range(1000):
    scene.step()
dt = time.time() - t0
print("qpos 形状:", tuple(franka.get_qpos().shape))  # (256, 9)
print(f"吞吐: {256 * 1000 / dt:,.0f} env-step/s")
EOF

python parallel_demo.py
```
> **建议**：可以先用 `speed_benchmark/franka.py` 测出您机器上的吞吐，再确定环境数量。
3.2 相机渲染与录像

默认无界面运行，通过相机出图和录像：

```python
source /opt/genesis/env.sh
mkdir -p /root/workspace/genesis/out && cd /root/workspace/genesis

cat > camera_demo.py <<'EOF'
import genesis as gs

gs.init(backend=gs.gpu, logging_level="warning")
scene = gs.Scene(sim_options=gs.options.SimOptions(dt=0.01), show_viewer=False)
scene.add_entity(gs.morphs.Plane())
scene.add_entity(gs.morphs.MJCF(file="xml/franka_emika_panda/panda.xml"))
cam = scene.add_camera(res=(640, 480), pos=(2.0, -1.2, 1.3), lookat=(0.5, 0.0, 0.2), fov=40)
scene.build()  # 相机必须在 build 之前添加

cam.start_recording(save_to_filename="out/demo.mp4", fps=25)
for i in range(300):
    scene.step()
    if i % 4 == 0:  # dt=0.01、fps=25 → 每 4 步出一帧
        rgb, depth, seg, normal = cam.render(rgb=True, depth=True)
cam.stop_recording()
print("rgb:", rgb.shape, "depth:", depth.shape)  # (480, 640, 3) (480, 640)
```

```
print('rgb', rgb.shape, 'depth', depth.shape)"
EOF

python camera_demo.py
```

> **注意**：v1.4.5 中视频文件名和帧率要传给 `start_recording`，`stop_recording()` 不带参数。帧率需满足 `1/(fps×dt)` 为整数，`dt=0.01` 时可选 10 / 20 / 25 / 50 / 100，其他值会被自动调整。

四、关键参数说明

抓取数据生成示例参数（gs_smoke.py）

参数	必选	说明	默认值	示例
`--out`	否	产出目录	`out/smoke`	`./out/run01`
`--res`	否	相机分辨率（宽 高）	`640 480`	`1280 720`
`--fps`	否	视频帧率，需满足 `1/(fps×0.01)` 为整数	`25`	`50`
`--cpu`	否	改用 CPU 后端（仅用于排查问题，速度很慢）	关闭	`--cpu`

环境变量

变量	设置方式	说明
`PYOPENGL_PLATFORM=egl`、`MUJOCO_GL=egl`	`env.sh` 自动设置	强制使用 EGL 无头渲染
`XDG_CACHE_HOME=/opt/genesis/cache`	`env.sh` 自动设置	资产和内核编译缓存目录
`PYGLET_HEADLESS=1`	**需手动 export**	让写死 `show_viewer=True` 的示例在无显示器环境下运行；以后接入远程桌面查看交互窗口时不要设置

官方示例常用参数

参数	说明	适用示例
`-g` / `--gpu`	使用 GPU 后端（不加默认 CPU）	`rigid/single_franka_envs.py`、`coupling/*`、`deformable/*` 等
`-r` / `--record`	录制视频到 `examples/out/`	`rigid/single_franka_envs.py`
`-b` / `--num-envs`	并行环境数	`rigid/single_franka_envs.py`、`rigid/single_franka_batch_render.py`
`-s` / `--steps`	仿真步数	`rigid/single_franka_envs.py`、`rigid/single_franka_batch_render.py`
`-v` / `--vis`	打开可视化窗口（无显示器时需配合 `PYGLET_HEADLESS=1`）	大部分示例

五、常用示例列表

> 以下命令均在 `/root/workspace/genesis/examples` 目录下运行，需先执行 `source /opt/genesis/env.sh` 和 `export PYGLET_HEADLESS=1`。

入门教程：

命令	说明
`python tutorials/hello_genesis.py`	官方快速开始，地面 + Franka 的最小场景
`python tutorials/control_your_robot.py`	位置 / 速度 / 力控制
`python tutorials/visualization.py`	相机录像，产出 `out/visualization.mp4`
`python tutorials/parallel_simulation.py`	20 个并行环境
`python tutorials/IK_motion_planning_grasp.py`	逆运动学 + 运动规划抓取
`python tutorials/batched_IK.py`	16 个环境批量逆运动学

刚体：

命令	说明
`python rigid/franka_cube.py`	Franka 抓方块
`python rigid/single_franka_envs.py -g -r`	GPU 运行并录像，产出 `out/single_franka_envs.mp4`
`python rigid/domain_randomization.py`	8 个环境随机化质量（域随机化）
`python rigid/single_franka_batch_render.py -b 4 -s 30`	多环境批量渲染，产出 `out/batch_render/`

可变形体与耦合：

命令	说明
`python coupling/rigid_mpm_attachment.py -g`	MPM 弹性体挂在机械臂上
`python coupling/cloth_on_rigid.py -g`	PBD 布料落在刚体上
`python deformable/elastic_dragon.py -g`	PBD 弹性体
`python coupling/fem_cube_linked_with_arm.py -g`	FEM 软块 + 机械臂，产出 `out/cube_link_arm_*.mp4`

传感器与性能：

命令	说明
`python sensors/imu_franka.py`	IMU 传感器，产出 `out/imu_data.npz`
`python sensors/camera_as_sensor.py`	相机传感器，产出 `out/camera_as_sensor/*.png`
`python speed_benchmark/franka.py`	3 万个并行环境吞吐测试

六、常见问题（FAQ）

Q1：运行报错 `GenesisException: No display detected`**，怎么办？**

示例写死了打开可视化窗口，而开发机没有显示器。执行 `export PYGLET_HEADLESS=1` 后重新运行即可，无需修改示例源码。

Q2：部分 tutorials 示例（如 **`mpm.py`、**`pbd_cloth.py`**、**`sph_liquid.py`**）一直跑不完？**

这几个官方示例在源码中写死了 CPU 后端，CPU 上不到 2 FPS。请改用同类求解器、支持 `-g` 参数的示例，如 `coupling/rigid_mpm_attachment.py -g`、`coupling/cloth_on_rigid.py -g`。

Q3：开启可视化窗口的示例只有约 100 FPS，是性能问题吗？

不是。打开窗口时 Genesis 按实时速率运行（`dt=0.01` 即每秒 100 步）。需要全速运行时，请在脚本中设置 `show_viewer=False`。

Q4：首次运行很慢，卡在编译阶段？

Genesis 首次运行会编译 GPU 内核，约 2 分钟。镜像已预置编译缓存（`/opt/genesis/cache`），正常情况下约 15 秒即可启动。如果删除了该目录，首次运行会重新编译，之后恢复正常。

Q5：录像时报错 `TypeError: Camera.stop_recording() got an unexpected keyword argument`**？**

v1.4.5 的录像接口有变化：文件名和帧率需传给 `cam.start_recording(save_to_filename=..., fps=...)`，`cam.stop_recording()` 不带参数。网上部分旧教程的写法已不适用。

Q6：可以安装其他 Python 包吗？

可以。执行 `uv pip install -p /opt/genesis/venv/bin/python <包名>`。请勿升级 `torch`、`genesis-world` 等核心依赖，完整版本清单见 `/root/workspace/genesis/requirements.lock`。

Q7：目录里都有什么？

路径	说明
`/root/workspace/genesis/gs_smoke.py`	抓取数据生成示例
`/root/workspace/genesis/examples/`	v1.4.5 官方示例原版
`/root/workspace/genesis/regress.sh`	一键回归脚本
`/root/workspace/genesis/install.sh`	环境重装脚本
`/root/workspace/genesis/requirements.lock`	依赖版本锁定清单
`/root/workspace/genesis/out/`	示例和回归的产出
`/opt/genesis/`	Python 环境与编译缓存（请勿移动）

通过网页方式可视化连接 MuJoCo

MuJoCo介绍

MuJoCo 全称 Multi-Joint dynamics with Contact，是一个物理仿真引擎，主要用于机器人、生物力学和强化学习研究。它的特点是计算快、接触力学模拟准确，适合需要大量仿真的场景，比如训练控制策略。

MuJoCo 仿真可视化镜像预置 MuJoCo Studio、MuJoCo Web Viewer 和 mjlab Cartpole 示例。用户可以通过百舸开发机入口，在浏览器中快速查看仿真画面、编辑并保存 XML 场景，也可以在终端启动 Web Viewer、训练和评估示例。

本文介绍如何创建开发机、打开 MuJoCo Studio、管理显示模式、编辑保存场景，以及运行镜像内置的 Web Viewer 和 Cartpole 示例。

部署环境要求

组件	部署要求
CPU	按任务选择
内存	按场景规模选择
GPU	GPU 显示模式需要支持 Vulkan 图形渲染和 NVENC 硬件编码的 NVIDIA GPU
训练资源	Cartpole 训练和回放示例需要 NVIDIA CUDA 训练环境
浏览器	支持 WebSocket 的最新版 Chrome 或 Edge

使用须知

- GPU 模式下自动检测同时检查 Vulkan/NVIDIA 渲染和 NVENC 硬件编码。
- 原生 Studio 和 Web Viewer 使用 MuJoCo 版本与mjlab 训练环境使用 MuJoCo / MuJoCo Warp版本相互独立。

1. 通过快速开始创建开发机

1.1 创建开发机

1. 在百舸控制台，快速开始下找到该卡片【通过网页方式可视化连接 MuJoCo】
2. 在卡片中，点击【通过开发机打开】
3. 根据您的需求选择开发机配置，MuJoCo Studio 可以通过 CPU 进行渲染，因此不强制依赖显卡，但流畅度会受到影响
4. 在开发机状态为运行中后，点击登录进入开发机

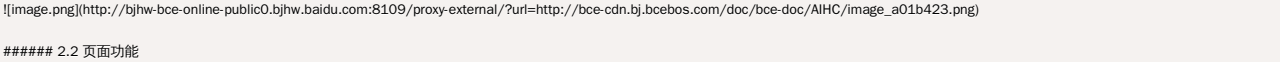
2. 进入 MuJoCo 可视化界面

2.1 启动 MuJoCo Studio

新开发机或新容器首次启动时，镜像会自动检测 GPU 渲染和硬件编码能力，并启动 Studio。打开 **MuJoCo Studio** 卡片即可进入页面。

如果用户在同一开发机中手动停止了 Studio，之后重启 Jupyter 不会再次自动拉起；如果页面提示 Studio 尚未启动，请在 JupyterLab Terminal 中执行：

```bash
sim-studio start --auto
```

启动完成后，页面会自动显示原生 MuJoCo Studio 编辑器。关闭浏览器标签页或 Terminal 不会停止 Studio。
请注意：如果您挂载了显卡，请确认页面左上角渲染模式为【GPU】，如果显示为【CPU】，请参考 2.3 命令与配置在终端中尝试重启或参考 2.5 常见问题咨询客服。


2.2 页面功能

功能	说明
原生编辑器	使用 MuJoCo Studio 的菜单、3D 视口和 Editor 编辑场景
打开模型	在 Studio 中选择**File → Open Model File**，打开工作区中的 XML 文件
编译场景	在**View → Editor** 中编辑后，单击 **Compile and Reload** 使修改生效
保存 XML	选择**File → Save XML**，将编译后的模型保存到工作区
恢复窗口	页面中的【恢复窗口】按钮或`sim-studio restore`，恢复最小化窗口，不重启编辑器
Web Viewer	在 Terminal 运行`sim-example web`，按命令输出的链接打开浏览器 Viewer
Cartpole 回放	在 Terminal 运行`sim-example cartpole play`，打开训练策略的可视化回放

原生 Studio 保存的是模型定义。运行时的位置、速度、仿真时间和编辑器撤销记录不会写入 XML。关闭页面前请先进行保存。

2.3 命令与配置

Studio 启动和模式管理

命令	说明
`sim-studio status`	查看 Studio 状态、当前模式和保存偏好；不输出浏览器访问地址
`sim-studio start --auto`	自动检测 GPU 渲染和编码能力，检测通过使用 GPU，否则使用 CPU
`sim-studio start --cpu`	使用软件渲染和 CPU 视频编码
`sim-studio start --gpu`	强制使用 GPU；检测未通过时拒绝启动，不自动回退 CPU
`sim-studio restart --auto --saved`	在确认场景已保存后重启，并重新自动检测模式
`sim-studio stop --saved`	在确认场景已保存后停止 Studio
`sim-studio restore`	恢复最小化窗口，不重启 Studio

模式偏好会持久化。后续执行不带模式参数的 `sim-studio start` 时，会沿用上次选择；需要恢复自动检测时，明确使用 `--auto`。

`--saved` 只表示用户已经完成保存确认，不会自动保存场景，也不会检查编辑器是否存在未保存修改。

Studio 已运行时，`start` 不会替换当前实例。重启或切换模式会关闭当前 Studio，执行前先在原生 Studio 中选择 **File → Save XML**。

切换模式或加载其他场景时，使用 `restart`：

```bash
sim-studio restart --auto --saved \
  --model /root/workspace/my-scene/scene.xml
```

Web Viewer 和 Cartpole 示例

命令	说明
`sim-example web`	启动官方 MuJoCo 4.0 Web Viewer，默认打开 humanoid 示例；命令保持运行并输出访问链接
`sim-example web --model /root/workspace/my-scene/scene.xml`	使用指定 XML 启动 Web Viewer，这个路径请填写自己文件的路径
`sim-example cartpole play`	打开预置 Cartpole 策略的 Viser 回放
`sim-example cartpole train --iterations 500 --num-envs 1024`	训练 Cartpole 策略并生成 checkpoint
`sim-example cartpole play --checkpoint /实际输出目录/model_499.pt`	回放指定 checkpoint，这个路径请填写自己文件的路径
`sim-example cartpole evaluate --num-envs 128 --steps 1000`	使用默认 checkpoint 评估策略并输出 JSON 结果
`sim-example cartpole evaluate --checkpoint /实际输出目录/model_499.pt --num-envs 128 --steps 1000`	评估指定 checkpoint 并输出 JSON 结果

Web Viewer 和 Cartpole 命令需要保持对应的 Terminal 运行；按 Ctrl+C 结束本次 Viewer 或回放。Viewer 可以与 Studio 同时运行。

Web Viewer 默认加载 humanoid 示例并处于暂停状态，页面提供 Run、Pause 和 Reset 操作。该模型没有控制策略，运行后倒下属于预期行为。保存 XML 后，Web Viewer 会监视文件并自动重新加载；如果进程已结束，再重新执行启动命令。

Cartpole 示例使用 mjlab 官方任务，训练、回放和评估都要求 CUDA 训练环境。未指定 checkpoint 时，`play` 和 `evaluate` 使用镜像预置的示例 checkpoint；训练完成后请使用命令输出的实际 checkpoint 路径。Studio 选择 CPU 模式不会把 Cartpole 训练切换为 CPU。修改 Cartpole 模型时，请使用自包含 XML；该示例不接受 include 或外部资产，并且新模型仍需要匹配任务的观测、动作和奖励定义。

2.5 常见问题

Q1：页面显示 Studio 没有启动？

在 JupyterLab Terminal 中执行：

```bash
sim-studio start --auto
```

然后返回 MuJoCo Studio 页面等待连接。首次启动可能需要等待初始化完成。

Q2：有 GPU，但 Auto 进入了 CPU 模式？

Auto 要求 Vulkan/NVIDIA 渲染和 NVENC 硬件编码同时通过。显卡可见或 CUDA 可用，不代表 NVENC 可用。继续使用 CPU 模式时无需再次启动；如果要切换模式，请先在 Studio 中保存 XML，再执行：

```bash
sim-studio restart --gpu --saved \
  --model /root/workspace/my-scene/scene.xml
```

如果只是明确切换到 CPU，可将 `--gpu` 替换为 `--cpu`。GPU 模式仍要求图形渲染和 NVENC 检测同时通过。如果必须使用 GPU 模式，请联系平台管理员检查实例的图形驱动、容器图形能力和 NVENC 支持。

Q3：执行 `sim-studio start --gpu` 提示检测未通过？

这表示 GPU 渲染或硬件编码至少有一项未通过。GPU 强制模式会拒绝启动，不会自动回退到 CPU。确认实例环境后，可临时使用 `--auto` 或 `--cpu`。

Q4：修改场景后重新打开，修改没有保留？

在 Editor 中修改后，先执行 **Compile and Reload**，再执行 **File → Save XML**。`sim-studio stop --saved` 和 `sim-studio restart --saved` 都不会代替保存操作。

Q5：需要加载自己的 XML，应该如何上传？

将 XML 及其纹理、网格和 include 文件一起上传到 `/root/workspace` 下，并保持相对路径。然后在 Studio 中通过 **File → Open Model File** 打开，或执行：

```bash
sim-studio restart --auto --saved \
  --model /root/workspace/my-scene/scene.xml
```

Q6：Web Viewer 页面没有打开或模型没有更新？

确认 Terminal 中的 `sim-example web` 进程仍在运行，并打开该命令输出的最新链接。Web Viewer 会监视已保存的 XML，保存后通常会自动重新加载；如果进程已经结束或页面仍未更新，再按 Ctrl+C 结束旧 Viewer，并重新执行命令。

Q7：Cartpole 训练无法启动？

Cartpole 训练、回放和评估都使用 CUDA，并要求使用示例规定的自包含 XML 和任务结构。请确认当前开发机有可用 NVIDIA CUDA 资源；Studio 的 CPU/GPU 显示模式不改变训练设备。评估指定策略时，可以追加 `--checkpoint /实际输出目录/model_499.pt`。

Q8：窗口被最小化后看不到编辑器？

点击页面中的【恢复窗口】，或在 Terminal 中执行：

```bash
sim-studio restore
```

该操作只恢复窗口，不会重启 Studio 或改变当前场景。

常见问题

资源池常见问题

> 本文为您介绍资源池的相关问题。

目录

- [一、资源池创建 & 网络配置](#一资源池创建--网络配置)
- [二、节点管理](#二节点管理)
- [三、资源队列与 GPU 虚拟化](#三资源队列与-gpu-虚拟化)
- [四、预留实例券](#四预留实例券)
- [五、集群预安装组件](#五集群预安装组件)

> **名词说明**：本文涉及以下缩写。
> - **资源池**：百舸 AI 计算平台中算力资源的组织单元，分为全托管资源池与自运维资源池。
> - **全托管资源池**：平台提供的免运维资源池，用户无需关注底层物理资源和算力插件的运维管理。
> - **资源队列**：算力分配调度的最小单元，工作负载（开发机、训练任务、推理服务）需提交到队列以获得算力。
> - **MIG**：Multi-Instance GPU，NVIDIA 提供的硬件级 GPU 切分能力。
> - **预留实例券**：计费权益与资源锁，用于锁定库存资源并抵扣同规格按量付费节点的费用。

一、资源池创建 & 网络配置

Q：全托管资源池具有什么优势？

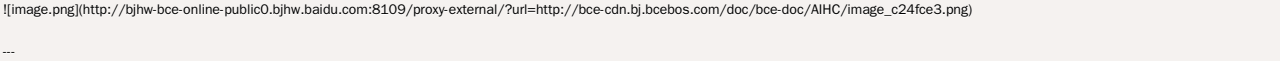
全托管资源池是百舸 AI 计算平台提供的免运维资源池。用户无需关注底层物理资源和算力插件的运维管理，由平台保障稳定性。

Q：创建资源池时，网络配置有哪些限制？

- 必须配置 VPC 网络、节点子网和容器子网。
- 容器子网不能与平台预留网段 `11.0.0.0/12` 冲突。

Q：可以在资源池创建后添加不同可用区的百舸节点吗？

支持在存量资源池中跨可用区新建节点。如需添加不同可用区的节点，请先添加对应可用区的容器子网，再在该可用区新建节点。



二、节点管理

Q：百舸节点的"封锁"和"释放"有什么区别？

- **封锁**：节点进入不可调度状态，新的 Pod 不会调度到该节点上，但存量业务不受影响。
- **释放**：节点停止使用并停止计费。仅支持"按量付费"和"已到期"的包年包月节点。

三、资源队列与 GPU 虚拟化

Q：为什么创建了资源池还要创建资源队列？

资源队列是算力分配调度的最小单元。所有工作负载（开发机、训练任务、推理服务）必须提交到对应的队列中才能获得算力分配。

Q：什么是 GPU 虚拟化？创建后可以更改吗？

GPU 虚拟化支持将单张物理 GPU 切分给多个容器使用：

- 开启后支持按显存切分，允许给容器分配小数卡（如 0.5 卡）。
- 队列创建后，该设置**不可修改**。且节点和容器镜像需适配特定的 OS、驱动和 CUDA 版本。

Q：开启虚拟化后支持碎片治理吗？

开启 **MIG 虚拟化**的队列和节点不支持平台侧的自动碎片治理功能。

Q：可以在一个任务中同时使用 MIG 实例和普通物理 GPU 吗？

不可以。任务只能提交至单一类型的队列中，无法跨越开启了 MIG 的队列和普通队列进行联合计算。

四、预留实例券

Q：购买了预留实例券，还需要单独去开通机器吗？

需要。预留实例券是计费权益和资源锁，不是机器本身。购买实例券后，您需要前往计算页面创建按量付费的同规格节点，系统会自动将两者关联进行抵扣。

Q：对于预留实例券，如果把机器释放了，实例券还收费吗？

收费。即使您释放了实例，实例券依然在为您锁定库存资源，因此会正常计算有效期和费用。

Q：预留实例券可以随意变更 EHC 集群吗？

不支持变更。

Q：预留实例券支持跨地域（Region 级）抵扣吗？

暂不支持。目前仅支持可用区级（AZ 级）的严格匹配。

五、集群预安装组件

Q：百舸集群默认预安装了哪些组件？

百舸集群默认预安装以下组件，覆盖 GPU 调度、分布式训练、高性能网络、存储与日志等能力：

预安装组件名称	说明
CCE-CPU-Manager	CPU 架构感知等高级调度，支持 CPU 算力和显存的共享与隔离，详见 https://cloud.baidu.com/doc/CCE/s/(1kp80bcb4)

CCE GPU Manager	GPU 架构感知等高级调度，支持 GPU 算力和显存的共享与隔离。详见 https://cloud.baidu.com/doc/CCE/s/1kp8o0cb4
CCE AI Job Scheduler	适配 AI 分布式训练场景，支持批量调度、队列管理等。详见 https://cloud.baidu.com/doc/CCE/s/akp81ro8j
CCE Deep Learning Frameworks Operator	提供分布式模型训练、部署等功能，支持 TensorFlow、Pytorch、Mxnet、PaddlePaddle 框架。详见 https://cloud.baidu.com/doc/CCE/s/Dkp814hyo
CCE RDMA Device Plugin	高性能实例之间的网络通信，提供高性能网络。详见 https://cloud.baidu.com/doc/CCE/s/Qkp81mtzj
CCE CSI PFS Plugin	支持在 Pod 中以 PV/PVC 方式挂载百度云 PFS 存储，并支持动态创建 PV。详见 https://cloud.baidu.com/doc/CCE/s/slc792j5c
CCE Log Operator	采集集群中的日志信息传输至日志服务（BLS）中进行持久化存储。详见 https://cloud.baidu.com/doc/CCE/s/Glc8zpk34

开发机常见问题

> 本文为您介绍开发机的相关问题。

目录

- [一、开发机创建 & 启动](#一开发机创建--启动)
- [二、开发机管理与使用](#二开发机管理与使用)
- [三、镜像管理](#三镜像管理)
- [四、挂载配置](#四挂载配置)
- [五、网络问题](#五网络问题)
- [六、第三方库安装](#六第三方库安装)
- [七、实例运行](#七实例运行)
- [八、卡顿问题](#八卡顿问题)
- [九、相关文档](#九相关文档)

> **名词说明**：本文涉及以下缩写。
> - **BLB**：负载均衡（Baidu Load Balance），开发机 SSH 与自定义服务的流量入口。
> - **PFS / CFS / BOS**：并行文件存储 / 文件存储 / 对象存储，均为可挂载的外部存储。
> - **CCR**：容器镜像服务（Baidu Cloud Container Registry），分个人版 / 企业版。
> - **C-prom**：托管 Prometheus 监控服务，"停止实例通知"功能依赖其实例绑定。

一、开发机创建 & 启动

Q：开发机为什么依赖云磁盘？对云磁盘的依赖和关联有哪些？

开发机目前依靠云磁盘作为数据持久化载体。挂载云磁盘后，开发机数据会被自动保存，实例关机、异常等情况下数据不会丢失。

由于物理节点网卡特性，对云磁盘的挂载数量有限制，这决定了单个物理节点上可创建的、支持数据持久化的开发机数量（开发机本质是物理节点上的容器实例）。

因此在选购节点套餐时，建议优先选择云磁盘配额高的节点。

Q：如果节点不支持挂载云磁盘怎么办？

- **情况 1：队列中部分节点支持挂载云磁盘、部分不支持。**
  在全局云磁盘选项未关闭时，开发机默认只会在支持挂载云磁盘的节点上创建并调度。如需允许队列中所有节点创建无云磁盘挂载的开发机，管理员可按以下步骤设置：
  1. 进入 **资源池 → 全局配置 → 开发机配置**。
  2. 设置 **是否强制挂载云磁盘**。
  3. 关闭该选项后，开发机可在不支持挂载云磁盘的节点上创建并调度。

  > ⚠ 未挂载云磁盘的开发机默认**不支持数据持久化**，请谨慎设置。

- **情况 2：队列中所有节点都不支持挂载云磁盘。**
  该队列只允许创建无云磁盘挂载的开发机。此场景下，建议通过共享存储实现部分数据持久化：
  - 创建开发机时同时挂载 PFS，并将工作目录挂载到 PFS 上；
  - 存放在 PFS 上的代码和数据将被持久化。

Q：开发机创建表单的部分选项如何理解？

- **环境变量**：出于服务稳定性考虑，自定义环境变量不允许与系统环境变量重名；若重名，表单会提醒并要求修改。
- **绑定 BLB**：开发机 SSH 和自定义服务通过 BLB 作为流量入口。初次使用开发机需预先创建一个 BLB。BLB 只能被同一个集群绑定，每个自定义服务默认占用一个 BLB 端口和 IP 组配额；配额占满后 BLB 无法被绑定。
- **停止实例通知**：开启此选项需资源池完成 C-prom 实例绑定。

Q：开发机长时间排队的常见原因有哪些？

- 队列节点资源不足（CPU / GPU / 内存）；
- 队列节点云磁盘已挂满；
- BLB 配额不足（IP 组或端口在创建时间窗口内被抢占打满，常见于选择了配额临近打满的 BLB）；
- 账号欠费。

> 可通过时间线，或点击开发机名称进入事件页面，查看具体报错详情。

Q：开发机实例启动后长时间处于"部署中"怎么办？

排查方法：

1. 查看时间线报错提示；
2. 单击开发机实例名称，在"事件"页签下查看报错信息。

常见原因：

- 资源账号欠费，导致云磁盘创建失败；
- 关联资源被释放（如已绑定的 PFS 实例在开发机启动前被删除或释放）；
- 镜像拉取失败（网络问题、镜像仓库用户名密码错误、镜像本身问题等）；
- 节点系统盘空间不足（容器数据目录容量不足，无法拉取镜像）。

二、开发机管理与使用

Q：开发机启动时可以开机执行 Python 文件吗？

可以。您可以在创建开发机实例时，或通过变更实例配置的方式，设置自定义启动脚本。

Q：找不到开发机实例怎么办？

请在导航栏切换不同地域，查看不同创建地、不同类型的实例。

Q：为什么启动开发机实例的时间越来越长？

每次启动时间逐渐变长，通常是因为您保存的镜像体积较大所致。

Q：停止或删除开发机实例后，数据和代码会丢失吗？

是否保留取决于您的操作以及实例是否挂载了云盘：

- **停止实例**：包含云盘系统盘的实例，停止时数据已持久化到云盘，数据不会丢失；
- **删除实例**：云盘将被释放，数据会被清除。

Q：为什么正在运行的开发机实例会自动停止？

可能有两类原因：

- 实例配置了闲时关机、低利用率关机、自动关机等策略，用于节省资源、提升集群利用率；
- 若开发机对队列其他成员可见，也可能被其他成员操作停止。

Q：使用开发机时，软件包、代码和数据推荐如何管理？

总体原则：**运行环境装在开发机内，代码和数据存在外部存储中**。具体建议：

- 数据集、模型权重等数据，推荐存储在外部存储（PFS、CFS、BOS 等）并挂载使用，不建议拷贝到开发机内部；
- 软件包（如 conda）推荐安装到开发机内；除非希望在多个开发机或训练环境间共享，才装到外部存储；
- 不需要打包镜像的场景，推荐代码也存外部存储；若需中途打包镜像，可将代码当前版本拷贝到开发机内。

三、镜像管理

Q：制作镜像时，哪些数据会被打包进镜像？

- 所有从外部挂载的存储卷（PFS、BOS、CFS 等）**都不会**被打包进镜像；
- 如需将外部存储里的数据打包进镜像，请事先（或在打包前）规划好存储路径。

Q：镜像拉取失败的常见原因有哪些？

- 镜像仓库用户名 / 密码输错；
- 镜像地址错误、镜像错误、镜像不存在等；
- 网络异常。

Q：为什么制作开发机镜像时会失败或超时？

- **镜像大小超限**：单层镜像保存的数据量不应超过 **40 GiB**，否则构建失败。镜像可写层过大（如放入大容量数据集）是最常见原因。建议仅保留必要的代码与环境，数据集存到 PFS 等共享存储；可通过清理云磁盘、将非必要数据转移到共享存储来降低可写层大小。
- **地域不匹配**：开发机实例与容器镜像服务 CCR 实例必须位于同一地域，否则制作镜像时找不到对应的镜像仓库。
- **系统盘空间不足**：剩余可用空间小于待写入镜像层的数据大小时，会报 `insufficient capacity of ephemeral storage` 错误。
- **网络问题**：个人版 CCR 通过公网推送镜像，较大镜像可能因网络波动或传输超时而失败；企业版 CCR 若与开发机在同一 VPC，可走内网推送，更快更稳定。

四、挂载配置

> 本节示例统一使用挂载路径 `/mnt/workspace/pfs/data`（工作目录下）与 `/mnt/pfs/data`（工作目录外）做对比说明。

Q：开发机实例如何挂载和使用自己的文件系统？

您可以在新建实例时挂载 PFS / CFS / BOS，然后通过开发机 Terminal 进入挂载目录查看并使用文件。

> 目前开发机仅支持在**创建实例时**挂载**同地域**下的文件系统。

Q：成功挂载了 PFS，为什么在 JupyterLab 左侧文件浏览器里看不到？

这是因为文件浏览器默认显示实例的工作目录（路径一般为 `/mnt/workspace`）。若挂载路径（如 `/mnt/pfs/data`）不在工作目录下，就不会显示在左侧列表中。可通过下列任一方式解决：

- **通过代码访问**：文件其实已挂载成功，在代码中使用完整挂载路径即可，例如 `open('/mnt/pfs/data/my_file.csv')`。
- **更改挂载点**：配置挂载时把挂载路径设到工作目录的子目录下，例如 `/mnt/workspace/pfs/data`，挂载后即可在文件浏览器中看到。
- **通过终端访问**：在 Terminal 中 `cd /mnt/pfs/data`，再用 `ls` 等命令查看和操作文件。

Q：两个开发机实例之间如何互传、分享数据？

让两个实例挂载**相同的**文件系统路径（PFS / CFS / BOS / 数据集），将数据存放在该路径下即可实现共享。

五、网络问题

Q：开发机实例如何访问公网，或被公网访问？

- **访问公网**：参见[《容器实例开启访问公网》](https://cloud.baidu.com/doc/AIHC/s/Jm3rdwin7)。
- **被公网访问**：开发机可通过绑定已挂载公网 IP 的 BLB 实现被公网访问，参见[《公网访问开发机实例》](https://cloud.baidu.com/doc/AIHC/s/smjshd4u8)。

六、第三方库安装

Q：如何在开发机中安装并使用第三方库？

在开发机 Terminal 中执行以下命令：

```bash
#### Python 3
pip install --user xxx

#### Python 2
source activate python2
pip install --user xxx
```

将 `xxx` 替换为待安装的库名。安装成功后，单击 **kernel → restart kernel** 重启服务。

Q：为什么安装的第三方包没有生效？

`import` 时找不到包，先尝试重启服务；若仍报错，确认当前使用的 Python 环境是否与安装时一致。

Q：pip install 报依赖冲突或版本错误怎么办？

通常由环境不兼容导致，按以下顺序排查：

1. **更换镜像（首选）**：停止当前实例，新建实例时选择一个不同的官方镜像。
2. **指定版本安装**：查阅库的官方文档，选一个适配当前环境（Python / CUDA 版本）的版本：
   ```bash
   pip install package_name==x.y.z
   ```
3. **更换下载源**：使用国内镜像源，例如清华源：
   ```bash
   pip install -i https://pypi.tuna.tsinghua.edu.cn/simple package_name
   ```

Q：Terminal 里装好了库，为什么 Jupyter Notebook 里 import 找不到？

可能是 Terminal 和 Jupyter 使用了不同的 Python 环境。用 `which python` 确认当前环境，或直接在 Notebook 中安装所需库。

Q：开发机里没有 unzip 或 7z 命令，如何解压文件？

通过 `apt-get` 安装：

```bash
#### 安装 unzip
apt-get update && apt-get install -y unzip
unzip your_file.zip

#### 安装 p7zip（用于 7z）
apt-get update && apt-get install -y p7zip-full
7z x your_file.7z
```

Q：用 pip 安装时一直卡住或超时怎么办？

通常是网络问题：

1. 检查实例是否配置了公网访问。若创建时选了"无公网访问"，或配置了 VPC 但未设置 NAT 网关，将无法连接外部下载源。
2. 更换下载源，例如切换到清华源：
   ```bash
   pip install -i https://pypi.tuna.tsinghua.edu.cn/simple <包名>
   ```
3. 若网络确实不通，可在本地电脑下载好 `.whl` 安装包，上传到开发机离线安装。

Q：可以在开发机里用 Docker 部署应用吗？

> **关于 Docker in Docker**：开发机本身运行在容器中，目前**暂不支持 Docker in Docker**，该功能已在规划中，请持续关注。

七、实例运行

Q：关闭浏览器或电脑关机后，开发机中运行的训练任务还会继续吗？

会继续。开发机实例运行在云端，关闭本地设备不影响其运行状态。

> ⚠ 注意：部分实例可能配置了**闲置自动关机策略**。若 CPU / GPU 等资源在一段时间内持续低于阈值，可能被判定为闲置并自动停止，从而中断任务。

Q：如何 Debug 代码？

- **Debug Notebook 代码**：Notebook 天然适合调试，每个 Cell 单独执行即可即时看到输出，也可用 `print()` 输出关键变量值。
- **Debug Python 代码**：在 WebIDE 中使用 VSCode 标准的断点与调试功能。

Q：用 VSCode 客户端通过 SSH 连接开发机，频繁提示 OOM 并断开怎么办？

开发机负载很低却频繁断开、连上几分钟就报 OOM，通常是**客户端 VSCode（remote server）侧的 OOM**，而非开发机后端问题。

1. **不要直接打开超大目录**：避免在 VSCode Explorer 中直接展开或打开 PFS 挂载的数据集目录（如 `/media` 下文件量巨大的目录），大量文件会撑爆 remote server 内存。
2. **调大 remote server 预留内存**：修改用户维度 VSCode remote server 的预留内存。
3. **回滚 VSCode 版本**：若问题在升级 VSCode 后才出现，可尝试回滚到此前可用的版本。

> 排查提示：报错弹窗若来自 VSCode 客户端，且开发机后端 code-server 服务正常（无重启等异常），即可判定为客户端环境问题。

Q：开发机实例规格不满足要求怎么办？

按以下步骤更新实例规格：

1. 在开发机实例列表中，找到 **操作 → 更多**。
2. 单击 **变更配置**。
3. 在变更配置面板中更新实例规格。

> 说明：变更规格时若实例正在运行，会重建开发机；资源不足时会排队等待空闲资源。如有数据和环境需保存，请先通过**制作镜像**保存环境。

Q：开发机关机（停止）后，pip 安装的包和代码会丢失吗？

若使用云盘作为系统盘则不会丢失，实例磁盘数据（含 `/mnt/workspace` 和 `/root` 下的环境）都会保留，下次启动时环境和文件都还在。只有**删除实例**才会彻底清空所有数据。

八、卡顿问题

Q：使用 WebIDE（网页版 VSCode）或本地 VSCode 通过远程连接开发机时出现卡顿，如何排查？

开发机负载正常却出现 WebIDE / VSCode 卡顿，常见原因是 VSCode 的 Git 扩展递归扫描了工作目录下的 PFS、CFS、BOS、RapidFS、NFS 等远端 / 网络挂载存储，导致对远端存储元数据服务产生大量访问请求，延迟叠加后表现为开发机整体变慢。CDS 云盘、本地盘挂载后为本地块设备，不受此问题影响。

如何判断是否由此导致：

1. 检查工作目录：在 WebIDE 终端执行 `pwd`，并查看左侧文件树根目录，确认当前打开的是否为 `/root`、`/mnt`、`/data` 等包含远端存储挂载点的父目录。
2. 确认挂载点：执行 `mount | grep -Ei 'pfs|cfs|bos|rapidfs|nfs'` 或 `df -h | grep -Ei 'pfs|cfs|bos|rapidfs|nfs'`，查看远端存储挂载路径是否落在当前工作区内。
3. 观察扫描现象：文件树、搜索、Git 面板长时间加载或刷新，但终端执行普通命令正常，是典型表现；也可执行 `ps -ef | grep -Ei 'code-server|vscode|rg|git'` 查看相关进程是否持续活跃。
4. 验证：执行 `File > Close Folder`，重新 `File > Open Folder` 仅打开代码所在子目录（不包含远端存储挂载点的父目录），若卡顿消失即可确认。

处理方式： [开发机 WebIDE 或本地 IDE 卡顿：避免扫描共享存储目录](https://cloud.baidu.com/doc/AIHC/s/kmrym1fbz)

如上述步骤仍无法确认卡顿原因，请联系客服协助排查。

Q：使用开发机 WebIDE 或本地 IDE（如VSCode）通过 **Remote SSH 方式**远程连接开发机时出现卡顿

开发机正常运行但出现 WebIDE / 本地 IDE 卡顿，一般原因为 IDE 的 Git 扩展递归扫描了工作目录下的文件树，触发对共享存储（PFS、CFS、BOS、RapidFS 等百度云存储）元数据服务大量并发请求，延迟叠加后表现为开发机响应变慢。

如何判断是否由以上原因导致：

1. 检查工作目录：在 WebIDE 终端执行 `pwd`，查看左侧文件树根目录，确认当前打开的是否为 `/root`、`/mnt`、`/data` 等包含共享存储挂载点的父目录。
2. 确认挂载点：执行 `mount | grep -Ei 'pfs|cfs|bos|rapidfs|nfs'` 或 `df -h | grep -Ei 'pfs|cfs|bos|rapidfs|nfs'`，查看共享存储挂载路径是否包含在当前工作区内。
3. 观察扫描现象：文件树、搜索、Git 面板长时间加载或刷新，但终端执行普通命令正常，是典型表现；也可执行 `ps -ef | grep -Ei 'code-server|vscode|rg|git'` 查看相关进程是否持续活跃。
4. 验证：执行 `File > Close Folder`，重新 `File > Open Folder` 仅打开代码所在子目录（不包含共享存储挂载点的父目录），若卡顿消失即可确认。

处理方式： [开发机 WebIDE 或本地 IDE 卡顿：避免扫描共享存储目录](https://cloud.baidu.com/doc/AIHC/s/kmrym1fbz)

如上述步骤仍无法确认卡顿原因，请联系百度云技术支持人员协助排查。

九、相关文档

- [《容器实例开启访问公网》](https://cloud.baidu.com/doc/AIHC/s/Jm3rdwin7)
- [《公网访问开发机实例》](https://cloud.baidu.com/doc/AIHC/s/smjshd4u8)
- [《开发机 WebIDE 或本地 IDE 卡顿：避免扫描共享存储目录》](https://cloud.baidu.com/doc/AIHC/s/kmrym1fbz)

在线服务部署常见问题

> 本文为您介绍在线服务部署过程中的常见问题。

目录

- [一、推理服务部署与启动](#一推理服务部署与启动)
- [二、模型运行与资源配置](#二模型运行与资源配置)

> **名词说明**：本文涉及以下缩写。
> - **AIAK**：百度 AI 加速套件（AI Accelerator Kit），用于推理 / 训练加速。
> - **PFS**：并行文件存储（Parallel File Storage），可挂载的外部高性能存储。
> - **POD**：Kubernetes 中的最小调度单元，一个推理副本通常对应一个 POD。
> - **OOM**：内存溢出（Out Of Memory）。

一、推理服务部署与启动

Q：首次启动多副本推理实例时，只有一个副本能成功，其他副本返回失败是什么原因？

当您使用 AIAK 加速引擎进行推理服务部署时，首次需要进行模型权重转换，该过程需要一段时间（具体时长取决于模型参数大小，参数越大转换越久）。在此期间，多副本中的其他 POD 会不断进行失败重试，直到模型权重转换完成后恢复正常。

> ⚠ 这是首次部署时的预期行为，无需手动干预，待权重转换完成后其余副本会自动启动成功。

Q：模型权重转换中途失败，再次启动时报错，应该如何解决？

当推理服务部署过程中，因 OOM、POD 被驱逐等因素导致模型权重转换失败后，需要手动清理 PFS 中的 `/mnt/model/triton` 文件夹，然后重新进行推理服务部署。

二、模型运行与资源配置

Q：模型进行较长上下文输入输出时，最后的输出出现大量乱码是什么原因？

Llama 1 系列模型单次最大 token 支持 2048（包括输入和输出）。超出 2048 限制后，模型会出现乱码等异常情况，请注意控制输入和输出的 token 长度。

Q：单副本 POD 为什么不支持使用多张 A800 GPU 卡运行推理服务？

建议按照百舸推理服务的推荐配置相关资源，使用 1 张 A800 GPU 卡进行 Llama 7B 或 13B 模型的推理服务部署。

> ⚠ 若为 Llama 7B / 13B 这类参数量相对较小的模型配置多卡运行，由于引入多卡间的通信损失，反而可能带来性能下降。

Q：部署 Llama 7B 和 13B 模型推理服务的 CPU 和内存默认推荐规格是什么？

- **Llama 7B**：默认推荐规格为 12C CPU 和 40GB 内存；
- **Llama 13B**：默认推荐规格为 24C CPU 和 80GB 内存。

服务等级协议SLA

AIHC全托管资源池服务等级协议SLA

协议生效时间：2026年8月19日

本服务等级协议（Service Level Agreement，以下简称“SLA”）规定了百度智能云向客户提供百度百舸·AI计算平台（AI Heterogeneous Compute，简称“AIHC”）全托管资源池的服务可用性等级指标。除非另有约定，本协议不适用于AIHC内测、公测、免费的资源池类型、功能、版本。

1. 定义

1.1 百度百舸·AI计算平台AIHC：指百度智能云提供的面向大规模深度学习的高性能云原生AI计算平台，为模型算法专家和运维专家提供全面的集群运维支持和任务全生命周期管理，同时具备训练/推理加速、故障容错、智能故障诊断等高级功能。

1.2 服务周期：一个服务周期为一个自然月。

1.3 单节点服务周期总分钟数：按照单节点服务周期内的总天数 × 24（小时）× 60（分钟）计算。

1.4 节点不可用：因百度智能云故障导致了，一台设置了出入允许规则百舸节点以TCP或者UDP协议与任一IP地址的双向（出/入）都无法连通，且该状态持续一分钟以上，视为该分钟内百舸节点不可用。

1.5 单节点服务周期内不可用分钟数：一个服务周期内AIHC单节点计入不可用分钟数之和。

1.6 单节点单地域多可用区服务不可用分钟数：在一个服务周期内，单百舸节点的单地域多可用区服务不可用的分钟数之和。

1.7 月度服务费用：为客户在一个服务周期（即自然月）内单个百舸全托管资源池节点所支付的服务费用总额，如客户一次性支付多个月的服务费，则将按照所购买的月份分摊计算月度服务费。

2. 服务可用性

2.1 服务可用性计算方式

百舸节点的服务可用性将根据服务周期，按如下维度分别统计每个百舸节点的可用性：

单节点服务可用性 = (单节点服务周期总分钟数 - 单节点服务不可用分钟数) / 单节点服务周期总分钟数 × 100%

2.2 服务可用性承诺

百度智能云承诺一个服务周期内百舸节点的服务可用性不低于99.9%；如AIHC服务未达到上述服务可用性承诺，客户可以根据本协议第三条约定获得赔偿。

赔偿范围不包括以下原因所导致的不可用：

(1) 百度智能云预先通知客户后进行系统维护所引起的，包括割接、维修、升级和模拟故障演练；

(2) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的；

(3) 应用程序或数据信息受到黑客攻击而引起的；

(4) 客户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的；

(5) 客户的疏忽或由客户授权的操作所引起的；

(6) 不可抗力以及意外事件引起的；

(7) 百舸服务只赔付百舸自身，不赔付百舸实例所关联的其他云服务；

(8) 其他非百度智能云原因所造成的百舸服务无法正常使用。

(9)百度云预先通知用户后，用户未按提示进行相关任务操作所产生的时间不计入赔付时间范围；

(10) 由于客户所安装软件或者其他非百度智能云直接运营的第三方软件或者配置引起的实例出现错误；

(11) 本地盘导致的不可用（使用本地盘存储的实例的本地磁盘中数据有丢失的风险，比如本地磁盘损坏）；

(12) 使用本地盘存储的实例、带有异构卡（GPU/NPU/FPGA等）的实例因硬件等故障等待客户授权及维修期间造成的服务不可用；

(13)由于账户欠费/实例到期导致百舸节点被暂停服务或被释放等；

(14) 由于客户违反《百度智能云服务协议》导致的百舸节点被暂停或终止；

(15) 属于相关法律法规、相关协议、相关规则或百度智能云单独发布的相关规则、说明等中所述的百度智能云可以免责、免除赔偿责任等的情况；

(16) 客户未遵循百度智能云产品使用文档或使用建议引起的，如：客户手动重启、删除AIHC核心系统组件；客户自行通过一些手段修改或攻击AIHC服务，如安全攻击、压力测试等操作引起的不可用。

3. 赔偿方案

3.1 赔偿标准

每个百舸全托管资源池节点按单节点月度服务可用性，按照下表中的标准计算赔偿金额，赔偿方式仅限于用于购买AIHC产品的代金券，且赔偿总额不超过未达到服务可用性承诺当月，客户就该全托管资源池节点支付的月度服务费用（不含用代金券抵扣的费用）。

服务可用性	赔付代金券金额
低于99.9%但是等于或高于99%	2倍超出服务可用性标准的不可用分钟数
低于99%但等于或高于95%	3倍超出服务可用性标准的不可用分钟数
低于95%	4倍超出服务可用性标准的不可用分钟数

3.2 赔偿申请时限

客户可以在每月的第五（5）个工作日后对上个月发生故障的百舸全托管资源池节点提出赔偿申请，赔偿申请必须限于在相关节点故障的相关月份结束后的两（2）个月内提出。超出申请时限的赔偿申请将不被受理。

4. 其他说明

* 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。
* 本协议一经公布立即生效，百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云将以网站公示或发送邮件等方式通知客户。如客户不同意百度智能云对SLA所做的修改，客户有权停止使用本服务，如客户继续使用本服务，则视为客户接受修改后的SLA。
* 本协议项下百度智能云对于客户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该等通知于发送之日视为已送达收件人。因客户自身未及时获知百度智能云的服务变更或终止条款遭受损失的，百度智能云不承担**任何责任**。
* 本协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。
* 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。

* 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍应有效并且有约束力。
* 关于用户约束条款，详见[《用户服务协议》](https://cloud.baidu.com/doc/Agreements/s/mjwvy1waw)中的"用户的权利与义务"相关条款内容。
* 关于服务商免责条款，详见[《用户服务协议》](https://cloud.baidu.com/doc/Agreements/s/ujwvy1vjn)中的"免责声明"相关条款内容。