# 高性能计算集群

# HyperAcc 训练加速套件

腾讯云

# 文档目录

# HyperAcc 训练加速套件
# 新功能发布记录

最近更新时间：2026-08-14 18:27:00

本文为您介绍腾讯云 HyperAcc 训练加速套件的版本更新、能力变化和重要公告。发布时间以产品版本或功能发布时间为准。

## 2026年07月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 环境变量可视化配置页面发布 | 新增环境变量可视化配置能力，支持点选操作并自动生成最简配置命令；同时支持通过功能开关独立管控各类算子的启用与禁用。 | 2026-07-06 |

## 2026年05月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 升级激活入口 | 升级自动激活入口、透明替换加载机制和 Python 3.8 兼容性，提升训练任务接入稳定性。 | 2026-05-06 |

## 2026年04月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 产品接入体验优化 | 完善 wheel 安装包接入说明，明确环境变量配置和 `import hyperacc.auto` 的推荐使用流程。 | 2026-04-27 |
| 安全启用策略增强 | 未设置环境变量时，`import hyperacc.auto` 默认不做任何操作；您可按需显式启用指定能力，便于灰度接入和快速回退。 | 2026-04-23 |
| 功能开关体系升级 | 统一采用 `HA_USE_*` 功能开关，支持全部启用、按能力启用和单项覆盖，提升配置一致性。 | 2026-04-15 |
| 环境变量配置向导上线 | 提供环境变量配置向导，帮助按训练场景生成推荐配置命令，降低首次接入成本。 | 2026-04-15 |
| 内存格式优化能力增强 | 增强 `channels_last` 相关优化能力，适配更多卷积和归一化训练场景。 | 2026-04-14 |

## 2026年03月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 海光 DCU 支持增强 | 增强 DCU 训练加速能力，覆盖更多大模型、多模态和检测任务中的常用计算场景。 | 2026-03-30 |
| 多卡资源亲和性优化 | 优化多卡训练场景下的 CPU/NUMA 资源匹配策略，提升训练任务运行稳定性。 | 2026-03-30 |
| Auto-Compile 能力增强 | 优化自动编译策略，提升对复杂 PyTorch 模型结构的适配能力，降低调优成本。 | 2026-03-25 |
| 大模型训练场景扩展 | 增强 LLM/VLM 训练相关优化能力，提升大模型和多模态模型训练场景覆盖范围。 | 2026-03-17 |
| 内存布局优化能力上线 | 新增训练过程中的内存布局优化能力，适用于卷积、归一化等相关模型结构。 | 2026-03-04 |

## 2026年02月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 海光 DCU 适配增强 | 提升 DCU 的设备识别和系统亲和性适配能力。 | 2026-02-28 |
| GC 优化策略扩展 | 新增更灵活的 GC 优化策略，帮助长时间训练任务降低垃圾回收带来的运行抖动。 | 2026-02-26 |
| NUMA 绑核稳定性优化 | 优化单卡、多卡和不同启动方式下的 NUMA 绑核行为，提升系统级优化稳定性。 | 2026-02-09 |
| NVIDIA GPU 训练场景兼容性优化 | 优化 NVIDIA GPU 相关训练场景的兼容性，提升安装包在不同环境中的可用性。 | 2026-02-04 |

## 2026年01月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 系统级优化能力升级 | 支持通过环境变量统一管理 GC、NUMA 绑核和透明大页等系统级优化能力。 | 2026-01-29 |
| 透明大页配置优化 | 优化透明大页启用和恢复策略，降低多进程训练场景下的配置冲突风险。 | 2026-01-29 |
| DataLoader 调优能力增强 | 完善 DataLoader 参数控制能力，支持通过环境变量管理 worker 数、预取批次和 pin memory。 | 2026-01-28 |

| | | |
|---|---|---|
| 安装包平台适配增强 | 提升安装包对不同 NVIDIA GPU 训练环境的适配能力，扩大可交付环境范围。 | 2026-01-23 |
| 训练循环调优能力扩展 | 新增面向训练循环的调优能力，支持在更多训练框架场景中使用 HyperAcc。 | 2026-01-13 |
| 图形与几何计算场景扩展 | 新增 rasterizer 相关优化能力，扩展图形、几何和感知类训练任务覆盖范围。 | 2026-01-06 |

## 2025年12月

| 动态名称 | 动态描述 | 发布时间 |
|---|---|---|
| 图形与几何算子优化增强 | 新增 rasterizer 相关优化能力，为依赖图形或几何计算的训练任务提供加速支持。 | 2025-12-29 |
| 分布式训练与数据加载能力增强 | 完善 SyncBatchNorm 和 DataLoader 相关能力，提升分布式训练和数据加载场景适配能力。 | 2025-12-28 |
| 透明替换能力上线 | 新增透明替换能力，支持在少量改造训练入口的情况下启用部分加速能力。 | 2025-12-18 |
| DataLoader 优化能力上线 | 新增 DataLoader 调优能力，支持通过环境变量统一管理数据加载参数。 | 2025-12-18 |
| 自动驾驶 3D 感知场景扩展 | 新增点云和 3D 感知相关训练加速能力，面向自动驾驶感知模型训练场景。 | 2025-12-11 |
| 训练数据处理优化扩展 | 新增地图线采样等数据处理优化能力，提升相关自动驾驶训练流程的适配范围。 | 2025-12-09 |

## 2025年11月

| 动态名称 | 动态描述 | 发布时间 |
|---|---|---|
| 系统内存优化能力上线 | 新增透明大页配置能力，用于改善大内存训练场景下的系统内存分配表现。 | 2025-11-30 |
| Auto-Compile 能力上线 | 新增基于 PyTorch 编译能力的自动编译功能，帮助降低模型编译接入成本。 | 2025-11-19 |
| GC 优化能力上线 | 新增 GC 优化能力，帮助训练任务降低 Python 垃圾回收带来的运行抖动。 | 2025-11-11 |

# HyperAcc 训练加速套件概述

最近更新时间：2026-08-14 18:27:00

## 产品概述

腾讯云 HyperAcc 训练加速套件是面向 PyTorch 训练任务的高性能训练加速组件。安装已编译的 HyperAcc wheel 包后，您可通过环境变量启用不同加速能力，并在训练入口 Python 脚本开头导入 `import hyperacc.auto` 完成自动激活。

HyperAcc 采用显式启用策略：默认情况下，未设置任何环境变量时，`import hyperacc.auto` 不执行任何操作，不会引入额外行为。您可通过 `HA_USE_*` 环境变量按需显式启用指定能力。

该策略支持在生产训练环境中进行灰度接入和按场景启用，同时具备快速回退能力，在保障现有训练流程稳定性的同时，降低新能力接入风险。

## 核心能力

| 能力 | 说明 | 适用场景 |
| --- | --- | --- |
| 无侵入接入 | 通过 `import hyperacc.auto` 和环境变量启用能力，减少对原训练逻辑的改动。 | 需要快速接入已有 PyTorch 训练脚本。 |
| 透明替换 | 透明替换等能力的实际启用取决于模型结构、PyTorch 版本及运行环境；不满足条件或发生不兼容时，会跳过并保持原有设置。建议先在测试环境验证后再用于生产训练。 | 在兼容场景下，无需改造训练框架，即可按需启用算子及流程优化能力 |
| 算子加速 | 提供 NVIDIA GPU 与 DCU 的部分算子优化能力。 | 3D 检测、多模态训练、LLM/VLM 训练等计算密集场景。 |
| 系统级优化 | 提供 GC 优化、CPU/NUMA 绑核和透明大页配置能力。 | 多卡训练、大内存训练、对训练时延稳定性敏感的任务。 |
| 自动编译 | 基于 `torch.compile` 对 PyTorch 模型进行自动编译和粒度控制。自动编译的能力的实际启用取决于模型结构、PyTorch 版本及运行环境；不满足条件或发生不兼容时，会回退到原有的模式。建议先在测试环境验证后再用于生产训练。 | 希望利用 PyTorch 2.x 编译能力提升训练或推理性能。 |
| DataLoader 优化 | 调整 `pin_memory`、`num_workers`、`prefetch_factor` 等参数。 | 数据加载成为训练瓶颈或需要统一 DataLoader 参数的任务。 |

> ⓘ **注意：**

> 自动编译、透明替换等能力的实际启用取决于模型结构、PyTorch 版本及运行环境；不满足条件或发生不兼容时，具体跳过、回退或报错行为以日志及产品文档为准。建议先在测试环境验证后再用于生产训练。

## 产品优势

- 有助于减少接入成本：以 wheel 包形式交付，您安装后即可在训练入口导入使用。
- 有助于减少训练脚本改造：通过环境变量控制功能开关，保留原训练脚本主体结构。
- 主要场景：可用于自动驾驶、3D 检测、多模态、LLM/VLM 等训练场景的部分优化。
- 便于灰度和回退：功能默认关闭，单项能力可独立启用或关闭。
- 提升资源利用效率：通过 CPU/NUMA 绑核、GC 调优、DataLoader 调优等方式，优化训练过程中的系统开销。

## 应用模型

- 自动驾驶（Autonomous Driving）：BEVFormer、BevFusion、Sparse4D
- 具身智能（VLA Model）：PI、PI0.5、lingbot-va
- 语言模型：Qwen、QwenVL 系列

> ⓘ **注意：**
> - 文中涉及的第三方产品、软件、框架、模型及商标名称归其各自权利人所有，仅用于说明 HyperAcc 的适配或使用场景，不构成任何认可、合作、担保或背书。
> - 已验证或可参考的模型包括 BEVFormer、BevFusion、Sparse4D、PI、PI0.5、lingbot-va、Qwen、QwenVL 系列。具体支持的模型版本、算子能力和效果以交付 wheel 包及产品文档为准。

## 应用场景

### 自动驾驶模型训练

HyperAcc 提供点云体素化、动态散射、多相机特征聚合、光栅化、SyncBatchNorm 等能力，适用于 3D 检测、BEV 感知、多传感器融合等自动驾驶训练任务。

### 大模型和多模态模型训练

HyperAcc 提供 RMSNorm、RoPE、Dropout、GELU 等部分 GPU/DCU 优化能力，并支持基于 torch.compile 的自动编译，可用于大模型和多模态模型训练中的算子与模型执行优化。

### 多卡分布式训练

HyperAcc 可根据可见 GPU、进程 rank 和 NUMA 拓扑进行 CPU 亲和绑定，有助于减少跨 NUMA 访存带来的训练抖动。

### 数据加载优化

HyperAcc 可通过环境变量统一调整 DataLoader 参数，适合数据加载瓶颈明显或容器环境参数不一致的训练任务。

## 支持环境

| 项目 | 说明 |
| --- | --- |
| Python | 支持 Python 3.8 及以上版本，且需要与交付 wheel 包的 Python ABI 匹配。 |
| PyTorch | 支持 PyTorch 2.0 及以上版本，启用自动编译能力时依赖 `torch.compile` 。 |
| GPU 硬件平台 | 支持 NVIDIA GPU 和 DCU，具体能力以交付 wheel 包和环境变量开关为准。 |
| 操作系统 | 面向 Linux 训练环境。透明大页和 NUMA 绑核能力依赖 Linux 系统接口。 |

## 使用限制

- 不同硬件平台支持的算子能力不同，实际启用取决于模型结构、PyTorch 版本及运行环境，不符合当前硬件平台的算子开关会被自动跳过。建议先在测试环境验证后再用于生产训练。
- 配置透明大页要求具备系统写入权限。权限不足场景下，训练进程不会中止，透明大页相关配置无法完成。
- 自动编译依赖模型结构、PyTorch 版本及第三方依赖库兼容性，建议在测试环境完成充分验证后，再应用至生产环境。
- 以上性能提升方向为能力说明，不代表所有场景下的实际表现。实际性能效果可能因模型结构、数据加载、硬件平台和训练配置不同而存在差异，具体以实际使用情况为准。

# 快速入门

最近更新时间：2026-08-14 18:27:00

腾讯云 HyperAcc 训练加速套件（以下简称 HyperAcc）面向 PyTorch 训练任务提供训练加速能力。您获取已编译的 wheel 安装包后，可通过安装 Python 包、配置环境变量、在训练入口导入激活模块的方式接入，在兼容场景下，无需改造训练框架。
本文介绍如何在已获取 HyperAcc wheel 安装包的情况下，完成首次安装和训练任务接入。

## 前提条件

开始前，请确认已具备以下条件：

| 条件 | 说明 |
| --- | --- |
| wheel 安装包 | 已联系 在线客服 获取 `hyperacc-*.whl` 文件。 |
| Python 环境 | Python 3.8 及以上版本，且需要与 wheel 包匹配。 |
| PyTorch 环境 | PyTorch 2.0 及以上版本，且训练脚本已可在当前环境中正常运行。 |
| 系统权限 | 如需启用透明大页，可能需要系统写权限或管理员预配置。 |

## 操作步骤

### 步骤1：安装 HyperAcc

在训练环境中执行以下命令，安装 HyperAcc wheel 包：

```
pip install /path/to/hyperacc-*.whl
```

安装完成后，执行以下命令检查 Python 包是否可正常导入：

```
python -c "import hyperacc.auto; print('hyperacc auto import ok')"
```

### 步骤2：启用加速能力

如需快速体验全部能力，可设置总开关：

```
export HA_USE_ALL=1
```

如需按需启用能力，可设置单项开关：

```
export HA_USE_COMPILE=1
export HA_USE_GC=1
export HA_USE_DATALOADER=1
```

环境变量优先级从高到低依次为：

- `HA_USE_<FEATURE>`：单项功能开关，优先级最高。显式设置为 `0` 时，可覆盖上层总开关的启用状态。
- `HA_USE_ALL_HOOKS`：Hook 类功能总开关。
- `HA_USE_ALL`：全局总开关。
- 默认关闭：未设置任何环境变量时，所有能力默认关闭。

## 步骤3：在训练入口导入

在 Python 训练入口脚本开头导入 `import hyperacc.auto`。除 `from __future__ import ...` 等必须置于文件开头的语句外，建议将导入语句放置在 `torch`、训练框架、模型及 `DataLoader` 等相关模块导入之前，以确保优化能力能够正确生效。

```
import hyperacc.auto

# 保持原有训练逻辑不变
# from your_project import train
# train()
```

## 步骤4：启动训练任务

单卡训练场景下，在启动命令前设置环境变量即可：

```
export HA_USE_ALL=1
python train.py
```

分布式训练场景下，同样在启动命令前设置环境变量：

```
export HA_USE_GC=1
export HA_USE_NUMABIND=1
export HA_USE_DATALOADER=1
torchrun --nproc_per_node=8 train.py
```

## 步骤5：验证是否生效

1. 默认日志级别为 INFO，可输出关键初始化信息，用于初步确认 HyperAcc 是否正常生效。如需获取更详细的运行与调试信息，可将日志级别设置为 DEBUG：

```
export HA_LOG_LEVEL=DEBUG
```

2. 观察训练启动日志中是否出现 HyperAcc 相关初始化信息。初始化日志示例如下：

```
HyperAcc.Sys.CPUAffinity - INFO - Rank 0: GPU 0 -> NUMA 0, L3 [0, 4, 8], CPUs [0, 192, 1, 193, ...], Memory bound
HyperAcc.Sys.CPUAffinity - INFO - Rank 6: GPU 6 -> NUMA 1, L3 [19, 23, 27], CPUs [144, 336, ...], Memory bound
```

3. 检查训练任务是否正常进入训练循环。
4. 如果启用 DataLoader、GC、NUMA 等能力，可结合训练日志、CPU 亲和性或系统监控确认行为变化。

# 常用配置

## 启用全部能力

```
export HA_USE_ALL=1
python train.py
```

## 仅启用系统优化

```
export HA_USE_GC=1
export HA_USE_NUMABIND=1
export HA_USE_THP=1
python train.py
```

## 仅启用自动编译

```
export HA_USE_COMPILE=1
export HA_COMPILE_MODE=default
python train.py
```

## 仅启用 DataLoader 优化

```
export HA_USE_DATALOADER=1
export HA_DATALOADER_WORKERS=16
export HA_DATALOADER_PREFETCH=2
```

```
python train.py
```

# 操作指南
# HyperAcc 训练加速套件常见优化实践

最近更新时间：2026-08-14 18:27:00

完成 HyperAcc wheel 包安装后，您可以参考本文进行常见训练任务的接入与性能调优。

## PyTorch 训练脚本无侵入接入

### 前提条件

- 已安装 HyperAcc wheel 包。
- 原训练脚本可在当前环境中正常运行。

### 注意事项

- 如果训练入口必须包含 `from __future__ import ...` ，请保留这些语句在最前面，再导入 `hyperacc.auto` 。
- 如果先导入了 `torch` 、训练框架、模型或 DataLoader 相关模块，部分透明替换能力可能无法覆盖已加载的目标组件。
- 首次接入生产任务时，建议先按单项能力启用，确认训练结果后再启用全部能力。

### 操作步骤

1. 设置需要启用的功能：

```
export HA_USE_ALL=1
# HA_USE_ALL启动的是无风险项， 有风险的都是默认关闭， 需要手动配置才打开
```

如需按需启用能力，可只设置单项开关：

```
export HA_USE_COMPILE=1
export HA_USE_GC=1
export HA_USE_DATALOADER=1
```

环境变量优先级从高到低依次为： `HA_USE_<FEATURE>` > `HA_USE_ALL_HOOKS` > `HA_USE_ALL` > 默认关闭。其中，单项功能开关 `HA_USE_<FEATURE>` 的优先级最高。显式设置为 0 时，可覆盖上层总开关的启用状态。

2. 在 Python 训练入口脚本开头导入 `import hyperacc.auto` 。除 `from __future__ import ...` 等必须置于文件开头的语句外，建议将导入语句放置在 `torch` 、训练框架、模型及 `DataLoader` 等相关模块导入

之前：

```
import hyperacc.auto

# 原有训练入口
# main()
```

3. 启动训练：

```
python train.py
```

# GC 优化实践

Python GC 在长时间训练和频繁创建临时对象的任务中可能带来停顿。HyperAcc 支持通过环境变量调整 GC 策略。

## 注意事项

- `disable` 模式会关闭 GC，建议仅在充分验证后使用。
- 对内存敏感的任务，应监控内存使用情况，避免长期禁用 GC 导致内存占用升高。

## 操作步骤

```
export HA_USE_GC=1
export HA_GC_MODE=reduce_frequency
python train.py
```

## 参数说明

| 参数 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_GC_MODE` | `reduce_frequency` | GC 策略，取值如下：<br>• `reduce_frequency` ：降低 GC 频率。<br>• `disable` ：禁用 GC。<br>• `custom` ：自定义 GC 策略。 |
| `HA_GC_MONITOR` | `0` 或 `false` | 是否启用 GC 监控。取值如下：<br>• `0` 或 `false` ：禁用 GC 监控。<br>• `1` 或 `true` ：启用 GC 监控。 |

| | | |
|---|---|---|
| `HA_GC_GEN0` | `5000` | 第 0 代 GC 阈值。 |
| `HA_GC_GEN1` | `20` | 第 1 代 GC 阈值。 |
| `HA_GC_GEN2` | `20` | 第 2 代 GC 阈值。 |

# DataLoader 调优实践

HyperAcc 可通过环境变量调整 DataLoader 参数，有助于减少数据加载瓶颈。

## 操作步骤

```
export HA_USE_DATALOADER=1
export HA_DATALOADER_WORKERS=16
export HA_DATALOADER_PREFETCH=2
python train.py
```

## 参数说明

| 参数 | 默认值 | 说明 |
|---|---|---|
| `HA_DATALOADER_NO_PIN_MEMORY` | `0` | 是否禁用 `pin_memory` 。取值如下：<br>• 0：启用 `pin_memory` 。<br>• 1：禁用 `pin_memory` 。 |
| `HA_DATALOADER_WORKERS` | `16` | `num_workers` 数量。负数表示不修改用户配置。 |
| `HA_DATALOADER_PREFETCH` | `2` | `prefetch_factor` ，DataLoader 预取批次数。负数表示不修改用户配置。 |

## 注意事项

- `num_workers` 过大可能增加 CPU 和内存压力。
- 共享内存较小或容器环境异常时，可尝试设置 `HA_DATALOADER_NO_PIN_MEMORY=1` 。

# torch.compile 自动编译实践

HyperAcc 可通过 `HA_USE_COMPILE=1` 启用基于 `torch.compile` 的自动编译能力。

## 操作步骤

```
export HA_USE_COMPILE=1
export HA_COMPILE_BACKEND=inductor
export HA_COMPILE_MODE=default
python train.py
```

## 参数说明

| 参数 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_COMPILE_BACKEND` | `inductor` | 编译后端，支持 `inductor`，`cudagraphs` 和 `eager` 等。 |
| `HA_COMPILE_MODE` | `default` | 编译模式，取值如下：<br>• `default`：默认编译模式。<br>• `reduce-overhead`：降低运行时开销模式。<br>• `max-autotune`：最大自动调优模式。 |
| `HA_COMPILE_FULLGRAPH` | `false` | 是否要求捕获完整计算图。 |
| `HA_COMPILE_DYNAMIC` | 空 | 是否启用动态形状。 |
| `HA_COMPILE_EXPLAIN` | `0` | 是否输出计算图断裂（graph break）诊断信息。 |
| `HA_COMPILE_MAX_PARALLEL` | `2` | 后台并行编译线程数。 |
| `HA_COMPILE_DRILLDOWN_THRESHOLD` | `1000000000` | 大模型分层编译阈值。 |

## 调试建议

如遇到编译失败或训练启动变慢，可先使用默认模式并打开诊断：

```
export HA_USE_COMPILE=1
export HA_COMPILE_MODE=default
export HA_COMPILE_EXPLAIN=1
export HA_LOG_LEVEL=INFO
python train.py
```

# CPU/NUMA 绑核实践

CPU/NUMA 绑核用于将训练进程绑定到 GPU 所在 NUMA 节点的 CPU，有助于降低跨 NUMA 访存开销。

### 操作步骤

```
export HA_USE_NUMABIND=1
python train.py
```

分布式训练中，HyperAcc 会读取 `LOCAL_RANK` 、`LOCAL_WORLD_SIZE` 、`CUDA_VISIBLE_DEVICES` 或 `HIP_VISIBLE_DEVICES` 等环境变量进行映射。

## 透明大页实践

透明大页配置可减少页表开销，有助于提升大内存训练任务的性能。

### 操作步骤

```
export HA_USE_THP=1
python train.py
```

### 注意事项

- 透明大页配置依赖 Linux 系统接口，通常需要写入 `/sys/kernel/mm/transparent_hugepage/` 下的系统配置。
- 权限不足时，对应配置可能无法生效，但不会中断训练。

# 日志与调试

## 透明替换系统参数

| 参数 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_HOOKS_LAZY` | `1` | 目标组件加载时再应用透明替换能力。 |
| `HA_HOOKS_SILENT` | `1` | 减少透明替换过程中的日志输出。 |

## 日志参数

| 参数 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_LOG_LEVEL` | `INFO` | 日志级别，支持 `DEBUG` 、`INFO` 、`WARNING` 、`ERROR` 、`CRITICAL` 。 |

| | | |
|---|---|---|
| `HA_SHIFT_COMPARE` | `0` | 启用 shift patch 对比测试，主要用于调试验证。 |

### 支持的日志级别

| 取值 | 映射到的 Python logging 级别 |
|---|---|
| `DEBUG` | logging.DEBUG |
| `INFO` | logging.INFO |
| `INFO` | logging.WARNING |
| `ERROR` | logging.ERROR |
| `CRITICAL / FATAL` | logging.CRITICAL |

**如需输出详细调试信息：**

```
export HA_LOG_LEVEL=DEBUG
export HA_HOOKS_SILENT=0
python train.py
```

**如需减少日志输出：**

```
export HA_HOOKS_SILENT=1
export HA_LOG_LEVEL=WARNING
python train.py
```

## 透明替换能力

**透明替换能力用于在不改造训练框架的情况下，将部分训练组件替换为 HyperAcc 优化实现。**

### 适用场景

- **自动编译。**
- `channels_last` **内存格式转换。**
- **Lightning Trainer 相关优化。**
- **DataLoader 参数调优。**
- **NVIDIA 或海光 DCU 的部分算子替换。**

### 使用建议

- 先启用单项能力验证训练正确性，再逐步扩大到组合能力。
- 如果训练任务依赖的第三方库版本较特殊，建议在验证环境中先完成回归测试。
- 通过 `HA_LOG_LEVEL=INFO` 或 `DEBUG` 查看初始化信息。

## 透明替换功能开关

| 环境变量 | 默认值 | 平台 | 说明 |
| --- | --- | --- | --- |
| `HA_USE_COMPILE` | 0 | 全平台 | 启用自动编译能力。 |
| `HA_USE_CHANNELS_LAST` | 0 | 全平台 | 启用 `channels_last` 内存格式转换。 |
| `HA_USE_LIGHTNING` | 0 | 全平台 | 启用 Lightning Trainer 相关优化。 |
| `HA_USE_DATALOADER` | 0 | 全平台 | 启用 DataLoader 优化。 |
| `HA_USE_SYNCBN` | 0 | NVIDIA GPU/海光 DCU | 启用 SyncBatchNorm 优化。 |
| `HA_USE_VOXEL` | 0 | NVIDIA GPU | 启用点云体素化和动态散射优化。 |
| `HA_USE_DEFORM_AGG` | 0 | NVIDIA GPU | 启用多相机可变形特征聚合优化。 |
| `HA_USE_RASTERIZER` | 0 | NVIDIA GPU | 启用光栅化相关优化。 |
| `HA_USE_DEFORM_ATTN` | 0 | 海光 DCU | 启用多尺度可变形注意力优化。 |
| `HA_USE_DROPOUT` | 0 | 海光 DCU | 启用 Dropout 融合优化。 |
| `HA_USE_FOCAL_LOSS` | 0 | 海光 DCU | 启用 Softmax Focal Loss 融合优化。 |
| `HA_USE_RMSNORM` | 0 | 海光 DCU | 启用 RMSNorm 优化。 |
| `HA_USE_ROPE` | 0 | 海光 DCU | 启用 RoPE 旋转位置编码优化。 |

| | | | |
|---|---|---|---|
| `HA_USE_FAST_GELU` | 0 | 海光 DCU | 启用 FastGELU 优化。 |

# NVIDIA GPU/DCU 算子启用实践

最近更新时间：2026-08-14 18:27:00

不同硬件平台支持的算子能力不同，建议按任务类型启用对应能力。

## NVIDIA GPU 平台

| 能力 | 环境变量 | 默认值 | 说明 | 适用方向 |
| --- | --- | --- | --- | --- |
| 点云体素化和动态散射 | `HA_USE_VOXEL` | 0 | 启用 NVIDIA 点云体素化相关优化。 | 3D 检测、点云处理。 |
| 多相机可变形特征聚合 | `HA_USE_DEFORM_AGG` | 0 | 启用 NVIDIA 多相机特征聚合优化。 | Sparse4D 等多相机感知任务。 |
| 光栅化 | `HA_USE_RASTERIZER` | 0 | 启用 NVIDIA 光栅化优化。 | 需要 rasterizer forward/backward 的训练任务。 |
| SyncBatchNorm | `HA_USE_SYNCBN` | 0 | 移除多余的 CPU 同步操作。 | 分布式训练。 |

NVIDIA GPU 场景下，可启用以下算子加速开关：

```
export HA_USE_VOXEL=1
export HA_USE_DEFORM_AGG=1
export HA_USE_RASTERIZER=1
export HA_USE_SYNCBN=1
python train.py
```

## DCU

| 能力 | 环境变量 | 默认值 | 说明 | 适用方向 |
| --- | --- | --- | --- | --- |
| SyncBatchNorm | `HA_USE_SYNCBN` | 0 | 移除多余的 CPU 同步操作。 | 分布式训练。 |
| Dropout 融合 | `HA_USE_DROPOUT` | 0 | 启用 DCU Dropout 优化。 | Transformer/大模型训练。 |

| | | | | |
|---|---|---|---|---|
| Softmax Focal Loss | `HA_USE_FOCAL_LOSS` | 0 | 启用 DCU Softmax Focal Loss 优化。 | 检测任务。 |
| 多尺度可变形注意力 | `HA_USE_DEFORM_ATTN` | 0 | 启用 DCU 多尺度可变形注意力优化。 | 检测、多模态任务。 |
| RMSNorm | `HA_USE_RMSNORM` | 0 | 启用 DCU RMSNorm 优化。 | LLM/VLM 训练和推理。 |
| RoPE | `HA_USE_ROPE` | 0 | 启用 DCU RoPE 优化。 | LLM/VLM 训练和推理。 |
| FastGELU | `HA_USE_FAST_GELU` | 0 | 启用 DCU FastGELU 优化。 | 使用 GELU 激活的大模型。 |

**DCU 场景下，可启用以下算子加速开关：**

```
export HA_USE_DEFORM_ATTN=1
export HA_USE_DROPOUT=1
export HA_USE_FOCAL_LOSS=1
export HA_USE_RMSNORM=1
export HA_USE_ROPE=1
python train.py
```

**导入后，HyperAcc 会读取当前进程的 `HA_*` 环境变量，并按配置启用对应能力。**

> ⓘ **注意：**
> **文中涉及的第三方产品、软件、框架、模型及商标名称归其各自权利人所有，仅用于说明 HyperAcc 的适配或使用场景，不构成任何认可、合作、担保或背书。**

## 注意事项

- **不同硬件平台支持的算子能力不同，实际启用取决于模型结构、PyTorch 版本及运行环境，不符合当前硬件平台的算子开关会被自动跳过。建议先在测试环境验证后再用于生产训练。**
- **启用算子替换前，建议先确认原始训练任务正常运行，以便在出现异常时快速定位问题并明确优化引入的影响。**
- **如任务依赖的第三方库版本差异较大，建议先在验证环境测试。**

## 算子启用

**您可设置以下变量启用 HyperAcc 训练加速套件的算子加速能力。**

### 功能开关管理

未设置环境变量时，`import hyperacc.auto` 不会执行任何操作。您需通过 `HA_USE_*` 环境变量显式启用能力。

## 总开关

| 环境变量 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_USE_ALL` | 0 | 一键启用所有功能。 |
| `HA_USE_ALL_HOOKS` | 0 | 启用所有透明替换类能力，不影响系统级优化。 |

## 优先级规则

环境变量优先级从高到低依次为：

- `HA_USE_<FEATURE>`：单项功能开关，优先级最高。显式设置为 `0` 时，可覆盖上层总开关的启用状态。
- `HA_USE_ALL_HOOKS`：Hook 类功能总开关。
- `HA_USE_ALL`：全局总开关。
- 默认关闭：未设置任何环境变量时，所有能力默认关闭。

## 系统级优化开关

| 环境变量 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_USE_GC` | 1 | 启用 GC 优化。 |
| `HA_USE_NUMABIND` | 0 | 启用 CPU/NUMA 绑核。 |
| `HA_USE_THP` | 0 | 启用透明大页配置。 |

# HyperAcc 常用环境变量

最近更新时间：2026-08-14 18:27:00

本文介绍腾讯云 HyperAcc 训练加速套件的环境变量配置接口，包括自动激活入口、功能开关（总开关、透明替换能力、算子加速能力、系统级优化能力）及功能参数（编译、GC、DataLoader、日志等）的详细说明。HyperAcc 通过环境变量统一控制所有功能，您在训练入口 Python 脚本开头导入 `hyperacc.auto` 即可完成自动激活。

## 使用说明

- 将所有配置写入训练任务启动脚本、容器启动命令或调度平台环境变量配置中。
- 环境变量应在训练进程启动前设置。
- 设置环境变量后，需要在训练入口 Python 脚本开头导入 `hyperacc.auto` 才会自动激活对应能力。
- 系统级优化能力在权限不足或环境不满足时，可能无法生效，建议结合日志确认。

## 自动激活入口

推荐在训练入口 Python 脚本开头导入。除 `from_future_` 等必须位于文件开头的语句外，`import hyperacc.auto` 应放在 torch、训练框架、模型和 DataLoader 相关导入之前：

```
import hyperacc.auto
```

导入后，HyperAcc 会读取当前进程的 `HA_*` 环境变量，并按配置启用对应能力。

> ⓘ **注意：**
> - 未设置环境变量时，`import hyperacc.auto` 默认不做任何操作。
> - 自非主进程场景下，部分自动激活逻辑可能跳过，以避免影响 DataLoader worker 等子进程。

## 环境变量 API

### 功能开关总开关

| 环境变量 | 默认值 | 说明 |
| --- | --- | --- |
| `HA_USE_ALL` | 0 | 启用所有能力，支持0（关闭）和1（开启）。 |
| `HA_USE_ALL_HOOKS` | 0 | 启用所有透明替换能力，不影响系统级优化，支持0（关闭）和1（开启）。 |

### 通用透明替换能力

| 环境变量 | 默认值 | 说明 |
| --- | --- | --- |
| HA_USE_COMPILE | 0 | 启用自动编译，支持0（关闭）和1（开启）。 |
| HA_USE_CHANNELS_LAST | 0 | 启用 channels_last 内存格式转换，支持0（关闭）和1（开启）。 |
| HA_USE_LIGHTNING | 0 | 启用 Lightning Trainer 相关能力，支持0（关闭）和1（开启）。 |
| HA_USE_DATALOADER | 0 | 启用 DataLoader 优化，支持0（关闭）和1（开启）。 |
| HA_USE_SYNCBN | 0 | 启用 SyncBatchNorm 优化，支持0（关闭）和1（开启）。 |

### 系统级优化能力

| 环境变量 | 默认值 | 说明 |
| --- | --- | --- |
| HA_USE_GC | 1 | 启用 GC 优化。 |
| HA_USE_NUMABIND | 0 | 启用 CPU/NUMA 绑定。 |
| HA_USE_THP | 0 | 启用透明大页。 |

### 功能参数

| 环境变量 | 默认值 | 说明 |
| --- | --- | --- |
| HA_COMPILE_BACKEND | inductor | 编译后端，支持 `inductor`，`cudagraphs` 和 `eager` 等。 |
| HA_COMPILE_MODE | default | 编译策略，取值如下：<br>• `default`：默认编译模式。<br>• `reduce-overhead`：降低运行时开销模式。<br>• `max-autotune`：最大自动调优模式。 |
| HA_COMPILE_FULLGRAPH | false | 是否要求完整计算图。 |
| HA_COMPILE_DYNAMIC | 空 | 是否启用动态形状。 |
| HA_COMPILE_EXPLAIN | 0 | 是否输出计算图断裂（graph break）诊断信息。 |

| | | |
|---|---|---|
| HA_COMPILE_MAX_PARALLEL | 2 | 后台并行编译线程数。 |
| HA_COMPILE_DRILLDOWN_THRESHOLD | 1000000000 | 大模型分层编译阈值。 |
| HA_GC_MODE | reduce_frequency | GC 策略，取值如下：<br>• reduce_frequency ：降低 GC 频率。<br>• disable ：禁用 GC。<br>• custom：自定义 GC 策略。 |
| HA_GC_MONITOR | 0 或 false | 是否启用 GC 监控，取值如下：<br>• 0 或 false：禁用 GC 监控。<br>• 1 或 true：启用 GC 监控。 |
| HA_GC_GEN0 | 5000 | 第 0 代 GC 阈值。 |
| HA_GC_GEN1 | 20 | 第 1 代 GC 阈值。 |
| HA_GC_GEN2 | 20 | 第 2 代 GC 阈值。 |
| HA_DATALOADER_NO_PIN_MEMORY | 0 | 是否关闭 pin memory。取值如下：<br>• 0：启用 pin_memory 。<br>• 1：禁用 pin_memory 。 |
| HA_DATALOADER_WORKERS | 16 | DataLoader worker 数量。负数表示不修改用户配置。 |
| HA_DATALOADER_PREFETCH | 2 | DataLoader 预取批次数。负数表示不修改用户配置。 |
| HA_HOOKS_LAZY | 1 | 是否在目标组件加载时再应用透明替换能力。 |
| HA_HOOKS_SILENT | 1 | 是否减少透明替换相关日志。 |
| HA_LOG_LEVEL | INFO | 日志级别。支持 DEBUG 、 INFO 、 WARNING 、 ERROR 、 CRITICAL 。 |
| HA_SHIFT_COMPARE | 0 | 启用 shift patch 对比测试，主要用于调试验证。 |

# 常见问题

最近更新时间：2026-08-14 18:27:00

本文汇总 HyperAcc 训练加速套件使用过程中的常见问题，涵盖安装、能力启用、训练加速、系统优化、硬件算子、自动编译、DataLoader 及日志等方面，提供排查思路与解决方案。

## 安装相关

### 安装 wheel 包失败怎么办?

请按以下顺序检查：

1. 确认 wheel 包文件完整，文件名通常为 `hyperacc-*.whl` 。
2. 确认 Python 版本为 3.8 及以上，并与 wheel 包匹配。

```
python --version
#确认Python 版本
```

3. 确认当前环境已安装 PyTorch 2.0 及以上版本和训练任务所需依赖。
4. 使用绝对路径重新安装：

```
pip install /path/to/hyperacc-*.whl
# path是wheel包所放置的路径
```

### 如何确认 HyperAcc 已安装成功?

执行以下命令：

```
python -c "import hyperacc.auto; print('hyperacc auto import ok')"
```

如果命令正常输出，说明 Python 包可被当前环境导入。

### 为什么 import hyperacc.auto 后没有启用能力?

这是预期行为。未设置环境变量时， `import hyperacc.auto` 默认不做任何操作。如需启用能力，请先设置对应环境变量，再导入 `hyperacc.auto` ：

```
import hyperacc.auto
```

## 启用相关

## 如何一键启用全部能力?

```
export HA_USE_ALL=1
python train.py
```

并确保训练入口中已导入:

```
import hyperacc.auto
```

## 如何只启用某一项能力?

设置单项环境变量即可，例如:

```
export HA_USE_COMPILE=1
export HA_USE_GC=1
python train.py
```

单项开关优先级高于总开关。

## 环境变量优先级是什么?

优先级从高到低依次为: `HA_USE_<FEATURE>` > `HA_USE_ALL_HOOKS` > `HA_USE_ALL` > 默认关闭
例如，设置 `HA_USE_ALL=1` 后，再设置 `HA_USE_COMPILE=0` ，可单独关闭自动编译能力。

## 为什么启用了环境变量仍未生效?

请逐项排查:

- 环境变量是否在启动训练进程前设置。
- 训练入口 Python 脚本是否导入了 `hyperacc.auto` 。
- `import hyperacc.auto` 是否位于 torch、训练框架、模型和 DataLoader 相关导入之前（ `from __future__` 等必须位于文件开头的语句除外 ）。
- 当前硬件平台是否支持对应能力。
- 日志级别是否过低，可尝试设置 `HA_LOG_LEVEL=DEBUG` 获取更多信息。

# 训练加速相关

## HyperAcc 会保证所有任务都有固定性能提升吗?

不会。优化效果受模型结构、数据加载瓶颈、硬件平台、PyTorch 版本及训练配置多重因素影响，建议结合自身业务任务开展实测验证。

## 是否需要改造训练框架?

不需要。推荐方式是通过环境变量启用所需能力，并在训练入口 Python 脚本开头导入 `hyperacc.auto` 。

## 能否在生产任务中直接启用 HA_USE_ALL=1?

可以用于快速验证，但生产任务建议先按单项能力逐步启用，确认训练正确性和稳定性后再组合启用。

## 系统优化相关

## CPU/NUMA 绑核依赖哪些信息?

HyperAcc 会根据 `LOCAL_RANK` 、 `LOCAL_WORLD_SIZE` 、 `CUDA_VISIBLE_DEVICES` 或 `HIP_VISIBLE_DEVICES` 等环境变量，结合 GPU 与 NUMA 拓扑进行绑定。

## LOCAL_WORLD_SIZE 未设置时能否使用 NUMA 绑核?

可以。单卡直接运行时，HyperAcc 可按单卡场景处理，将进程绑定到 GPU 所在 NUMA 节点的 CPU。

## 透明大页为什么没有生效?

透明大页配置需要写入 Linux 系统配置，通常需要 root 或管理员预配置权限。权限不足时，对应能力可能无法完成配置。

```
/sys/kernel/mm/transparent_hugepage/enabled: Permission denied
# 权限不足时的展示
```

## GC 优化应该选择什么模式?

建议默认使用：

```
export HA_USE_GC=1
export HA_GC_MODE=reduce_frequency
```

`disable` 模式会完全关闭 GC，建议仅在充分验证并监控内存使用后使用。

## NVIDIA/ DCU 相关

## NVIDIA 和 DCU 支持的能力是否相同?

不完全相同。部分算子能力仅适用于 NVIDIA GPU，部分能力仅适用于 DCU。

## 如何启用 NVIDIA 相关算子能力?

算子能力是通过环境变量进行启用。

示例：

```
export HA_USE_VOXEL=1
export HA_USE_DEFORM_AGG=1
export HA_USE_RASTERIZER=1
export HA_USE_SYNCBN=1
python train.py
```

## 如何启用 DCU 相关算子能力?

示例：

```
export HA_USE_DEFORM_ATTN=1
export HA_USE_DROPOUT=1
export HA_USE_FOCAL_LOSS=1
export HA_USE_RMSNORM=1
export HA_USE_ROPE=1
python train.py
```

# torch.compile 相关

## 启用自动编译后首次启动变慢是否正常?

是正常现象。`torch.compile` 可能在首次运行时产生编译开销，后续运行性能表现需结合具体任务评估。

## 自动编译失败怎么办?

可先降低配置复杂度：

```
export HA_USE_COMPILE=1
export HA_COMPILE_MODE=default
export HA_COMPILE_FULLGRAPH=false
export HA_COMPILE_EXPLAIN=1
export HA_LOG_LEVEL=INFO
python train.py
```

如果仍有问题，建议先关闭自动编译，确认其他能力是否正常：

```
export HA_USE_COMPILE=0
python train.py
```

## HA_COMPILE_MODE 应该如何选择?

| 模式 | 说明 |
| --- | --- |
| default | 默认模式，平衡编译速度和性能。 |
| reduce-overhead | 降低运行时开销，适合部分推理或小 batch 场景。 |
| max-autotune | 更激进的优化策略，可能带来更长编译时间。 |

## DataLoader 相关

## HA_DATALOADER_WORKERS 应该设置多少?

可从默认值 16 开始，根据 CPU 核数、数据预处理开销和内存情况调整。过大的 worker 数可能造成 CPU 或内存压力。

## 什么时候设置 HA_DATALOADER_NO_PIN_MEMORY=1?

当容器共享内存不足、出现锁页内存相关报错，或 `pin_memory` 在当前环境中表现不稳定时，可尝试关闭。

## 日志相关

## 查看日志

```
#查看更多的日志
export HA_LOG_LEVEL=DEBUG
export HA_HOOKS_SILENT=0
python train.py

#查看更少的日志

export HA_LOG_LEVEL=WARNING
export HA_HOOKS_SILENT=1
python train.py
```

## 问题定位建议

在问题定位时，建议提供以下信息：

- HyperAcc 安装包版本。
- Python 和 PyTorch 版本。
- 硬件平台。
- 已设置的 `HA_*` 环境变量。

- 训练启动命令和关键日志。