# 高性能计算集群

# HyperTrace 训练效能保障套件

腾讯云

# 文档目录

# HyperTrace 训练效能保障套件 新功能发布记录

最近更新时间：2026-08-21 15:21:00

本文为您介绍腾讯云 HyperTrace 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 8 月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| preflight 预检测正式发布 | 发布面向分布式训练的 preflight 一体化预检入口，支持占用检测、健康检测、集群一致性检测和基线标准化检测，检测结果驱动 go/no-go 决策后自动拉起训练命令。 | 2026-08-06 |
| gpu health 集群批量健康检查 | `gpu health` 子命令新增 `--auto-vendor` 参数，一次命令覆盖混合 NVIDIA/海光 DCU 集群，并行 SSH 采集各节点健康报告后聚合输出。 | 2026-08-06 |

## 2026 年 7 月

| 动态名称 | 动态描述 | 发布时间 |
| --- | --- | --- |
| 集群一致性检测 | 跨节点集群一致性检测支持 OS/内核/驱动/VBIOS/固件等维度，rank 0 自动汇聚所有节点的检测日志。 | 2026-07-15 |
| 算子深度分析 | 基于驱动底层接口，获取算子执行耗时，自动分析热点算子，实现算子级别可观测追踪。 | 2026-07-10 |
| 全链路性能分析 | 集成 eBPF 全栈追踪（ `run` ）、GPU Tracer 注入（ `setup` ）和多节点检测核心能力。 | 2026-07-01 |

# HyperTrace 训练效能保障套件概述

最近更新时间：2026-08-21 15:21:00

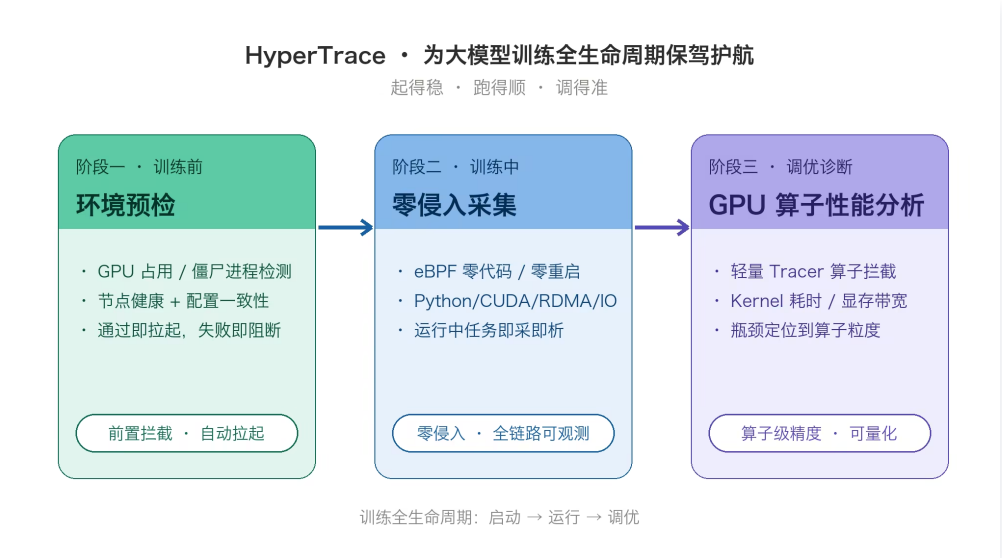
## 产品概述

随着大模型参数规模与训练集群节点数量持续攀升，分布式训练已从“单机可完成”演进为“百卡、千卡级协同工程”。训练任务一旦启动，可能面临环境异构、算力浪费、性能瓶颈难定位、故障难复现等多重挑战，任何一个节点的细微异常，都可能让整轮训练付诸东流。

HyperTrace 训练效能保障套件是面向大模型分布式训练的全链路护航与性能分析套件，围绕训练任务的启动、运行和调优三大阶段，按能力层级提供针对性的保障与观测支持：

- 训练前 – 节点环境预检： 无需侵入业务代码，自动排查多机环境与硬件隐患，提前阻断异常训练任务；
- 训练中 – 多维度性能观测： 基于 eBPF 技术实现系统与通信指标的零侵入采集，无需修改业务代码，无需重启训练进程；
- 调优阶段 – 算子级瓶颈定位： 支持深入定位 GPU Kernel 级性能瓶颈；仅需提前注入目标环境或通过 Preflight 模式拉起训练，即可在不改动训练脚本代码的前提下完成高精细度算子追踪。

该套件致力于为大模型训练全生命周期保驾护航，以降低训练中断风险、辅助减少问题遗漏、提升问题定位效率为目标，围绕训练任务的启动、运行和调优三大阶段，提供端到端的可观测与稳定性保障能力，帮助您在每一轮训练中“起得稳、跑得顺、调得准”。




## 核心能力

| 能力 | 说明 | 应用场景 |
| --- | --- | --- |
| 训练前环境预检 | 在每个训练节点启动前运行，检测 GPU 占用/僵尸进程、节点健康状态、集群配置一致性和基线标准 | 分布式训练启动前确保每个节点环境就绪。 |

| | | |
|---|---|---|
| | 化，检测通过后自动拉起训练命令；如任一节点未通过预设检查，则通知各节点暂不启动本次训练任务。 | |
| 零侵入全链路采集 | 通过 eBPF 采集 Python 函数栈、CUDA/HIP API 调用、GPU Kernel 活动、CPU 调度（具体采集范围以支持环境和产品文档为准），无需修改训练代码或重启进程。 | 需要快速分析已在运行的 GPU 训练任务。 |
| GPU 算子性能分析 | 通过轻量 Tracer 注入自动拦截算子执行与数据搬运事件，获取 kernel 耗时等关键指标。 | 定位具体 GPU kernel 的性能瓶颈。 |

> ⓘ **注意：**
> 如果是在宿主机环境执行性能数据采集，需要有 `root` 权限；如果是在容器环境执行性能数据采集，需要容器处于**特权模式**，或者容器启动参数中包含权限： `SYS_PTRACE`、`PERFMON`、`BPF` ，否则无法获取 CPU 函数栈和 eBPF 采集的数据。

## 产品优势

- **覆盖训练全生命周期**：围绕训练任务的启动、运行和调优三大阶段，提供从环境预检、多维度观测到算子级瓶颈定位的端到端保障。
- **训练前主动拦截风险**：在训练启动前自动排查 GPU 占用、僵尸进程、节点健康状态、集群配置一致性等隐患，发现问题时通知各节点暂不启动训练任务，减少因异常环境启动训练造成的算力和时间浪费。
- **训练过程中随时采集、按需诊断**：训练任务运行期间可随时采集，无需中断训练即可实时获取性能数据并定位瓶颈，也无需修改训练业务代码，尤其适合长周期、高成本的训练任务。
- **算子级瓶颈定位**：还原算子符号并分析线程、寄存器及显存微架构开销，辅助定位具体 GPU Kernel 的性能瓶颈，避免盲目调优。

## 支持环境

| 项目 | 说明 |
|---|---|
| 操作系统 | Linux 内核 5.4+（推荐 5.15+）。 |
| GPU 硬件平台 | 支持 NVIDIA GPU 和 DCU。 |
| 训练运行时 | 支持 CUDA 和 HIP。 |
| Python 版本 | 3.8 ~ 3.12。 |

## 使用限制

- 如果是在宿主机环境执行需要 root 权限才能加载 eBPF 程序；如果是容器环境需要提供特权模式，或者开启权限：`SYS_PTRACE`、`PERFMON`、`BPF`。
- 算子采集需要提前通过 `hyper-trace setup` 命令注入到目标环境，或通过 preflight 模式启动训练进程。
- 多节点模式需 root SSH 免密登录各训练节点。

# 快速入门

最近更新时间：2026-08-21 15:21:00

本文介绍如何完成 HyperTrace 训练效能保障套件的首次安装和 GPU 训练性能采集。

## 前提条件

| 条件 | 说明 |
| --- | --- |
| 已编译的 hyper-trace 二进制 | 已联系 在线客服 获取已编译的二进制工具。 |
| 操作系统 | Linux 内核 5.4+（推荐 5.15+）。 |
| 内核模块 | 启用 BTF 功能（检查是否存在虚拟文件 `/sys/kernel/btf/vmlinux` ）。 |
| GPU 训练进程正在运行 | NVIDIA CUDA 或 DCU 训练进程已启动（或即将启动）。 |
| root 权限 | 采集命令需要 sudo 运行。 |

## 步骤1：安装 HyperTrace 训练效能保障套件

HyperTrace 训练效能保障套件以二进制命令行工具形式提供，不依赖于操作系统发行版本。工具下载到训练环境后添加可执行权限即可使用，推荐放到 `/usr/local/bin` 目录下方便使用。

```
chmod +x hyper-trace
mv hyper-trace /usr/local/bin/

# 验证工具，查看帮助命令
hyper-trace --help
```

## 步骤2：训练环境检测

- **单节点场景下：**支持 NVIDIA/DCU 的环境检测，训练前预检测 GPU 故障，发现环境问题提前告警。

```
# NVIDIA 平台
hyper-trace gpu health

# DCU 平台
hyper-trace dcu health
```

- **多节点/集群场景下：**批量执行环境检测，需要节点间配置 SSH 免密。

```
# 集群节点信息提前配置到 hosts.txt 文件中
echo "10.0.0.1
10.0.0.2" >> hosts.txt

# 执行集群批量环境检测
hyper-trace gpu health -f hosts.txt
```

## 步骤3：集群一致性检测

集群一致性检测支持批量执行，批量检测前需要节点间配置 SSH 免密。

```
# 集群节点信息提前配置到 hosts.txt 文件中
echo "10.0.0.1
10.0.0.2" >> hosts.txt

# 执行集群一致性检测
hyper-trace collect -f hosts.txt
```

## 步骤4：接入训练任务

HyperTrace 训练效能保障套件支持以 preflight 方式提供训练任务的接入，并通过配置环境变量开启/关闭特定功能。执行后节点间会自动建连并识别拓扑，自动批量执行对应功能。

```
# master 节点
hyper-trace torchrun --nproc_per_node=8 --nnodes=2 --node_rank=0 --
master_addr=10.0.0.1 train.py

# worker 节点
hyper-trace torchrun --nproc_per_node=8 --nnodes=2 --node_rank=1 --
master_addr=10.0.0.1 train.py
```

## 步骤5：采集训练性能数据

在另一个终端中执行，采集进行中的训练任务：

```
# 自动检测 GPU 进程，全部探针，采集 10 秒
sudo hyper-trace run -t 10
```

```
# 采集完成后自动生成 HTML 报告
sudo hyper-trace run -t 10 --report
```

**采集完成后，输出目录结构如下：**

```
/tmp/ebpf-trace/<timestamp>/
├── trace_output.json                Chrome Trace 格式（可导入 Perfetto
UI）
├── trace_report.html                交互式 HTML 报告（ECharts 可视化）
└── trace_output_gpu_stats.xlsx      算子性能数据明细清单
```

# 操作指南
# HyperTrace 训练效能保障套件常见优化实践

最近更新时间：2026-08-21 15:21:00

本文面向已安装 HyperTrace 训练效能保障套件工具的用户，介绍常见训练任务的接入和信息采集方式。

## 场景一：训练前环境标准化预检

适用于多机分布式训练启动前，需要确保所有节点 GPU 就绪、配置一致且无僵尸进程的场景。

### 使用方式

在每个训练节点执行同一条命令，preflight 自动完成预检并在通过后拉起训练：

```
# 隐式形式（推荐）
hyper-trace torchrun --nproc_per_node=8 --nnodes=4 --node_rank=0 --
master_addr=10.0.0.1 train.py

# 显式形式
hyper-trace preflight -- torchrun --nproc_per_node=8 train.py
```

### 行为模式

通过设置环境变量 `HYPER_TRACE_PREFLIGHT_MODE` 可以选择标准化预检模式：

- **strict 模式（默认）**：任意节点 FAIL 时，rank 0 广播 no-go，所有节点退出非零状态，训练不启动。
- **warn 模式（** `HYPER_TRACE_PREFLIGHT_MODE=warn` **）**：告警但始终拉起训练命令。

## 场景二：GPU 训练全栈性能分析

适用于需要分析单机多卡训练任务中 Python 层、CUDA/HIP API 层和 GPU Kernel 层性能关联的场景。

### 使用方式

训练启动前，通过 `hyper-trace setup` 命令完成环境注入，或者通过 preflight 模式启动训练任务。

```
# 场景一：通过 hyper-trace setup 功能完成环境注入，然后启动训练进程
hyper-trace setup
bash start_train.sh
```

```
# 场景二：通过环境变量开启训练检测注入，然后通过 preflight 模式启动训练进程
export HYPER_TRACE_PREFLIGHT_SETUP=on
hyper-trace bash start_train.sh
```

新开启一个终端窗口，然后调用 hyper-trace 进行性能数据采集：

```
# 全栈式采集 10 秒，完成后生成报告
sudo hyper-trace run -t 10 --report

# 只采集 Python 函数栈 + GPU Kernel
sudo hyper-trace run --pystack --gpu -t 10
```

采集完成后，按以下方式分析结果，结果文件输出在 `/tmp/hyper-trace/<timestamp>/` 目录下：

- 打开 `trace_report.html`，重点查看：GPU 利用率、通信和计算占比、Python 函数热点、GPU 算子热点。
- 将 `trace_output.json` 导入 Perfetto UI，在时序图中找出高耗时 Kernel，定位 Python 函数与 GPU 执行的完整调用链路。

# HyperTrace 常用命令与环境变量

最近更新时间：2026-08-21 15:21:00

本文介绍 HyperTrace 训练效能保障套件各子命令的详细参数、输出文件以及环境变量的配置。

## 命令树总览

```
hyper-trace
├── run            全栈 eBPF 追踪采集
├── preflight      训练前环境预检，通过后自动拉起训练
├── setup          GPU Tracer 安装（CUPTI / ROCtracer）
├── collect        节点配置诊断（GPU/RDMA/System）
├── gpu            NVIDIA GPU 工具集
│   └── health     GPU 健康检查
├── dcu            DCU 工具集
│   └── health     DCU 健康检查
├── report         从 trace JSON 生成分析报告
├── pytrace        Python 函数栈轻量采样
└── init           生成配置文件（交互式 / 默认）
```

## 输出文件参考

| 路径 | 说明 |
| --- | --- |
| `/tmp/ebpf-trace/<timestamp>/` | 采集输出根目录 |
| `trace_output.json` | Chrome Trace 格式（Perfetto UI 可直接加载） |
| `trace_report.html` | 13 维度交互式 HTML 报告 |
| `/opt/ebpf_tracer/` | GPU Tracer 默认安装目录 |
| `/opt/ebpf_tracer/activate.sh` | 环境变量激活脚本 |
| `/tmp/hyper-trace-preflight/` | preflight 日志默认目录 |

## `hyper-trace run` 全栈追踪采集

```
sudo hyper-trace run [flags]
```

**常用示例：**

```
# 默认全探针，采集时间 10 秒
sudo hyper-trace run -t 10

# 仅 Python 栈 + GPU
sudo hyper-trace run --pystack --gpu -t 10

# 指定 PID
sudo hyper-trace run -p 12345,12346 -t 10

# 采集后自动生成报告
sudo hyper-trace run -t 10 --report
```

**全栈采集的详细参数如下：**

| 参数 | 说明 | 默认行为 |
| --- | --- | --- |
| `-t, --duration` | 采集时长（秒） | 10 (s) |
| `-c, --config` | 配置文件路径（JSON） | – |
| `-p, --pid` | 指定 PID（逗号分隔，覆盖自动检测） | 采集 GPU 上所有进程 |
| `-o, --output` | 输出文件名 | 输出到：`/tmp/hyper-trace/<timestamp>/` |
| `--output-mode` | • full（完整 Chrome Trace，~GB）<br>• detect（聚合统计，~KB）<br>• hybrid（统计+采样事件） | full 模式 |
| `--cpu` | CPU on/off 调度追踪 | 关闭 |
| `--gpu` | GPU Kernel profiling（CUPTI/ROCtracer） | 关闭 |
| `--cuda-api` | CUDA/HIP API uprobe 追踪 | 关闭 |

| | | |
|---|---|---|
| `--pystack` | Python 函数栈（USDT 优先，自动回退 uprobe） | 开启 |
| `--pystack-hz` | Python 栈采样频率 | 200 Hz |
| `--gc` | Python GC 追踪 | 关闭 |

## `hyper-trace preflight` 训练前环境预检

```
hyper-trace preflight [flags] -- <command> [args...]
```

**常用示例：**

```
# 显示调用 preflight
hyper-trace preflight -- torchrun --nproc_per_node=8 train.py

# 隐式调用 preflight，与显式调用无差别
hyper-trace python train.py
```

| 参数 | 说明 | 默认行为 |
|---|---|---|
| `--log-dir` | 本地日志目录 | 输出到：`/tmp/hyper-trace-preflight` |
| `--timeout` | 多机节点发现汇聚超时秒数 | 超时时间为：120s |
| `--skip-health` | 跳过健康检测 | 不跳过 |
| `--skip-collect` | 跳过标准化/基线检测 | 不跳过 |
| `--skip-consistency` | 跳过集群一致性检测 | 不跳过 |
| `--batch_detect` | 仅运行检测，不拉起训练命令；失败退出码 1 | 以拉起训练命令的形式执行 |

## `hyper-trace collect` 节点配置诊断及一致性检测

```
# 本地表格输出
sudo hyper-trace collect

# JSON 输出
sudo hyper-trace collect -o json

# 打印 baseline 标准
sudo hyper-trace collect --baseline

# 多节点集群采集
sudo hyper-trace collect -f hosts.txt
```

## `hyper-trace gpu health` NVIDIA GPU 健康检查

```
# 本地节点检查
hyper-trace gpu health

# JSON 输出
hyper-trace gpu health --json

# 集群批量检查
hyper-trace gpu health -f hosts.txt
```

**数据来源：** `nvidia-smi -q`（温度、功耗、ECC、PCIe、时钟）、`dmesg`（NVRM/Xid 错误）、`/dev/nvidia*`（驱动就绪）。

## `hyper-trace dcu health` DCU 健康检查

```
# 本地节点检查
hyper-trace dcu health

# JSON 输出
hyper-trace dcu health -o json

# 集群批量检查
hyper-trace dcu health -f hosts.txt
```

数据来源：`hy-smi`（温度、功耗、显存、时钟、PCIe 链路）、`dmesg`（Hygon/XCD 错误码）、`/dev/kfd` + `/dev/dri/renderD*`（驱动就绪）、DTK-ROCm 版本信息。

## 关键环境变量参考

| 环境变量名称 | 功能 |
| --- | --- |
| `CUPTI_TRACER_AUTOSTART` | CUPTI tracer 自动启动开关 |
| `HYPER_TRACE_PREFLIGHT_MODE` | preflight 模式：strict（默认）/ warn |
| `HYPER_TRACE_PREFLIGHT_HEALTH` | preflight 健康检测开关：on / off（默认 off） |
| `HYPER_TRACE_PREFLIGHT_COLLECT` | preflight 基线检测开关：on / off（默认 off） |
| `HYPER_TRACE_PREFLIGHT_OCCUPANCY` | preflight 占用检测开关：on / off（默认 on） |
| `HYPER_TRACE_PREFLIGHT_CONSISTENCY` | preflight 集群一致性检测开关（默认 on） |

# 常见问题

最近更新时间：2026-08-21 15:21:00

## 安装相关

### 运行时报 /sys/kernel/btf/vmlinux 不存在?

确认内核是否启用 BTF：

```
cat /boot/config-$(uname -r) | grep CONFIG_DEBUG_INFO_BTF
```

若输出 `CONFIG_DEBUG_INFO_BTF=y` 但文件仍不存在，检查 debugfs 是否挂载：

```
mount -t debugfs none /sys/kernel/debug
ls /sys/kernel/btf/vmlinux
```

若输出为 `is not set` 或无此配置项，说明内核编译时未开启 BTF，需升级到支持 BTF 的内核（5.4+）。

## 采集相关

### 执行 run 时提示 no GPU processes found?

确认训练进程是否有 GPU 活动，必要时手动指定 PID。

```
# NVIDIA 环境：确认训练进程是否有 GPU 活动
nvidia-smi --query-compute-apps=pid,process_name --format=csv

# DCU 环境
/opt/hyhal/bin/hy-smi --showpids

# 手动指定 PID
sudo hyper-trace run -p <pid> -t 10
```

### 训练进程没有 CUPTI tracer 激活日志?

```
# 检查环境变量
echo $CUDA_INJECTION64_PATH

# 手动激活
hyper-trace setup
source /opt/ebpf_tracer/activate.sh

# 确认 .so 文件存在
ls -la /opt/ebpf_tracer/libcupti_tracer.so
```

### DCU 训练进程没有 ROCtracer 激活日志?

```
# 检查 LD_PRELOAD 是否包含 roctracer_wrapper.so
echo $LD_PRELOAD

# 手动激活
hyper-trace setup
source /opt/ebpf_tracer/activate.sh

# 确认 .so 文件存在（需 --dcu setup）
ls -la /opt/ebpf_tracer/librocm_tracer.so
```

### Python 函数栈采集为空?

```
# 检查 USDT 探针是否可用
readelf -n /proc/<pid>/exe | grep function__entry
# 无输出时会回退到 uprobe 模式（自动）

# 确认已启用 --pystack
sudo hyper-trace run --pystack -t 10
```

## 预检相关

### preflight 报 cluster consistency FAIL?

rank 0 会汇聚所有节点的 OS/内核/驱动/VBIOS 信息到 `<log-dir>/cluster/` 目录，查看各节点 `preflight.log` 找出不一致字段，对齐配置后重新运行。

**preflight 在多节点场景下超时（节点发现阶段卡住）？**

检查各节点之间 `<master_port> + 200` 的 TCP 端口是否可达（防火墙/安全组）。也可增大超时：

```
HYPER_TRACE_PREFLIGHT_TIMEOUT=300 hyper-trace ...
```

## HyperTrace 与常见工具的功能对比

| 对比项 | 采集内容 | Nsight System | Torch-Profiler | HyperTrace |
| --- | --- | --- | --- | --- |
| 预检测 | -- | 不支持 | 不支持 | 支持 |
| 随时诊断 | -- | 不支持 | 不支持 | 支持 |
| 零代码入侵 | -- | 支持 | 不支持 | 支持 |
| Python 应用层 | Python 函数调用、训练 step 耗时 | 不支持 | 支持 | 支持 |
| 框架层 | PyTorch 算子执行 | 不支持 | 支持 | 支持 |
| GPU 计算层 | kernel 执行时间、类型分布 | 支持 | 支持 | 支持 |