

高性能计算集群

算子优化平台 Hyper Kernel



腾讯云

【 版权声明 】

©2013–2026 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

算子优化平台 Hyper Kernel

算子优化平台 Hyper Kernel 概述

操作指南

算子优化平台 Hyper Kernel

算子优化平台 Hyper Kernel 概述

最近更新时间：2026-09-15 17:28:10

Hyper Kernel 是腾讯云推出的、支持多硬件后端的 AI 算子自动优化平台，面向需要在 GPU 或国产芯片上进行算子性能优化与迁移适配的开发者 and 工程团队。它将原本依赖资深工程师、单个算子往往耗时数天的优化工作，转化为由 AI 自动执行的标准化流程，帮助您在更短时间内获得经过实测验证的高性能算子实现。

本文面向开发者与接入对接方，介绍平台的典型使用场景、接入方式、实测效果，以及背后的架构与工作原理，帮助您判断平台是否适用于自身的算子优化需求，并了解如何将其接入实际工作流程。CUDA 算子优化是一项高门槛的工作，需要资深 CUDA 工程师投入大量时间，单项算子优化通常耗时数天。随着国产芯片的普及，尤其在部分非 GPGPU 架构（如部分国产 AI 加速卡）中，算子优化任务更加复杂和艰巨。

① 说明：

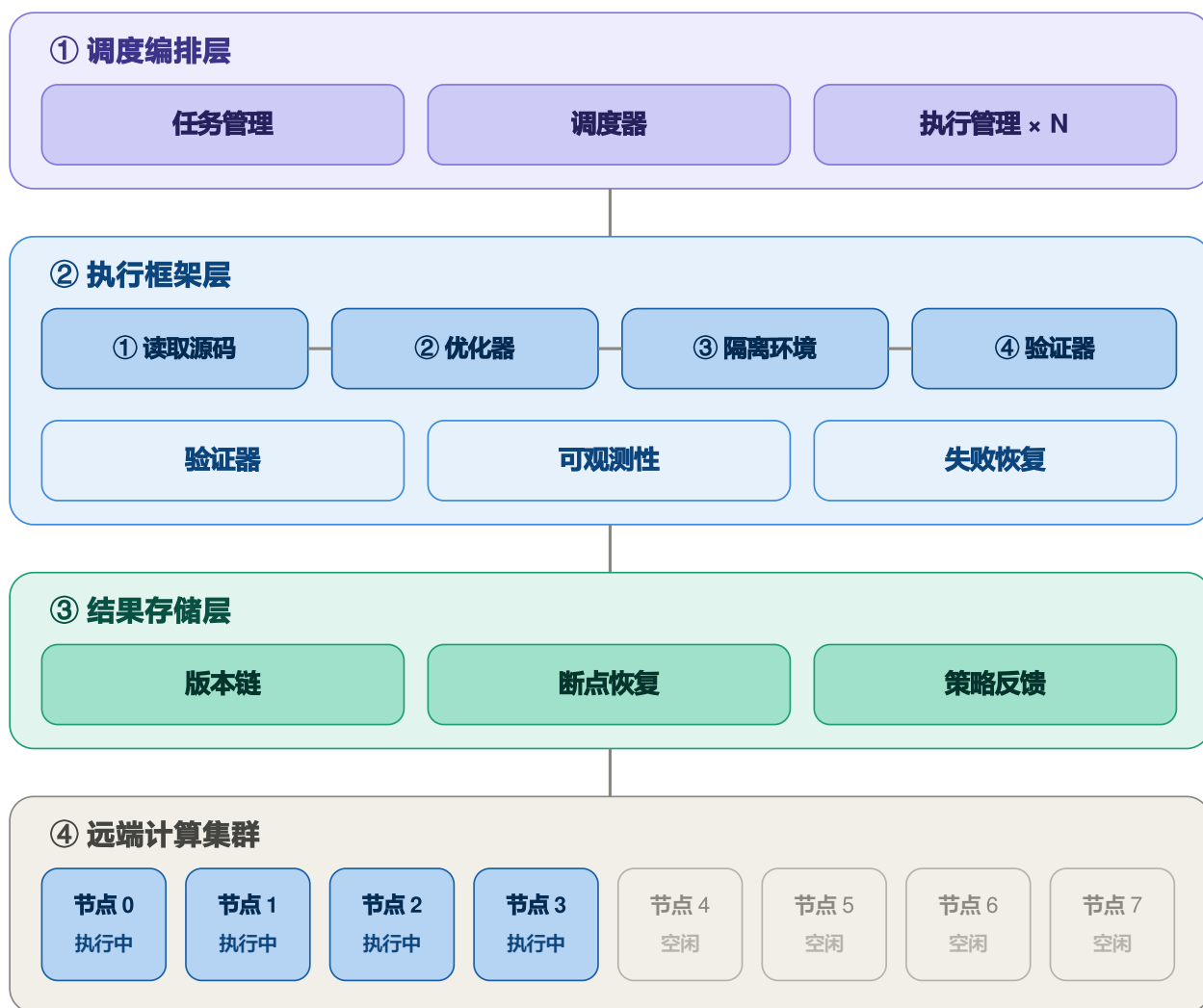
如需使用，请联系 [在线客服](#) 获取。

为提升算子优化效率，腾讯云推出支持多硬件后端的算子优化平台 Hyper Kernel。该平台基于“Agent + Skill + Orchestrator”范式，将领域专家经验凝练为结构化 Skill，将算子迭代优化过程沉淀为多 Agent 协同工作流程，实现 AI 算子优化的零人工干预执行。基于前沿模型，在本文列示的 Transformer Engine、mmcv 和 KernelBench 测试集中，获得有效提升的算子占比分别为 75%、78% 和 86%；平均加速比分别为 1.96 倍、4.75 倍和 4.77 倍。结果仅反映相应测试集和测试环境下的情况。

① 说明：

- 上述测试结果仅反映相应测试集和测试环境下的情况。
- Transformer Engine、mmcv 和 KernelBench 分别在各自固定的代码版本内进行测试，同一测试集内采用统一的原始实现作为比较基线，各测试集之间不作横向性能比较。

整体系统架构



国产硬件迁移挑战

跨平台迁移难点

不同硬件平台在执行模型、内存层级、API 兼容性及工具链方面存在差异，既有优化经验无法直接迁移，需要结合目标硬件的软件栈、开发文档和实际测试结果重新适配。API 名称变化仅是表面，底层的线程模型、内存层级、指令行为均存在差异，任何一个未适配的点都可能导致结果出错或性能下降。

跨平台迁移的难点还在于底层硬件抽象的差异——同一段 GPU 代码在不同厂商芯片上，并行模型、内存层次、同步机制各不相同。这些差异无法通过简单的 API 替换解决，需要对每个算子的并行逻辑重新审视。

核心挑战

以迁移到某款国产芯片为例，面临以下四个方面的核心挑战：

- **底层并行模型不同**：该国产芯片的执行单元与 NVIDIA GPU 的 Warp 存在核心区别。两者均遵循 SIMT（单指令多线程）编程模型，但底层最关键的执行单元大小和实现机制不同，直接导致性能特性和优化策略的差异。

- **API 表面相似但行为不同**：该芯片通过自定义接口对接 CUDA 生态，大部分 API 可一一映射，但部分高级特性（如计算图捕获、流同步等）行为不同或不支持。表面上的字符串替换可能埋下运行时的隐性缺陷。
- **代码与框架深度耦合**：部分算子并非独立 kernel，而是深度依赖 PyTorch 内部基础实现（如随机数状态管理、张量元信息描述等），这些内部接口在国产芯片端没有对应物，需要将核心计算逻辑提取后独立重写。
- **缺乏国产芯片调优经验**：国产芯片公开资料较少，资料相对少于类似 NVIDIA Nsight 的全套工具链。优化策略需从架构白皮书或编程手册逐项推导，试错成本高，且不同算子的最优策略可能截然不同。

使用场景

Hyper Kernel 适用于以下几类典型的算子优化与迁移场景。以下场景描述了平台在实际工作中的应用方式，具体案例与数据请结合实际情况补充。

场景一：国产芯片算子适配

当团队需要将算子迁移适配到国产芯片，但缺乏对应的调优经验、公开资料有限时，可将待适配算子提交平台，由平台结合已沉淀的硬件 Skill 自动完成优化，产出经过精度校验和性能验证的实现，降低从零摸索的试错成本。

场景二：存量算子批量提效

当团队已有一批算子需要统一做性能优化时，可通过关联代码仓库或批量上传的方式一次性提交，平台按可用硬件资源并行执行优化任务，并汇总每个算子的提升情况，适合规模化的性能提效工作。

场景三：框架/库移植后的性能补齐

当框架或算子库移植到新硬件后，机械转译得到的实现往往存在明显的性能缺口。此时可将这些实现提交平台做进一步优化，在保证精度的前提下补齐性能基线，减少人工逐个调优的投入。

典型算子优化实测

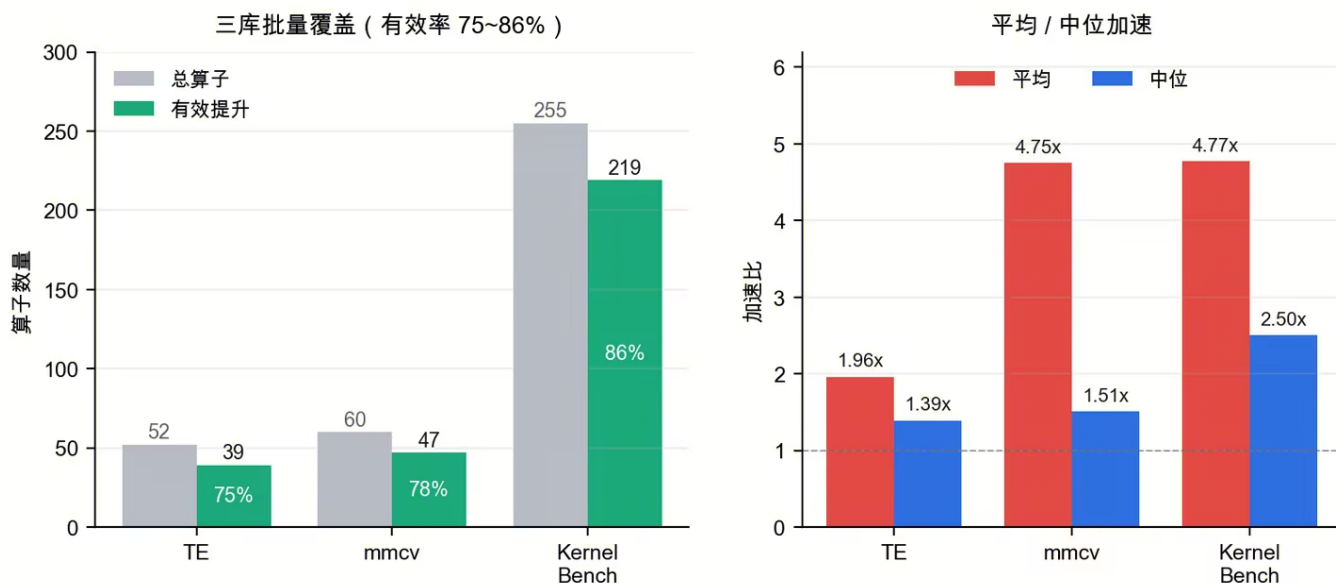
一套具备可信度的方法论，需基于规模化算子集开展验证，不应仅选取少量典型样例佐证结论。我们依托三套开源库完成批量实测。

Transformer Engine & mmcv & KernelBench

在 Transformer Engine(TE)、mmcv 和 KernelBench 三个库上的测试结果进一步验证了覆盖面：

测试库	算子总数	有效提升数	有效提升率	平均加速倍数
Transformer Engine	52	39	75%	1.96 倍
mmcv	60	47	78%	4.75 倍
KernelBench	255	219	86%	4.77 倍

下图展现的是 Kernel Engine 在 TE & mmcv & KernelBench 三个库的优化表现：



说明:

- 硬件型号及数量: DCU HCCPD1; 驱动/运行时版本: 6.3.31-V1.5.1; 操作系统: Ubuntu 22.04.5; 编译器: dcc 25.10.0-0; 框架: PyTorch 2.9.0; DTK 版本: dtk26.04。
- 模型/算子版本: 由开源库直接拉取, 具体版本及地址如下:
 - Transformer Engine (v2.16.1)
 - KernelBench (v0.1)
 - mmcv (v2.2.0)
- 输入 Shape、数据精度: 以测试用例为准。
- 有效提升指优化方案通过误差阈值校验, 且相较统一基线的加速比大于1。测试结果取平均值; 平均加速比采用算术平均计算, 并不包含未提升及失败样本。结果仅反映相应代码版本、输入 Shape、数据精度和软硬件环境下的表现。

结论表明, Transformer Engine 的52个算子中有39个获得有效提升, 平均加速1.96倍。mmcv 60 个算子中 47 个获得有效提升, 平均加速 4.75 倍, 其中 `active_rotated_filter` 算子最高加速达数十倍。KernelBench 是斯坦福发布的用于评测 LLM 生成 kernel 能力的标准 benchmark, 255 个算子中 219 个获得有效提升 (86%), 中位加速 2.5 倍, 平均加速 4.77 倍。

注意:

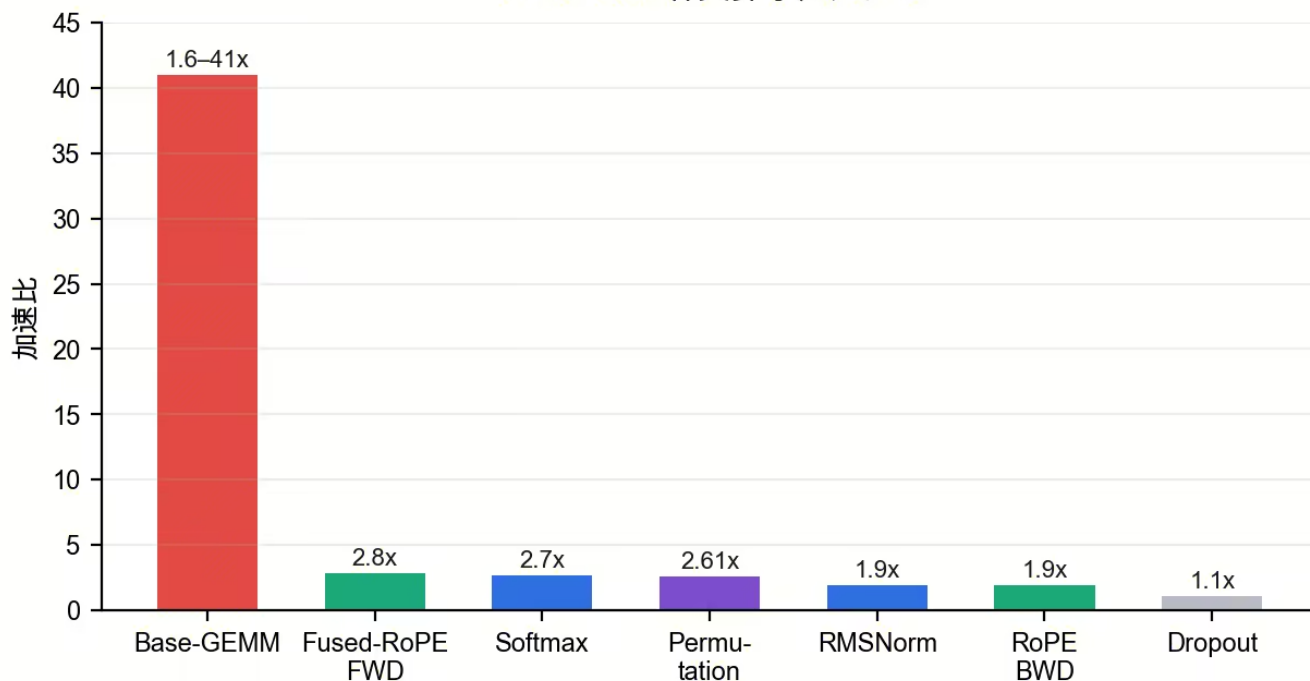
文中涉及的第三方产品、软件、框架、模型及商标名称归其各自权利人所有, 仅用于说明 Hyper Kernel 的适配或使用场景, 不构成任何认可、合作、担保或背书。

各类算子加速汇总

Kernel Skill 在不同类型算子上的加速效果如下:

算子类型	加速倍数
GEMM	1.6 – 41 倍
Softmax	2.7 倍
Fused-RoPE FWD	2.8 倍
Fused-RoPE BWD	1.9 倍

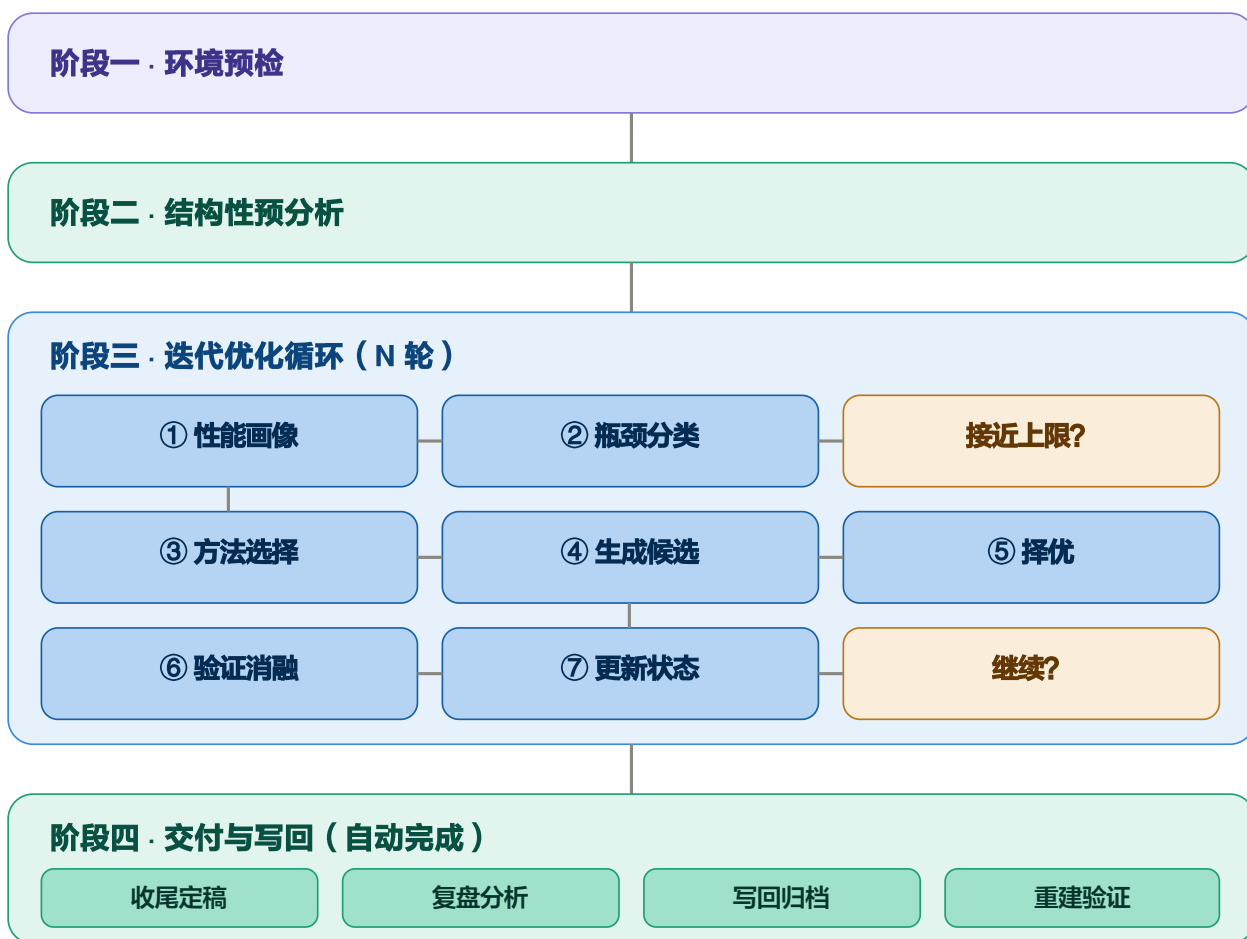
Kernel Skill 各类算子加速汇总



基于证据的 HIP Kernel 迭代优化流程

Skill 优化的核心是将各项硬规则串联为一条可迭代、可审计的完整流程。资深工程师优化算子时遵循严谨的方法论：先通过 profiling 分析性能瓶颈，判断算子属于算力受限还是访存受限；尝试优化方案后，与前一版本对比测试；最终保留更优结果。平台让 AI 严格遵循这一流程执行，确保每一步优化决策均有数据支撑，每一项变更均可追溯其优化依据。HIP Kernel 迭代优化流程如下：

HIP Kernel 迭代优化全流程



HIP Kernel 优化采用迭代式流程，整体分为四个阶段：环境预检、结构性预分析、迭代优化循环和交付写回。迭代优化循环是核心环节，通过反复执行 Profile、Roofline 分类、策略选择、候选生成、择优、验证和状态更新，逐步逼近性能最优解。

迭代优化循环的每一轮包含以下四个关键步骤：

- Profile 量化瓶颈：通过 Profiler 工具采集真实性能数据，获取算子的实际运行指标，而非依赖经验估算。
- Roofline 定位方向：判断当前瓶颈属于算力受限还是带宽受限，将优化资源分配至瓶颈所在维度。当性能接近理论极限时，自动触发早停机制。
- 选择方法并择优：Agent 从知识库中匹配与瓶颈类型对应的优化手段，生成多路候选方案，通过基准测试对比后选择最优结果。
- 证据验收：使用 Profiler 数据和 ISA 层级证据验证性能收益，确保优化效果可度量、可复现。缺乏证据支持的优化结果不予采纳。

持续迭代

平台支持优化结果的不断升级：

- 算子本身不变，优化结果以可更新的快照形式保存。
- 每轮优化完成后，自动对比性能，保留更优版本。

- 所有历史记录形成版本链，可回溯、可分析。
- 优化数据直接用于升级 Skill 规则，形成持续闭环。

行业价值

平台为国产芯片生态构建“加速器”与“连接器”，体现在以下三个方面：

- **降低生态门槛，实现经验资产化：**平台将稀缺的硬件调优经验结构化、代码化为可复用的 Skill 库，降低经验复用门槛。
- **缩短部署周期，提升国产硬件吸引力：**通过自动化优化，并复用已验证的优化策略和硬件 Skill，减少重复迁移和调试工作，提升国产硬件适配效率。同时可以保障性能基线，降低国产硬件的适配成本。
- **牵引软硬协同与工具链成熟：**平台在抽象多硬件差异、沉淀优化模式的过程中，形成的最佳实践和性能模型可与硬件厂商共建更高效、更统一的软硬件接口与工具链，助力构建健康、自主的算力生态。

展望

- **Skill 知识库持续进化：**覆盖多种国产硬件的架构差异、API 映射和优化策略，每次实战的经验自动回流，持续提升准确度。
- **多平台扩展：**Skill 按硬件平台分层组织，新增硬件平台时，可复用现有调度、验证和迭代框架，并结合目标硬件补充相应的硬件 Skill 和适配验证。

小结

从 Skill 迭代生成框架，到跨硬件后端的领域知识沉淀，再到 Harness Engineering 的工程底座，Hyper Kernel 构建了一个三层闭环系统：

- Skill 提供 Agent 决策的先验知识。
- Harness 提供将决策转化为可信实测结果的确定性基础设施。
- 实测结果反过来驱动 Skill 演进。

该架构的核心思路，是将大模型生成高性能 CUDA Kernel 这一高度非确定性问题，纳入确定性闭环控制系统进行约束，以此满足生产环境在可靠性、可观测性与可迭代性方面的严苛要求。在已完成硬件 Skill 构建和适配验证的场景中，平台可自动执行算子优化任务；当前支持范围以已验证硬件、软件版本及算子清单为准。

平台的价值不止于提升算子优化效率，更关键的是借助 AI 原生工程范式，面向持续迭代的新一代 AI 硬件搭建自适应、可持续演进的底层算力适配层。从长期视角来看，该体系能够降低 AI 产业对单一硬件生态的依附程度。

操作指南

最近更新时间：2026-09-10 11:27:30

前提条件

开始之前，请确认已具备以下条件：

- **已获得平台访问权限**：可正常登录并进入优化任务界面。
- **准备好待优化的算子代码**：可以是单个算子源码，也可以是包含依赖头文件、配置及测试用例的完整工程。
- **明确目标硬件**：已确认要优化到的目标芯片，且该芯片在平台当前支持范围内（如不确定，请参见 [使用限制](#)）。

⚠ 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 Hyper Kernel 的适配或使用场景，不构成任何认可、合作、担保或背书。

操作步骤

此处以优化单个算子为例，介绍从提交代码到查看结果的完整操作流程。



第 1 步：进入平台，创建优化任务

登录平台后，单击顶部**创建任务**，进入单算子优化页面。页面结构如下：

- 左侧为**提交算子代码**区，用于配置和提交待优化算子
- 右侧为执行区，实时展示优化进度与结果。

如需一次优化多个算子，可使用 [批量优化](#)。

第 2 步：选择目标芯片与优化模型

在左侧顶部的两个下拉框中，分别选择本次优化的**目标芯片**（算子将在该硬件上编译、运行与评测）和**优化所用模型**。

① 说明：

请确保所选芯片在平台当前已支持的范围内（目前仅支持 DCU、Nvidia）。

第 3 步：填写任务名称（可选）

在任务名称输入框中填写便于识别的任务名称（例如按算子名或优化目标命名）。该项为可选，留空时平台会自动生成默认名称（如 `RoPE FWD 优化`）。填写后便于后续在任务列表中检索和区分。

第 4 步：提交算子代码

平台为您提供以下三种算子提交方式，可使用标签页切换：

- **粘贴代码**：直接将 HIP/CUDA 算子代码粘贴到输入框，适合快速验证单个算子。可将完整算子代码、测试用例及编译脚本（例如 Makefile、CMakeLists）一并粘贴，平台将自动识别代码结构并进行优化。
- **上传文件**：上传算子源码及其依赖文件（支持 `.h`、`.hpp`、`.cu`、`.cuh`、`.cpp`、`.hip`、`.py`、`.json`、`.xml`、`.yaml` 等类型），文件夹上传时保留原有目录结构。
- **示例代码**：如仅需快速体验操作流程，可直接加载平台内置的示例代码。

① 说明：

为确保优化效果与精度校验结果的准确性，建议将算子依赖的头文件、配置及测试用例一并提交，避免因上下文缺失导致校验偏差。

第 5 步：开始优化并查看进度

确认配置无误后，单击开始优化提交任务。右侧执行区会按照初始化 > 基准 > 优化 > 精度 > 验证 > 报告的阶段实时展示进度。整个过程自动执行，无需人工干预。您可随时查看当前所处阶段，或在必要时中途终止任务。

第 6 步：查看优化报告与结果

优化完成后，右侧会生成优化报告，包含性能对比（加速比、优化前后耗时与带宽）、精度校验结果，以及优化后代码及其与原始代码的 diff。确认结果符合预期后，可下载优化后的代码。所有任务均可在**任务列表**中查看状态（运行中/已完成/已停止）、重入查看或复盘。

批量优化

当需要一次性优化多个算子时，单击顶部**批量优化**，通过关联代码仓库或批量上传的方式提交算子，并统一选择目标芯片、优化模型与全局优化 Prompt。平台将根据可用硬件资源并行执行各子任务，并在批量任务报告中汇总每个算子的优化情况。

输入

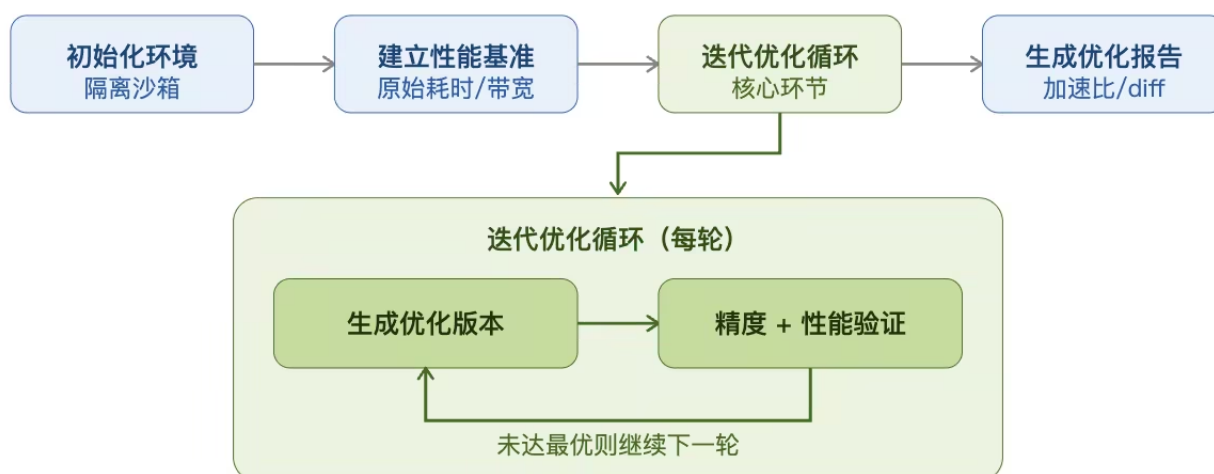
平台支持多种方式提交待优化的算子代码，您可以根据自身工程习惯选择：

- **直接粘贴代码**：适合快速验证单个算子。
- **上传文件或文件夹**：支持算子源码及其依赖的头文件、配置文件等常见类型（如 `.h`、`.hpp`、`.cu`、`.cu.h`、`.cpp`、`.hip`、`.py`、`.json`、`.xml`、`.yaml`），文件夹上传时保留原有相对路径结构。
- **关联代码仓库**：提供代码仓库地址，由平台拉取待优化算子，适合对接已有工程。

提交时可选择目标硬件后端与优化所用模型，并可填写任务名称、全局优化 Prompt 等可选项，用于对同一批任务做统一约束。

执行流程

提交算子后，平台以 Episode 为单位完成单个算子的优化循环，全程无需人工干预。一次完整优化通常经历以下阶段：初始化环境、建立性能基准、迭代优化、精度校验、性能验证，最终生成优化报告。其中，迭代优化环节由 Kernel Optimizer Skill 生成优化版本，在隔离环境中完成编译与 Benchmark，再由 Oracle 验证正确性并评测性能。您可在任务列表中实时查看每个任务的运行状态（运行中/已完成/已停止），随时重入查看进度，或中途终止任务。



输出

优化完成后，平台生成一份可查看、可下载的优化报告，主要包含：

- **性能对比**：优化前后的耗时与带宽、加速比等关键指标，直观反映优化收益。
- **精度校验结果**：标明优化后算子是否通过精度验证，确保正确性。
- **优化后代码**：可查看优化前后的代码 diff，支持保留多个优化版本并下载。

所有任务及其历史记录均可在任务列表中回溯，便于对比不同版本、复盘优化过程。

使用限制

平台架构预留 GPU/NPU 等后端扩展能力。当前已验证的硬件型号、驱动、运行时和算子范围以交付清单为准；其他硬件平台的适配能力需以实际验证结果为准。接入前请先确认目标硬件与算子是否在当前支持范围内（目前仅支持

DCU、Nvidia)。

常见问题

Q: 提交代码后一直没有开始优化怎么办?

请先确认所选目标芯片在平台支持范围内，且提交的代码结构完整（含必要的依赖文件）。若使用关联代码仓库方式，请确认仓库地址可访问。

Q: 优化后的算子精度没通过校验，说明什么?

说明该轮优化生成的实现未满足精度阈值要求，平台不会将其作为有效结果采纳。您可以查看报告中的精度校验详情，必要时补充更完整的测试用例后重新提交。

Q: 一次能优化多少个算子?

单任务针对单个算子；如需批量处理，请使用**批量优化**，平台将按照可用硬件资源并行执行多个子任务。

Q: 优化结果可以回溯吗?

可以。每个任务的优化结果以版本形式保存，可在任务列表中查看历史记录、对比不同版本，并下载所需版本的代码。

使用建议

为帮助您更高效地使用平台并获得可靠的优化结果，结合平台的工作方式，给出以下实践建议：

- **首先确认硬件与算子在支持范围内：**接入前核对目标硬件后端和算子类型是否已被平台验证支持。对于尚在扩展中的硬件，建议先做小批量试跑，再决定是否规模化投入。
- **从单算子试跑起步，再转批量：**首次使用时，建议先通过粘贴代码的方式，对单个代表性算子进行一次完整优化，以熟悉基准 > 优化 > 精度 > 验证 > 报告各阶段的产出形态。确认效果符合预期后，再通过关联代码仓库或批量上传，将优化范围扩展至批量算子。
- **提交完整的依赖上下文：**部分算子深度依赖框架内部实现或头文件、配置。上传时，建议保留目录结构，将相关依赖文件一并提交，避免因上下文缺失影响优化和精度校验。
- **善用全局优化 Prompt 统一约束：**批量优化时，可通过全局 Prompt 对精度阈值、优化偏好等参数进行统一设置，使同一批任务遵循相同标准，便于横向对比优化效果。
- **以报告数据为准做决策：**平台的加速比、精度校验、耗时与带宽等指标均基于实测数据。是否采纳某一优化版本，应以报告中的性能收益和精度结果为依据，而非仅凭预期。优化效果会因算子类型、起始版本、硬件环境和优化配置而有所差异。
- **利用版本链复盘与回溯：**优化结果以可更新的快照形式保存并形成版本链。当某次优化不理想时，可回溯历史版本对比分析，也可将失败尝试作为后续优化的经验参照。