arXiv:2603.21331v1 [cs.LG] 22 Mar 2026

AutoKernel：通过智能体驱动的迭代搜索实现 GPU 内核自主优化

Jaber Jaber*
RightNow AI
jaber@rightnowai.co

Osama
JaberRightNow AI
osama@rightnowai.co

摘要

编写高性能图形处理器内核是机器学习系统工程中最耗费人力的任务之一。面向张量核心硬件的单个矩阵乘法内核，可能需要专家花费数周时间，在分块策略、内存布局和精度配置方面进行调优。我们推出 AUTOKER-NEL，这是一个开源框架，它将自主智能体循环应用于任意 PyTorch 模型的 GPU 内核优化。给定一个模型，AUTOKERNEL 会对其进行性能分析以识别计算瓶颈，根据阿姆达尔定律的影响对瓶颈排序，并通过数百次实验迭代改进 Triton 或 CUDA C++内核实现，无需人工干预。五阶段正确性测试框架涵盖冒烟测试、跨 10+配置的形状遍历测试、对抗性输入下的数值稳定性、确定性验证以及边界情况覆盖，确保每个候选内核在记录任何加速比之前都经过验证。我们介绍了该系统的架构，其中包含超过 9,000 行 Python 代码和一份 909 行的智能体指令文档、两个后端上的 18 个初始内核实现、一套六层优化手册，以及与包含 250 个标准化问题的 Kernel-Bench 基准测试套件的集成。AUTOKERNEL 涵盖九种内核类型，涉及现代 Transformer 架构中的主要运算。在 NVIDIA H100 上，我们的 Triton 内核在大多数测试配置下都优于 PyTorch eager 和 torch.compile（max-autotune）：在 RMSNorm 上加速 5.29× 倍，在 Softmax 上加速 2.82× 倍，在交叉熵上加速 2.21× 倍；而在相同内核上，分别比 torch.compile 快 2.83× 倍、3.44× 倍和 2.94× 倍。在社区部署中，一个经 AutoKernel 优化的内核在 vectorsum_v2 B200 排行榜上取得第一名；另有报告称，一个通过单条提示生成的 Triton FP4 矩阵乘法内核性能比 CUTLASS 高 1.63 至 2.15× 倍。完整系统可在 https://github.com/RightNow-AI/autokernel 获取。

1 引言

大型 Transformer 模型在 GPU 硬件上的运行时间主要由少数几种计算内核决定。注意力和前馈层中的矩阵乘法通常占总 GPU 时间的 60% 到 80%；其余大部分时间则用于归一化、Softmax 和位置嵌入操作。cuBLAS 和 cuDNN 等供应商库能够很好地覆盖常见场景，但深度学习领域的架构创新步伐往往快于库的覆盖范围：分组查询注意力、SwiGLU 激活、旋转位置嵌入和均方根归一化都已在生产模型中投入使用，之后才获得专门的库支持。

*通讯邮箱：jaber@rightnowai.co

第 37 届神经信息处理系统大会（NeurIPS 2023）。

要缩小默认库性能与硬件能力之间的差距，需要深入掌握 GPU 微架构。工程师必须分析算术强度和屋顶线模型（Williams 等人，2009）、共享内存中的内存合并访问与存储体冲突、寄存器压力与占用率权衡、分块大小及其与 L2 缓存的相互作用、线程束级同步，以及张量核心指令的选择。单个高性能矩阵乘法内核可能包含 200+ 行 CUDA 或 Triton 代码，以及数十个相互依赖的参数。这类专业知识十分稀缺，而且调优过程难以扩展。

大语言模型能编写图形处理器内核吗？ KernelBench（Ouyang 等人，2025）系统地提出了这个问题，并在四个难度等级下，对大语言模型前沿模型解决 250 个图形处理器内核问题的表现进行了评估。结果令人沮丧：即使是最好的模型，采用一次性生成的方式，也只有不到 20% 的案例能够达到 PyTorch 基线性能。后续研究则探索了多智能体系统（Wei 等人，2025；Sun 等人，2026）、进化搜索（Wiedemann 等人，2026；Andrews 和 Witteveen，2025）、硬件感知反馈循环（Zhang 等人，2025）以及强化学习（Li 等人，2026；Dai 等人，2026），以进一步提升这些数字。

我们的方法。我们没有设计复杂的多智能体架构，而是注意到，专家内核工程师的工作流程本身就是一个简单的循环：编写候选项、进行基准测试、保留改进、舍弃退步，然后重复。AutoKernel 将这一流程自动化。一个大语言模型智能体修改单个内核文件，固定的基准测试框架负责验证正确性并测量吞吐量，结果则决定是否保留这项更改。每次迭代大约需要 90 秒。运行一整晚，可针对多个内核完成 300 到 400 次实验。
这一设计直接借鉴了 Karpathy 的 autoresearch 项目（Karpathy，2026）：该项目证明，让 AI 智能体在大语言模型训练代码上执行保留/回退循环，能够在单个图形处理器上用两天时间、经过 700 次实验发现 20 项优化。AutoKernel 将这一循环移植到内核代码中：搜索空间变为所有可能的内核实现，评估函数则是经过正确性门控的基准测试，而非验证损失。

关键洞见：优化真正重要的部分。与将内核问题孤立处理的既有工作不同，AUTOKERNEL 面向完整模型运行。它会分析任意 PyTorch 模型，找出消耗 GPU 时间最多的内核，并应用阿姆达尔定律，按比例分配优化资源。对于占总运行时间 60% 的内核，实现 1.5× 的加速比可带来 1.25×的端到端加速比。同样的加速比用于占比仅为 5% 的内核时，收益则只有 1.03×。该系统会将精力集中在能够产生累积效应的地方。

贡献。
1. 一个完整的开源流水线（9,200+ 行 Python 代码，以及智能体指令），用于自主 GPU 内核优化，涵盖从模型性能分析到端到端验证的全过程。2. 一个五阶段正确性测试框架，测试 10+ 种输入形状、三种数据类型、对抗性输入、确定性以及非二次幂边界情况。3. 一个双后端架构，包含 9 个 Triton 和 9 个 CUDA C++ 初始内核，覆盖 Transformer 模型的主要操作。4. 一个基于阿姆达尔定律的编排器，设有明确的继续条件（平台期检测、峰值利用率、时间预算、加速阈值）。5. 一套六层优化手册，将专家级内核工程知识编写为智能体可读的指令。6. 集成 KernelBench 基准测试套件与 HuggingFace Kernels 分发平台。

2 相关工作

2.1 GPU 内核语言与编译器

Triton（Tillet 等，2019）引入了基于分块的图形处理器编程方式，开发者只需考虑块级张量运算，而无需逐个处理线程。编译器负责内存合并访问、共享内存分配和指令调度，使 FP16 矩阵乘法内核仅用不到 25 行代码就能达到与 cuBLAS 相当的性能。CUDA C++（英伟达，2024）仍是

表 1：基于大语言模型的图形处理器内核优化系统比较。AUTOKERNEL 是唯一结合模型级性能分析并支持 Triton/CUDA C++ 双后端的系统。

系统	方法	后端	模型 迭代 开放 性能分析		src
KernelBench（Ouyang 等人，2025）	基准测试	多领域专用语言	✗	✗	✓
GEAK（Wang 等人，2025）	多智能体	Triton	✗	✓	✓
CudaForge（Zhang 等人，2025）	编码器+评判器	CUDA	✗	✓	✓
KernelFoundry（Wiedemann 等人，2026）	演化式	CUDA/SYCL	✗	✓	✓
内核科学家（Andrews 和 Witteveen，2025）	演化式	CUDA	✗	✓	✓
CUDA-L1（Li 等人，2026）	RL	CUDA	✗	✓	✓
CUDA Agent（Dai 等人，2026）	智能体强化学习	CUDA	✗	✓	✓
Astra（Wei 等人，2025）	多智能体	CUDA	✗	✓	✓
KernelSkill（Sun 等人，2026）	多智能体	CUDA	✗	✓	✓
AutoKernel	迭代循环	Triton+CUDA	✓	✓	✓

实现对线程束级原语、张量核心指令（WMMA/MMA）和无共享内存组冲突布局进行显式访问的必要手段。PyTorch 2 的 TorchInductor（Ansel 等，2024）可将捕获的计算图自动编译为 Triton 内核，在 180+ 种模型上实现 2.27× 的推理加速比和 1.41× 的训练加速比。AutoKernel 同时支持 Triton 和 CUDA C++：Triton 适合快速迭代（编译耗时 1 至 5 秒），CUDA C++ 则能提供最大的控制权。

2.2 优化后的内核库

FlashAttention（Dao 等，2022）证明，采用在线 Softmax 的分块注意力可以将高带宽内存的读写量从随序列长度呈平方级增长降至线性增长，实现 2 至 4× 倍的壁钟时间加速。FlashAttention-2（Dao，2023）在 A100 上达到了理论峰值性能的 50% 至 73%。FlashAttention-3（Shah 等，2024）利用 Hopper 的异步性（TMA、WGMMA）和 FP8 量化，在 H100 上实现了 840 万亿次浮点运算每秒（利用率为 85%）。CUTLASS（Thakkar 等，2023）提供了可组合的张量核心通用矩阵乘法构建模块。这些库代表了 AutoKernel 智能体所追求的性能上限。

2.3 基于大语言模型的内核生成

表 1 从多个维度比较了近期的系统。

KernelBench（Ouyang 等人，2025）提供了标准评估基础设施。GEAK（Wang 等人，2025）为 AMD 硬件上的 Triton 内核引入了生成、评估、反思和优化智能体。CudaForge（Zhang 等人，2025）将编码器与硬件感知评审器配对，由后者提供性能分析反馈。KernelFoundry（Wiedemann 等人，2026）采用结合元提示词演化的 MAP-Elites 质量-多样性搜索，并报告称在 SYCL 的 KernelBench 上平均加速比为 2.3× 。GPU Kernel Scientist（Andrews 和 Witteveen，2025）将优化表述为 AMD MI300 上的多阶段演化。CUDA-L1（Li 等人，2026）将监督微调与对比强化学习相结合，实现了 3.12× 的平均加速比，并被 ICLR 2026 接收。CUDA Agent（Dai 等人，2026）采用大规模智能体强化学习，在 KernelBench 的第 1 和第 2 级上实现了 100% 的性能优于 torch.compile 比例，在第 3 级上则达到 92%。Astra（Wei 等人，2025）协调专门的性能分析智能体和规划智能体，报告称平均加速比为 1.32× 。KernelSkill（Sun 等人，2026）引入了包含可复用专家优化技能的双层记忆，在第 1 级达到 5.44× 。

AutoKernel 与这些系统有三点不同。首先，它从完整的 PyTorch 模型出发，对其进行性能分析，并重点优化那些对总运行时间影响最大的内核。其次，它在单一框架中同时支持 Triton 和 CUDA C++ 后端，让智能体能够针对每个内核选择合适的抽象层级。第三，它采用简单的循环（编辑、基准测试、保留/回退），并配合五阶段正确性测试框架，而非依赖多智能体协作或学习型策略，以架构上的简洁换取透明性与可靠性。

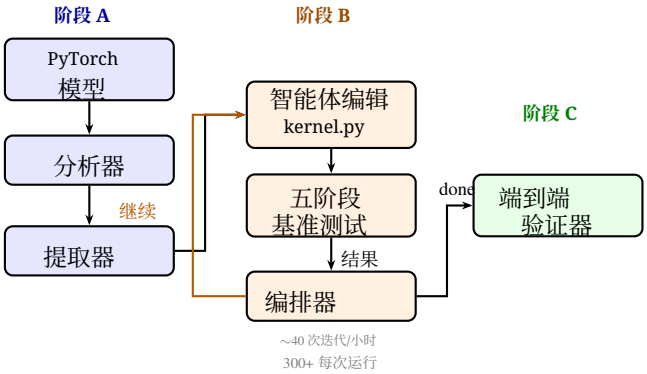


图 1：AutoKernel 架构。阶段 A 对模型进行分析并提取瓶颈内核。阶段 B 运行自主优化循环：智能体编辑 kernel.py，基准测试通过五个正确性阶段进行验证，编排器则决定保留、更改回退，还是转到下一个内核。阶段 C 验证端到端正确性和加速比。

2.4 自主研究智能体

Karpathy 的 autoresearch（Karpathy，2026）展示了自主大语言模型训练优化：智能体修改一份 630 行的训练脚本，运行固定的 5 分钟评估，并提交改进成果。它在两天内于单块图形处理器上运行了 700 次实验，无需人工介入便发现了 20 项训练优化。AutoKernel 将这一范式应用于内核优化，但采用了不同的搜索空间和评估函数。

3 系统设计

AutoKernel 包含 14 个核心脚本，共 9,200 多行 Python 代码、18 个内核实现（9 个 Triton、9 个 CUDA C++）、4 个模型定义，以及一份长达 909 行的智能体指令文档（program.md）。系统分三个阶段运行（图 1）。

3.1 模型分析（阶段 A）

分析器（profile.py，共 1,125 行）接受本地 Python 文件、HuggingFace 模型标识符或自定义模型类。它使用 torch.profiler 和形状记录功能，在可配置的预热迭代与分析迭代期间捕获每个内核的图形处理器时间（默认：预热 5 次，分析 10 次）。

每个 CUDA 内核都会通过对内核名称进行模式匹配，被归类为九种操作类型之一（第 6 节）。分类器能够处理特定于供应商的命名方式：cuBLAS 通用矩阵乘法变体、CUTLASS 内核、Triton 编译函数和 ATen 算子各自具有不同的命名模式，但都会映射到相同的操作类型。

分析器根据已知规格数据库检测 GPU 硬件，数据库涵盖英伟达（H100、A100、L40S、L4、A10、RTX 4090/4080/3090/3080）和 AMD（MI300X、MI325X、MI350X、MI355X）加速器。对于未知 GPU，分析器会根据 SM 数量、时钟频率和计算能力估算峰值 FP16 吞吐量。当设备名称为空时（这是 ROCm 中的常见情况），则通过 gcnArchName 属性检测 AMD GPU。

可选输出包括 Chrometrace JSON、CUDA 内存快照、torch.compile 日志和 HTA（整体跟踪分析）结果。

3.2 内核提取

提取器（extract.py，648 行）读取性能分析结果，筛选受支持的操作类型，并生成独立内核文件，其中包含初始实现、模型特定形状（主要、半尺寸和双倍尺寸变体）、用于屋顶线计算的浮点运算次数和字节数公式，以及数据类型特定容差。

算法 1 单个内核的 AutoKernel 优化循环 要求：内核文件 k，基准测试 B，智能体 A，继续条件 C1: $k_{\text{best}} \leftarrow k$； $t_{\text{best}} \leftarrow B(k)$； $n_{\text{rev}} \leftarrow 0$2: 对于 $i = 1,2,\ldots$ do 3: $k' \leftarrow A$. edit(k_{best}, history,roofline) 4: Git 提交 k'5: (pass t') $\leftarrow B(k')$ {五阶段正确性检查 + perf} 6: 如果通过且 $t' > 1.01 \cdot t_{\text{best}}$ 则 7: $k_{\text{best}} \leftarrow k'$； $t_{\text{best}} \leftarrow t'$； $n_{\text{rev}} \leftarrow 0$8: 否则 9: Git 重置 –hard HEAD~1； $n_{\text{rev}} \leftarrow n_{\text{rev}} + 1$10: 结束 if 11: 将 log($i$, t', decision, descriptio n) 写入制表符分隔值 12: 如果 $C(n_{\text{rev}}, t_{\text{best}}$, elapsed) 则 13: break 14: 结束 if 15: 结束 for 16: 返回 $k_{\text{best}}, t_{\text{best}}$ {**keep**}

{**revert**}

它会根据阿姆达尔定律的影响（公式 1）计算优化计划并排序，并在加速比达到 1.5×, 2×, 3× 和 5× 时进行假设性预测。

3.3 智能体优化循环（PhaseB）

AutoKernel 的核心是算法 1。智能体修改一个文件，测试框架对其进行验证和基准测试，并根据结果决定是否保留更改。

计时。 每次迭代约需 90 秒：30 秒用于五阶段正确性检查，30 秒通过 Triton 的 do_bench 进行性能基准测试，另 30 秒用于智能体推理和代码修改。每小时进行 40 次实验，运行 10 小时可完成 300 到 400 次实验。

单文件不变量。 智能体只修改一个文件。这样可以保持差异精简、轻松撤销，并防止出现难以隔离的耦合更改，以免发生性能回退时难以排查。

智能体指令。 智能体会读取 program.md（909 行），其中提供了一个六级优化指南：

1. **块大小调优**（10% 到 50%）：通过 2 的幂次扫描分块维度，尝试矩形分块，并调整 num_warps 和 num_stages。2. **内存访问**（10% 到 30%）：合并加载、软件预取、L2 交错、共享内存填充。3. **计算**（5% 到 15%）：TF32 累加、尾声融合、循环不变量外提。4. **进阶优化**（5% 到 20%）：分割 K、持久化内核、Triton 自动调优、线程束专门化。5. **架构专属优化**（5% 到 15%）：在 Hopper 上使用 TMA，在 Ampere 上使用 cp.async，并针对 L4/RTX 调整尺寸。6. **内核专属优化**：针对注意力使用在线 Softmax，针对归一化使用 Welford 算法，针对高瘦矩阵乘法使用分割 K。

3.4 多内核编排

编排器（orchestrate.py，共 842 行）运用阿姆达尔定律（Amdahl，1967）来确定多个内核的优化工作优先级：

$$S = \frac{1}{(1-f) + f/s} \tag{1}$$

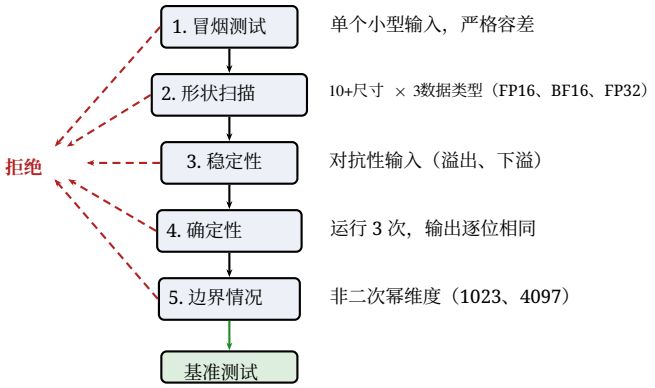


图 2：五阶段正确性流水线。任何失败都会立即拒绝该候选项。只有通过全部五个阶段后，才会测量吞吐量。每个阶段都会捕获一类不同的错误。

表 2：矩阵乘法的形状扫描。其他内核也采用类似的扫描，包含 8 到 10 种配置，涵盖极小、标准、大型、非方形以及特定工作负载维度。

Name	M	N	K	用途
tiny	128	128	128	最低可行
小型	512	512	512	小型工作负载
大型	2048	2048	2048	类生产环境
超大型	4096	4096	4096	压力测试
tall	8192	1024	1024	非方形
深度_k	1024	1024	8192	大规约维度
大语言模型_qkv	4096	4096	512	注意力模式
大语言模型_mlp	4096	11008	4096	前馈模式

其中， f 是该内核占 GPU 总时间的比例， s 是所实现的加速比。满足以下任一条件时，将转向下一个内核：(1) 连续 5 次回退；(2) 达到 GPU 峰值性能的 90%；(3) 已经过 2 小时；或 (4) 达到 $2\times$ 加速比。

4 五阶段正确性验证

基准测试框架（bench.py，共 1,416 行）通过五个阶段（图 2）确保正确性。所有阶段均须通过，才能测量性能。

阶段 1：冒烟测试。 在小型输入（例如 128×128）上执行一次前向传播，可在 1 秒内发现编译错误、形状不匹配和明显的数值错误。

阶段 2：形状扫描。 内核会针对 8 至 10 种输入配置和三种数据类型运行。表2展示了矩阵乘法扫描。此阶段可发现尺寸相关错误：边界处理、分块余数逻辑以及数据类型特定问题。

阶段 3：数值稳定性。 通过对抗性输入探测浮点边界情况。对于 Softmax：使用数值较大且相同的行。对于矩阵乘法：使用动态范围极大的数据。对于归一化：使用接近零的方差。

阶段 4：确定性。 同一输入运行三次，输出必须逐位相同。此阶段可发现并行归约中的竞态条件和非确定性原子操作。

阶段 5：边界情况。 非二次幂维度（1023、4097、1537）会暴露掩码错误和分块余数错误。

容差因数据类型而异：FP16 使用 atol $= 10^{-2}$，BF16 使用 2×10^{-2}，FP32 使用 10^{-4}.

Ta表 3：支持的内核类型、性能区间及初始实现技术。

内核	区间	指标	关键初始技术
矩阵乘法	计算	万亿次浮点运算每秒	128×128 分块，WMMA，双重-缓冲共享内存
Flash Attention_	计算	万亿次浮点运算每秒	分块在线 Softmax、因果掩码
融合_多层感知机	计算	万亿次浮点运算每秒	SwiGLU 门控与上投影融合
Softmax	内存	GB/s	Warp shuffle 归约、half2 加载
层归一化	内存	GB/s	Welford 单遍算法，float4 加载
RMSNorm	内存	GB/s	Warp 洗牌级联，快速倒数平方根
交叉熵_	内存	GB/s	在线对数求和指数，融合负对数似然
旋转位置编码_嵌入	内存	GB/s	交错旋转，sincosf
归约	内存	GB/s	分层式线程束洗牌

5 双后端：Triton 和 CUDA C++

Triton。 提供九个以类 Python 领域特定语言编写的初始内核，即时编译耗时 1 到 5 秒。智能体可以修改块大小、线程束、阶段、累加器精度和循环结构。对于矩阵乘法，Triton 通常可达到 cuBLAS 吞吐量的 80% 到 95%。

CUDA C++。 九个可直接访问张量核心（WMMA API、 16×16×16片段）的初始内核，采用线程束级洗牌（__shfl_xor_sync）、向量化加载（float4、half2）、无 bank 冲突的共享内存布局、双缓冲，以及__启动_边界__寄存器控制。编译使用 load_inline，结合基于哈希的缓存、架构自动检测和线程安全构建。

Bot这两个后端提供相同的 kernel_fn() 接口。无论使用哪个后端，基准测试的运行方式都相同 s of b后端。

6 内核覆盖

表3总结了支持的九种内核类型。

每种内核类型都在 reference.py 中有对应的 PyTorch 参考实现，作为正确性判定基准。基准测试会计算吞吐量（万亿次浮点运算每秒或 GB/s）以及相对于检测到的图形处理器峰值性能的屋顶线利用率。

系统随附四种独立的模型定义（GPT-2 124M、LLaMA 160M/7B、BERT-base 110M、自定义模板），无需依赖 transformers 等外部组件。

7 实验评估

我们在配备 80GB HBM3 的 NVIDIA H100 图形处理器上评估 AutoKernel（132 个流式多处理器，计算能力 9.0，CUDA 12.8）。所有测量均使用 FP16 精度和 CUDA 事件计时；每种配置运行 200 次，并取截尾均值（剔除最高和最低的 10%）。我们将其与两个基线进行比较：PyTorch 即时执行（矩阵乘法使用 cuBLAS，其他运算使用 ATen），以及采用最大自动调优模式的 torch.compile；后者会触发 TorchInductor 在多种 Triton 内核配置上进行自动调优。涵盖 7 种内核类型和 34 种基准测试配置的完整评估，实际耗时不到 10 分钟。全部 34 种配置均通过正确性检查，零失败。

7.1 内核性能

表 4 汇报了与生产环境相关的各种张量尺寸下的性能。我们重点展示最大尺寸下的结果，此时计算与开销比最能代表实际工作负载。

关键发现。

• **内存受限内核的性能提升最大**。 在测试的最大规模下，RMSNorm 相比 eager 提升 5.29× ，相比 torch.compile 提升 2.83×，达到 2,788 GB/s（H100 峰值带宽 3,352 GB/s 的 83%）。交叉熵达到 2,070 GB/s。Softmax 达到 2,800 GB/s。这些提升

表 4：H100（FP16）上的内核性能，所选配置。三个基线：PyTorch eager（cuBLAS/ATen）、torch.compile（max-autotune）和 AutoKernel Triton 起始配置。展示了从完整测试的 34 种配置中选出的 16 种代表性配置（均通过正确性检查）。完整评估在 10 分钟内完成。

内核	Size	急切模式 (μs)	编译模式 (μs)	我们的方法 (μs)	我们的方法 vs 急切模式	我们的方法 vs 编译模式	吞吐量。
矩阵乘法	2048^3	28.1	101.2	65.3	0.43×	**1.55**×	263 太拉浮点运算每秒
	4096^3	182.8	257.8	494.2	0.37×	0.52×	278 太拉浮点运算每秒
	8192^3	1679.5	1916.1	5773.1	0.29×	0.33×	190 太拉浮点运算每秒
Softmax	4096^2	58.9	96.1	40.1	**1.47**×	**2.40**×	1675GB/s
	8192^2	270.4	330.0	95.9	**2.82**×	**3.44**×	2800GB/s
层归一化	4096×5120	45.6	105.5	42.5	**1.07**×	**2.48**×	1974GB/s
	8192×4096	64.7	166.8	51.9	**1.25**×	**3.21**×	2586GB/s
RMSNorm	4096^2	142.8	99.9	39.1	**3.65**×	**2.56**×	1716GB/s
	8192×4096	262.4	138.1	51.2	**5.12**×	**2.70**×	2619GB/s
	8192^2	509.6	272.1	96.3	**5.29**×	**2.83**×	2788GB/s
交叉熵_	4096×32k	295.6	386.3	134.9	**2.19**×	**2.86**×	1943GB/s
	8192×32k	559.7	745.1	253.3	**2.21**×	**2.94**×	2070GB/s
归约	8192^2	60.7	185.2	62.2	0.98×	**2.98**×	2156GB/s
	16384×4096	50.4	185.9	52.8	0.95×	**3.52**×	2542GB/s
旋转	2×32×2k×128	211.4	106.9	117.4	**1.80**×	0.91×	576 GB/s
	2×32×4k×128	394.9	136.0	223.0	**1.77**×	0.61×	607 GB/s

这源于将多个 ATen 分解操作融合为单次处理的 Triton 内核，从而最大限度地减少高带宽内存流量。

• **AutoKernel 在大多数内核上都优于** torch.compile 。 尽管启用 max-autotune 的 torch.compile 会运行自己的 Triton 自动调优，但在表 4 所示的 16 种配置中，我们的初始内核有 12 种表现更佳。TorchInductor 的通用融合和自动调优并不总能找到内核专用实现所利用的专门分块与归约策略。

• **矩阵乘法仍然很难优化**。 PyTorch 的 cuBLAS 后端针对每种 GPU 架构都经过了广泛调优。我们的 Triton 初始内核达到 278 万亿次浮点运算每秒（相当于 H100 峰值性能 989.5 万亿次浮点运算每秒的 28%），远低于 cuBLAS。不过，在 2048_3 规模下，我们的内核比 torch.compile 快 1.55×，这表明 TorchInductor 的矩阵乘法自动调优也并非总是最优。这一差距正是智能体迭代优化循环的主要目标。

• **正确性普遍有保障**。 全部 34 种配置均通过了所有五个验证阶段，在 eager、编译后和自定义内核输出中均未出现失败。

7.2 社区部署成果

AutoKernel 已被社区用户部署，用于具有竞争力的图形处理器内核基准测试。以下两个结果展现了该系统的实际影响：

向量求和归约（B200）。在向量求和_v2 挑战赛中，参赛者需要为形状为（N，）且元素数量最多达数百万的张量实现求和归约内核。经过 AutoKernel 优化的 Triton 提交方案以 44.086μs 的延迟在 NVIDIA B200 排行榜上获得第一名，超过了第二名（44.249μs）和第三名（46.553μs）。获胜内核由 AutoKernel 的迭代优化循环在夜间运行期间生成，探索了块大小、warp 级 shuffle 归约以及向量化内存访问模式。

FP4 矩阵乘法：Triton 超越 CUTLASS。一位社区用户报告称，只需一次 AutoKernel 提示，并经过约 3 分钟的智能体交互，就生成了一个 Triton FP4 矩阵乘法内核，在 H100 上的多种形状下均优于 CUTLASS。表5展示了结果：与 CUTLASS 融合内核相比，Triton 内核的加速比达到 1.63× 至 2.15× 倍，在形状为 2048×18432×3072 时最高可达 2,898 万亿次浮点运算每秒。这一结果值得关注，因为 CUTLASS 代表着

表5：社区报告的 H100 上 FP4 矩阵乘法结果：AutoKernel Triton 内核与 CUTLASS 融合基线对比。Triton 内核由单条智能体提示生成（∼3 分钟）。

形状 (M×N×K)	Triton (太拉浮点运算每秒，毫秒)	CUTLASS 加速比 (太拉浮点运算每秒，毫秒)	
128×3072×3072	186 TFLOPS, 0.013	100 TF, 0.024	**1.86**×
128×18432×3072	1105 太浮点运算, 0.013	550 TF, 0.026	**2.01**×
1024×3072×3072	1477 TF, 0.013	686 TF, 0.028	**2.15**×
1024×18432×3072	2777TF, 0.042	1609 TF, 0.072	**1.73**×
2048×3072×3072	1662 TF, 0.023	964 TF, 0.040	**1.72**×
2048×18432×3072	2898TF, 0.080	1777 TF, 0.130	**1.63**×
4096×3072×3072	2443 TF, 0.032	1405 TF, 0.055	**1.74**×

专为 NVIDIA 张量核心设计的手工优化 C++ 模板代码，而通过一次智能体交互生成的 Triton 内核能够超越它，这证明了优化手册的有效性。

7.3 优化循环动态

迭代优化循环针对表 4 所示的初始内核运行。根据优化手册（第 3.3 节），智能体按层级顺序应用变更。对于 RMSNorm 这类内存受限的内核，典型的优化进程如下：

1. **层级 1**（块大小）：将 BLOCK_SIZE 从 256 扫描到 4096，通常在前 5 到 10 次实验中可带来 10% 到 30% 的提升。2. **层级 2**（内存）：向量化加载（tl.load 与 BLOCK_SIZE 对齐）和合并访问修复，在接下来的 10 到 20 次实验中可再提升 10% 到 20%。3. **层级 3**（计算）：将权重乘法融合到归一化尾处理中，省去一次全局内存往返，可带来 5% 到 10% 的提升。4. **平台期**：经过 30 到 50 次实验后，随着内核逐渐接近硬件极限，连续回退的情况会增加。编排器会触发转向下一项。

对于矩阵乘法等计算受限内核，情况则有所不同：第 1 层块大小调优的影响更大（我们的初始方案为 278 TFLOPS，而 cuBLAS 达到 800+ TFLOPS，两者差距很大），但每次改进都需要进行更多实验，因为分块维度、线程束数量、流水线阶段和累加器精度构成的搜索空间更大。

继续条件（连续 5 次回退、达到峰值的 90%、超时 2 小时或 2× 加速比）可避免智能体在收益递减的内核上耗费过多时间。对于一个矩阵乘法占 GPU 时间 62%、RMSNorm 占 5% 的模型，根据阿姆达尔定律，即使矩阵乘法只有小幅改进（例如 1.3×），其端到端收益也会超过 RMSNorm 的大幅改进（而 RMSNorm 已经达到 5.29×）。

8 KernelBench 集成

AutoKernel 通过三个组件与 KernelBench（Ouyang 等人，2025）集成：

1. **问题加载器**（bridge.py，674 行）。从 HuggingFace 数据集或本地克隆版本获取数据。分析运算并生成初始 ModelNew 类。2. **评估框架**（bench_kb.py，726 行）。分四个阶段进行验证：正确性（5 次试验，atol $= 10^{-2}$）、稳定性（NaN/Inf）、确定性（运行 3 次）、性能（CUDA 事件，截尾中位数）。3. **批量评分器**（scorer.py，351 行）。在七个阈值下计算 fast_p：1.0× 至 5.0×。

虽然大多数 KernelBench 评测报告的是单次运行得分，但 AutoKernel 会针对每个问题进行 50 到 300 次迭代实验。

9 HuggingFace 内核导出

一个导出工具（export_hf.py，共 868 行）会将优化后的 CUDA 内核打包，以便通过 HuggingFace Hub 分发（HuggingFace，2025）。它会生成：包含后端和 Hub 元数据的 build.toml；使用 TORCH_LIBRARY_EXPAND 进行基于 schema 的注册的 torch_binding.cpp；以及导入 huggingface/kernels 内核构建器的 flake.nix，以便在不同 PyTorch/CUDA/ABI 变体之间进行交叉编译。用户可通过以下命令安装优化后的内核：module =get_kernel("rightnow-ai/matmul").

10 设计依据

简洁胜于繁复。 由专门的规划、编码、审查和性能分析智能体组成的多智能体系统会增加协调开销和故障模式。单个智能体在紧凑的循环中运行，可生成线性的实验历史，并避免寻求共识所带来的延迟。

评估固定，代码可变。 智能体绝不会修改基准测试。这道防火墙可防止操纵评估结果；当候选生成器同时控制评分器时，这确实是一个风险。

将 Git 用于实验跟踪。 每次实验都对应一个 Git 提交。保留的实验会推进分支；撤销的实验则通过 Git 重置清除。使用标准 Git 工具即可浏览历史记录。

屋顶线引导搜索。 每次实验后，基准测试都会报告计算受限或内存受限的分类，以及峰值百分比。智能体据此选择合适的优化层级。

制表符分隔值优于数据库。 实验结果保存为制表符分隔文件。无依赖、便于人类阅读、适合 Git 管理，并且智能体可以轻松解析。

11 局限性与未来工作

AutoKernel 的代码生成能力受其底层大语言模型所限。软件流水线、自定义 PTX 生成和多 CTA 协作策略等复杂技术可能超出当前智能体的能力范围，不过随着前沿模型不断进步，这一上限也会逐步提高。目前，该系统针对单个图形处理器上的单个内核进行优化；分布式内核和多设备内存管理不在其范围内。

值得探索的方向包括：在多个 GPU 实例间进行基于种群的搜索；利用历史实验数据训练学习型搜索策略；通过 SASS 分析或内存吞吐量计数器进行性能分析引导的变异；以及在单个内核优化完成后发现跨内核融合。

12 结论

我们介绍了 AutoKernel，这是一种通过智能体驱动的迭代循环自动进行图形处理器内核优化的系统。模型级性能分析用于确定优化目标；阿姆达尔定律用于确定优化顺序；五阶段正确性测试框架确保数值有效性；双后端 Triton/CUDA C++兼顾迭代速度与硬件层面的深入控制；六级操作指南则编码了专家知识。该系统将内核优化从需要专家耗费数周的工作转变为一夜之间即可完成的自主流程。代码和初始内核可在 https://github.com/ RightNow-AI/autokernel 获取。

参考文献

G. M. Amdahl. 实现大规模计算能力的单处理器方法的有效性。载于*AFIPS* 春季联合计算机会议，第 483–485 页，1967 年。

M. Andrews 和 S. Witteveen。《GPU Kernel Scientist：一种由大语言模型驱动的迭代式内核优化框架》。*arXiv:2506.20807*，2025。

J. Ansel、E. Yang、H. He、N. Gimelshein、A. Jain、M. Voznesensky、B. Bao、P. Bell、D. Berard、E. Burber 等。PyTorch 2：通过动态 Python 字节码转换和图编译加速机器学习。发表于 *ASPLOS*，2024。W. Dai、H. Wu、Q. Yu、H. Gao、J. Li、C. Jiang、W. Lou、Y. Song、H. Yu、J. Chen、W. Ma、Y. Zhang、J. Liu、M. Wang、X. Liu 和 H. Zhou。CUDA Agent：面向高性能 CUDA 内核生成的大规模智能体强化学习。*arXiv:2602.24286*，2026。T. Dao、D. Y. Fu、S. Ermon、A. Rudra 和 C. Ré。FlashAttention：具备 I/O 感知能力的快速且节省内存的精确注意力机制。发表于 *NeurIPS*，2022。T. Dao。FlashAttention-2：通过更好的并行性和工作划分实现更快的注意力计算。*arXiv:2307.08691*，2023。HuggingFace。Kernels：直接从 Hub 加载计算内核。https://huggingface.co/ docs/kernels，2025。A. Karpathy。autoresearch：运行单 GPU nanochat 训练研究的人工智能智能体。https: // github.com/karpathy/autoresearch，2026。X. Li、X. Sun、A. Wang、J. Li 和 C. Shum。CUDA-L1：通过对比式强化学习改进 CUDA 优化。发表于 *ICLR*，2026。英伟达。CUDA C++编程指南，v12.6。https://docs.nvidia.com/cuda/ cuda-c-programming-guide/，2024。A. Ouyang、S. Guo、S. Arora、A. L. Zhang、W. Hu、C. Ré 和 A. Mirhoseini。KernelBench：大语言模型能编写高效的图形处理器内核吗？*arXiv:2502.10517*，2025。J. Shah、G. Bikshandi、Y. Zhang、V. Thakkar、P. Ramani 和 T. Dao。FlashAttention-3：具备异步性和低精度计算能力的快速、准确注意力机制。发表于 *NeurIPS*，2024。Q. Sun、J. Han、T. Li、Z. Tang、S. Chen、F. Yang、A. Liu、X. Liu 和 Y. Liu。KernelSkill：用于图形处理器内核优化的多智能体框架。*arXiv:2603.10085*，2026。V. Thakkar 等。CUTLASS。https://github.com/NVIDIA/cutlass，2023。P. Tillet、H.-T. Kung 和 D. D. Cox。Triton：用于分块神经网络计算的中间语言和编译器。发表于 *MAPL@PLDI*，第 10–19 页，2019。J. Wang、V. Joshi、S. Majumder、X. Chao、B. Ding、Z. Liu、P. P. Brahma、D. Li、Z. Liu 和 E. Barsoum。GEAK：推出 Triton 内核人工智能智能体和评测基准。*arXiv:2507.23194*，2025。A. Wei、T. Sun、Y. Seenichamy、H. Song、A. Ouyang、A. Mirhoseini、K. Wang 和 A. Aiken。Astra：用于图形处理器内核性能优化的多智能体系统。*arXiv:2509.07506*，2025。N. Wiedemann、Q. Leboutet、M. Paulitsch、D. Wofk 和 B. Ummenhofer。KernelFoundry：具备硬件感知能力的图形处理器内核进化式优化。*arXiv:2603.12440*，2026。S. Williams、A. Waterman 和 D. Patterson。Roofline：一种用于多核架构的直观可视化性能模型。《*ACM* 通讯》，52(4)：65–76，2009。Z. Zhang、R. Wang、S. Li、Y. Luo、M. Hong 和 C. Ding。CudaForge：一种具备硬件反馈能力的 CUDA 内核优化智能体框架。*arXiv:2511.01884*，2025。