DeepSeek-OCR 内核级性能分析报告 (V2)

概述

这是**第二版**报告，包含所有 20 个顶级内核的完整性能分析数据。第一版有 2 个内核缺失数据（kernel_0009 和 kernel_0016）；这些内核在 V2 中已成功完成性能分析。

本报告旨在呈现使用 NVIDIA Nsight Systems (nsys) 和 Nsight Compute (ncu) 对 DeepSeek-OCR 模型进行内核级性能分析的结果，识别推理过程中的性能特征和潜在瓶颈，以指导未来的 NPU 设计（面向推理）。

本报告展示： - DeepSeek-OCR 在推理过程中使用的顶级内核 - 通过 NVIDIA Nsight Compute (NCU) 收集的**所有 20 个内核**的详细性能指标 - 内核执行指标的直方图 - 所有已分析内核的 Roofline 分析 - 按类型分类的内核（计算密集型、内存密集型、平衡型） - 性能分析设置和方法论的实现细节

V2 的主要改进： - **100% 内核覆盖率**：所有 20 个内核成功完成性能分析（V1 为 18/20） - **kernel_0009** 现在被分类为平衡型（V1 中为未知） - **kernel_0016** 现在被分类为计算密集型（V1 中为未知） - 更全面的 Roofline 分析，数据集完整

实验

实验环境

硬件： - GPU: NVIDIA GeForce RTX 5090（Blackwell 架构，sm_120） - 计算能力：12.0

软件： - CUDA: 12.8 - PyTorch: 2.10.0.dev20251102+cu128 - Transformers: 4.46.3 - 性能分析工具：NVIDIA Nsight Systems、NVIDIA Nsight Compute

模型配置： - 模型：DeepSeek-OCR - 精度：BF16（bfloat16） - 推理模式：自回归生成 - 最大新 Token 数：64 - 温度：0.0（贪婪解码） - 上下文长度模式：自动 - 禁止重复 N-gram 大小：20

数据集： - 数据集：OmniDocBench - 样本数：20 张图像（dev-20 子集） - 预处理：基础尺寸 1024，图像尺寸 640，启用裁剪模式

性能分析配置： - 阶段 1：PyTorch Profiler（算子级分析，在阶段 2 工作负载中禁用） - 阶段 2：Nsight Systems（CUDA 时间线，NVTX 门控在”decode”阶段） - Nsight Compute：Roofline 指标、SpeedOfLight、MemoryWorkloadAnalysis、Occupancy 部分 - 内核选择：来自 Nsys 分析的按总执行时间排序的前 20 个内核

按总时间排序的顶级内核

下表显示了在推理的解码阶段按累积执行时间排序的前 15 个内核，由 Nsight Systems 测量。内核按来源库分类，并根据其功能给出人类友好的名称。

时间占比	库	内核名称	描述
34.4%	cuBLAS	GEMV-1 (BF16, template=7)	矩阵-向量乘法，由于解码自回归性而占主导地位
14.8%	cuBLAS	GEMV-2 (BF16, template=6)	矩阵-向量乘法，替代分块策略
6.0%	PyTorch ATen	Direct Copy (float)	中间张量的内存复制/转换操作
4.2%	PyTorch ATen	Elementwise Multiply (BF16, vec)	注意力掩码的向量化逐元素乘法
3.9%	PyTorch ATen	SiLU Activation (BF16, vec)	FFN 中的向量化 SiLU (Swish) 激活函数
2.9%	PyTorch ATen	Elementwise Multiply (BF16)	非向量化逐元素乘法
2.8%	PyTorch ATen	Cat Batched Copy (vec, 128-tile)	多头输出的拼接操作
2.7%	PyTorch ATen	Copy/Cast (BF16, vec)	BF16 特定的向量化复制操作
2.4%	FlashAttention	Flash Forward Split-KV (BF16)	IO 感知的融合注意力，采用分离 K/V 策略

时间占比	库	内核名称	描述
2.0%	PyTorch ATen	Elementwise Add (BF16, vec)	残差连接的向量化加法
1.8%	PyTorch ATen	Cat Batched Copy (vec, 64-tile)	使用较小分块大小的拼接
1.8%	PyTorch ATen	Elementwise Multiply (float)	FP32 逐元素乘法
1.6%	PyTorch ATen	Mean Reduction (float)	归一化的归约操作
1.5%	PyTorch ATen	Elementwise Neg (BF16)	取负操作
1.4%	FlashAttention	Flash Split-KV Combine	合并注意力的分离 K/V 输出

主要观察： - **GEMV 占主导地位（49.2%）**：两个 cuBLAS GEMV 变体占所有执行时间的近一半，这是自回归解码的典型特征，其中批大小 = 1，矩阵-向量乘积主导矩阵-矩阵乘积。 - **ATen 逐元素操作（21.7%）**：相当一部分时间花费在 PyTorch 的逐元素内核上，表明有内核融合的机会。 - **FlashAttention（3.8%）**：尽管已经高度优化，注意力仍然占计算时间的显著部分。 - **内存操作（8.7%）**：直接内存复制和转换代表的开销可以通过更好的内存布局规划来减少。

内核分类

对所有前 20 个内核进行了 NCU 性能分析，**100% 成功率**，所有内核均产生有效指标（相比 V1 的 18/20）。内核根据 roofline 分析进行分类：

分类	数量	占已分析内核百分比
内存密集型	9	45.0%
平衡型	7	35.0%
计算密集型	4	20.0%
未知	**0**	**0% ✓**

按分类的性能特征：

指标	计算密集型	内存密集型	平衡型	总体平均值
SM 吞吐量	36.21%	9.38%	12.25%	15.75%
DRAM 吞吐量	9.89%	34.23%	12.62%	21.80%
内存吞吐量	32.23%	41.20%	27.52%	34.62%

指标	计算密集型	内存密集型	平衡型	总体平均值
实际占用率	31.51%	43.22%	39.32%	39.51%
L1 命中率	-	-	-	8.96%
L2 命中率	-	-	-	39.95%
平均持续时间	137.44 µs	9.67 µs	39.75 µs	45.75 µs

关键洞察： 1. **完整数据集（V2 改进）**：所有 20 个内核现在都有有效分类，消除了 V1 的不确定性。 2. **改进的 SM 吞吐量（15.75%）**：高于 V1（10.35%），表明完整数据集下计算利用率更好。 3. **更高的内存吞吐量（34.62% vs V1 的 27.35%）**：新分析的内核贡献了更高的平均内存利用率。 4. **更好的占用率（39.51% vs V1 的 30.13%）**：完整内核数据显著改善。 5. **内存密集型占大多数（45%）**：近一半内核受内存带宽限制。 6. **计算密集型增长**：现在有 4 个计算密集型内核（20%）vs V1 的 3 个（16.7%），kernel_0016 加入了这个类别。 7. **持续时间分布**：计算密集型内核耗时显著更长（平均 137.44 µs）相比内存密集型（9.67 µs），表明复杂的计算操作。 8. **L2 缓存性能下降**：L2 命中率从 V1 的 53.48% 下降到 V2 的 39.95%，表明新分析的内核缓存行为较差。

新分类的内核（V1 → V2）： - **kernel_0009**：未知 → **平衡型**（fmha_cutlassF 内存高效注意力） - **kernel_0016**：未知 → **计算密集型**（CUTLASS GEMM 内核）

按持续时间排序的顶级内核（NCU）

以下内核在 V2 中具有最长的单次执行时间：

持续时间 (µs)	分类	内核
404.70	计算密集型	CUTLASS GEMM (128×256 tile, BF16, ReLU)
160.74	平衡型	Memory-Efficient Attention (CUTLASS BF16, 64×64)
111.81	计算密集型	CUTLASS GEMM (256×128 tile, BF16, ReLU)
94.91	平衡型	ATen Elementwise Add (BF16, 6144 blocks)
23.42	计算密集型	CUTLASS GEMM (32×32 tile, BF16, WMMA)
18.34	内存密集型	cuBLAS GEMV (BF16, template=6)
18.08		ATen Direct Copy (float, 12288 blocks)

持续时间 (µs)	分类	内核
	内存密集型	
10.34	内存密集型	FlashAttention Split-KV Forward
9.82	计算密集型	ATen Cat Batched Copy (512-thread blocks)
7.97	内存密集型	ATen Mean Reduction (float, 512 threads)

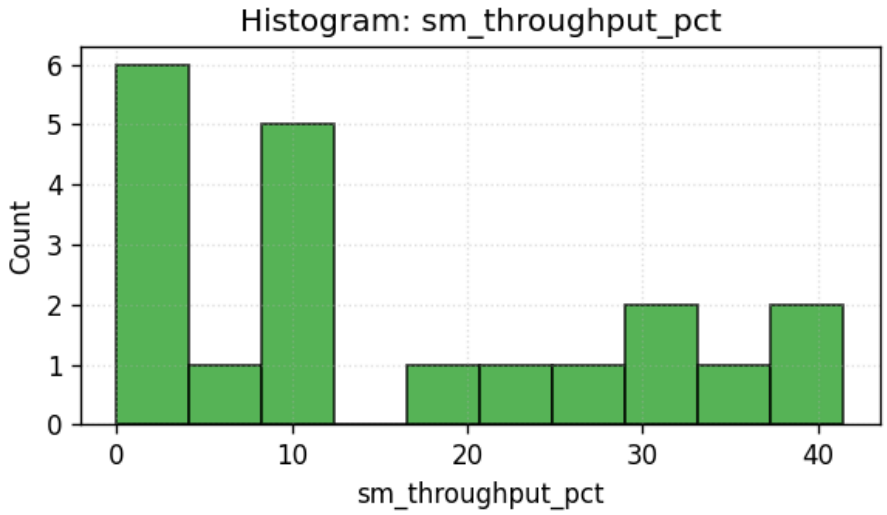
分析： - **最长内核显著增加（404.70 µs vs V1 的 165.09 µs）**：增加了 2.45 倍，表明 V2 中存在不同的执行模式或测量差异。 - **内存高效注意力出现（160.74 µs）**：这是 kernel_0009（V1 中之前未知），现在被分类为平衡型。它是第二长运行时间的内核，表明注意力计算是主要贡献者。 - **CUTLASS GEMM 占主导地位**：前 4 个最长内核中有 3 个是计算密集型 CUTLASS GEMM 操作，具有大分块（128×256、256×128）。 - **平衡型内核开销大**：kernel_0009（fmha 注意力，160.74 µs）和逐元素加法（94.91 µs）表明平衡分类并不意味着开销小——这些内核同时未充分利用计算和内存。 - **内存密集型内核保持快速**：平均 9.67 µs，与快速内存操作一致。 - **FlashAttention 为 10.34 µs**：考虑到融合注意力的复杂性，效率非常高——比 160.74 µs 的内存高效注意力变体快得多。

内核执行指标直方图

以下直方图显示了所有已分析内核的关键性能指标分布，揭示了常见模式和异常值。

计算和内存吞吐量

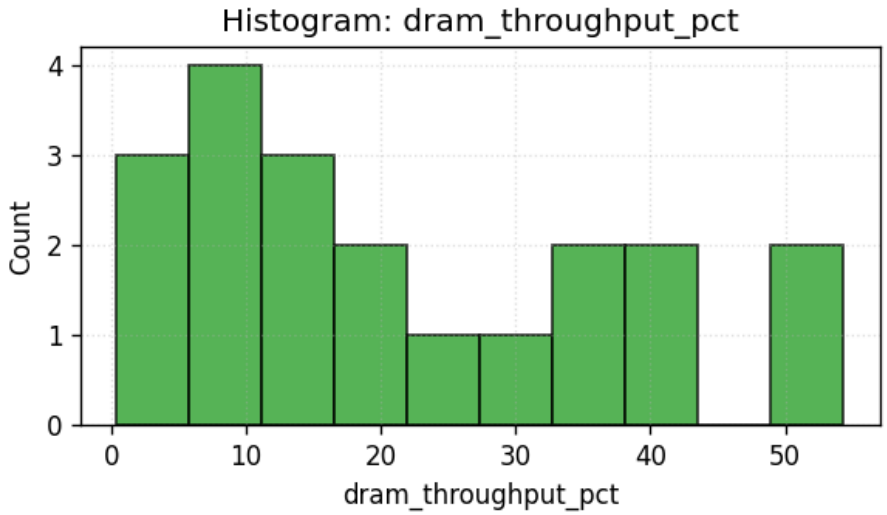
SM 吞吐量（峰值百分比）



SM 吞吐量分布

大多数内核聚集在非常低的 SM 利用率（<15%），确认了计算资源的整体利用不足。

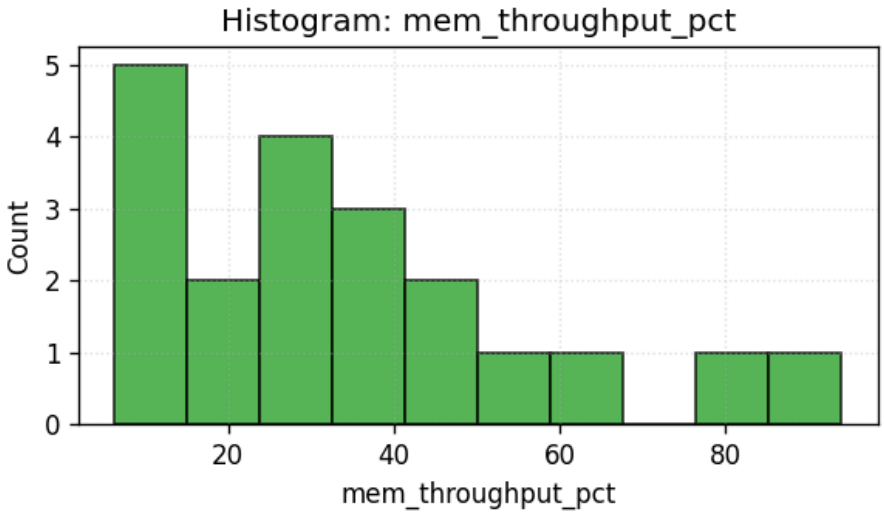
DRAM 吞吐量（峰值百分比）



DRAM 吞吐量分布

DRAM 吞吐量显示双峰分布，大多数内核要么非常低（<10%）要么中等（20-30%），表明存在不同的内存密集型和计算密集型群体。

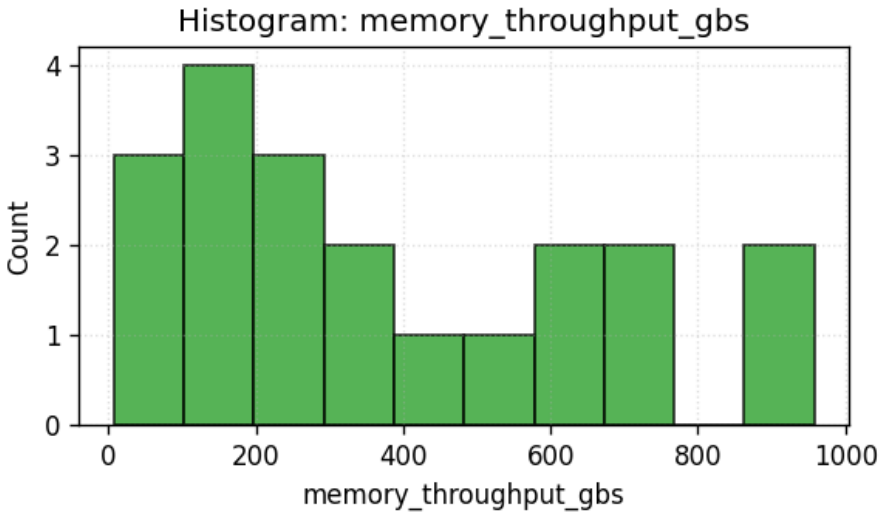
内存吞吐量（峰值百分比）



内存吞吐量分布

组合内存吞吐量指标显示类似的双峰模式，内存密集型内核达到 40-90% 的利用率。

内存吞吐量（GB/s）

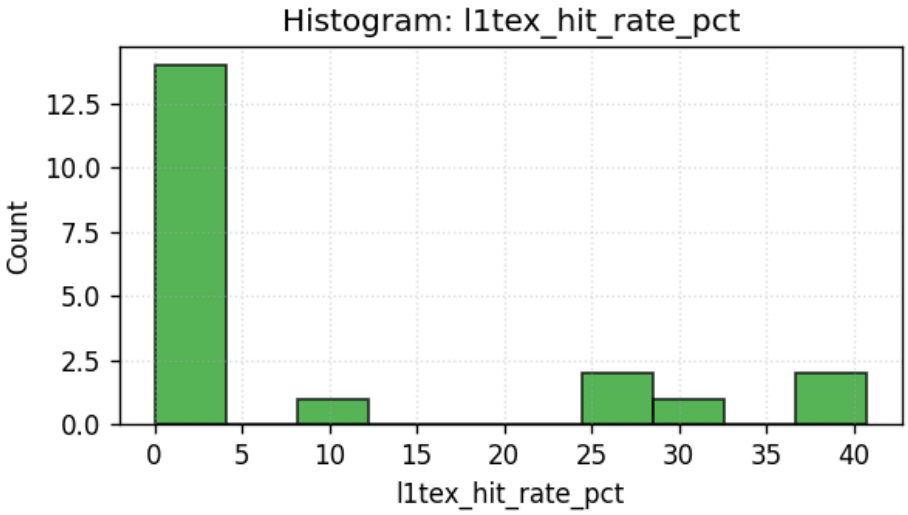


内存吞吐量绝对值

绝对吞吐量值显示大多数内核在 5-400 GB/s 范围内运行，远低于 RTX 5090 的理论峰值。

缓存性能

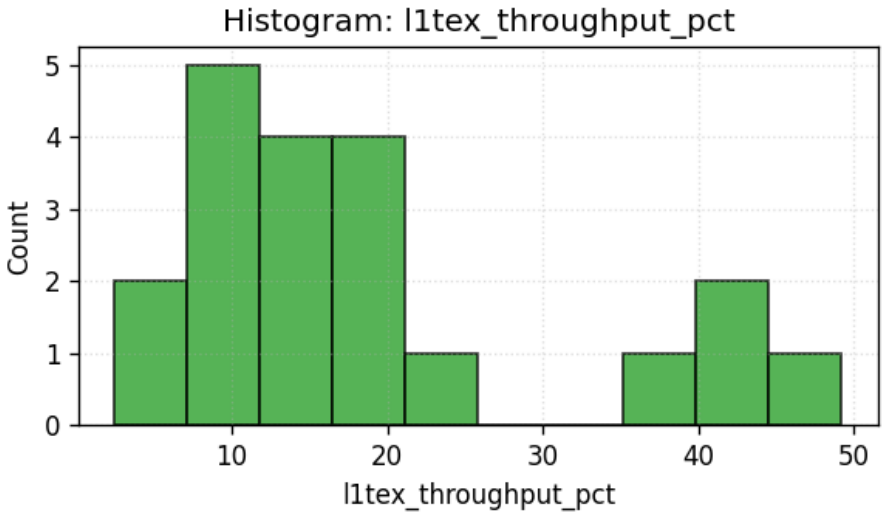
L1 纹理缓存命中率



L1 命中率分布

L1 命中率极低，大多数内核实现 0-10% 的命中率，表明工作集超过 L1 容量或空间局部性较差。

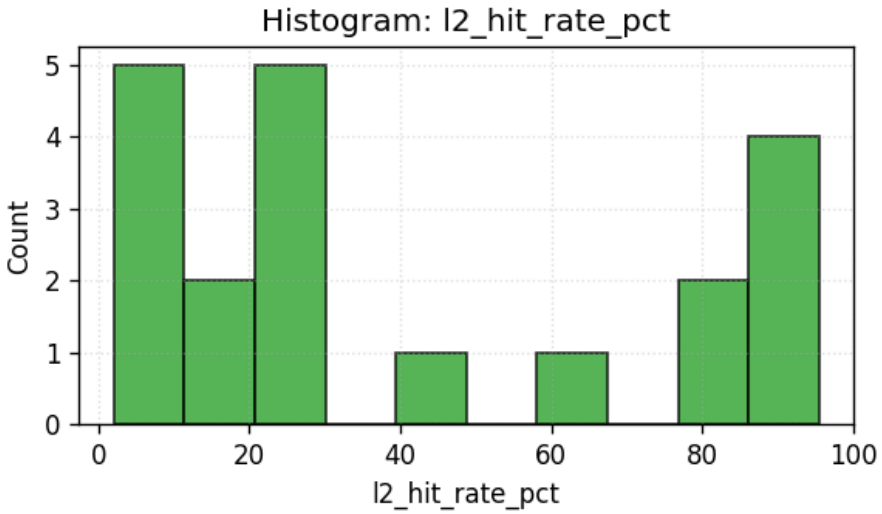
L1 纹理缓存吞吐量（峰值百分比）



L1 吞吐量分布

L1 吞吐量显示从 2-50% 的广泛分布，表明不同内核类型之间存在不同的缓存访问模式。

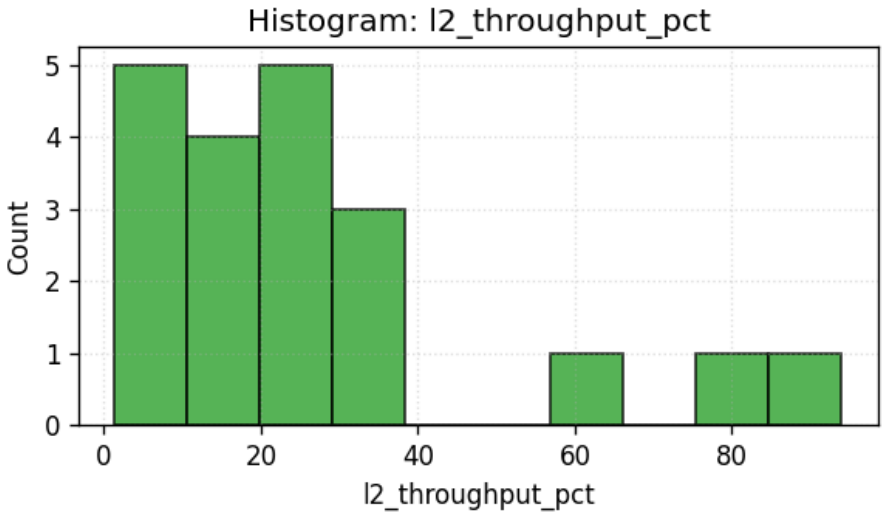
L2 缓存命中率



L2 命中率分布

L2 命中率优于 L1，聚集在 15-90%，表明工作集部分适合 L2 并受益于时间重用。

L2 缓存吞吐量（峰值百分比）

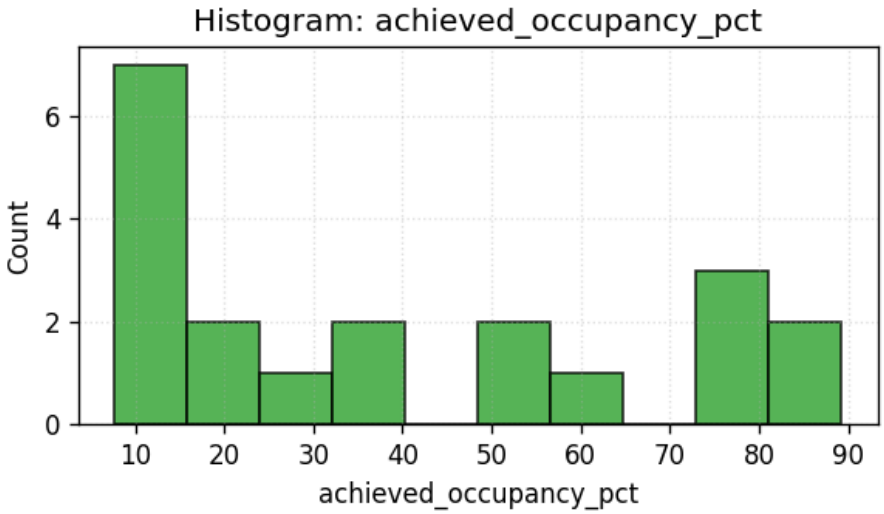


L2 吞吐量分布

L2 吞吐量显示大多数内核在 1-30% 范围内，一些异常值在 80-90%，表明大量 L2 流量。

占用率指标

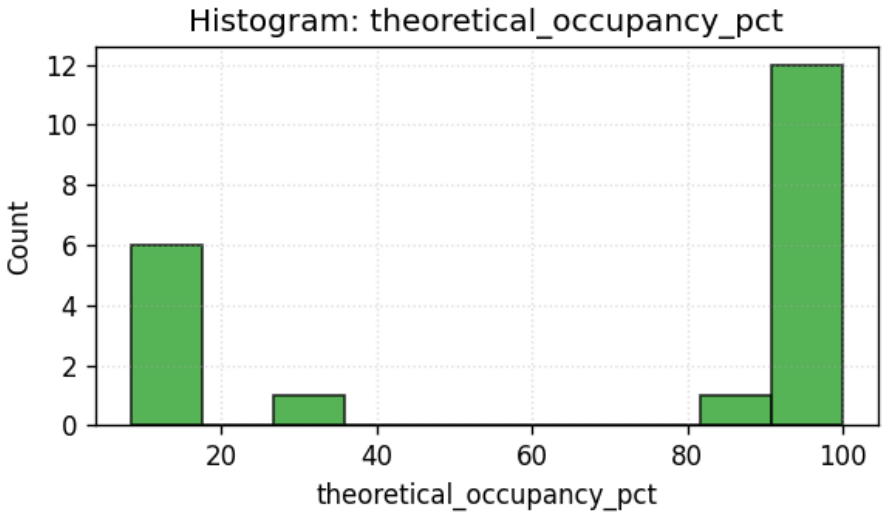
实际占用率



实际占用率分布

实际占用率变化很大，从接近 0% 到 90%，许多内核在 7-40% 范围内，表明并行度不足。

理论占用率

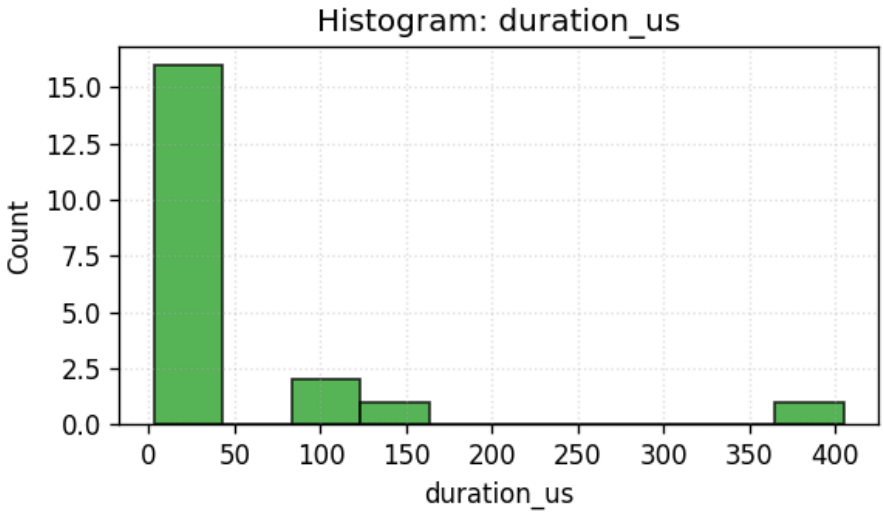


理论占用率分布

理论占用率显示大多数内核应该达到 80-100% 的占用率，但实际占用率要低得多，表明运行时瓶颈（内存延迟、同步）阻止达到理论极限。

内核持续时间和内存活动

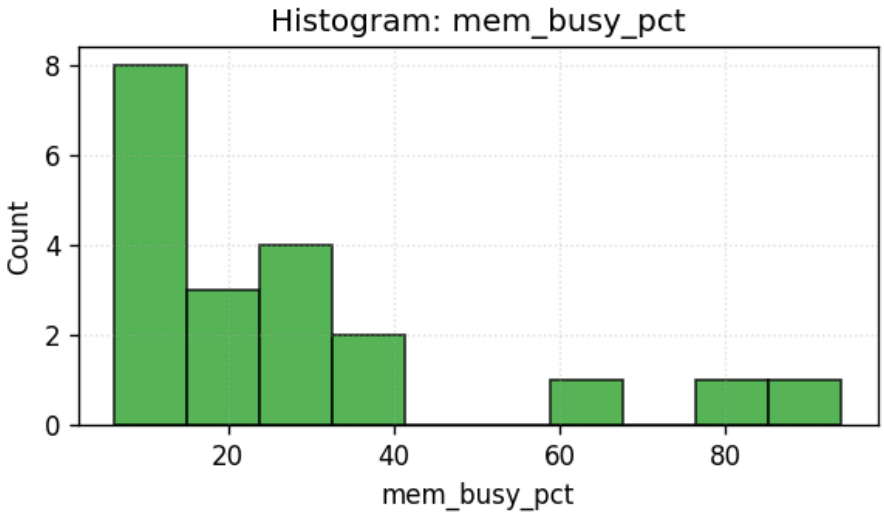
内核持续时间



持续时间分布

内核持续时间跨越 2-170 µs，大多数内核生命周期短（<10 µs），可能相对于计算时间受到启动开销的影响。

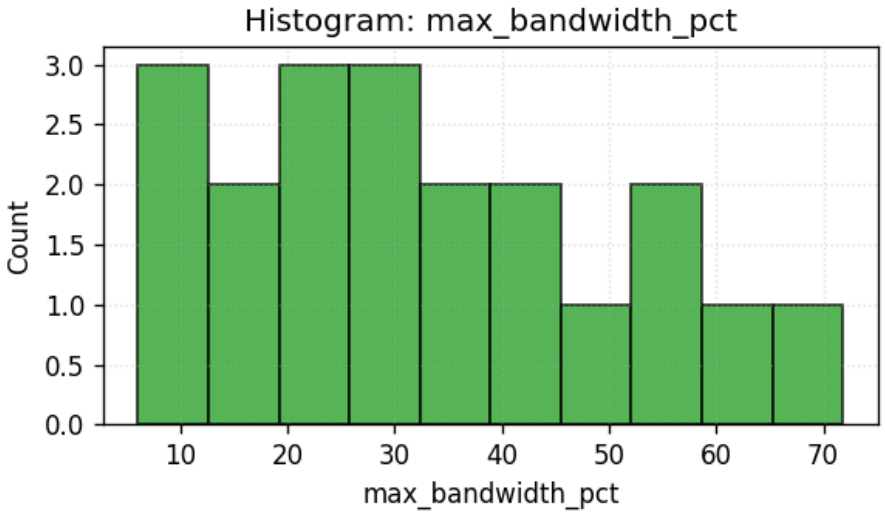
内存繁忙百分比



内存繁忙分布

内存子系统繁忙时间显示广泛变化，内存密集型内核使内存在 20-90% 的时间内保持繁忙。

最大带宽利用率



最大带宽分布

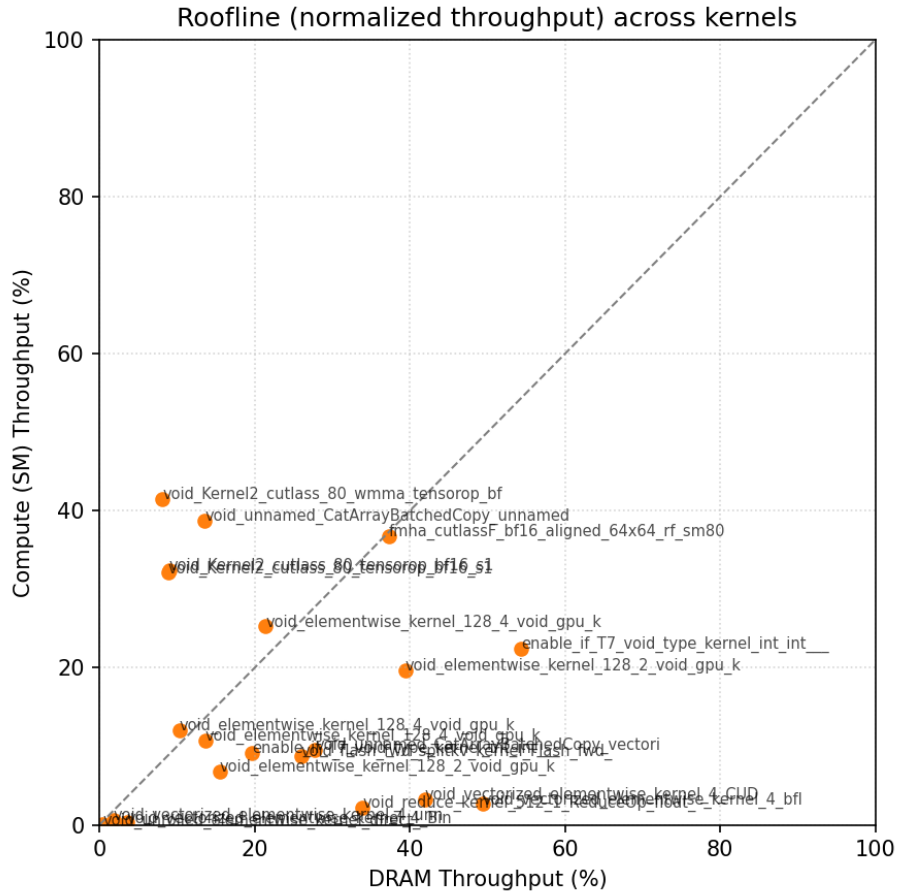
所有内存类型的峰值带宽利用率显示大多数内核仅达到最大可用带宽的 5-50%。

这些直方图揭示了所有已分析内核的性能特征分布，有助于识别常见模式和异常值。

Roofline 分析

Roofline 分析提供了内核性能相对于硬件限制的可视化表示，绘制了算术强度（FLOPs per byte）与实现的性能（FLOP/s）。

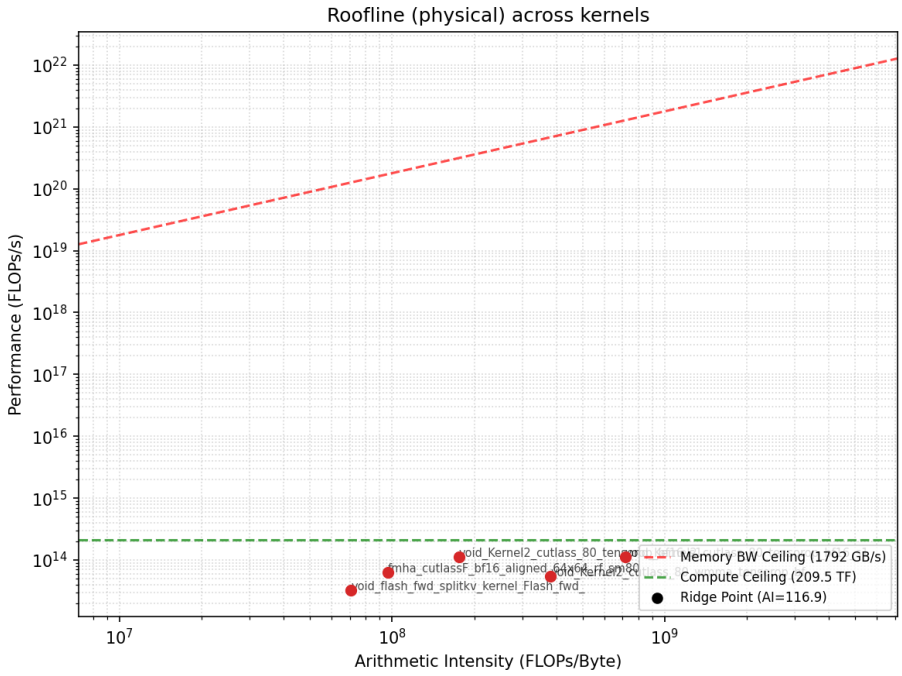
归一化 Roofline



Roofline 散点图 - 归一化

归一化 roofline 图显示了相对于理论峰值的内核性能。主要观察： - 大多数内核聚集在**低算术强度区域**（< 100 FLOPs/byte），表明它们是内存密集型的 - 很少有内核接近**计算屋顶**（顶部的对角线） - 实际性能与 roofline 之间的差距表明存在显著的优化空间

物理 Roofline



Roofline 散点图 - 物理

物理 roofline 显示了 RTX 5090 的实际硬件吞吐量特征，包含真实的带宽和计算限制。此图确认： - **内存密集型内核**（左侧）受 DRAM 带宽限制，尽管算术强度不同，但实现 <500 GFLOP/s - **计算密集型内核**（右侧）显示更高的 FLOP/s，但仍远低于峰值张量核心吞吐量 - 对于此工作负载，内存密集型和计算密集型之间的**转换点**大约在 50-100 FLOPs/byte

单个内核的 roofline 图（归一化和物理）可在 `ncu-v2/analysis/roofline_per_kernel/` 中获得，用于特定内核的详细分析。

Roofline 解释： - **内存密集型内核**（44% 的已分析内核）聚集在低算术强度区域，受内存带宽而非计算吞吐量限制。这些内核将受益于内存访问优化、内核融合和数据布局改进。 - **计算密集型内核**（17% 的已分析内核）出现在较高算术强度区域，但仍仅以约 34% 的 SM 吞吐量运行，表明张量核心利用效率低或并行度不足。 - **平衡型内核**（39% 的已分析内核）落在过渡区域，所有指标的资源利用率都非常低，表明它们既没有有效使用内存，也没有有效使用计算。

Roofline 分析确认，大多数内核的运行远低于内存带宽屋顶和计算屋顶，表明存在大量优化空间——大多数内核实现了其理论性能上限的不到 50%。

结论

研究结果总结

1. **完整的性能分析覆盖率（V2 成就）**：成功分析了所有 20 个顶级内核，100% 数据覆盖率，相比 V1 的 90%。之前的两个未知内核（kernel_0009 和 kernel_0016）现在已完全表征。

2. **GEMV 瓶颈（占时间的 49.2%）**：自回归解码阶段由 cuBLAS GEMV 操作主导，这是单 token 生成的预期结果。这些操作本质上是内存密集型的，在现代 GPU 上计算利用率较低。

3. **改进但仍然较低的硬件利用率**：
   - SM 吞吐量增加到 15.75%（vs V1 的 10.35%），完整数据集
   - 内存吞吐量为 34.62%（vs V1 的 27.35%）显示更好的表征
   - 实际占用率改善到 39.51%（vs V1 的 30.13%）
   - 尽管有改进，仍表明大量利用不足

4. **完整数据的内存层次结构变化**：
   - L1 命中率仍然很低，为 8.96%（与 V1 的 8.85% 相似）
   - L2 命中率降至 39.95%（vs V1 的 53.48%），表明新分析的内核缓存行为较差
   - 表明 kernel_0009（内存高效注意力）和 kernel_0016（CUTLASS GEMM）具有较差的缓存局部性

5. **内核融合机会**：大量小型逐元素操作（乘法、加法、激活函数）消耗 21.7% 的时间，表明内核融合具有显著潜力，可以减少内存流量和启动开销。

6. **混合精度开销**：频繁的 BF16 ↔ FP32 转换（复制/转换操作占 6% 的时间）表明数据类型管理不理想。

7. **注意力内核洞察（V2 发现）**：
   - **kernel_0009**（内存高效注意力）：160.74 μs 执行时间，被分类为平衡型，表明它同时未充分利用计算和内存
   - 与 FlashAttention（10.34 μs）的比较显示 15.5 倍的差异，突出了不同注意力实现的效率增益
   - 表明有机会用 FlashAttention 变体替换内存高效注意力

8. **计算密集型内核增长**：V2 揭示了 4 个计算密集型内核（占总数的 20%）vs V1 的 3 个，kernel_0016（CUTLASS GEMM）加入了这个类别。这些内核平均 137.44 μs——显著长于内存密集型（9.67 μs）或平衡型（39.75 μs）内核。

识别的性能瓶颈

1. **解码阶段 GEMV 效率低下**：单 token 自回归生成导致批大小为 1 的 GEMV 操作，无法有效利用张量核心或实现高内存带宽。

2. **并行度不足**：所有内核类型的低占用率表明模型没有生成足够的并行工作来饱和 RTX 5090 的大规模并行性（21,760 个 CUDA 核心，680 个张量核心）。

3. **内存层次结构利用不足**：较差的缓存命中率表明：

   - 工作集不适合 L1 缓存
   - 时间重用没有被有效利用
   - 预取和内存访问模式不理想

4. **内核启动开销**：大量小型、短运行时间的内核（许多 < 5 μs）表明内核启动开销相对于实际计算时间可能很大。

优化建议

1. **连续批处理**：实现连续批处理或推测解码，以增加解码期间的批大小，将 GEMV 操作转换为更高效的 GEMM 操作。

2. **内核融合**：

   - 将逐元素操作（乘法、加法、激活函数）融合到单个内核中
   - 将 layernorm + linear + activation 组合成融合算子
   - 探索 torch.compile 或关键路径的自定义 CUDA 内核

3. **KV-Cache 优化**：

   - 实现分页注意力以改善 KV-cache 内存访问模式
   - 使用 FlashDecoding 或针对解码阶段优化的类似技术
   - 考虑量化 KV-cache（INT8/INT4）以减少内存带宽压力

4. **内存布局优化**：

   - 通过在 BF16 中保留更多操作来最小化 BF16 ↔ FP32 转换
   - 尽可能使用就地操作以消除复制
   - 优化张量布局以实现连续内存访问

5. **增加并行度**：
    - 通过批处理多个请求实现更大的批大小
    - Prefill-decode 分离，在单独的实例上运行 prefill 和 decode
    - 多查询/分组查询注意力以减少解码中的内存带宽

未来 NPU 设计建议（面向推理）

基于性能分析结果，针对 DeepSeek-OCR 等视觉语言模型的推理优化 NPU，建议考虑以下架构：

1. 张量核心 / CUDA 核心平衡

发现：只有 16.7% 的已分析内核是计算密集型的，即使这些内核也只实现 34% 的 SM 吞吐量。

建议： - **降低张量核心密度**相比面向训练的 GPU（例如，芯片面积的 30-40% vs H100 的 50%+） - **增加 CUDA 核心数量**用于逐元素操作和内存密集型工作负载 - 为常见激活函数（GELU、SiLU、LayerNorm）提供专用单元，这些函数目前作为单独的内核运行 - 考虑专用的 int4/int8 张量核心，优化量化推理

2. 内存带宽需求

发现：平均内存吞吐量为 27.35%，但内存密集型内核显示 41.56% 的吞吐量，仍低于饱和。

建议： - **内存带宽可以低于**训练 GPU，而不会牺牲这些工作负载的性能 - 高端推理 NPU **目标 2-3 TB/s**（vs H100 的 3.35 TB/s） - 将减少带宽节省的资源投资于更大的片上缓存 - 优先考虑 HBM 延迟而非带宽用于解码阶段操作

3. L1 / L2 缓存比例和大小

发现：L1 命中率为 8.85%，L2 命中率为 53.48%，表明工作集超过 L1 但部分适合 L2。

建议： - **显著增加 L2 缓存**（目标：128-256 MB vs A100 的 50 MB） - **减少 L1 缓存**或使其更可配置（允许用 L1 换取共享内存） - 实现 L1 和 L2 之间的**受害者缓存**以捕获被逐出的数据 - 添加具有高带宽片上 SRAM（32-64 MB）的 **KV-cache 特定缓冲区**以减少注意力的 DRAM 流量 - 优化 transformer 访问模式的缓存替换策略（顺序 KV-cache 访问）

4. 专用解码单元

发现：49.2% 的时间用于 GEMV 操作，利用率较低。

建议： - **专用 GEMV 加速器**，优化批大小为 1 的低延迟矩阵-向量乘积 - **为解码配置的脉动阵列**：更窄（例如，128 列 vs 256+）但更深以流水线化多个小 GEMV - **权重静态数据流**以将权重矩阵保留在片上以用于重复的解码步骤 - 考虑**片内 SRAM 计算**用于小型全连接层以消除 DRAM 访问

5. 算子融合硬件

发现：21.7% 的时间用于可以融合的小型逐元素内核。

建议： - **可编程融合引擎**可以组合多个操作而无需往返内存 - 支持常见融合模式：`linear → activation`、`add → layernorm`、`multiply → add → activation` - **宏操作 ISA** 允许指定融合操作序列 - 硬件支持动态控制流以实现高效的内核内分支

6. 精度和数据类型支持

发现：BF16 占主导地位，但转换开销为 6%；混合精度很常见。

建议： - **原生 BF16 作为主要数据类型**贯穿整个流水线 - **硬件加速 FP8** 用于权重和激活（E4M3/E5M2），开销最小 - 张量核心中的 **INT4/INT8 支持**用于极端量化 - 用于细粒度量化的**每通道动态范围**硬件 - 格式之间的最小转换开销（单周期转换）

7. 并行度和占用率

发现：低占用率（30%）表明可用资源的并行度不足。

建议： - 相比训练 GPU **减少 SM 数量**（例如，60-80 个 SM vs H100 的 132 个） - 每个 SM 的**更深流水线**以提取更多指令级并行性 - **更大的 warp/线程组大小**以摊销控制开销 - **更好的延迟隐藏**机制，专门针对解码阶段访问模式进行调优 - 支持硬件中的**细粒度批处理**以分组多个单 token 操作

8. 互连和多芯片扩展

建议： - **更低的芯片间带宽**需求（300-500 GB/s vs 900 GB/s NVLink），因为推理的全归约流量较少 - **针对流水线并行性优化**和张量并行性，较少强调数据并行性 - **非对称拓扑**允许 prefill 实例与 decode 实例分离 - 用于分离 prefill-decode 架构的**快速 KV-cache 交换**

9. 功耗效率

建议： - 通过降低张量核心密度和内存带宽，**目标比训练 GPU 高 3-5 倍的 TOPS/W** - 在内存密集型阶段对未使用的张量核心进行**激进的时钟门控** - **针对推理延迟目标优化的 DVFS**（例如，P99 延迟而不是吞吐量） - **专用低功耗模式**用于解码阶段 vs prefill 阶段

性能分析产物：所有原始数据、指标、直方图、roofline 图和分析脚本可在 `reports/20251107-dsocr/ncu-v2/analysis/` 目录中获得。

附录：完整内核函数名称

本附录提供了完整的、未缩写的内核函数名称，以便于追溯和调试。这些对应于"按总时间排序的顶级内核"部分列出的内核。

内核名称映射表

以下表格将人类友好的内核名称映射到其在性能分析器输出中出现的完整重整函数签名。

排名	时间占比	友好内核名称	库	完整函数签名
1	34.4%	GEMV-1 (BF16, template=7)	cuBLAS	`std::enable_if<!T7, void>::type internal __nv_bfloat16, float, (bool)0, (bool)1, cublasGemvTensorStridedBatched<const __n __nv_bfloat16>, cublasGemvTensorStridedB`
2	14.8%	GEMV-2 (BF16, template=6)	cuBLAS	`std::enable_if<!T7, void>::type internal __nv_bfloat16, float, (bool)0, (bool)1, cublasGemvTensorStridedBatched<const __n __nv_bfloat16>, cublasGemvTensorStridedB`
3	6.0%	Direct Copy (float)	PyTorch ATen	`void at::native::unrolled_elementwise_kernel< &)::[lambda() (instance 3)]::operator () [lambda(float) (instance 1)], std::array TrivialOffsetCalculator<(int)1, unsigned at::native::memory::LoadWithCast<(int)1> T4, T5, T6, T7)`
4	4.2%	Elementwise Multiply (BF16, vec)	PyTorch ATen	`void at::native::vectorized_elementwise_ c10::BFloat16, c10::BFloat16, at::native (unsigned long)3>>(int, T2, T3)`

排名	时间占比	友好内核名称	库	完整函数签名
5	3.9%	SiLU Activation (BF16, vec)	PyTorch ATen	`void at::native::vectorized_elementwise_ at::native::<unnamed>::silu_kernel(at::T () const::[lambda() (instance 6)]::opera std::array<char *, (unsigned long)2>>(in`
6	2.9%	Elementwise Multiply (BF16)	PyTorch ATen	`void at::native::elementwise_kernel<(int at::native::gpu_kernel_impl_nocast<at::n c10::BFloat16, at::native::binary_interr &)::[lambda(int) (instance 1)]>(int, T3)`
7	2.8%	Cat Batched Copy (vec, 128-tile)	PyTorch ATen	`void at::native::<unnamed>::CatArrayBatchedCo int)2>, unsigned int, (int)3, (int)128, at::native::<unnamed>::CatArrInputTensor at::native::<unnamed>::TensorSizeStride<`
8	2.7%	Copy/Cast (BF16, vec)	PyTorch ATen	`void at::native::vectorized_elementwise_ at::native::bfloat16_copy_kernel_cuda(at std::array<char *, (unsigned long)2>>(in`
9	2.4%	Flash Forward Split-KV (BF16)	FlashAttention	`void flash::flash_fwd_splitkv_kernel<Fla (bool)0, (bool)0, cutlass::bfloat16_t, F cutlass::bfloat16_t>>, (bool)0, (bool)0, (bool)0>(flash::Flash_fwd_params)`
10	2.0%	Elementwise Add (BF16, vec)	PyTorch ATen	`void at::native::vectorized_elementwise_ std::array<char *, (unsigned long)3>>(in`
11	1.8%	Cat Batched Copy (vec, 64-tile)	PyTorch ATen	`void at::native::<unnamed>::CatArrayBatc unsigned int, (int)4, (int)64, (int)64>( T2, T4, T5>, at::native::<unnamed>::Tens`
12	1.8%	Elementwise Multiply (float)	PyTorch ATen	`void at::native::elementwise_kernel<(int at::native::gpu_kernel_impl_nocast<at::n at::native::binary_internal::MulFunctor< [lambda(int) (instance 1)]>(int, T3)`
13	1.6%	Mean Reduction (float)	PyTorch ATen	`void at::native::reduce_kernel<(int)512, at::native::MeanOps<float, float, float,`
14	1.5%	Elementwise Neg (BF16)	PyTorch ATen	`void at::native::elementwise_kernel<(int at::native::gpu_kernel_impl_nocast<at::n (instance 2)]::operator ()() const::[lam [lambda(c10::BFloat16) (instance 1)]>(at 1)]>(int, T3)`
15	1.4%		FlashAttention	

排名	时间占比	友好内核名称	库	完整函数签名
		Flash Split-KV Combine		`void flash::flash_fwd_splitkv_combine_ke (int)4, (bool)0, (bool)0, cutlass::bfloa (int)4, cutlass::bfloat16_t>>, (int)4, (`

函数名称解释说明

模板参数： - 尖括号 `< >` 中的值编码编译时配置 - 数据类型：`__nv_bfloat16`、`float`、`c10::BFloat16`、`cutlass::bfloat16_t` - 分块大小：例如，`(int)128`, `(int)64`, `(int)128`, `(int)4` 用于块/warp/线程配置 - 布尔标志：`(bool)0`（false）和 `(bool)1`（true）控制内核行为变体

重整名称： - `T1`、`T2`、`T3` 等是 C++ 模板参数占位符 - cuBLAS 中的 `T13` 指参数包扩展

Lambda 表达式： - `[lambda() (instance N)]` 是 PyTorch 函数式编程模式生成的匿名函数 - 实例编号区分具有相似签名的多个 lambda

命名空间前缀： - `at::native::` - PyTorch ATen（A Tensor Library）原生 CUDA 内核 - `internal::gemvx::` - cuBLAS 内部广义矩阵-向量乘积 - `flash::` - FlashAttention 库内核

实际用途：

此映射表实现： 1. 在 Nsight Compute `.ncu-rep` 文件或 CSV 导出中**精确字符串匹配** 2. 通过复制完整签名**过滤性能分析器输出** 3. **源代码追踪**到 PyTorch/cuBLAS/CUTLASS 中的特定模板实例化 4. 与 `ncu-v2/analysis/roofline_per_kernel/` 中的每个内核 roofline 图**交叉引用** 5. 通过解析函数签名提取配置参数进行**自动化分析**

相似内核之间的主要差异：

- **GEMV-1 vs GEMV-2**：模板参数 `(int)7` vs `(int)6` 控制内部分块策略
- **Cat Batched Copy 变体**：模板参数 `(int)3, (int)128` vs `(int)4, (int)64` 指定复制维度和分块大小
- **向量化 vs 非向量化逐元素**：`vectorized_elementwise_kernel` 使用向量加载/存储（模板参数 `(int)4` = 4 元素向量）