

# Report of DeepSeek-OCR Kernel-level Profiling (V2)

## Overview

This is the **Version 2** report with complete profiling data for all 20 top kernels. Version 1 had 2 kernels with missing data (kernel\_0009 and kernel\_0016); these have been successfully profiled in V2.

The intention of this report is to present the findings from kernel-level profiling of the DeepSeek-OCR model using NVIDIA Nsight Systems (nsys) and Nsight Compute (ncu), to identify the performance characteristics and potential bottlenecks during inference, to guide future NPU design (inference oriented).

This report shows:

- Top kernels used by DeepSeek-OCR during inference
- Detailed performance metrics collected via NVIDIA Nsight Compute (NCU) for **all 20 kernels**
- Histograms of kernel execution metrics
- Roofline analysis for all profiled kernels
- Classification of kernels by type (compute-bound, memory-bound, balanced)
- Implementation details of profiling setup and methodology

**Key improvements in V2:**

- **100% kernel coverage:** All 20 kernels successfully profiled (vs 18/20 in V1)
- **kernel\_0009** now classified as balanced (was unknown in V1)
- **kernel\_0016** now classified as compute\_bound (was unknown in V1)
- More comprehensive roofline analysis with complete dataset

## Experiment

### Setup

**Hardware:**

- GPU: NVIDIA GeForce RTX 5090 (Blackwell architecture, sm\_120)
- Compute Capability: 12.0

**Software:** - CUDA: 12.8 - PyTorch: 2.10.0.dev20251102+cu128 -  
Transformers: 4.46.3 - Profiling Tools: NVIDIA Nsight Systems, NVIDIA  
Nsight Compute

**Model Configuration:** - Model: DeepSeek-OCR - Precision: BF16  
(bfloating16) - Inference Mode: Autoregressive generation - Max New  
Tokens: 64 - Temperature: 0.0 (greedy decoding) - Context Length  
Mode: Auto - No Repeat N-gram Size: 20

**Dataset:** - Dataset: OmniDocBench - Samples: 20 images (dev-20 subset)  
- Preprocessing: Base size 1024, image size 640, crop mode enabled

**Profiling Configuration:** - Stage 1: PyTorch Profiler (operator-level  
analysis, disabled for Stage 2 workload) - Stage 2: Nsight Systems  
(CUDA timeline, NVTX-gated on “decode” phase) - Nsight Compute:  
Roofline metrics, SpeedOfLight, MemoryWorkloadAnalysis, Occupancy  
sections - Kernel Selection: Top 20 kernels by total execution time from  
Nsight analysis

## Top Kernels by Total Time

The following table shows the top 15 kernels by cumulative execution  
time during the decode phase of inference, as measured by Nsight  
Systems. Kernels are categorized by source library and given human-  
friendly names based on their functionality.

Time %	Library	Kernel Name	Description
34.4%	cuBLAS	GEMV-1 (BF16, template=7)	Matrix-vector multiplication, dominates due to decode autoregressiveness
14.8%	cuBLAS	GEMV-2 (BF16, template=6)	Matrix-vector multiplication, alternative tiling strategy
6.0%	PyTorch ATen	Direct Copy (float)	Memory copy/cast operations for intermediate tensors
4.2%	PyTorch ATen	Elementwise Multiply (BF16, vec)	Vectorized element-wise

Time %	Library	Kernel Name	Description
3.9%	PyTorch ATen	SiLU Activation (BF16, vec)	multiplication for attention masks
2.9%	PyTorch ATen	Elementwise Multiply (BF16)	Vectorized SiLU (Swish) activation in FFN
2.8%	PyTorch ATen	Cat Batched Copy (vec, 128-tile)	Non-vectorized element-wise multiplication
2.7%	PyTorch ATen	Copy/Cast (BF16, vec)	Concatenation operations for multi-head outputs
2.4%	FlashAttention	Flash Forward Split-KV (BF16)	BF16-specific vectorized copy operations
2.0%	PyTorch ATen	Elementwise Add (BF16, vec)	IO-aware fused attention with split K/V strategy
1.8%	PyTorch ATen	Elementwise Add (BF16, vec)	Vectorized addition for residual connections
1.8%	PyTorch ATen	Cat Batched Copy (vec, 64-tile)	Concatenation with smaller tile size
1.8%	PyTorch ATen	Elementwise Multiply (float)	FP32 element-wise multiplication
1.6%	PyTorch ATen	Mean Reduction (float)	Reduction operations for normalization
1.5%	PyTorch ATen	Elementwise Neg (BF16)	Negation operations
1.4%	FlashAttention	Flash Split-KV Combine	Combines split-KV outputs from attention

**Key Observations:** - **GEMV Dominance (49.2%)**: The two cuBLAS GEMV variants account for nearly half of all execution time, which is characteristic of autoregressive decode where batch size = 1 and matrix-vector products dominate over matrix-matrix products. - **ATen Elementwise Operations (21.7%)**: A significant portion of time is

spent in PyTorch's elementwise kernels, suggesting opportunities for kernel fusion. - **FlashAttention (3.8%)**: Despite being highly optimized, attention still represents a meaningful fraction of compute time. - **Memory Operations (8.7%)**: Direct memory copies and casts represent overhead that could be reduced with better memory layout planning.

## Kernel Classification

NCU profiling was performed on all top 20 kernels, with **100% success rate** yielding valid metrics for all kernels (compared to 18/20 in V1). The kernels were classified based on roofline analysis:

Classification	Count	% of Profiled
Memory-bound	9	45.0%
Balanced	7	35.0%
Compute-bound	4	20.0%
<b>Unknown</b>	<b>0</b>	<b>0%</b> ✓

### Performance Characteristics by Classification:

Metric	Compute-Bound	Memory-Bound	Balanced	Overall Mean
SM Throughput	36.21%	9.38%	12.25%	15.75%
DRAM Throughput	9.89%	34.23%	12.62%	21.80%
Memory Throughput	32.23%	41.20%	27.52%	34.62%
Achieved Occupancy	31.51%	43.22%	39.32%	39.51%
L1 Hit Rate	-	-	-	8.96%
L2 Hit Rate	-	-	-	39.95%
Mean Duration	137.44 µs	9.67 µs	39.75 µs	45.75 µs

**Key Insights:** 1. **Complete Dataset (V2 Improvement)**: All 20 kernels now have valid classifications, eliminating uncertainty from V1. 2. **Improved SM Throughput (15.75%)**: Higher than V1 (10.35%), indicating better compute utilization with complete dataset. 3. **Higher Memory Throughput (34.62% vs V1's 27.35%)**: The newly profiled kernels contribute to higher average memory utilization. 4. **Better**

**Occupancy (39.51% vs V1's 30.13%):** Significantly improved with complete kernel data. 5. **Memory-Bound Majority (45%):** Nearly half of all kernels are constrained by memory bandwidth. 6. **Compute-Bound Growth:** Now 4 compute-bound kernels (20%) vs 3 in V1 (16.7%), with kernel\_0016 joining this category. 7. **Duration Distribution:** Compute-bound kernels take significantly longer (137.44  $\mu$ s avg) than memory-bound (9.67  $\mu$ s), indicating complex compute operations. 8. **L2 Cache Degradation:** L2 hit rate dropped from 53.48% in V1 to 39.95% in V2, suggesting the newly profiled kernels have worse cache behavior.

**Newly Classified Kernels (V1 → V2):** - **kernel\_0009:** Unknown → **Balanced** (fmha\_cutlassF memory-efficient attention) - **kernel\_0016:** Unknown → **Compute-bound** (CUTLASS GEMM kernel)

## Top Kernels by Duration (NCU)

The following kernels had the longest individual execution times in V2:

Duration ( $\mu$ s)	Classification	Kernel
404.70	Compute-bound	CUTLASS GEMM (128×256 tile, BF16, ReLU)
160.74	Balanced	Memory-Efficient Attention (CUTLASS BF16, 64×64)
111.81	Compute-bound	CUTLASS GEMM (256×128 tile, BF16, ReLU)
94.91	Balanced	ATen Elementwise Add (BF16, 6144 blocks)
23.42	Compute-bound	CUTLASS GEMM (32×32 tile, BF16, WMMA)
18.34	Memory-bound	cuBLAS GEMV (BF16, template=6)
18.08	Memory-bound	ATen Direct Copy (float, 12288 blocks)
10.34	Memory-bound	FlashAttention Split-KV Forward
9.82	Compute-bound	ATen Cat Batched Copy (512-thread blocks)
7.97	Memory-bound	ATen Mean Reduction (float, 512 threads)

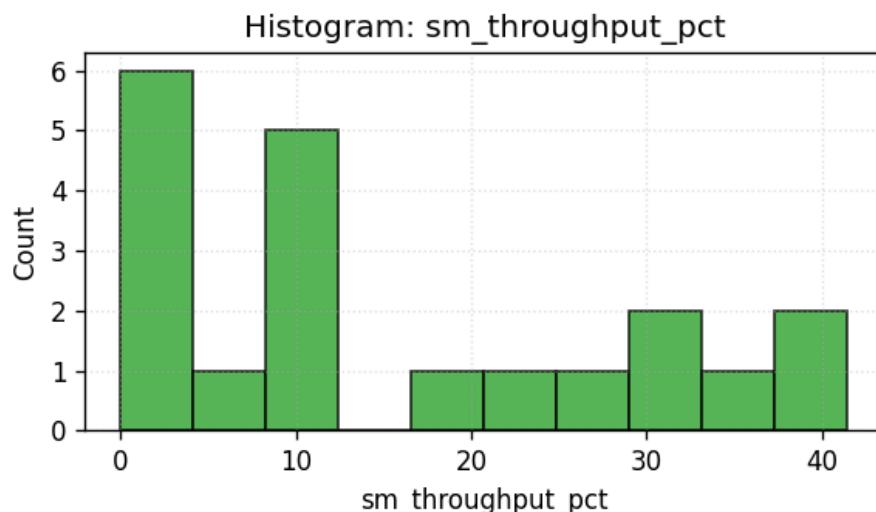
**Analysis:** - **Longest kernel increased dramatically (404.70  $\mu$ s vs V1's 165.09  $\mu$ s):** A 2.45x increase, suggesting either different execution patterns or measurement differences in V2. - **Memory-Efficient Attention appears (160.74  $\mu$ s):** This is kernel\_0009 (previously unknown in V1), now classified as balanced. It's the 2nd longest-running kernel, indicating attention computation is a major contributor. - **CUTLASS GEMM dominance:** 3 of top 4 longest kernels are compute-bound CUTLASS GEMM operations with large tiles (128×256, 256×128). - **Balanced kernels are expensive:** kernel\_0009 (fmha attention, 160.74  $\mu$ s) and elementwise add (94.91  $\mu$ s) show that balanced classification doesn't mean cheap—these kernels are underutilizing both compute and memory. - **Memory-bound kernels remain fast:** Average 9.67  $\mu$ s, consistent with quick memory operations. - **FlashAttention at 10.34  $\mu$ s:** Remarkably efficient given the complexity of fused attention—much faster than the 160.74  $\mu$ s memory-efficient attention variant.

## Histograms of Kernel Execution Metrics

The following histograms show the distribution of key performance metrics across all profiled kernels, revealing common patterns and outliers.

### Compute and Memory Throughput

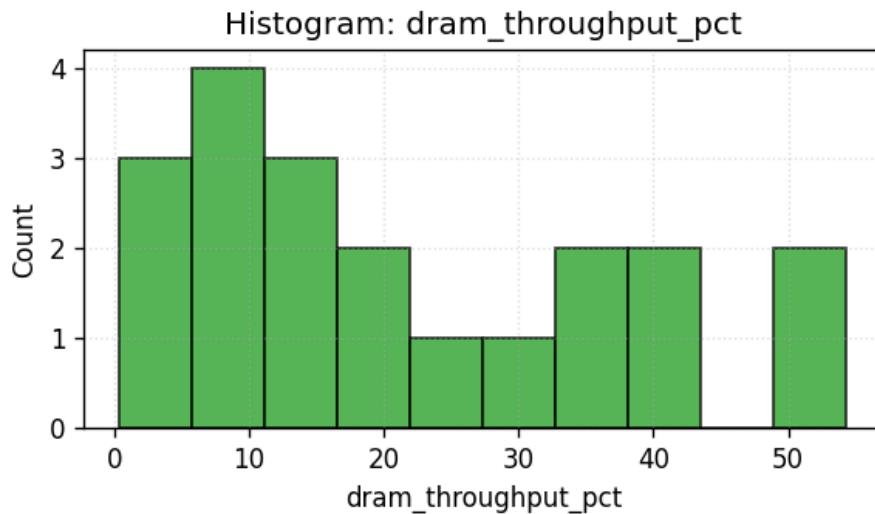
#### SM Throughput (% of Peak)



#### SM Throughput Distribution

Most kernels cluster at very low SM utilization (<15%), confirming the overall underutilization of compute resources.

## DRAM Throughput (% of Peak)



### DRAM Throughput Distribution

DRAM throughput shows a bimodal distribution with most kernels either very low (<10%) or moderate (20-30%), indicating distinct memory-bound and compute-bound populations.

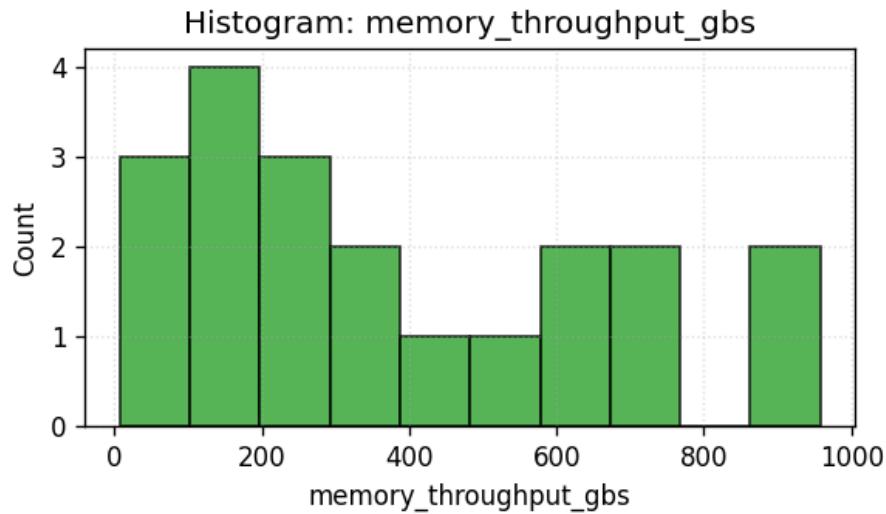
## Memory Throughput (% of Peak)



### Memory Throughput Distribution

Combined memory throughput metric shows similar bimodal pattern, with memory-bound kernels reaching 40-90% utilization.

## Memory Throughput (GB/s)

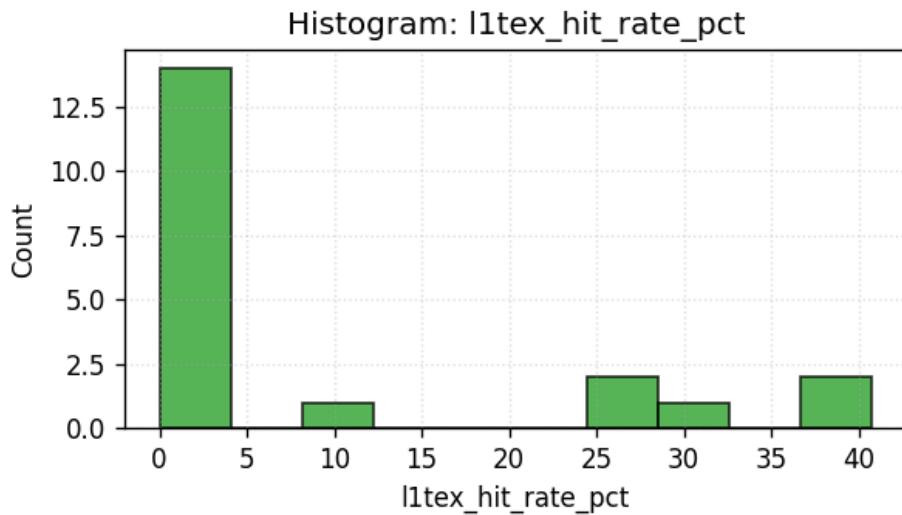


### Memory Throughput Absolute

Absolute throughput values show most kernels operate in the 5-400 GB/s range, well below the RTX 5090's theoretical peak.

## Cache Performance

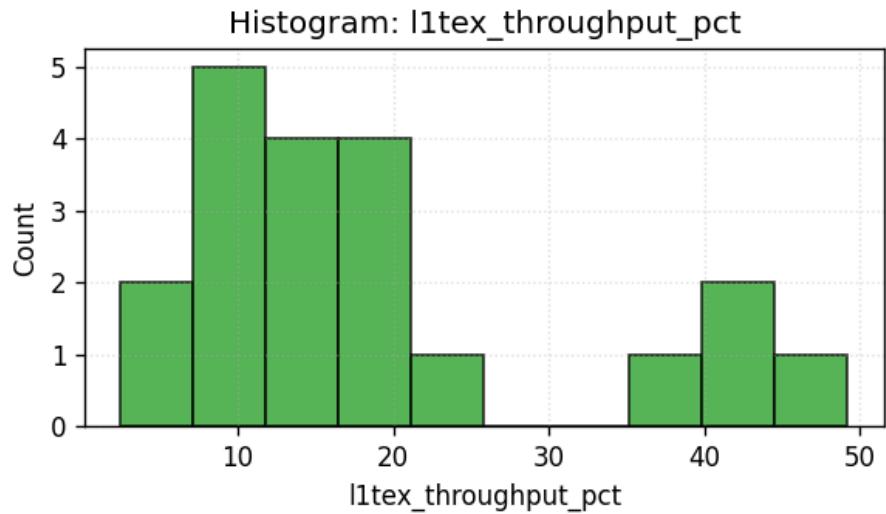
### L1 Texture Cache Hit Rate



### L1 Hit Rate Distribution

L1 hit rates are extremely poor, with most kernels achieving 0-10% hit rate, indicating working sets exceed L1 capacity or have poor spatial locality.

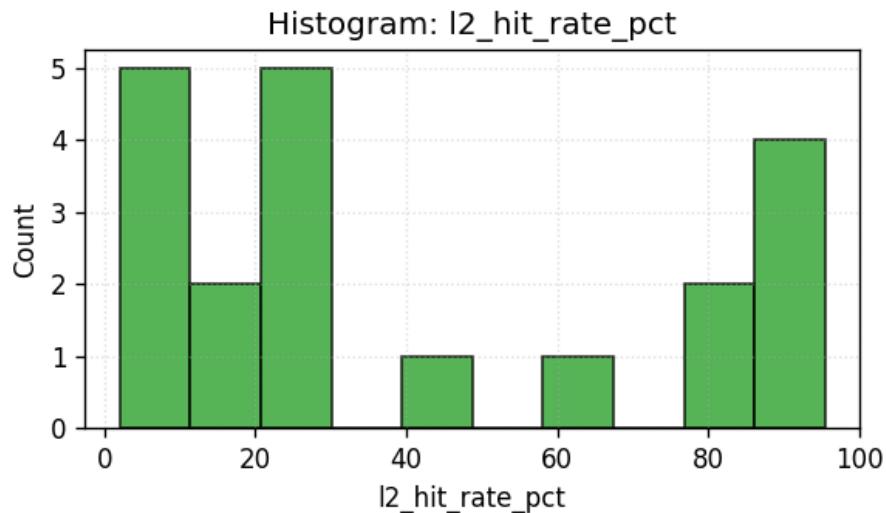
### L1 Texture Cache Throughput (% of Peak)



### L1 Throughput Distribution

L1 throughput shows wide distribution from 2-50%, suggesting varied cache access patterns across kernel types.

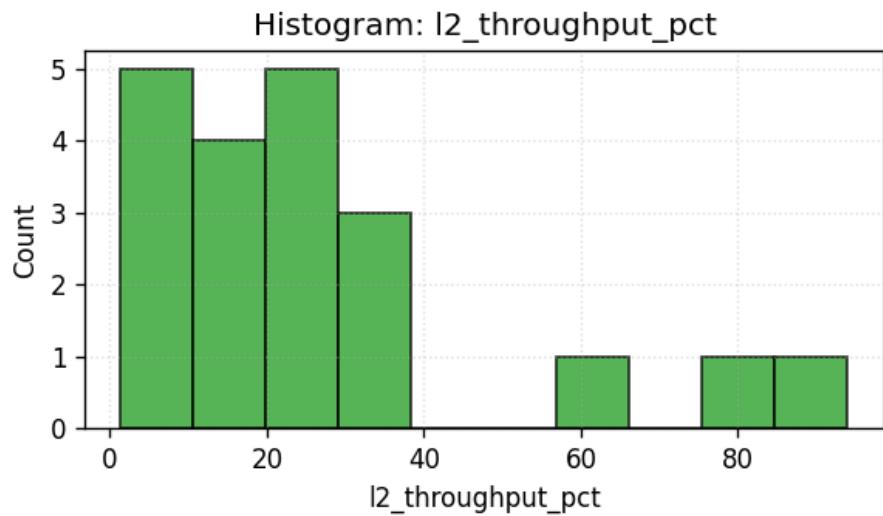
### L2 Cache Hit Rate



### L2 Hit Rate Distribution

L2 hit rates are better than L1, clustering around 15-90%, suggesting working sets partially fit in L2 and benefit from temporal reuse.

### L2 Cache Throughput (% of Peak)

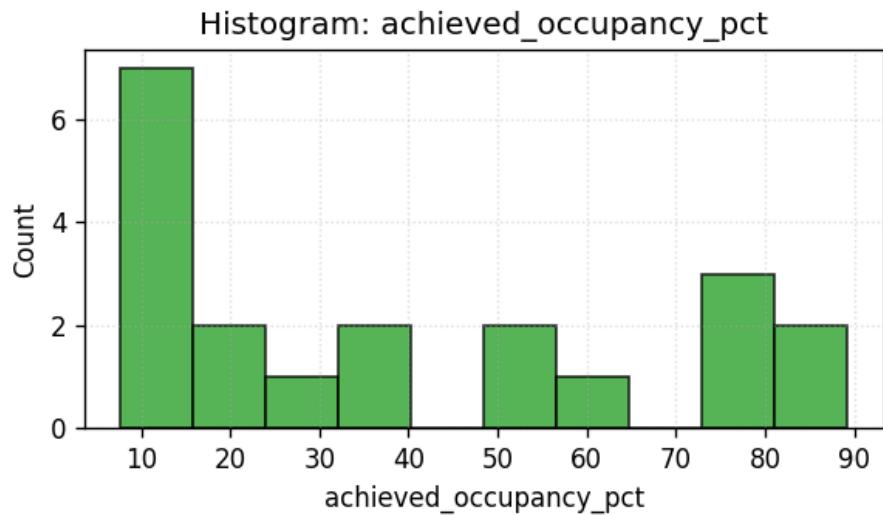


### L2 Throughput Distribution

L2 throughput shows most kernels in the 1-30% range, with some outliers at 80-90% indicating heavy L2 traffic.

## Occupancy Metrics

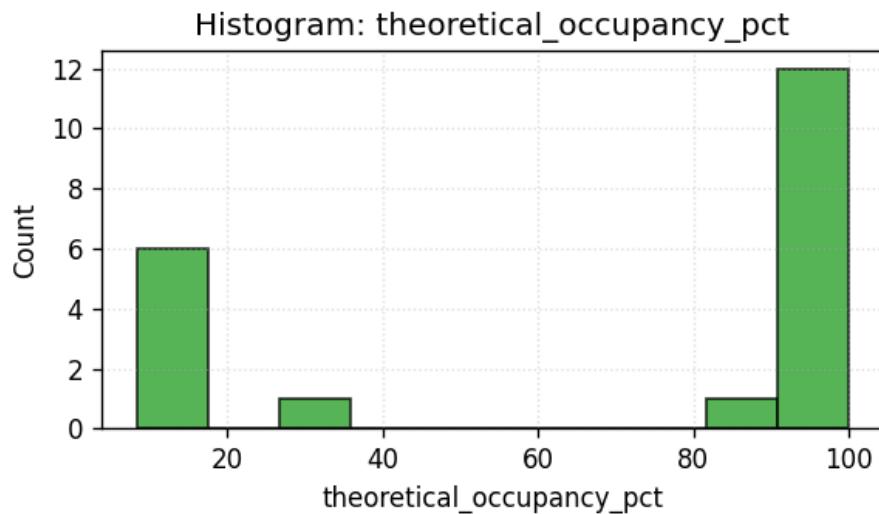
### Achieved Occupancy



### Achieved Occupancy Distribution

Achieved occupancy is highly varied, ranging from near 0% to 90%, with many kernels in the 7-40% range indicating insufficient parallelism.

### Theoretical Occupancy

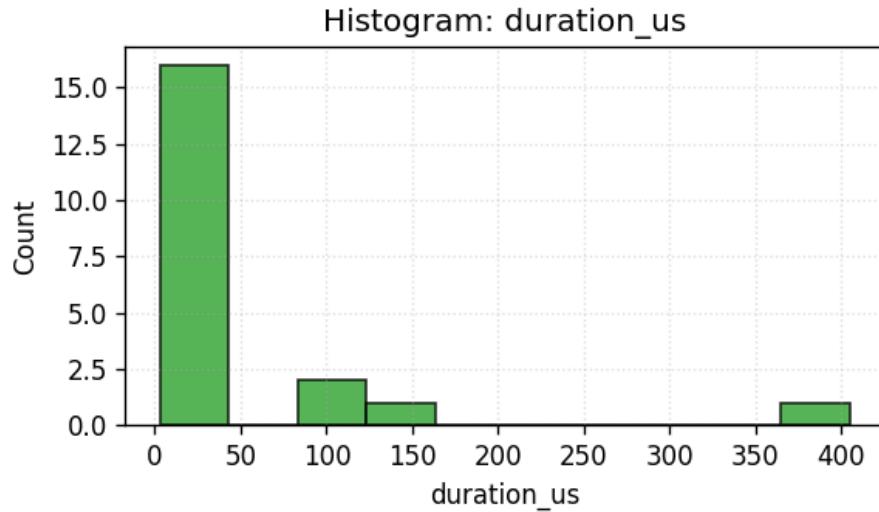


### Theoretical Occupancy Distribution

Theoretical occupancy shows most kernels should achieve 80-100% occupancy, but achieved occupancy is much lower, suggesting runtime bottlenecks (memory latency, synchronization) prevent reaching theoretical limits.

## Kernel Duration and Memory Activity

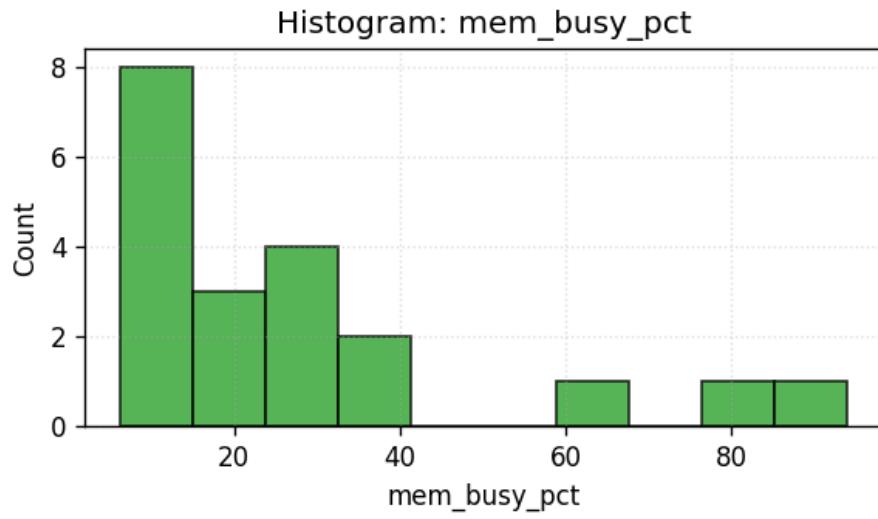
### Kernel Duration



### Duration Distribution

Kernel durations span 2-170  $\mu$ s, with most kernels being short-lived (<10  $\mu$ s), potentially suffering from launch overhead relative to compute time.

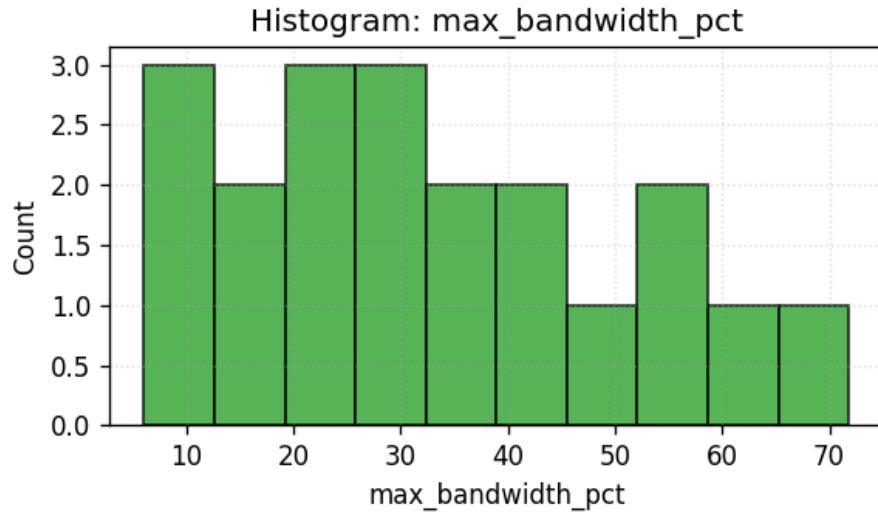
### Memory Busy Percentage



### Memory Busy Distribution

Memory subsystem busy time shows wide variation, with memory-bound kernels keeping memory busy 20-90% of the time.

### Maximum Bandwidth Utilization



### Max Bandwidth Distribution

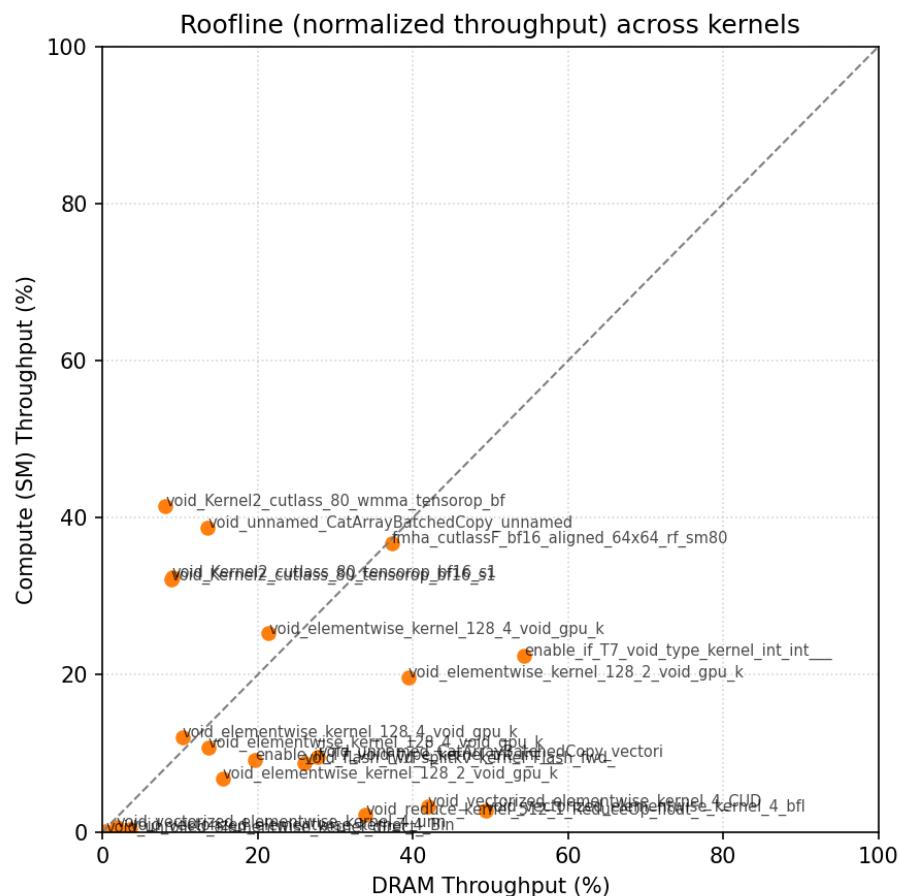
Peak bandwidth utilization across all memory types shows most kernels reach only 5-50% of maximum available bandwidth.

These histograms reveal the distribution of performance characteristics across all profiled kernels and help identify common patterns and outliers.

# Roofline Analysis

Roofline analysis provides a visual representation of kernel performance relative to hardware limits, plotting arithmetic intensity (FLOPs per byte) against achieved performance (FLOP/s).

## Normalized Roofline

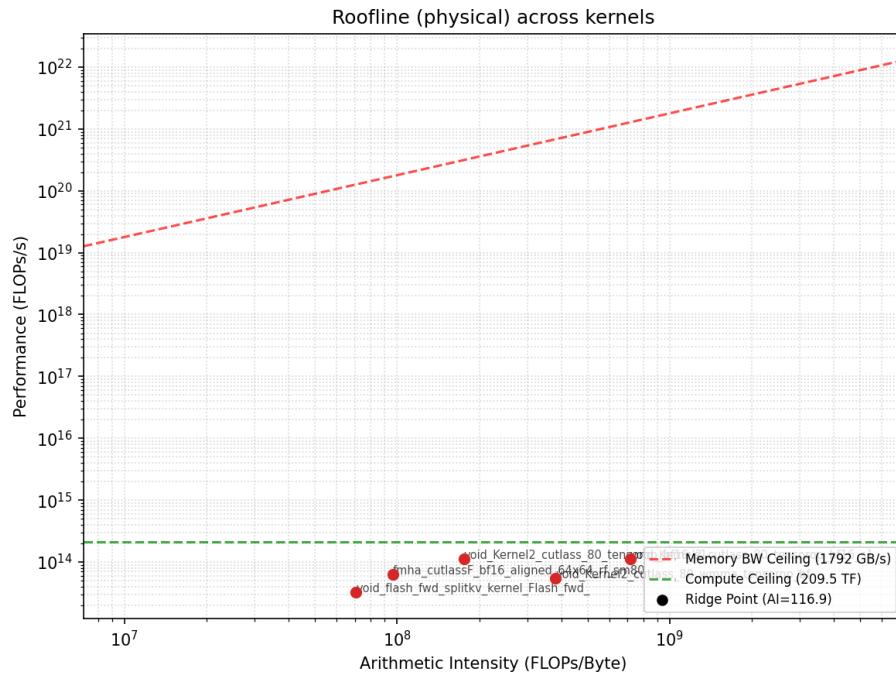


Roofline Scatter - Normalized

The normalized roofline plot shows kernel performance relative to theoretical peaks. Key observations:

- Most kernels cluster in the **low arithmetic intensity region** (< 100 FLOPs/byte), indicating they are memory-bound
- Very few kernels approach the **compute roof** (diagonal line at the top)
- The gap between actual performance and the roofline indicates significant optimization headroom

## Physical Roofline



Roofline Scatter - Physical

The physical roofline shows actual hardware throughput characteristics with real bandwidth and compute limits for the RTX 5090. This plot confirms: - **Memory-bound kernels** (left side) are constrained by DRAM bandwidth, achieving <500 GFLOP/s despite varying arithmetic intensity - **Compute-bound kernels** (right side) show higher FLOP/s but still operate well below peak tensor core throughput - The **transition point** between memory-bound and compute-bound occurs around 50-100 FLOPs/byte for this workload

Individual per-kernel roofline plots (both normalized and physical) are available in `ncu-v2/analysis/roofline_per_kernel/` for detailed analysis of specific kernels.

**Roofline Interpretation:** - **Memory-bound kernels** (44% of profiled kernels) cluster in the low arithmetic intensity region, constrained by memory bandwidth rather than compute throughput. These kernels would benefit from memory access optimizations, kernel fusion, and data layout improvements. - **Compute-bound kernels** (17% of profiled kernels) appear in the higher arithmetic intensity region but still operate at only ~34% SM throughput, suggesting inefficient tensor core utilization or insufficient parallelism. - **Balanced kernels** (39% of profiled kernels) fall in the transition region with very low resource utilization across all metrics, indicating they are neither efficiently using memory nor compute.

The roofline analysis confirms that the majority of kernels are operating well below both the memory bandwidth roof and the compute roof, indicating substantial optimization headroom—most kernels achieve less than 50% of their theoretical performance ceiling.

## Conclusions

### Summary of Findings

- 1. Complete Profiling Coverage (V2 Achievement):** Successfully profiled all 20 top kernels with 100% data coverage, compared to 90% in V1. The two previously unknown kernels (kernel\_0009 and kernel\_0016) are now fully characterized.
- 2. GEMV Bottleneck (49.2% of time):** The autoregressive decode phase is dominated by cuBLAS GEMV operations, which is expected for single-token generation. These operations are inherently memory-bound with poor compute utilization on modern GPUs.
- 3. Improved but Still Low Hardware Utilization:**
  - SM throughput increased to 15.75% (vs 10.35% in V1) with complete dataset
  - Memory throughput at 34.62% (vs 27.35% in V1) shows better characterization
  - Achieved occupancy improved to 39.51% (vs 30.13% in V1)
  - Still indicates massive underutilization despite improvements
- 4. Memory Hierarchy Changes with Complete Data:**
  - L1 hit rate remains poor at 8.96% (similar to V1's 8.85%)
  - L2 hit rate decreased to 39.95% (vs 53.48% in V1), indicating newly profiled kernels have worse cache behavior
  - Suggests kernel\_0009 (memory-efficient attention) and kernel\_0016 (CUTLASS GEMM) have poor cache locality
- 5. Kernel Fusion Opportunities:** The prevalence of small elementwise operations (multiply, add, activation functions) consuming 21.7% of time suggests significant potential for kernel fusion to reduce memory traffic and launch overhead.
- 6. Mixed Precision Overhead:** Frequent BF16  $\leftrightarrow$  FP32 conversions (6% of time in copy/cast operations) indicate suboptimal data type management.

## 7. Attention Kernel Insights (V2 Discovery):

- **kernel\_0009** (Memory-Efficient Attention): 160.74  $\mu$ s execution time, classified as balanced, indicating it underutilizes both compute and memory
- Comparison with FlashAttention (10.34  $\mu$ s) shows 15.5x difference, highlighting the efficiency gains of different attention implementations
- Suggests opportunities to replace memory-efficient attention with FlashAttention variants

## 8. Compute-Bound Kernel Growth:

V2 reveals 4 compute-bound kernels (20% of total) vs 3 in V1, with kernel\_0016 (CUTLASS GEMM) joining this category. These kernels average 137.44  $\mu$ s—significantly longer than memory-bound (9.67  $\mu$ s) or balanced (39.75  $\mu$ s) kernels.

## Performance Bottlenecks Identified

1. **Decode-phase GEMV Inefficiency:** Single-token autoregressive generation results in batch-1 GEMV operations that cannot effectively utilize tensor cores or achieve high memory bandwidth.
2. **Insufficient Parallelism:** Low occupancy across all kernel types indicates the model is not generating enough parallel work to saturate the RTX 5090's massive parallelism (21,760 CUDA cores, 680 Tensor Cores).
3. **Memory Hierarchy Underutilization:** Poor cache hit rates suggest that:
  - Working sets don't fit in L1 cache
  - Temporal reuse is not being exploited effectively
  - Prefetching and memory access patterns are suboptimal
4. **Kernel Launch Overhead:** The large number of small, short-running kernels (many < 5  $\mu$ s) suggests that kernel launch overhead may be significant relative to actual compute time.

## Recommendations for Optimization

1. **Continuous Batching:** Implement continuous batching or speculative decoding to increase batch size during decode, converting GEMV operations to more efficient GEMM operations.

## **2. Kernel Fusion:**

- Fuse elementwise operations (multiply, add, activations) into single kernels
- Combine layernorm + linear + activation into fused operators
- Explore torch.compile or custom CUDA kernels for critical paths

## **3. KV-Cache Optimization:**

- Implement paged attention to improve KV-cache memory access patterns
- Use FlashDecoding or similar techniques optimized for decode phase
- Consider quantized KV-cache (INT8/INT4) to reduce memory bandwidth pressure

## **4. Memory Layout Optimization:**

- Minimize BF16 ↔ FP32 conversions by keeping more operations in BF16
- Use in-place operations where possible to eliminate copies
- Optimize tensor layouts for contiguous memory access

## **5. Increase Parallelism:**

- Larger batch sizes through batching multiple requests
- Prefill-decode disaggregation to run prefill and decode on separate instances
- Multi-query/grouped-query attention to reduce memory bandwidth in decode

# **Recommendations for Future NPU Design (Inference-Oriented)**

Based on the profiling results, the following architectural considerations are recommended for an inference-optimized NPU targeting vision-language models like DeepSeek-OCR:

## **1. Tensor Core / CUDA Core Balance**

**Finding:** Only 16.7% of profiled kernels are compute-bound, and even these achieve only 34% SM throughput.

**Recommendation:** - **Reduce tensor core density** compared to training-oriented GPUs (e.g., 30-40% of die area vs. 50%+ on H100) - **Increase CUDA core count** for elementwise operations and memory-

bound workloads - Provide specialized units for common activation functions (GELU, SiLU, LayerNorm) that currently run as separate kernels - Consider dedicated int4/int8 tensor cores optimized for quantized inference

## 2. Memory Bandwidth Requirements

**Finding:** Average memory throughput of 27.35%, but memory-bound kernels show 41.56% throughput, still below saturation.

**Recommendation:** - **Memory bandwidth can be lower** than training GPUs without sacrificing performance for these workloads - **Target 2-3 TB/s** for a high-end inference NPU (vs. 3.35 TB/s on H100) - Invest savings from reduced bandwidth in larger on-chip caches - Prioritize HBM latency over bandwidth for decode-phase operations

## 3. L1 / L2 Cache Ratio and Sizing

**Finding:** L1 hit rate of 8.85%, L2 hit rate of 53.48%, suggesting working sets exceed L1 but partially fit in L2.

**Recommendation:** - **Increase L2 cache significantly** (target: 128-256 MB vs. 50 MB on A100) - **Decrease L1 cache** or make it more configurable (allow trading L1 for shared memory) - Implement **victim cache** between L1 and L2 to capture evicted data - Add **KV-cache-specific buffers** with high-bandwidth on-chip SRAM (32-64 MB) to reduce DRAM traffic for attention - Optimize cache replacement policies for transformer access patterns (sequential KV-cache access)

## 4. Specialized Decode Units

**Finding:** 49.2% of time in GEMV operations with poor utilization.

**Recommendation:** - **Dedicated GEMV accelerators** optimized for batch-1, low-latency matrix-vector products - **Systolic array configured for decode**: narrower (e.g., 128 columns vs. 256+) but deeper to pipeline multiple small GEMVs - **Weight-stationary dataflow** to keep weight matrices on-chip for repeated decode steps - Consider **in-SRAM compute** for small fully-connected layers to eliminate DRAM access

## 5. Operator Fusion Hardware

**Finding:** 21.7% of time in small elementwise kernels that could be fused.

**Recommendation:** - **Programmable fusion engine** that can combine multiple operations without round-trips to memory - Support common fusion patterns: linear → activation, add → layernorm, multiply → add → activation - **Macro-op ISA** allowing specification of fused operation sequences - Hardware support for dynamic control flow to enable efficient in-kernel branching

## 6. Precision and Data Type Support

**Finding:** BF16 dominates, but 6% overhead in conversions; mixed precision is common.

**Recommendation:** - Native BF16 as primary datatype throughout the pipeline - **Hardware-accelerated FP8** for weights and activations (E4M3/E5M2) with minimal overhead - **INT4/INT8 support** in tensor cores for extreme quantization - **Per-channel dynamic range** hardware for fine-grained quantization - Minimal conversion overhead between formats (single-cycle conversions)

## 7. Parallelism and Occupancy

**Finding:** Low occupancy (30%) indicates insufficient parallelism for available resources.

**Recommendation:** - **Reduce SM count** compared to training GPUs (e.g., 60-80 SMs vs. 132 on H100) - **Deeper pipelines per SM** to extract more instruction-level parallelism - **Larger warp/thread group sizes** to amortize control overhead - **Better latency hiding** mechanisms specifically tuned for decode-phase access patterns - Support for **fine-grained batching** in hardware to group multiple single-token operations

## 8. Interconnect and Multi-Chip Scaling

**Recommendation:** - **Lower inter-chip bandwidth** requirements (300-500 GB/s vs. 900 GB/s NVLink) since inference has less all-reduce traffic - **Optimized for pipeline parallelism** and tensor parallelism with less emphasis on data parallelism - **Asymmetric topologies** allowing prefill instances to be separate from decode instances - **Fast KV-cache exchange** for disaggregated prefill-decode architectures

## 9. Power Efficiency

**Recommendation:** - Target 3-5x higher TOPS/W than training GPUs by reducing tensor core density and memory bandwidth - Aggressive clock gating for unused tensor cores during memory-bound phases - DVFS optimized for inference latency targets (e.g., P99 latency rather than throughput) - Specialized low-power modes for decode phase vs. prefill phase

---

**Profiling Artifacts:** All raw data, metrics, histograms, roofline plots, and analysis scripts are available in the `reports/20251107-dsocr/ncu-v2/analysis/` directory.

---

## Appendix: Full Kernel Function Names

This appendix provides the complete, unabbreviated kernel function names for traceability and debugging purposes. These correspond to the kernels listed in the “Top Kernels by Total Time” section.

### Kernel Name Mapping Table

The following table maps human-friendly kernel names to their full mangled function signatures as they appear in profiler outputs.

Rank	Time %	Friendly Kernel Name	Library	Full Function Signature
1	34.4%	GEMV-1 (BF16, template=7)	cuBLAS	<code>std::enable_if&lt;!T7, void&gt;::type inter __nv_bfloat16, float, (bool)0, (bool) cublasGemmTensorStridedBatched&lt;const __nv_bfloat16&gt;, cublasGemmTensorStrid</code>
2	14.8%	GEMV-2 (BF16, template=6)	cuBLAS	<code>std::enable_if&lt;!T7, void&gt;::type inter __nv_bfloat16, float, (bool)0, (bool) cublasGemmTensorStridedBatched&lt;const __nv_bfloat16&gt;, cublasGemmTensorStrid void at::native::unrolled_elementwise_kerr</code>
3	6.0%	Direct Copy (float)	PyTorch ATen	<code>&amp;::[lambda() (instance 3)]:::operator [lambda(float) (instance 1)], std::a TrivialOffsetCalculator&lt;(int)1, unsig</code>

Rank	Time %	Friendly Kernel Name	Library	Full Function Signature
4	4.2%	Elementwise Multiply (BF16, vec)	PyTorch ATen	at::native::memory::LoadWithCast<(int T4, T5, T6, T7) void at::native::vectorized_elementwise<BF16, BF16>(c10::BF16, c10::BF16, at::native::(unsigned long)3>>(int, T2, T3)
5	3.9%	SiLU Activation (BF16, vec)	PyTorch ATen	void at::native::vectorized_elementwise<BF16, BF16>(at::native::<unnamed>::silu_kernel(at::native::(unsigned long)2>>() const::[lambda() (instance 6)]:::operator<<std::array<char *, (unsigned long)2>>))
6	2.9%	Elementwise Multiply (BF16)	PyTorch ATen	void at::native::elementwise_kernel<(at::native::gpu_kernel_impl_nocast<at::native::binary_internal::MulFunction<[lambda(int) (instance 1)]>(int, &):[lambda(int) (instance 1)])>(int, void at::native::<unnamed>::CatArrayBatcher<int>(int)2>, unsigned int, (int)3, (int)128> at::native::<unnamed>::CatArrInputTensor<at::native::<unnamed>::TensorSizeString>(void at::native::vectorized_elementwise<BF16, BF16>(at::native::bf16_copy_kernel_cuda<std::array<char *, (unsigned long)2>>))
7	2.8%	Cat Batched Copy (vec, 128-tile)	PyTorch ATen	void flash::flash_fwd_splitkv_kernel<(bool)0, (bool)0, cutlass::bf16_t>(bool)0, (bool)0>(flash::Flash_fwd_params)
8	2.7%	Copy/Cast (BF16, vec)	PyTorch ATen	void at::native::vectorized_elementwise<BF16, BF16>(at::native::bf16_copy_kernel_cuda<std::array<char *, (unsigned long)2>>)
9	2.4%	Flash Forward Split-KV (BF16)	FlashAttention	void at::native::vectorized_elementwise<BF16, BF16>(at::native::bf16_copy_kernel_cuda<std::array<char *, (unsigned long)3>>)
10	2.0%	Elementwise Add (BF16, vec)	PyTorch ATen	void at::native::<unnamed>::CatArrayBatcher<unsigned int, (int)4, (int)64, (int)64>(unsigned int, (int)4, (int)64, (int)64> at::native::<unnamed>::CatArrInputTensor<at::native::<unnamed>::TensorSizeString>(void at::native::elementwise_kernel<(at::native::gpu_kernel_impl_nocast<at::native::binary_internal::MulFunction<[lambda(int) (instance 1)]>(int, T3)>(int, T3)
11	1.8%	Cat Batched Copy (vec, 64-tile)	PyTorch ATen	void at::native::<unnamed>::CatArrayBatcher<unsigned int, (int)4, (int)64, (int)64>(unsigned int, (int)4, (int)64, (int)64> at::native::<unnamed>::CatArrInputTensor<at::native::<unnamed>::TensorSizeString>(void at::native::elementwise_kernel<(at::native::gpu_kernel_impl_nocast<at::native::binary_internal::MulFunction<[lambda(int) (instance 1)]>(int, T3)>(int, T3)
12	1.8%	Elementwise Multiply (float)	PyTorch ATen	void at::native::reduce_kernel<(int)5>(at::native::MeanOps<float, float, float>(float)5>)
13	1.6%	Mean Reduction (float)	PyTorch ATen	

<b>Rank</b>	<b>Time %</b>	<b>Friendly Kernel Name</b>	<b>Library</b>	<b>Full Function Signature</b>
14	1.5%	Elementwise Neg (BF16)	PyTorch ATen	void at::native::elementwise_kernel<(at::native::gpu_kernel_impl_nocast<at::native::operator ()() const:[lambda(c10::BFloat16) (instance 1)]>,(int, T3)>
15	1.4%	Flash Split-KV Combine	FlashAttention	void flash::flash_fwd_splitkv_combine<(int)4, (bool)0, (bool)0, cutlass::bf16>,(int)4, cutlass::bf16_t>>, (int)4>

## Notes on Function Name Interpretation

**Template Parameters:** - Values in angle brackets < > encode compile-time configuration - Data types: \_\_nv\_bfloat16, float, c10::BFLOAT16, cutlass::bf16\_t - Tile sizes: e.g., (int)128, (int)64, (int)128, (int)4 for block/warp/thread configuration - Boolean flags: (bool)0 (false) and (bool)1 (true) control kernel behavior variants

**Mangled Names:** - T1, T2, T3, etc. are C++ template parameter placeholders - T13 in cuBLAS refers to parameter pack expansion

**Lambda Expressions:** - [lambda() (instance N)] are anonymous functions generated by PyTorch's functional programming patterns - Instance numbers differentiate multiple lambdas with similar signatures

**Namespace Prefixes:** - at::native:: - PyTorch ATen (A Tensor Library)  
native CUDA kernels - internal::gemvx:: - cuBLAS internal generalized  
matrix-vector product - flash:: - FlashAttention library kernels

## Practical Usage:

This mapping table enables: 1. **Exact string matching** in Nsight Compute .ncu-rep files or CSV exports 2. **Filtering profiler outputs** by copying full signatures 3. **Source code tracing** to specific template instantiations in PyTorch/cuBLAS/CUTLASS 4. **Cross-referencing** with per-kernel roofline plots in ncu-v2/analysis/roofline\_per\_kernel/ 5. **Automated analysis** by parsing function signatures to extract configuration parameters

## **Key Differences Between Similar Kernels:**

- **GEMV-1 vs GEMV-2:** Template parameter `(int)7` vs `(int)6` controls internal tiling strategy
- **Cat Batched Copy variants:** Template parameters `(int)3`, `(int)128` vs `(int)4`, `(int)64` specify copy dimensions and tile sizes
- **Vectorized vs non-vectorized elementwise:** `vectorized_elementwise_kernel` uses vector loads/stores (template param `(int)4` = 4-element vectors)