SnapMLA: Efficient Long-Context MLA Decoding via Hardware-Aware FP8 Quantized Pipelining

Yifan Zhang[1†] **Zunhai Su**[1,2†] **Shuhao Hu**[1†*] **Rui Yang**[1]
Wei Wu[1] **Yulei Qian**[1] **Yuchen Xie**[1] **Xunliang Cai**[1]

[1]Meituan [2]Tsinghua University

Abstract

While FP8 attention has shown substantial promise in innovations like FlashAttention-3, its integration into the decoding phase of the DeepSeek Multi-head Latent Attention (MLA) architecture presents notable challenges. These challenges include numerical heterogeneity arising from the decoupling of positional embeddings, misalignment of quantization scales in FP8 PV GEMM, and the need for optimized system-level support. In this paper, we introduce *SnapMLA*, an FP8 MLA decoding framework optimized to improve long-context efficiency through the following hardware-aware algorithm-kernel co-optimization techniques: *(i) RoPE-Aware Per-Token KV Quantization:* Motivated by our analysis of the heterogeneous quantization sensitivity inherent to the MLA KV cache, this approach preserves the RoPE part in high precision. Furthermore, per-token granularity is employed to align with the autoregressive decoding process and maintain quantization accuracy. *(ii) Quantized PV Computation Pipeline Reconstruction:* Addresses the misalignment of quantization scales in FP8 PV computation caused by the shared KV structure of the MLA. *(iii) End-to-End Dataflow Optimization:* Establishes an efficient data read-and-write workflow using specialized kernels, ensuring streamlined data flow and improved performance. Extensive experiments on state-of-the-art MLA LLMs show that SnapMLA achieves up to ***a 1.91× improvement in throughput on long-output decoding workloads while maintaining near-parity benchmark quality compared with the BF16 baseline*** on the evaluated reasoning and code-generation benchmarks. The code corresponding to this work is included in the supplementary material.

1 Introduction

Large language models (LLMs) have revolutionized the field of artificial intelligence[3, 24, 1, 2, 8, 7, 37, 38], with their true potential increasingly dependent on the ability to efficiently process long-context data [46, 6, 43, 42]. However, this advancement places significant strain on the underlying serving systems. As sequence length grows, the Key-Value (KV) cache, which stores intermediate attention states, expands linearly, rapidly consuming GPU memory and limiting both system throughput and the maximum supportable batch size [32, 18]. Addressing this memory bottleneck has become critical for enabling efficient long-context inference [6, 45, 26, 44]. To mitigate this issue, DeepSeek introduces the Multi-head Latent Attention (MLA) architecture [24, 2, 1, 37], which reduces KV cache storage requirements through low-rank KV joint compression. However, in high-concurrency inference scenarios, architectural compression alone is insufficient. To further enhance throughput and support larger batch sizes, quantizing the KV cache into low-bit formats has emerged as a crucial strategy [32, 18, 1, 14, 16].

*Work done at Meituan. † Equal Contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

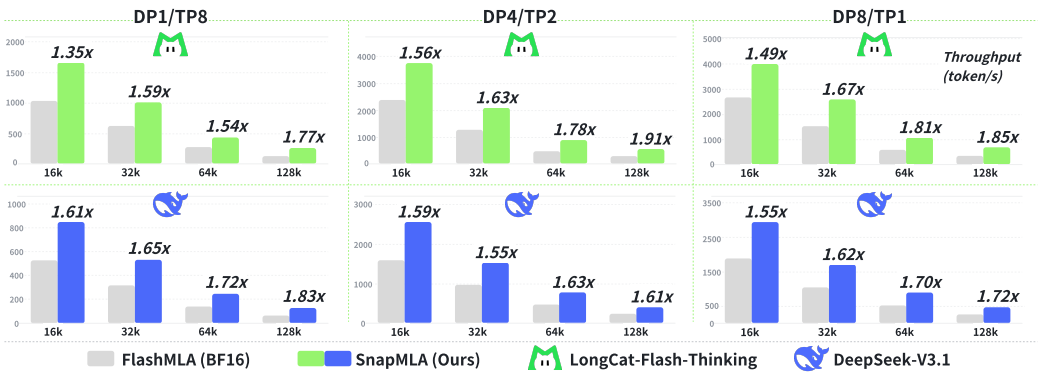


Figure 1: End-to-end decoding throughput comparison. We evaluate the generation throughput of SnapMLA on *LongCat-Flash-Thinking* and *DeepSeek-V3.1* across various parallelization configurations and context lengths. For additional details please refer to Section 4.3.

With its exceptional representational capacity and impressive efficiency in both memory and computation, the FP8 data format for quantization has gained significant attention and widespread adoption [30, 17, 39, 14, 2]. FP8 Tensor Cores, introduced with NVIDIA's Hopper architecture [29], are specifically designed to accelerate machine learning workloads using 8-bit floating point precision. However, fully harnessing the potential of FP8 during the decoding phase of MLA LLMs presents several distinct challenges:

(i) FP8 KV Cache Quantization: MLA introduces architecture-induced numerical heterogeneity by decoupling KV vectors into a compressed content component and a precision-sensitive Rotational Position Embedding (RoPE) component. The application of uniform quantization does not effectively address this disparity. Furthermore, when quantization is applied across tokens (e.g., at per-channel or per-block granularity), the autoregressive decoding prevents the immediate quantization of newly generated tokens. This entails complex buffer management, which undermines efficiency.

(ii) FP8 MLA Computation: In MLA, a shared term is used to store and project both KVs. This results in a misalignment of the quantization scale of V during the FP8 PV GEMM on Hopper Tensor Cores due to the ***k-major layout constraint*** [14], rendering the standard post-GEMM dequantization process unfeasible.

(iii) System-Level Support for FP8 MLA Decoding: Effective FP8 MLA decoding depends on essential system-level optimizations. Minimizing memory access overhead and reducing kernel launch latencies are crucial for fully unlocking the potential of FP8 decoding, ensuring the effective realization of algorithmic advancements and other optimizations.

To address these challenges, we introduce *SnapMLA*, an FP8 hardware-aware algorithm-kernel co-optimization framework designed to optimize long-context MLA decoding efficiency. This framework tackles the identified challenges through three key components: *RoPE-Aware Per-Token Quantization* (Section 3.1), *Quantized PV Computation Pipeline Reconstruction* (Section 3.2), and *End-to-End Dataflow Optimization* (Section 3.3). Our main contributions are summarized as follows:

- We introduce ***the first open-source FP8 decoding framework tailored to MLA decoding***. By leveraging hardware-aware algorithm-kernel co-optimization, our framework effectively overcomes the challenges associated with MLA KV cache quantization, quantized MLA computation, and system-level support for FP8 MLA decoding.
- Building on a comprehensive analysis of MLA KV cache numerical and quantization errors, we empirically identify, to the best of our knowledge for the first time, the divergent quantization sensitivity between the latent content and decoupled RoPE components in the MLA KV cache. We propose ***RoPE-Aware Per-Token Quantization***, along with supporting system optimizations, to ensure the preservation of the critical RoPE part from both algorithmic and system-level perspectives.
- We develop custom CUDA kernels for SnapMLA, and extensive experiments on state-of-the-art MLA LLMs demonstrate that our approach achieves up to a ***1.91× improvement in throughput on long-output decoding workloads (Figure 1) while maintaining near-parity***

benchmark quality compared with the BF16 baseline (Table 1) on the evaluated reasoning and code-generation benchmarks.

2 Preliminaries of Multi-Head Latent Attention

Multi-Head Latent Attention (MLA), first introduced in DeepSeek-V2 [24], has since been integrated into several state-of-the-art LLMs, including DeepSeek [2, 1, 11] and the LongCat-Flash series [8, 7, 38, 37, 25]. The key innovation of MLA lies in the utilization of low-rank KV joint compression, which projects both KVs into a shared, low-dimensional latent space. In the following, we explore the core components of MLA.

Low-Rank Compression and Decoupled RoPE Let $h_t \in \mathbb{R}^d$ denote the input hidden state at time step t. MLA begins by compressing the joint KV representation into a latent vector $c_{KV} \in \mathbb{R}^{d_c}$ using a down-projection matrix $W^{DKV} \in \mathbb{R}^{d_c \times d}$:

$$c_{KV} = W^{DKV} h_t \tag{1}$$

where d_c is the compression dimension, with $d_c \ll d_h n_h$. To resolve the incompatibility between low-rank compression and position-sensitive embeddings, MLA employs a *Decoupled RoPE* strategy. The K tensor for each attention head i, is partitioned into two components: a compressed content part k_i^C and a RoPE part k^R:

$$k_i^C = W^{UK} c_{KV}, \quad k^R = \text{RoPE}(W^{KR} h_t) \tag{2}$$

Here, W^{UK} is the up-projection matrix for the K, and W^{KR} is the matrix for the RoPE term. k^R carries the RoPE, which is shared across all heads. The final K used for attention is the concatenation of k_i^C and k^R:

$$k_i = [k_i^C; k^R] \tag{3}$$

The V vector is derived solely from the latent vector:

$$v_i^C = W^{UV} c_{KV} \tag{4}$$

Inference-Optimized Absorbed Mode The matrix absorption technique is a crucial process in MLA that ensures inference efficiency. During the decoding phase, the up-projection matrices for K (W^{UK}) and V (W^{UV}) are integrated into the Q projection (W^Q) and the output projection (W^O), respectively. This optimization eliminates the need for explicit reconstruction of the compressed content components k^C and v^C. As a result, the attention score calculation for the i-th head between the current Query $q_{t,i}$ and a past token j is reformulated as:

$$q_{t,i}^T k_{j,i} = \underbrace{(q_{t,i}^C W^{UK})^T c_{KV,j}}_{\text{Content Term}} + \underbrace{(q_{t,i}^R)^T k_j^R}_{\text{RoPE Term}} \tag{5}$$

Challenges of MLA Quantization The distinctive attention structure of MLA presents two major challenges in achieving efficient quantization. First, the MLA architecture splits the KV cache for each token into two distinct components, each serving a specific function:

- ***Compressed Latent Vector*** $c_{KV} \in \mathbb{R}^{d_c}$***:*** A dense, low-dimensional vector containing contextual content information.
- ***Decoupled RoPE*** $k^R \in \mathbb{R}^{d_h^R}$***:*** A position-sensitive vector that requires high precision to effectively preserve positional information.

The functional heterogeneity between these components introduces numerical heterogeneity, leading to differences in quantization sensitivity—an issue not extensively explored in prior work.

Second, executing the PV GEMM under FP8 precision on Hopper Tensor Cores requires adherence to a strict ***k-major layout constraint*** [14]. However, MLA introduces a distinct challenge: V inherits per-token quantization scales from the latent cache (C_{KV}), which are aligned along the *reduction dimension*. This alignment disrupts the conventional post-GEMM dequantization process.

Along with the preliminaries of MLA, the preliminaries of low-bit quantization are presented in Appendix C.

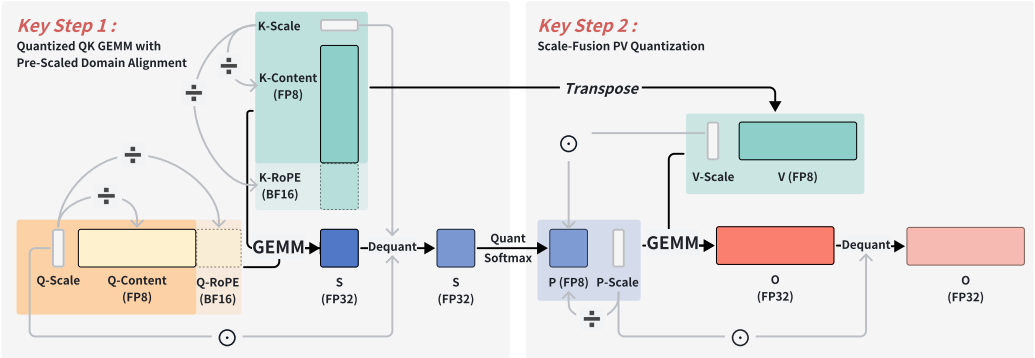


Figure 2: Overview of the scale fusion pipeline in SnapMLA. Note that K and V content share the latent cache ($\mathbf{c}_{KV}$). ***Key Step 1*** illustrates *RoPE-Aware Per-Token KV Quantization*, where the BF16 RoPE part of the QK is pre-scaled with the quantization scale of the content part, thereby unifying the numerical domains of the quantized content part and unquantized RoPE part. ***Key Step 2*** demonstrates *Quantized PV Computation Pipeline Reconstruction*, where the scale of V is pre-fused into the P, circumventing quantization dimension mismatch.

3 Methodology

In this section, we present *SnapMLA*, a hardware-aware algorithm-kernel co-optimization framework designed to optimize long-context MLA decoding. We begin by introducing *RoPE-Aware Per-Token KV Quantization* in Section 3.1. In Section 3.2, we explore *Quantized PV Computation Pipeline Reconstruction*, which resolves the misalignment of quantization scales. In Section 3.3, we introduce *End-to-End Dataflow Optimization*, where we establish a streamlined data read-and-write workflow. Figure 2 provides an overview of the critical scale fusion pipeline in SnapMLA. The algorithm for the SnapMLA decoding process is presented in Algorithm 1.

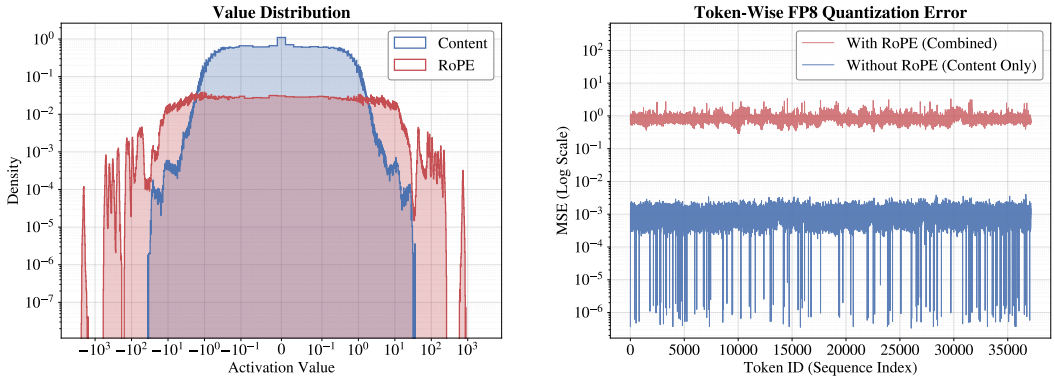
3.1 RoPE-Aware Per-Token KV Quantization

Building upon a comprehensive analysis of numerical values and quantization errors in the MLA KV cache, we introduce *RoPE-Aware Per-Token Quantization* in this section, supported by targeted system-level optimizations to enhance both its efficiency and performance.

3.1.1 MLA KV Quantization: An Algorithmic Perspective

Heterogeneous Quantization Sensitivity The structure of the MLA KV cache represents a notable departure from traditional multi-head attention (MHA), as detailed in Section 2. Effective and reliable quantization hinges on a deep understanding of the numerical distribution and a thorough re-evaluation of quantization sensitivity. To this end, we conduct an extensive analysis of the content and RoPE components of the MLA KV cache, based on *LongCat-Flash-Thinking* [7]. Our empirical findings uncover a pronounced divergence in the distributional characteristics of these two parts. As shown in Figure 3a, the RoPE component spans a significantly wider dynamic range (reaching $\pm 10^3$), exhibiting distinct outlier tails, while the content component is tightly concentrated around zero (within $\pm 10^1$), highlighting a stark contrast in their statistical behavior.

This scale mismatch renders uniform quantization strategies ineffective. Specifically, FP8 quantization leads to an order-of-magnitude increase in Mean Squared Error (MSE) for the RoPE component (as shown in Figure 3b), highlighting their heightened sensitivity to quantization. To address these issues, we propose a *RoPE-aware* strategy that applies FP8 quantization exclusively to the content components, while retaining the RoPE components in BF16 to preserve positional robustness. Although this quantization scheme is straightforward, it is firmly grounded in analysis and plays a pivotal role in preserving accuracy while optimizing KV cache compression rates.



(a) Numerical values analysis.

(b) Quantization error analysis.

Figure 3: Analysis of the numerical value distribution and quantization error comparison for the content and RoPE components of MLA KV cache in *LongCat-Flash-Thinking*.

Decoding-Centric Quantization Granularity for Accuracy Preservation While approaches like FA3 [14] demonstrate effective FP8 quantization during the prefill stage through block-wise quantization, this method encounters limitations in autoregressive decoding. In the decoding stage, tokens are generated sequentially, leading to "page tail" tokens within incomplete KV quantization blocks. This requires complex buffer management, which renders block-level quantization inefficient. To address this, we adopt per-token quantization granularity, offering two primary advantages:

- ***Instant quantization***, which enables immediate quantization of newly generated KV tokens and eliminates the need for "tail buffer" management.
- ***Framework compatibility***, which allows processing all tokens with unified logic, thereby simplifying integration into inference frameworks (e.g., vLLM, SGLang).

Building on the comprehensive algorithmic insights presented above, we adopt *RoPE-Aware Per-Token KV Quantization* as the foundation of our core quantization strategy.

3.1.2 Hardware-Optimized RoPE-Aware Per-Token Quantization

Eliminating Mixed-Precision Accumulation in Quantized QK GEMM While the algorithmic details of *RoPE-Aware Per-Token KV Quantization* were discussed in the previous section, its efficient hardware implementation introduces the challenge of mixed-precision accumulation in quantized QK GEMM. The basic MLA implementation (i.e., FlashMLA [16]) leverages instruction-level parallelism by partitioning the QK GEMM reduction dimension into nine thread groups (each spanning 64 dimensions) and employing a permuted execution schedule. The RoPE-aware quantization method introduces an inefficient mixed-precision constraint: the first eight groups (content components) use FP8 and require dequantization, while the ninth group (RoPE component) retains BF16 precision.

A potential solution is mixed-precision accumulation, where FP8 blocks are first aggregated, followed by dequantization and the addition of RoPE. However, this approach introduces a strict synchronization barrier, preventing interleaved execution, which is essential for optimal pipelining. As a result, pipeline bubbles are introduced, leading to a reduction in the actual throughput gains.

Quantized QK GEMM with Pre-Scaled Domain Alignment Instead of modifying the kernel's accumulation logic, we strategically project the BF16 RoPE components into the FP8 quantization domain during the data preparation phase. As shown in ***Key Step 1*** of Figure 2, we pre-scale the RoPE terms by the inverse of the content quantization scale:

$$\mathbf{Q}^R = \frac{\mathbf{Q}^R}{\mathbf{S}^{Q_c}}, \quad \mathbf{K}^R = \frac{\mathbf{K}^R}{\mathbf{S}^{K_c}}. \tag{6}$$

This alignment unifies the numerical domains, effectively "disguising" the high-precision RoPE segments as scaled quantized values. As a result, the kernel can process all groups uniformly, reusing the original highly optimized accumulation order without the need for intermediate dequantization or synchronization steps.

3.2 Quantized PV Computation Pipeline Reconstruction

3.2.1 Memory Layout Constraints and Quantization Mismatch

The execution of the second GEMM (PV) under FP8 precision on Hopper Tensor Cores requires strict adherence to the ***k-major layout constraint*** for V [14]. This hardware constraint mandates that V must be contiguous along the sequence dimension. However, in the context of MLA, V inherits per-token quantization scales from the latent cache ($\mathbf{c}_{KV}$). These scales are aligned with the GEMM's *reduction dimension*, resulting in a fundamental architectural conflict. Consequently, the conventional post-GEMM dequantization paradigm fails to work as intended.

3.2.2 Scale-Fusion PV Quantization: A Hardware-Algorithm Co-Optimization

To address the quantization scale mismatch, we propose a unified algorithmic design that seamlessly integrates three tightly coupled components, collectively enabling efficient and hardware-optimized quantized PV computation:

(i) Scale Fusion: In the first step, we resolve the scale mismatch by leveraging the associative property of multiplication (see ***Key Step 2*** in Figure 2). The quantization scale of V, denoted S_V, is fused into the attention probability matrix P, resulting in the scaled matrix $P' = P \odot S_V$.

(ii) Block-Wise Dynamic P Quantization: Since incorporating S_V modifies the original data distribution and extends the dynamic range of P, we apply block-wise dynamic quantization to P' to maintain robust quantization accuracy. The block size is set to the PV GEMM kernel's tiling parameter ($BlockN = 64$), ensuring numerical precision across dynamic value ranges while seamlessly integrating with the kernel's tiled execution pattern.

(iii) Implicit Dequantization: Block-wise P quantization introduces scale variations across blocks. To preserve arithmetic accuracy during tiled accumulation without incurring explicit quantization scaling overhead, we integrate the quantization scaling operation into the online Softmax accumulation states. This integration allows the tiled accumulation to account for scale differences across blocks. As a result, the online accumulation realizes *implicit dequantization*, ensuring correctly scaled outputs within a unified computational flow. Importantly, this fusion does not modify the QK logits or the resulting Softmax distribution; the scale handling is applied to the post-Softmax probability blocks and their accumulation states. A more detailed description of the scale-aware online accumulation mechanism is provided in Appendix D.

3.2.3 Summary of the Quantized PV Computation Pipeline

We present the reconstructed pipeline as four block-wise stages:

1. **Online Softmax**: Process the restored QK logits block by block, updating the running maximum and normalization statistic for the global Softmax.
2. **Scale Fusion**: Integrate the per-token V scale S_V into the current probability block P to obtain the fused probability block $P' = P \odot S_V$.
3. **Quantization**: Apply block-wise dynamic quantization to P', producing a quantized probability block and its scale.
4. **Computation**: Perform tiled PV GEMM and update O with scale-aware accumulation, which implicitly accounts for the probability-block quantization scale.

Together, these co-designed components seamlessly integrate scale management into the attention computation, eliminating traditional scale mismatches and dequantization overhead, while fully adhering to hardware execution constraints. Additional implementation details for the quantized PV computation can be found in Appendix E.

3.3 End-to-End Dataflow Optimization

While Section 3.2 addresses the computational challenges in quantized attention, this section focuses on mitigating critical *memory and data movement bottlenecks* in the end-to-end pipeline. We propose a systematic optimization strategy consisting of three interconnected layers: (i) fused kernel design to

eliminate intermediate memory traffic, (ii) memory subsystem optimization via cache-aware tiling, and (iii) zero-overhead data layout transformation to meet Tensor Core requirements.

3.3.1 Layer 1: Fused Compute-Memory Operators

Fused Token Preparation: Q-Quant and K-Append We replace the traditional sequential workflow—comprising statistic computation, quantization, and memory copying—with two specialized atomic kernels. The ***Fused-Q-Quant*** kernel consolidates per-token scale calculation, mixed-precision conversion, and *Scale Domain Alignment* (by injecting quantization scales into RoPE dimensions) into a single operation. The ***Fused-K-Append*** kernel extends this paradigm to KV cache management. In addition to performing quantization and alignment, it integrates PagedAttention-style non-contiguous writes directly into the computational stream. This enables the system to complete quantization, scale alignment, and cache updates in a single kernel launch, thereby eliminating intermediate memory buffers and reducing launch overhead.

Fused Fetch-and-Dequant for Chunk Processing For decoding phases that require high-precision reuse of cached data (e.g., chunk prefill, prefix caching), we introduce the ***Fused-Fetch-Dequant*** operator. This kernel performs on-the-fly, register-level dequantization immediately after fetching data from the quantized KV cache. By combining the dequantization operation directly with the load instructions, we streamline the memory retrieval and precision conversion into a single step. This eliminates the need for the traditional two-step process—loading quantized data into shared memory followed by a separate dequantization kernel—thereby reducing global memory I/O and kernel scheduling latency.

3.3.2 Layer 2: Memory Subsystem Optimization

Cache-Aligned Tiling and Swizzling To fully leverage the benefits of FP8's reduced bit-width, we increase the tiling size along the content dimension from 64 to 128. This strategic adjustment ensures that each atomic memory load operation is aligned with the 128-byte L2 cache lines and the Swizzle-128B shared memory (SMEM) layout in the Hopper architecture. As a result, this access pattern generates fully coalesced Tensor Memory Accelerator (TMA) descriptors, optimizing High Bandwidth Memory (HBM) utilization and eliminating shared memory bank conflicts.

3.3.3 Layer 3: Zero-Overhead Data Layout Transformation

To satisfy the strict ***k-major layout constraint*** required by FP8 WGMMA instructions, we implement an in-kernel transformation strategy tailored to our quantized pipeline. This involves two synchronized, fine-grained operations:

(i) V-Tile Transposition via Register File: V tensor tiles are transposed using a SMEM $\rightarrow$ RF $\rightarrow$ SMEM routing path, leveraging the Register File for low-latency data reordering.

(ii) P-Accumulator Byte Permutation: The accumulator holding intermediate attention scores undergoes byte-level register permutations to match the WGMMA output layout.

Critically, we schedule these data movements to leverage Hopper's asynchronous execution capabilities. By overlapping the V transposition and P permutation with the computation intervals of the preceding QK GEMM, we effectively mask their latency. This fine-grained overlap, combined with the extended 64×128 tiling scheme, enables zero-overhead layout adaptation while maintaining full compatibility with the mixed-precision pipeline.

4 Experiments

In this section, we present an extensive evaluation of SnapMLA, assessing its effectiveness from three crucial perspectives: (i) *Benchmark Results* (Section 4.2), where we rigorously evaluate SnapMLA across a wide range of tasks, including general QA, mathematical reasoning, general reasoning, coding, and others, to ensure its robustness across diverse domains; (ii) *Numerical Accuracy* (Appendix G), focusing on the precision of SnapMLA to verify its ability to preserve data fidelity during quantization; and (iii) *Efficiency Analysis* (Section 4.3), where we evaluate the efficiency of SnapMLA.

Table 1: Main benchmark accuracy and quality evaluation across various task domains.

Benchmark	DeepSeek-V3.1		LongCat-Flash-thinking	
	FlashMLA BF16	**SnapMLA (Ours) FP8**	FlashMLA BF16	**SnapMLA (Ours) FP8**
General QA				
MMLU-Pro$_{(acc)}$	84.41	84.43	82.60	82.73
MMLU-Redux$_{(acc)}$	90.48	90.89	89.30	88.27
Alignment				
IFEval$_{(strict\ prompt)}$	86.32	87.25	86.90	87.80
Arena-Hard$_{(hard\ prompt\ gemini)}$	57.10	55.50	69.90	70.40
Mathematical Reasoning				
MATH-500$_{(Mean@1)}$	98.80	98.20	99.20	98.80
AIME-24$_{(Mean@32)}$	93.85	93.65	91.87	91.67
AIME-25$_{(Mean@32)}$	87.92	85.42	89.58	88.44
BeyondAIME$_{(Mean@10)}$	71.80	69.90	69.50	70.20
General Reasoning				
GPQA-Diamond$_{(Mean@16)}$	84.15	82.57	81.50	80.24
ZebraLogic$_{(Mean@1)}$	96.10	96.00	95.50	95.00
Coding				
LCB (24.08-25.05)$_{(Mean@4)}$	73.46	72.74	79.40	79.74

4.1 Experimental Setup

Models and Baselines We evaluate state-of-the-art MLA LLMs from two different series: *DeepSeek-V3.1* [2] and *LongCat-Flash-Thinking* [8]. *DeepSeek-V3.1* is a mixture-of-experts (MoE) LLM with 671 billion total parameters, activating 37 billion parameters per token. *LongCat-Flash* is a 560-billion-parameter MoE LLM optimized for computational efficiency. It dynamically allocates computational resources by activating between 18.6 billion and 31.3 billion parameters per token, depending on the *Zero-Computation Expert* mechanism. Additionally, it enhances the computation-communication overlap through the *Shortcut-connected MoE* design.

We use FlashMLA [14] (the standard BF16 implementation) as our baseline, serving as the reference for both benchmark accuracy and efficiency performance. Details of the benchmarks used in the experiments are provided in Appendix F. FlashMLA is a specialized attention kernel optimized for the MLA proposed by DeepSeek [2], specifically tailored for NVIDIA Hopper GPUs. For the FP8 experiments, we adopt the E4M3 format for efficient KV cache and MLA computation.

Because SnapMLA targets decoding attention rather than model weights, the benchmark scores in Table 1 evaluate whether the FP8 decoding pipeline preserves model quality on representative tasks. Many reasoning and coding workloads produce long generations, exercising the decoding kernel over thousands to tens of thousands of generated tokens; Appendix F reports average generated lengths for a partially overlapping workload-characterization set, separately from the main accuracy evaluation. Separately, Section 4.3 sweeps runtime KV-cache lengths from 16k to 128k to evaluate throughput under controlled serving configurations.

Infrastructure Details All experiments are conducted on a high-performance computing node equipped with eight NVIDIA H20 GPUs. The GPUs are interconnected via NVLink, ensuring high-bandwidth communication.

4.2 Benchmark Results

Main Results As shown in Table 1, SnapMLA maintains near-parity benchmark quality relative to the BF16 FlashMLA baseline across various task domains, even with FP8 quantization. The results demonstrate that SnapMLA preserves accuracy across general QA, alignment, and reasoning tasks, highlighting the effectiveness of our sensitivity-aware mixed-precision strategy. In mathematical reasoning tasks, SnapMLA remains close to the BF16 baseline, while in coding tasks, it either

matches or slightly surpasses FlashMLA. Overall, SnapMLA preserves benchmark quality across diverse tasks, validating its ability to maintain accuracy while reducing computational overhead. Experimental results on kernel numerical precision are provided in Appendix G, demonstrating that our method maintains numerical accuracy comparable to the BF16 baseline.

4.3 Efficiency Analysis

End-to-End Throughput Figure 1 reports end-to-end decoding throughput under matched per-rank input shapes for BF16 FlashMLA and SnapMLA. We assess generation throughput across different parallelism strategies, examining both data parallelism (DP) and tensor parallelism (TP) configurations (DP1/TP8, DP4/TP2, DP8/TP1), as well as varying context lengths ranging from 16k to 128k.

As shown in Figure 1, SnapMLA consistently outperforms FlashMLA. The largest observed speedup reaches up to $1.91\times$, demonstrating the execution efficiency of the FP8 MLA decoding pipeline under matched workload shapes. Figure 7 further isolates kernel-level TFLOPS under fixed-batch synthetic configurations; no batch-size search is used in that sensitivity study. Additional efficiency analyses, including kernel Roofline performance and input configuration sensitivity, can be found in Appendix H and I. Appendix J summarizes how the numerical, end-to-end, and kernel-level evaluations support the coupled design choices in SnapMLA, while also clarifying that they are not intended as a complete factorial ablation of every kernel subcomponent.

5 Related Work

FP8 Attention and MLA Decoding FP8 attention kernels such as FlashAttention-3 [14] demonstrate that Hopper FP8 Tensor Cores can substantially accelerate attention when the computation is organized around hardware layout constraints such as the k-major operand format. However, these designs target standard attention and do not directly address MLA's shared latent KV cache or its decoupled RoPE component, whose numerical sensitivity motivates the mixed-precision treatment in SnapMLA. FlashMLA [16] provides a highly optimized BF16 MLA decoding kernel on Hopper GPUs, but directly extending it to FP8 introduces scale-layout mismatches and pipeline hazards under per-token quantization. SnapMLA builds on these lines of work by co-designing RoPE-aware KV-cache quantization, scale-aware FP8 PV computation, and fused data movement for MLA decoding. Additional related-work discussion is provided in Appendix B.

6 Conclusion

In this paper, we introduce SnapMLA, an FP8 decoding framework specifically designed to optimize long-context inference for MLA-based LLMs. By leveraging hardware-aware algorithm-kernel co-optimization, our framework effectively addresses the challenges associated with MLA KV cache quantization, quantized MLA computation, and system-level support for FP8 MLA decoding. Extensive experiments show that SnapMLA achieves up to a $1.91\times$ improvement in throughput on long-output decoding workloads while maintaining near-parity benchmark quality compared with the BF16 baseline on the evaluated reasoning and code-generation benchmarks. These contributions position SnapMLA as an effective framework for efficient decoding of long-context MLA LLMs, ensuring both high computational throughput and the preservation of model accuracy.

Acknowledgements and Disclosures

Funding: This work was supported by Meituan.

Competing interests: The authors declare no competing interests beyond their disclosed affiliations.

References

[1] Liu A, Mei A, Lin B, et al. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

[2] Liu A, Feng B, Xue B, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[3] Yadagiri Annepaka and Partha Pakray. Large language models: a survey of their development, capabilities, and applications. *Knowledge and Information Systems*, 67(3):2967–3022, 2025.

[4] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

[5] ByteDance-Seed. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads, 2025.

[6] Hooper C, Kim S, Mohammadzadeh H, et al. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In *Advances in Neural Information Processing Systems*, volume 37, pages 1270–1303, 2024.

[7] Team M L C, Gui A, Li B, et al. Introducing longcat-flash-thinking: A technical report. *arXiv preprint arXiv:2509.18883*, 2025.

[8] Team M L C, Li B, Lei B, et al. Longcat-flash technical report. *arXiv preprint arXiv:2509.01322*, 2025.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[10] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with mmlu? *arXiv preprint arXiv:2406.04127*, 2025.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[12] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2021.

[13] HMMT. Hmmt 2025, 2025.

[14] Shah J, Bikshandi G, Zhang Y, et al. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems*, volume 37, pages 68658–68685, 2024.

[15] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

[16] Shengyu Liu Jiashi Li. Flashmla: Efficient multi-head latent attention kernels. `https://github.com/deepseek-ai/FlashMLA`, 2025.

[17] Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems*, 35:14651–14662, 2022.

[18] Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442*, 2024.

[19] Qingyuan Li, Yifan Zhang, Liang Li, Peng Yao, Bo Zhang, Xiangxiang Chu, Yerui Sun, Li Du, and Yuchen Xie. Fptq: Fine-grained post-training quantization for large language models. *arXiv preprint arXiv:2308.15987*, 2023.

[20] Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. From live data to high-quality benchmarks: The arena-hard pipeline, April 2024.

[21] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[22] Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of LLMs for logical reasoning. In *Forty-second International Conference on Machine Learning*, 2025.

[23] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

[24] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[25] Hong Liu, Jiaqi Zhang, Chao Wang, Xing Hu, Linkun Lyu, Jiaqi Sun, Xurui Yang, Bo Wang, Fengcun Li, Yulei Qian, et al. Scaling embeddings outperforms scaling experts in language models. *arXiv preprint arXiv:2601.21204*, 2026.

[26] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

[27] MAA. Aime 2024, 2024.

[28] MAA. Aime 2025, 2025.

[29] NVIDIA. Nvidia h100 tensor core gpu, 2026. Accessed: 2026-02-06.

[30] Micikevicius P, Stosic D, Burgess N, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

[31] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[32] Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.

[33] Zunhai Su, Qingyuan Li, Hao Zhang, Weihao Ye, Qibo Xue, YuLei Qian, Yuchen Xie, Ngai Wong, and Kehong Yuan. Unveiling super experts in mixture-of-experts large language models. *arXiv preprint arXiv:2507.23279*, 2025.

[34] Zunhai Su, Wang Shen, Linge Li, Zhe Chen, Hanyu Wei, Huangqi Yu, and Kehong Yuan. Akvq-vl: Attention-aware kv cache adaptive 2-bit quantization for vision-language models. *arXiv preprint arXiv:2501.15021*, 2025.

[35] Zunhai Su and Kehong Yuan. Kvsink: Understanding and enhancing the preservation of attention sinks in kv cache quantization for llms. *arXiv preprint arXiv:2508.04257*, 2025.

[36] Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024.

[37] Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chen Gao, Chen Zhang, Chengcheng Han, et al. Longcat-flash-thinking-2601 technical report. *arXiv preprint arXiv:2601.16725*, 2026.

[38] Meituan LongCat Team, Bairui Wang, Bin Xiao, Bo Zhang, Bolin Rong, Borun Chen, Chang Wan, Chao Zhang, Chen Huang, Chen Chen, et al. Longcat-flash-omni technical report. *arXiv preprint arXiv:2511.00279*, 2025.

[39] Mart Van Baalen, Andrey Kuzmin, Suparna S Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, et al. Fp8 versus int8 for efficient deep learning inference. *arXiv preprint arXiv:2303.17951*, 2023.

[40] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

[41] He Xiao, Qingyao Yang, Dirui Xie, Wendong Xu, Zunhai Su, Wenyong Zhou, Haobo Liu, Zhengwu Liu, Ngai Wong, et al. Exploring layer-wise information effectiveness for post-training quantization in small language models. *arXiv preprint arXiv:2508.03332*, 2025.

[42] Jing Xiong, Liyang Fan, Hui Shen, Zunhai Su, Min Yang, Lingpeng Kong, and Ngai Wong. Dope: Denoising rotary position embedding. *arXiv preprint arXiv:2511.09146*, 2025.

[43] Jing Xiong, Jianghan Shen, Chuanyang Zheng, Zhongwei Wan, Chenyang Zhao, Chiwun Yang, Fanghua Ye, Hongxia Yang, Lingpeng Kong, and Ngai Wong. Parallelcomp: Parallel long-context compressor for length extrapolation. *arXiv preprint arXiv:2502.14317*, 2025.

[44] Weihao Ye, Qiong Wu, Wenhao Lin, and Yiyi Zhou. Fit and prune: Fast and training-free visual token pruning for multi-modal large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22128–22136, 2025.

[45] Su Z, Chen Z, Shen W, et al. Rotatekv: Accurate and robust 2-bit kv cache quantization for llms via outlier-aware adaptive rotations. *arXiv preprint arXiv:2501.16383*, 2025.

[46] Chen Zhang, Yang Bai, Jiahuan Li, Anchun Gui, Keheng Wang, Feifan Liu, Guanyu Wu, Yuwei Jiang, Defei Bu, Li Wei, et al. Efficient context scaling with longcat zigzag attention. *arXiv preprint arXiv:2512.23966*, 2025.

[47] Hengyuan Zhang, Zhihao Zhang, Mingyang Wang, Zunhai Su, Yiwei Wang, Qianli Wang, Shuzhou Yuan, Ercong Nie, Xufeng Duan, Qibo Xue, et al. Locate, steer, and improve: A practical survey of actionable mechanistic interpretability in large language models. *arXiv preprint arXiv:2601.14004*, 2026.

[48] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Limitations

SnapMLA is optimized for MLA decoding on NVIDIA Hopper-class GPUs, leveraging FP8 Tensor Cores, WGMMA, TMA, and Hopper-specific memory layouts. The reported kernel-level speedups are specific to this execution substrate and require architecture-specific adaptation on other GPUs. In particular, native block-scaled GEMM on newer GPUs may simplify or subsume the V-tile transposition, accumulator permutation, and scale/layout handling used here. In contrast, the heterogeneous quantization sensitivity of the RoPE and content components and the per-token V-scale semantics of the shared latent cache arise from MLA itself; native block scaling does not by itself eliminate these numerical requirements.

In addition, SnapMLA employs FP8 E4M3 quantization for the content portion of the MLA KV cache while preserving the RoPE component at higher precision. Our numerical analyses and benchmark results validate this mixed-precision strategy for the evaluated decoding workloads. Exploring more aggressive precision formats, alternative calibration policies, or substantially longer context lengths remains a promising direction for future investigation.

B Additional Related Work

FlashAttention-3 and FP8 Attention Acceleration The NVIDIA Hopper architecture introduces fourth-generation Tensor Cores, offering substantial throughput gains via FP8 computation. However, unlocking this potential requires adhering to strict hardware constraints: the FP8 WGMMA instructions mandate that the $\mathbf{V}$ matrix in the PV GEMM be contiguous along the sequence dimension (k-major). This requirement conflicts with the conventional head-contiguous memory layout used in standard attention implementations.

FlashAttention-3 [14] enables efficient FP8 attention acceleration by resolving this layout mismatch without expensive offline data conversion. Instead of relying on hardware-specific memory instructions, it adopts an on-the-fly transposition strategy within the computation kernel. By dynamically reshaping the $\mathbf{V}$ blocks during the loading phase, it aligns the data layout with the Tensor Core requirements while pipelining memory access with computation to hide latency. Furthermore, to align the $\mathrm{QK}^\top$ GEMM accumulator with the input requirements of the subsequent PV GEMM, FlashAttention-3 utilizes register-level permutations. By ensuring the operands match the specific layout required by WGMMA instructions, it achieves mathematical exactness while maximizing the utilization of FP8 compute units.

While FlashAttention-3 successfully accelerates standard MHA through this uniform FP8 pipeline, its design premises restrict its direct applicability to MLA. Specifically, MLA's decoupled RoPE component imposes heterogeneous precision requirements that are incompatible with a uniform FP8 data path. SnapMLA addresses this challenge by preserving the RoPE component in higher precision while quantizing the latent content path.

FlashMLA: Architecture-Aware MLA Decoding on NVIDIA Hopper GPUs FlashMLA [16] is a specialized kernel engineered for the NVIDIA Hopper architecture to accelerate MLA decoding. Unlike traditional kernels that often suffer from memory-bound bottlenecks, FlashMLA is designed to maximize instruction throughput in compute-bound scenarios by strictly aligning algorithmic execution with hardware resources.

The technical core of FlashMLA lies in its asynchronous pipelining. By leveraging TMA and WGMMA instructions, the kernel enables direct matrix operations on shared memory, reducing pipeline bubbles caused by register movement and synchronization. It uses a warp-specialized scheduling model where data movement and computation are decoupled through parameterizable buffers, mitigating register pressure within the SM. Its fine-grained TMA-to-GEMM pipeline also allows GEMM operations to trigger immediately upon partial data arrival, helping maintain Tensor Core utilization. SnapMLA extends this line of work to FP8 MLA decoding by introducing RoPE-aware quantization, scale-aware PV reconstruction, and fused dataflow support.

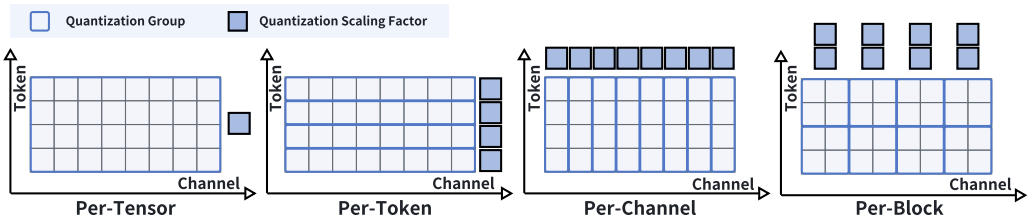


Figure 4: Illustration of various quantization granularities.

C Preliminaries of Low-Bit Quantization

Extensive prior research has focused on the quantization of LLMs, addressing both linear layers [9, 23, 6, 19, 41, 47] and KV caches [18, 6, 26, 45, 34]. This technique not only reduces memory footprint and data transfer latency but also facilitates acceleration through low-precision computation kernels. The evolution of hardware has driven a shift from integer (INT) to floating-point (FP) quantization [14, 30, 17]. On older architectures like NVIDIA Ampere, INT quantization prevails, with accuracy enhanced by tailoring weight and activation distributions [23, 4]. Meanwhile, newer generations such as Hopper and Blackwell unlock native support for low-bit FP computation. With its wider dynamic range, FP quantization inherently captures distribution characteristics more effectively [39], yielding higher accuracy than INT at the same bit-width.

The granularity of quantization plays a significant role in determining accuracy, computational efficiency, and engineering feasibility. Typically, several levels of quantization granularity exist (as illustrated in Figure 4), each determined by the method used to apply scaling factors to the input tensor $\mathbf{X} \in \mathbb{R}^{M \times N}$:

(1) ***Per-Tensor Quantization***

A single scalar scaling factor $S \in \mathbb{R}^1$ is shared across the entire tensor:

$$\mathbf{X}_q = \text{round}\left(\frac{\mathbf{X}}{S}\right). \tag{7}$$

This method minimizes metadata overhead but may compromise accuracy due to limited adaptability to the tensor's diverse values.

(2) ***Per-Token Quantization***

An independent scaling factor is computed for each token (i.e., each row), resulting in a column vector $S \in \mathbb{R}^{M \times 1}$:

$$\mathbf{X}_q^{(i,:)} = \text{round}\left(\frac{\mathbf{X}^{(i,:)}}{S_i}\right), \quad i = 1, \ldots, M. \tag{8}$$

Previous research has shown that significant outlier activation tokens [33, 36] and KV cache outlier tokens [35] exist in LLMs. This approach captures cross-token activation variations, making it particularly effective for handling outliers.

(3) ***Per-Channel Quantization***

In this scheme, each channel (i.e., each column) of the tensor is quantized independently using a separate scaling factor. The scaling factors are organized as a vector $S \in \mathbb{R}^{1 \times N}$:

$$\mathbf{X}_q^{(:,j)} = \text{round}\left(\frac{\mathbf{X}^{(:,j)}}{S_j}\right), \quad j = 1, \ldots, N. \tag{9}$$

This method allows for more flexibility in capturing variations across different channels [6, 26], improving accuracy over per-tensor quantization.

(4) ***Per-Block Quantization***

The tensor is partitioned into fixed $B \times B$ sub-blocks (e.g., 128×128 or 64×64), with each block assigned an independent scaling factor arranged in a matrix:

$$S \in \mathbb{R}^{\lceil \frac{M}{B} \rceil \times \lceil \frac{N}{B} \rceil}. \tag{10}$$

The quantization is then applied per block:

$$\mathbf{X}_q^{(\text{block }(i,j))} = \text{round}\left(\frac{\mathbf{X}^{(\text{block }(i,j))}}{S_{i,j}}\right). \tag{11}$$

This scheme balances accuracy with hardware-friendly block-wise computation, particularly beneficial in scenarios where parallel processing of smaller blocks is required.

D Online Scale Fusion for Softmax Accumulation

Scale Convention and Exactness Throughout this section, a quantized tensor X_q with scale σ_X represents the real-valued tensor $X \approx \sigma_X X_q$, with scales broadcast along the corresponding GEMM reduction dimension. For QK computation, σ_q and σ_K correspond to the content scales S^{Q_c} and S^{K_c} used in the pre-scaled RoPE alignment. For PV computation in the absorbed MLA form, the value operand reuses the latent content cache; therefore, the per-token V scale S_V follows the same latent-cache scale convention as the content cache. We use $\sigma_P^{(k)}$ exclusively for the dynamic quantization scale of the fused probability block at tile k. The scale relocation $P(S_V \odot V_q) = (P \odot S_V)V_q$ and the online state rescaling are algebraically exact up to finite-precision rounding; approximation is introduced by FP8 quantization of the content cache and fused probability blocks. In implementation, dynamic scales are lower-bounded by a small ϵ before division to avoid zero-scale cases.

Block-wise P quantization introduces scale variations across blocks. To maintain arithmetic accuracy during tiled accumulation without incurring explicit dequantization overhead, we fold the probability-block quantization scale into the online Softmax accumulation updates. Specifically, we track two intermediate states during the tiled computation: the scaled normalization statistic L, which tracks the accumulated sum of exponentials in the current probability-scale domain, and the partial attention output O. The QK logits and the Softmax distribution are unchanged; scale handling is applied after exponentiation, when the per-token V scale is fused into the current probability block.

To ensure hardware efficiency and minimize computational overhead, we maintain O and L in the current probability-scale domain. Let $\sigma_P^{(k)}$ denote the dynamic quantization scale of the fused probability block at tile k, and let $S_V^{(k)}$ denote the per-token scale of V. The following update rules describe how the normalization statistic and the attention output are computed in this hardware-optimized pipeline:

Hardware-Optimized Online Fusion (With L and O Updates) For each block k, let s_j be the restored QK logit and $m^{(k)} = \max(m^{(k-1)}, \max_{j \in \text{Block } k} s_j)$. We first compute the unnormalized Softmax term $e_j = e^{s_j - m^{(k)}}$, then form the fused probability term $\tilde{e}_j = e_j S_{V,j}^{(k)}$ before block-wise quantization. Here, $V_{q,j}$ denotes the quantized V value used by the FP8 PV GEMM. The states $L^{(k)}$ and $O^{(k)}$ are updated iteratively as follows:

$$L^{(k)} = L^{(k-1)} \cdot e^{(m^{(k-1)} - m^{(k)})} \cdot \frac{\sigma_P^{(k-1)}}{\sigma_P^{(k)}} + \frac{\sum_{j \in \text{Block } k} e_j}{\sigma_P^{(k)}} \tag{12}$$

$$O^{(k)} = O^{(k-1)} \cdot e^{(m^{(k-1)} - m^{(k)})} \cdot \frac{\sigma_P^{(k-1)}}{\sigma_P^{(k)}} + \frac{\sum_{j \in \text{Block } k} \tilde{e}_j V_{q,j}}{\sigma_P^{(k)}} \tag{13}$$

This formulation ensures that the output O and normalization statistic L are dynamically re-scaled to account for both the shifting maximum $m^{(k)}$ and the block-specific probability scale $\sigma_P^{(k)}$, thus eliminating the need for a separate dequantization pass.

E Supplementary Implementation Details for Quantized PV Computation

The Scale Hazard in Double-Buffered Execution The original FlashMLA kernel utilizes a double-buffered pipeline, coordinated by two warp groups (WGs), to parallelize the $\mathbf{PV}$ computation. A key design feature is the inverted execution order of WG1: it computes the contribution from the second block ($\mathbf{P}_1\mathbf{V}_1^R$) before the first block ($\mathbf{P}_0\mathbf{V}_0^R$), maximizing instruction-level overlap and

Algorithm 1 Decoding Process with the Dual-WG Pipeline

Require: $\mathbf{q} = [\mathbf{q}_c, \mathbf{q}_r] \in \mathbb{R}^d$ in HBM, with $\mathbf{q}_c \in \mathbb{R}^{d_c}$ (FP8) and $\mathbf{q}_r \in \mathbb{R}^{d_r}$
Require: Cached $\mathbf{K} = [\mathbf{K}_c, \mathbf{K}_r] \in \mathbb{R}^{N \times d}$ in HBM, with $\mathbf{K}_c \in \mathbb{R}^{N \times d_c}$ (FP8) and $\mathbf{K}_r \in \mathbb{R}^{N \times d_r}$; in absorbed MLA, the PV value operand reuses the latent content cache, denoted $\mathbf{V}_q \equiv \mathbf{K}_c$
Require: Fp8 scaling factors $\boldsymbol{\sigma}_\mathbf{q} \in \mathbb{R}$ (for $\mathbf{q}_c$) and $\boldsymbol{\sigma}_\mathbf{K} \in \mathbb{R}^{N \times 1}$ (for $\mathbf{K}_c$) in HBM
Require: $\boldsymbol{\sigma}_\mathbf{q}$ and $\boldsymbol{\sigma}_\mathbf{K}$ are pre-injected into $\mathbf{q}_r$ and $\mathbf{K}_r$
Require: Key block size $B_c = 64$ with $T_c = \lceil \frac{N}{B_c} \rceil$, two warp groups: WG0, WG1.

Initialize $p_0, p_1 \in \mathbb{R}^{B_c}$, γ_0, γ_1, $m = -\infty$, $\sigma_p = 1.0$ in SRAM
Initialize $\mathbf{o}^L = \mathbf{o}^R = (0) \in \mathbb{R}^{\frac{d_c}{2}}$, $\ell = 0$ in Register
for $j = 0$ **to** $Tc - 1$ **step** 2 **do**
  Load $\mathbf{K}_0 = \mathbf{K}_j$, $\mathbf{K}_1 = \mathbf{K}_{j+1}$ in SRAM

Warp Group 0 (WG0)

$\mathbf{s}_j = \mathbf{q}\mathbf{K}_0^\top$
$\mathbf{V}_0^L = ({}^\top\mathbf{K}_0^L)$, $\mathbf{V}_0^R = ({}^\top\mathbf{K}_0^R)$
$\mathbf{s}_j = \mathbf{s}_j \odot (\boldsymbol{\sigma}_\mathbf{q} \boldsymbol{\sigma}_{\mathbf{K}_0}^\top)$
$m^{cur} = \max(\mathbf{s}_j)$, $m^{new} = \max(m, m^{cur})$
$p_j = \exp(s_j - m^{new})$, $\ell^{cur} = \mathrm{sum}(p_j)$
$p_j = p_j \odot \boldsymbol{\sigma}_{\mathbf{K_0}}$, $m^{cur} = \max(p_j)$
$\sigma_p^{cur} = m^{cur}/448.0$, $p_j^{'} = \mathrm{Quant}(p_j;\, \sigma_p^{cur})$
$\ell^{cur} = \ell^{cur}/\sigma_p^{cur}$
$\gamma = \exp(m - m^{new})\sigma_p/\sigma_p^{cur}$
$\mathbf{o}^L = \gamma \mathbf{o}^L$, $\ell = \gamma\ell + \ell^{cur}$
Store $\gamma_0 = \gamma$, $m = m^{new}$, $\sigma_p = \sigma_p^{cur}$
Arrive(γ_0Ready)
Store $p_0 = p_j^{'}$
Arrive(p_0Ready)
$\mathbf{o}^L = \mathbf{o}^L + p_j^{'}\mathbf{V}_0^L$
Wait($p_0\mathbf{V}_0^R$Issued)
$\mathbf{o}^L = \gamma_1 \mathbf{o}^L$, $\ell = \gamma_1 \ell$
$\mathbf{o}^L = \mathbf{o}^L + p_1 \mathbf{V}_1^L$

Warp Group 1 (WG1)

$\mathbf{s}_{j+1} = \mathbf{q}\mathbf{K}_1^\top$
$\mathbf{V}_1^L = ({}^\top\mathbf{K}_1^L)$, $\mathbf{V}_1^R = ({}^\top\mathbf{K}_1^R)$
Wait(γ_0Ready)
$\mathbf{s}_{j+1} = \mathbf{s}_{j+1} \odot (\boldsymbol{\sigma}_\mathbf{q} \boldsymbol{\sigma}_{\mathbf{K}_1}^\top)$
$m^{cur} = \max(\mathbf{s}_{j+1})$, $m^{new} = \max(m, m^{cur})$
$p_{j+1} = \exp(s_{j+1} - m^{new})$, $\ell^{cur} = \mathrm{sum}(p_{j+1})$
$p_{j+1} = p_{j+1} \odot \boldsymbol{\sigma}_{\mathbf{K_1}}$, $m^{cur} = \max(p_{j+1})$
$\sigma_p^{cur} = m^{cur}/448.0$,
$p_{j+1}^{'} = \mathrm{Quant}(p_{j+1};\, \sigma_p^{cur})$
$\ell^{cur} = \ell^{cur}/\sigma_p^{cur}$
$\gamma = \exp(m - m^{new})\sigma_p/\sigma_p^{cur}$
$\mathbf{o}^R = \gamma_0 \mathbf{o}^R$, $\ell = \gamma\gamma_0\ell + \ell^{cur}$
Store $\gamma_1 = \gamma$, $m = m^{new}$, $\sigma_p = \sigma_p^{cur}$
Store $p_1 = p_{j+1}^{'}$
Wait(p_0Ready)
$\mathbf{o}^R = \mathbf{o}^R + p_0\mathbf{V}_0^R$
Arrive($p_0\mathbf{V}_0^R$Issued)
$\mathbf{o}^R = \gamma \mathbf{o}^R$
$\mathbf{o}^R = \mathbf{o}^R + p_{j+1}^{'}\mathbf{V}_1^R$

end for
$\ell = BlockReduceSum(\ell)$
Merge $\mathbf{o}_L$, $\mathbf{o}_R$ and Normalize $\mathbf{o} = \mathbf{o}/\ell$
$\mathbf{L} = m + \log(\sigma_p \ell)$
Write $\mathbf{o}$, $\mathbf{L}$ to HBM
Return the output $\mathbf{o}$ and the logsumexp $\mathbf{L}$.

hiding memory latency. However, this optimization introduces a fundamental numerical issue when using FP8 quantization. The problem arises from synchronizing the accumulator $\mathbf{O}_{acc}$, which holds the running sum of partial results, with the dynamically scaled $\mathbf{P}$ matrices.

- **Problem 1 (Rescaling $\mathbf{P}_0$)**: After processing $\mathbf{P}_1$, $\mathbf{O}_{acc}$ resides at the scale of $\mathbf{P}_1$. The original strategy rescales $\mathbf{P}_0$ to match this scale before accumulation. For FP8, this is problematic. The already quantized $\mathbf{P}_0$ is constrained within a severely limited dynamic range. Applying a large rescaling factor (when $\sigma_{\mathbf{P}_1} \gg \sigma_{\mathbf{P}_0}$) disrupts its value distribution, leading to irreversible precision loss.
- **Problem 2 (Rolling Back $\mathbf{O}_{acc}$)**: The alternative—temporarily reverting $\mathbf{O}_{acc}$ to the scale of $\mathbf{P}_0$ and restoring it later—is equally problematic. This approach requires bidirectional rescaling, involving multiplication by both the ratio $\sigma_{\mathbf{P}_0}/\sigma_{\mathbf{P}_1}$ and its reciprocal $\sigma_{\mathbf{P}_1}/\sigma_{\mathbf{P}_0}$. As a result, each scale becomes a denominator in one of these operations. When a significant disparity exists between $\sigma_{\mathbf{P}_0}$ and $\sigma_{\mathbf{P}_1}$, these ratios can explode or vanish, posing a severe risk to numerical stability.

Lossless Pipeline Reconstruction via Order Enforcement To resolve this issue, we abandon the inverted execution logic and enforce a strictly monotonic execution order for WG1: $\mathbf{P}_0\mathbf{V}_0^R \rightarrow \mathbf{P}_1\mathbf{V}_1^R$.

This reconstruction guarantees that the accumulator's scale updates follow a strict, unidirectional progression, aligned with the computation order. The bidirectional rescaling dependency—and the associated risk of division by extreme scale differences—is completely eliminated.

Although this serialization alters the original memory-compute overlap pattern, we recover the lost performance by fine-tuning inter-group synchronization barriers and instruction scheduling. The

redesigned pipeline, detailed in Algorithm 1, successfully balances the strict numerical stability required for FP8 computation with the high-throughput demands of a production-grade kernel.

F Benchmarks Details

The main accuracy evaluation in Table 1 uses the following 11 benchmarks:

- **General QA:** MMLU-Pro [40], a robust re-evaluated version of MMLU [12] that corrects errors and reduces contamination. MMLU-Redux [10], another high-quality variant of the MMLU benchmark.
- **Alignment:** IFEval [48], an instruction-following benchmark consisting of a set of prompts with programmatically verifiable constraints, offering an objective score on the model's fidelity to complex instructions. Arena-Hard [20], a benchmark from the Chatbot Arena platform for assessing a model's helpfulness and conversational quality on difficult, open-ended user queries.
- **Mathematical Reasoning:** Olympiad-level mathematical benchmarks, including MATH-500 [21], AIME-24 [27] and AIME-25 [28] (American Invitational Mathematics Examinations), and BeyondAIME [5].
- **General Reasoning:** GPQA-Diamond [31], a benchmark for evaluating deep reasoning on graduate-level questions across several science domains. ZebraLogic [22], consisting of classic logic grid puzzles that require multi-step deductive reasoning and constraint satisfaction.
- **Coding:** LiveCodeBench (LCB) [15], a dynamic benchmark for evaluating coding problems, with specific problems ranging from 2408 to 2505.

Generated Length Statistics SnapMLA optimizes the autoregressive decoding stage, where reasoning and code-generation tasks can produce long outputs. Table 2 reports the average generated lengths observed on DeepSeek-V3.1. This workload-characterization set partially overlaps with the main accuracy evaluation in Table 1: HMMT-25 [13] and OJBench are additional workloads included only in the length statistics, while BeyondAIME lengths are not reported here. These statistics characterize the output lengths of the evaluated decoding workloads and show that several benchmarks exercise the attention kernel over thousands to tens of thousands of generated tokens. They are intended to characterize long-output decoding behavior, rather than to replace dedicated long-context input-understanding benchmarks. After FP8 quantization, the generated lengths remain close to their BF16 counterparts across benchmarks, with no consistent shortening trend. This provides additional evidence that the small benchmark accuracy differences in Table 1 are not caused by shortened or truncated outputs.

G Experiments on Numerical Accuracy

Settings To further evaluate the quantization accuracy of SnapMLA, we performed a layer-wise numerical analysis using real inference data from *LongCat-Flash-Thinking*. We compare the attention outputs from SnapMLA with the BF16 ground truth, evaluating the discrepancies using three metrics: Root Mean Square Error (RMSE), Cosine Difference ($1 - \text{cosine similarity}$), and Relative L2 Error, across all layers. To analyze the impact of quantization errors under different granularities and provide a comprehensive evaluation of SnapMLA's effectiveness in reducing errors, we also set up four alternative quantization configurations, as detailed in Table 3.

Analysis The layer-wise error comparison, shown in Figure 5, reveals two key insights. First, including RoPE term in the quantization process *(Config A)* results in a substantial increase in error metrics, particularly in the deeper layers. This "error explosion" empirically demonstrates the sensitivity of positional embeddings, underscoring the necessity of our RoPE-aware quantization. Second, as highlighted in the zoomed-in insets, coarse-grained quantization strategies *(Config B&C)* struggle to capture the dynamic range variations across tokens. Even the block-wise strategy *(Config D)* exhibits slightly higher degradation compared to our approach. In contrast, our method, which utilizes RoPE-Aware Per-token quantization, consistently delivers the lowest error rates across all metrics, maintaining numerical accuracy on par with the BF16 baseline.

Table 2: Average generated lengths (tokens) of DeepSeek-V3.1 decoding workloads. These statistics characterize long-output behavior rather than the main accuracy evaluation: HMMT-25 and OJBench are additional workloads, and BeyondAIME lengths are not reported.

Benchmark	BF16	FP8	Relative Diff.
General			
MMLU-Pro$_{\text{(acc)}}$	2,447	2,471	+1.0%
MMLU-Redux$_{\text{(acc)}}$	562	558	-0.7%
Instruction			
IFEval$_{\text{(strict prompt)}}$	680	672	-1.2%
Arena-Hard$_{\text{(hard prompt gemini)}}$	3,275	3,254	-0.6%
Math			
MATH-500$_{\text{(Mean@1)}}$	2,346	2,397	+2.2%
HMMT-25$_{\text{(Mean@32)}}$	16,618	16,989	+2.2%
AIME-24$_{\text{(Mean@32)}}$	11,909	11,616	-2.5%
AIME-25$_{\text{(Mean@32)}}$	15,208	15,327	+0.8%
Reasoning			
GPQA-Diamond$_{\text{(Mean@16)}}$	9,183	8,942	-2.6%
ZebraLogic$_{\text{(Mean@1)}}$	5,091	4,972	-2.3%
Coding			
LCB (24.08-25.05)$_{\text{(Mean@4)}}$	13,034	13,047	+0.1%
OJBench$_{\text{(Mean@1)}}$	21,174	22,041	+4.1%

Table 3: Different quantization configurations for KV cache in the numerical accuracy analysis.

Method	**Configuration**	**KV Cache Quantization**	
		Content Part	**RoPE Part**
SnapMLA (Ours)	***Per-Token RoPE-Aware***	***Per-Token***	***w/o quantization***
Config A	Per-Token RoPE-Unaware	Per-Token	Per-Token
Config B	Per-Tensor Static RoPE-Aware	Per-Tensor Static (Fixed Scale 1.0)	w/o quantization
Config C	Per-Tensor Dynamic RoPE-Aware	Per-Tensor Dynamic	w/o quantization
Config D	Per-Block RoPE-Aware	Per-Block	w/o quantization

H Kernel Efficiency and Roofline Analysis

To gain a deeper understanding of the sources of the efficiency gains, we perform a comprehensive analysis of the compute throughput of the SnapMLA kernel under end-to-end workloads. We now derive the effective theoretical peak performance. The computational core of the MLA kernel consists of sixteen FP8 tiles (content term) and one BF16 tile (RoPE term), with the equivalent computational cost in BF16 units reduced from 17 to $16/2 + 1 = 9$. Given the theoretical BF16 peak of 148 TFLOPS for the GPU we used, the effective FP8 peak performance is calculated as:

$$\text{Peak}_{\text{effective}} = 148 \times \frac{17}{9} \approx 279.6 \text{ TFLOPS} \tag{14}$$

As shown in Figure 6, our kernel performance closely aligns with this effective peak, demonstrating that the overhead from pipeline reconstruction and layout transformation is negligible.

I Sensitivity to Input Configurations

In this section, we assess the kernel's robustness by evaluating its performance across various input configurations, including the number of heads ($H \in \{16, 32, 64, 128\}$) and multi-token prediction (MTP) settings ($MTP \in \{1, 2\}$), with a fixed batch size of 32.

As illustrated in Figure 7, throughput improves as the number of heads increases, reaching saturation at $H \geq 64$, where it attains approximately ***85% of the effective theoretical peak***. Moreover, increasing

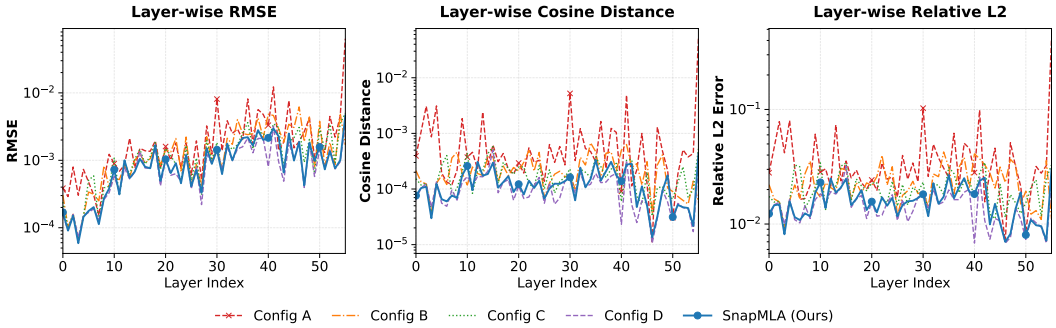


Figure 5: Layer-wise numerical fidelity analysis (context length = 32k). For details on the quantization configurations, please refer to Table 3.

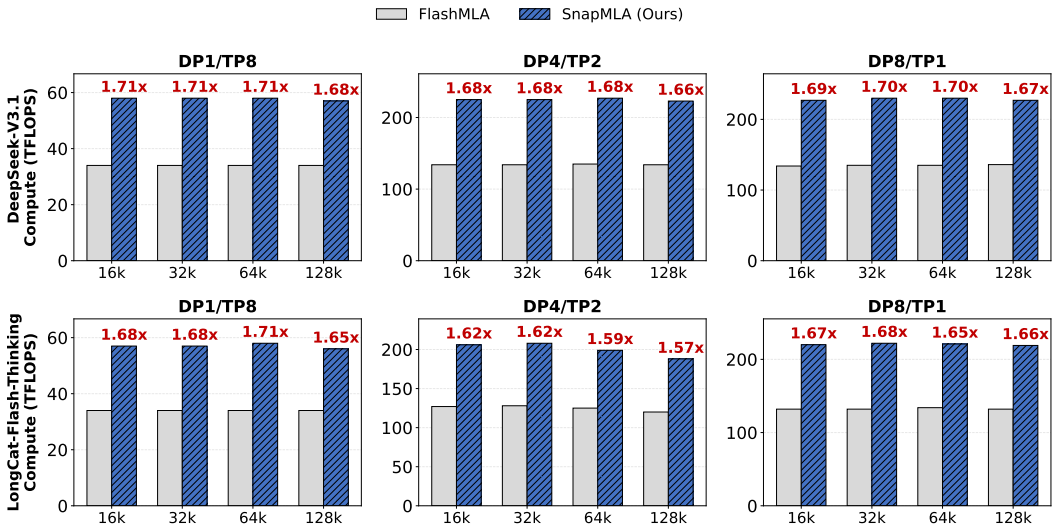


Figure 6: Kernel-level compute performance (TFLOPS). We measure the compute throughput of SnapMLA (blue hatched) versus the FlashMLA baseline (gray) across varying sequence lengths. The workload configurations are derived from the corresponding end-to-end DP/TP settings. Our kernel closely tracks the trajectory of the effective theoretical peak.

the query length (MTP=2) provides a moderate performance boost. Throughout all configurations tested, and particularly for standard settings ($H \geq 32$), SnapMLA consistently outperforms the baseline, demonstrating its robustness across varying input conditions.

J Evidence for Design Choices

SnapMLA is designed and evaluated as an integrated decoding pipeline rather than as a collection of independently removable optimizations. Several of its components are coupled by the MLA shared latent KV representation and the Hopper FP8 WGMMA layout constraints. As a result, removing one component can produce an invalid kernel, a substantially different data path, or an unfair comparison rather than a clean ablation point. We therefore interpret the current evaluation as evidence for the integrated co-design, not as a complete factorial ablation of every kernel subcomponent.

RoPE-Aware Per-Token Quantization Appendix G provides the most direct evidence for this design choice. The layer-wise numerical analysis compares SnapMLA with alternative KV-cache quantization configurations, including RoPE-unaware quantization and coarser content quantization granularities. The results show that quantizing the RoPE component or using coarser content

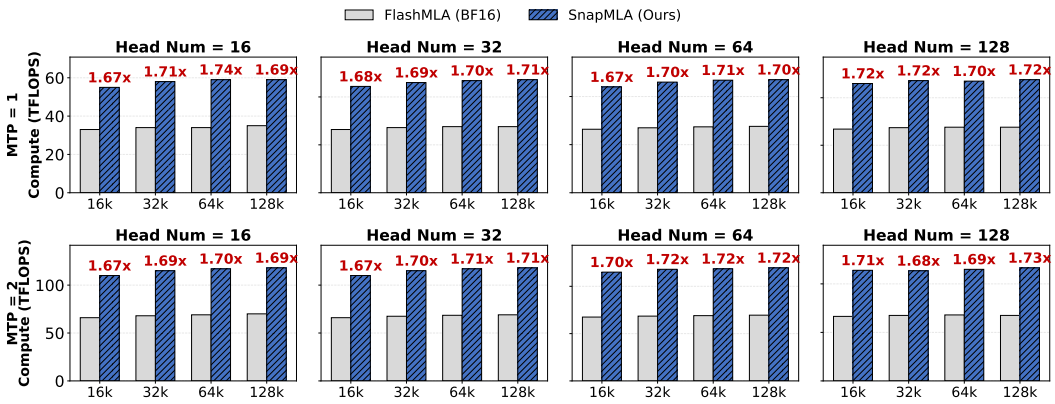


Figure 7: Kernel performance across different input configurations. We evaluate the compute throughput (TFLOPS) with a fixed batch size of 32, varying the number of heads from 16 to 128 and the query token length (MTP, rows) from 1 to 2. The results show that kernel performance improves as the number of heads increases, stabilizing at higher configurations. Additionally, increasing the query length (MTP=2) yields a slight throughput gain compared to the standard decoding setting (MTP=1). Across these configurations, SnapMLA consistently outperforms the baseline.

granularity leads to larger attention-output errors, supporting the choice of retaining the RoPE part in higher precision while applying per-token FP8 quantization to the content part.

Quantized PV Pipeline Reconstruction The PV reconstruction addresses the scale-layout mismatch caused by applying per-token scales to the MLA latent cache under the k-major layout required by FP8 WGMMA. The kernel-level analysis in Appendix H shows that the resulting FP8 MLA kernel tracks the effective hardware roofline under end-to-end workload configurations. This suggests that the additional scale handling, probability-block quantization, and layout transformation needed by the reconstructed pipeline do not dominate runtime in the evaluated settings.

End-to-End Dataflow Optimization The end-to-end throughput results in Figure 1 reflect the combined effect of FP8 KV-cache storage, quantized PV computation, and fused data movement under realistic DP/TP serving configurations. The input-sensitivity study in Appendix I further shows that the kernel maintains its advantage across relevant head-count and query-length configurations. Together, these results support the practical effectiveness of treating quantization, computation, and data movement as a coupled serving pipeline.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the scope of SnapMLA, its hardware-aware FP8 MLA decoding design, and the evaluated throughput and benchmark-quality claims; limitations are discussed in Appendix A.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in Appendix A, including hardware dependence, evaluated model scope, FP8 sensitivity, KV-cache memory savings, and inference-time serving scope.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present standalone theorems; the algebraic scale-fusion transformations and assumptions are described in Section 3 and the appendices.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Experimental settings, evaluated models, baselines, hardware, benchmark definitions, numerical fidelity analysis, and throughput measurements are described in Section 4 and the appendices; code is included in the supplementary material.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The implementation corresponding to SnapMLA is included in the supplementary material together with the paper submission.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer:[Yes]

   Justification: Section 4.1 specifies the evaluated models, BF16 FlashMLA baseline, FP8 E4M3 format, and H20 hardware; Appendix F details the benchmark suite and generated-length statistics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports aggregate benchmark scores following established benchmark protocols, including Mean@N where applicable; it does not introduce stochastic training runs for the proposed method.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experiments are conducted on an eight-GPU NVIDIA H20 node, and the evaluated DP/TP configurations, context-length sweeps, and kernel-level analyses are reported in Section 4.3 and Appendices H–I.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work targets inference efficiency for existing LLM serving workloads and does not introduce data collection, user-facing deployment, or human-subject components.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: The work improves inference efficiency for long-context MLA LLM serving; broader impacts are primarily related to computational efficiency and deployment cost rather than new model capabilities.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release an autonomous system or new model capability requiring additional safeguards; it focuses on low-precision inference kernels for existing MLA models.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the existing models, benchmarks, and kernel baselines used in the evaluation; benchmark and model references are listed in the bibliography.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new implementation is provided in the supplementary material, and the method, dataflow, kernel design, and experimental setup are documented in the main paper and appendices.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The work does not involve research with human subjects and therefore does not require IRB approval.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are evaluated as target workloads, but the core method is a low-precision inference and kernel optimization framework rather than a method that uses LLMs as a research component.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.