Diagonal-Tiled Mixed-Precision Attention for Efficient Low-Bit MXFP Inference

Anonymous CVPR submission

Paper ID *****

Abstract

Transformer-based large language models (LLMs) have demonstrated remarkable performance across a wide range of real-world tasks, but their inference cost remains prohibitively high due to the quadratic complexity of attention and the memory bandwidth limitations of high-precision operations. In this work, we present a low-bit mixed-precision attention kernel using the microscaling floating-point (MXFP) data format, utilizing the computing capability on next-generation GPU architectures. Our Diagonal-Tiled Mixed-Precision Attention (DMA) incorporates two kinds of low-bit computation at the tiling-level, and is a delicate fused kernel implemented using Triton, exploiting hardware-level parallelism and memory efficiency to enable fast and efficient inference without compromising model performance. Extensive empirical evaluations on NVIDIA B200 GPUs show that our kernel maintains generation quality with negligible degradation, and meanwhile achieves significant speedup by kernel fusion. We release our code at https://anonymous.4open.science/r/MP-Sparse-Attn.

1. Introduction

The rapid development of large language models (LLMs) has created a growing demand for faster throughput. Inside the transformer architecture, self-attention becomes the main inference bottleneck because its cost scales quadratically with sequence length. To reduce this cost, prior work has explored several directions, including quantization-based compression [5], structured sparsity [6], and kernel-based approximations such as linear attention [10]. At the same time, modern GPU architectures are adding stronger support for low-precision computation. In particular, NVIDIA Blackwell introduces native support for microscaling number formats, including MXFP8, MXFP4, and NVFP4, achieves lower bitwidth with better quantization quality [1, 8]. These advances make low-precision LLM inference practical in real deployment settings.

Despite this promising hardware support, effectively leveraging MX formats to accelerate attention in LLMs presents significant challenges. ***Challenge 1:*** low-bit quantization can lead to severe accuracy degradation. Directly applying 4-bit MXFP formats to attention computation causes substantial quantization error (see Tab. 2). While recent work [15] introduces smoothing techniques to mitigate quantization errors, it requires additional floating-point `GEMV` operations that reduce kernel throughput. ***Challenge 2:*** unfused operations in quantization process undermine the efficiency of low-bit computation. Our experiment reveals that without kernel fusion, the quantization and format conversion brings redundant memory accesses and kernel launch cost that cannot be underestimated (refer to Tab. 7).

Table 1. Comparison of representative MXFP data formats, including their block sizes, element formats, and shared scale formats.

Name	Block Size	Element Format	Element Bits	Scale Format	Scale Bits
MXFP8	32	FP8 (E4M3 / E5M2)	8	E8M0	8
MXFP4	32	FP4 (E2M1)	4	E8M0	8
NVFP4	16	FP4 (E2M1)	4	E4M3	8

In this paper, we propose **Diagonal-Tiled Mixed-Precision Attention** (DMA), the first attention workflow designed to operate hybrid MX formats. DMA addresses the above challenges via two key techniques: First, we adopt a tiling-level mixed-precision design that partitions the attention matrix into low- and high-precision regions to retain the most salient information for critical tokens in high-precision, while leveraging faster low-precision MX formats elsewhere to ensure the speedup. Second, we build a full-stack attention computation workflow into a fused memory-efficient kernel to ensure the end-to-end efficiency. It covers quantization, microscaling transformation, low-bit encoding and packing, and attention computation. This fusion allows fine-grained parallel execution on GPU thread blocks without storing intermediate results, significantly reducing memory pressure and synchronization overhead.

We evaluate DMA on Longbench dataset. Experimental results show that DMA achieves lossless generation quality compared to full-precision attention baseline. Furthermore,

our ablation studies demonstrate the effectiveness of mixed-precision tiling, quantization granularity, and diagonal window design in balancing performance and precision.

Our main contributions are summarized as follows:

- We propose Diagonal-Tiled Mixed-Precision Attention (DMA), a new attention workflow that operates with hybrid MX formats for efficient LLM inference.
- We develop a fully fused GPU kernel that integrates quantization, microscaling transformation, and attention computation into one end-to-end workflow, reducing memory access and kernel launch overhead.
- We conduct extensive experiments and detailed ablation studies to show the lossless generation quality of our kernel, and offer practical insights into the trade-off between efficiency and accuracy.

2. Related Works

2.1. Efficient Attention

Under the context of large language models (LLMs), the computational overhead of attention scales quadratically with sequence length, making efficient attention a critical research direction. FlashAttention [4] introduced a highly efficient attention kernel that uses tiling and an online softmax approach to eliminate the need to buffer full attention matrices. Quantization-based approaches such as INT-FlashAttention [3] and SageAttention [13] series compress attention operands into low-bit formats. They are plug-and-play quantized attention kernels that achieves 2–3× speedup over FlashAttention. Sparse attention methods like SparseAttention [14] limit the number of active token pairs to reduce complexity while preserving important information. As a lightweight sparse attention alternative, TurboAttention [7] combines head-wise quantization and sparsity-aware softmax approximation to deliver 1.2–1.8× attention speed up and 4× KV cache compression.

2.2. MXFP Quantization

The recent introduction of Microscaling Floating-Point (MXFP) formats is an advancement over traditional Block Floating Point (BFP), designed to improve the numerical efficiency and deployment flexibility of low-precision computations in AI workloads [9]. As supported by NVIDIA's latest Blackwell GPU architecture, MXFP4 and MXFP8 allow for significantly higher theoretical throughput, up to 2× to 4× compared to FP16 while still maintaining competitive numerical accuracy in mixed-precision settings [8]. However, despite these theoretical advantages, systematic and practical software support for MXFP-based attention kernels remains limited, especially for end-to-end pipelines that operate below the standard IEEE precision formats. This motivates our first contribution: a full-stack low-bit attention pipeline for MXFP formats (e.g., MXFP8, MXFP4, NVFP4), including efficient quantization and packing, low-bit computation combined with OnlineSoftmax technique.

3. Preliminaries

3.1. MXFP Data Format

Microscaling (MX) formats decompose a tensor into low-precision elements and a shared exponent scale per block (typically 32 or 16 elements). This approach allows dynamic range coverage much larger than conventional floating-point formats while significantly reducing storage and compute cost. Table 1 summarizes the MX formats used in this work. MX formats adopt a shared scale in E8M0 for every 32 elements per block and are named by prefixing "MX" to the element data format, i.e., MXFP8 (with FP8 data of E5M2/E4M3) or MXFP4 (FP4 data of E2M1). Since the exponent of FP32 uses 8-bit, the representable range of these formats covers that of FP32, ensuring compatibility with high-dynamic-range input distributions. While NVFP4 has FP8 (E4M3) shared scale for every 16 elements in each block. The finer-grained scaling and quantization granularity significantly reduces quantization error compared to MXFP4, enhancing downstream accuracy.

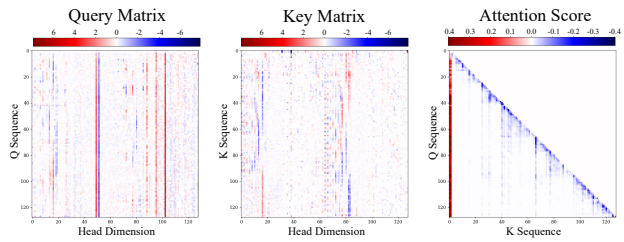
3.2. Online Softmax

To enable faster attention computation by dividing them into blocks, many efficient attention kernels uses OnlineSoftmax [4, 13] to ensure the equivalence of the results. For example, FlashAttention avoids materializing and storing the full attention matrix by computing the attention output in a tile-wise fashion. It fuses the attention score computation, softmax normalization, and value aggregation into a single memory-efficient kernel, using online row-wise softmax.

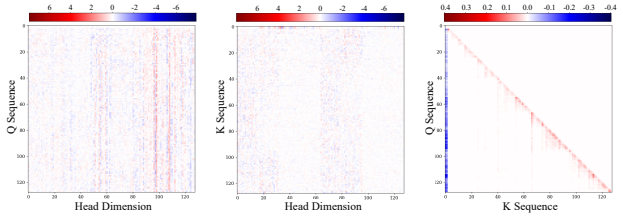
Each attention head is processed in parallel across blocks. L_Q, L_K are the sequence lengths of query and key/value matrices, respectively. D is the head dimension. We divide $\mathbf{Q}$ into tiles of size B_M. This results in $\lceil L_Q/B_M \rceil$ thread blocks executing in parallel per batch per head, each responsible for a tile of query $\mathbf{Q}_i \in \mathbb{R}^{B_M \times D}$. Within each thread block, it iterate over the key/value for $\lceil L_k/B_N \rceil$ tiles. Let $\mathbf{Q}_i \in \mathbb{R}^{B_M \times D}$ denote the i^{th} query tile, and $\mathbf{K}_j, \mathbf{V}_j \in \mathbb{R}^{B_N \times D}$ denote the j^{th} key and value tiles respectively. For each (i, j) tile pair, the attention score matrix is computed as:

$$\mathbf{S}_{ij} = \tilde{\mathbf{Q}}_i \times \mathbf{K}_j^\top, \quad \tilde{\mathbf{Q}}_i = \frac{\mathbf{Q}_i}{\sqrt{D}}. \tag{1}$$

It then applies OnlineSoftmax across key/value tiles to eliminate the need to store the full matrices of intermediate results. It tracks the running maximum of $\mathbf{S}_i$ in $\boldsymbol{m}_i$ and accumulates the intermediate attention outputs $\mathbf{O}_{i-1}$ across all



(a) MXFP4 quantization error



(b) NVFP4 quantization error

Figure 1. Visualization of quantization error of MXFP4 and NVFP4 format for query, key and attention score.

previous tiles, which is normalized by the ratio of the adjacent maximum values. The accumulated output $\mathbf{O}_i$ and normalization factor $\boldsymbol{l}_i$ are updated incrementally, tile by tile. We finalize the results of the i^{th} query tile by $\text{diag}(\boldsymbol{l}_i)^{-1}\mathbf{O}_i$.

4. Challenges in Low-bit MXFP Attention

Although full 4-bit quantization can reduce computation and memory cost, directly applying MXFP4 to the $QK^\top$ leads to clear degradation in attention quality. As shown in Fig. 1, the quantization error of MXFP4 is much pronounced than that of NVFP4. Table 2 confirms this trend quantitatively. Lower-bitwidth MX formats introduce larger quantization errors under all metrics. Direct MXFP4 causes a clear drop in attention-score fidelity, with cosine similarity decreasing from 0.988 to 0.714, PSNR from 71.70 to 60.82, and L1 error increasing from 0.246 to 0.924. NVFP4 is notably more stable, and adding token-wise quantization on top of it brings only marginal changes.

We observe that the error exhibits a clear channel-wise structure in query and key matrices, indicating that some feature dimensions are consistently more sensitive to low-bit quantization than others. However, this channel dimension is exactly the reduction axis in the $QK^\top$ multiplication. Therefore, it is unable to use channel-wise scaling in a fused low-bit attention kernel without introducing substantial implementation complexity.

5. Diagonal-Tiled Mixed-Precision Attention

In this section, we present the overall design of our method, which is built on the typical tiling-style of FlashAttention, but extends it with two components tailored for low-bit MXFP inference: a diagonal-tiled mixed-precision computation scheme to preserve the most sensitive attention regions (Sec. 5.2), and a fused quantization kernel to avoid the overhead of separate low-bit pre-processing steps (Sec. 5.3).

Table 2. Quantization error of attention score using different data formats. "+" means combining with tokenwise quantization.

Format	Cos Sim (↑)	PSNR (↑)	L1 (↓)	RMSE (↓)
MXFP8	0.988	71.70	0.246	0.003
MXFP4	0.714	60.82	0.924	0.009
NVFP4	0.982	69.37	0.309	0.003
NVFP4+	0.983	69.63	0.312	0.003
Ours	**0.988**	**71.70**	**0.248**	**0.003**

5.1. Overall Workflow

We follow the tiled execution pattern of FlashAttention to partition attention into sub-tensors for memory-efficient and parallel computation. Built on this workflow, DMA introduces a quantization-aware attention kernel that supports tile-level mixed precision on GPU tensor cores with native MXFP support. Meanwhile, we fuse the low-bit preprocessing pipeline into a single Triton implementation. Instead of executing quantization, low-bit encoding, packing, and scale conversion as separate operators, DMA performs them within the kernel, which reduces intermediate memory traffic, kernel launch overhead, and synchronization cost. This fused design is important for maintaining the end-to-end efficiency of mixed-precision attention. The overall framework is illustrated in Fig. 2.

5.2. Diagonal-Tiled Attention Workflow

We propose Diagonal-Tiled Mixed-Precision strategy to mitigate the accuracy degradation. As detailed in Sec. 4, low-bit quantization introduces significant quantization error, and the most influential scores typically concentrate along the diagonal of the attention matrix. Prior solutions, such as SageAttention2 [12], compensates for quantization error by performing additional GEMV operations between full-precision K and the mean-pooled query vector, which brings extra full-precision computation overhead. To mitigate this, we propose a mixed-precision strategy in tiling-level that selectively retains higher-precision tiles near diagonal, while aggressively quantizing peripheral regions to ensure the overall throughput.

Based on the tiling-wise attention paradigm, we introduce a diagonal window with size T, determining the amount of tokens for higher-precision computation along the token dimension. For each query tile indexed by n_Q, only the last T tokens are computed using higher-precision representations of query and key, while all preceding tiles are aggressively quantized. Taking causal attention as an

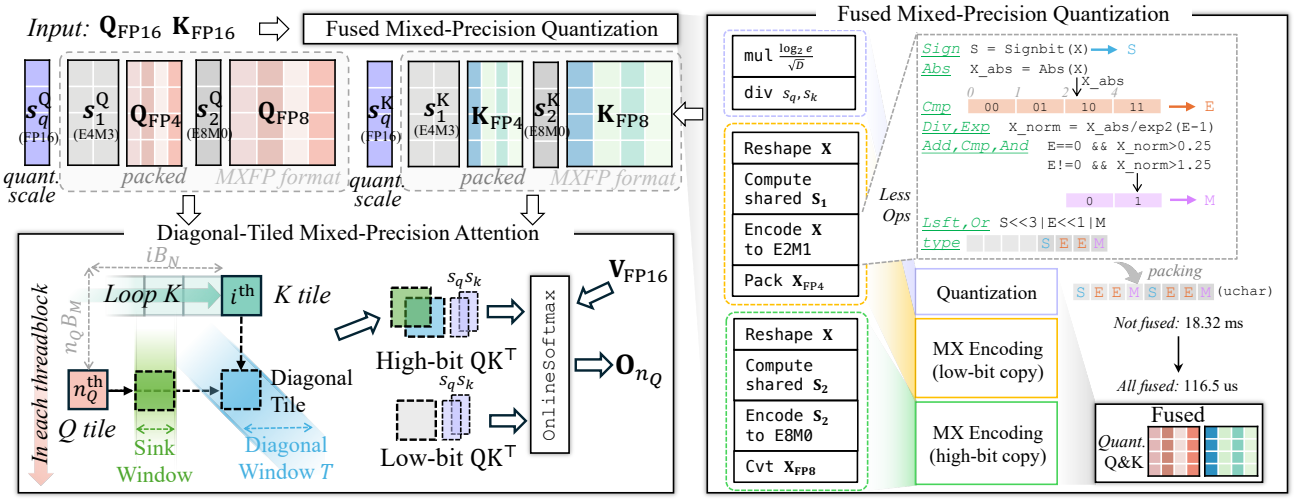


Figure 2. Overview workflow of our Diagonal-Tiled Mixed-Precision Attention. It first applies fused mixed-precision quantization to produce low-bit and high-bit Q and K representations, and then performs diagonal-tiled mixed-precision attention, using higher precision near the diagonal and lower precision elsewhere to balance accuracy and efficiency.

Algorithm 1 Diagonal-Tiled Mixed-Precision Attention

Input: $\tilde{\mathbf{Q}}^l \in \mathbb{R}^{B_M \times D}_{\text{FP4}}$, $\tilde{\mathbf{Q}}^h \in \mathbb{R}^{B_M \times D}_{\text{FP8}}$ are pre-processed with softmax scale. $\mathbf{K}^l \in \mathbb{R}^{B_N \times D}_{\text{FP4}}$. $\mathbf{K}^h \in \mathbb{R}^{B_N \times D}_{\text{FP8}}$. $V \in \mathbb{R}^{B_N \times D}_{\text{FP16}}$. B_M, B_N: tile sizes in sequence dimension for query and key, respectively. n_Q: thread block index. T: diagonal window size for high-precision.

Initialize: $\boldsymbol{m}_0 \leftarrow -\infty, \boldsymbol{l}_0 \leftarrow 1, \mathbf{O}_0 \leftarrow \mathbf{0}$

Phase 1: Low-precision tiles

for $i \leftarrow 0$ **to** $\lceil (n_Q B_M - T)/B_N \rceil$ **do**

  $\mathbf{S}_i \leftarrow \tilde{\mathbf{Q}}^l_{n_Q} \times \mathbf{K}^{l\top}_i$

  $\mathbf{O}_i, \boldsymbol{l}_i, \boldsymbol{m}_i \leftarrow \text{OnlineSoftmax}(\mathbf{S}_i, \mathbf{V}_i, \mathbf{O}_{i-1}, \boldsymbol{l}_{i-1}, \boldsymbol{m}_{i-1})$

end for

Phase 2: High-precision tiles in diagonal window

for $i \leftarrow \lceil (n_Q B_M - T)/B_N \rceil$ **to** $\lceil n_Q B_M / B_N \rceil$ **do**

  ▷ Skip upper triangular part for *causal attention*

  $\mathbf{S}_i \leftarrow \tilde{\mathbf{Q}}^h_{n_Q} \times \mathbf{K}^{h\top}_i$

  $\Delta_i \leftarrow (n_Q B_M + \mathcal{I}_{B_M}) < (iB_N + \mathcal{I}_{B_N})$

  $\mathbf{S}_i \leftarrow \mathbf{S}_i - \inf \cdot \Delta_i$

  ▷ Mask upper triangular of the global attention

  $\mathbf{O}_i, \boldsymbol{l}_i, \boldsymbol{m}_i \leftarrow \text{OnlineSoftmax}(\mathbf{S}_i, \mathbf{V}_i, \mathbf{O}_{i-1}, \boldsymbol{l}_{i-1}, \boldsymbol{m}_{i-1})$

end for

$\mathbf{O}_{n_Q} \leftarrow \text{diag}(\boldsymbol{l}_i)^{-1}\mathbf{O}_i$

Output: Causal attention output $\mathbf{O}_{n_Q} \in \mathbb{R}^{B_M \times D}$

example, we detail the workflow in Algorithm 1 in two phases.

Phase 1. Starting from the left columns in head dimension. Low-precision attention tiles are computed over the lower triangular region using low-bit quantized query and key matrices. We follow the standard OnlineSoftmax to incrementally accumulate the intermediate output $\mathbf{O}_i$. The first phase terminates after completing the computation for the first $\lceil (n_Q B_M - T)/B_N \rceil$ tokens.

Phase 2. Key/value tiles between $\lceil (n_Q B_M - T)/B_N \rceil$ and $\lceil n_Q B_M / B_N \rceil$ will be processed in this phase. To obtain a more precised attention weight, $\tilde{\mathbf{Q}}^h_{n_Q}, \mathbf{K}^h_i$ of the high-precision copy will be used. In causal attention, Δ_i masks scores within the current tile that correspond to the upper triangular region of the global attention matrix, where $\mathcal{I}_{B_M}$ and $\mathcal{I}_{B_N}$ are indices within each tile. It ensures that no query attends to future positions globally. Final outputs are normalized by the accumulated softmax scaling factors $\boldsymbol{l}_i$.

This diagonal-tiled execution maintains the performance and memory benefits of quantized attention while preserving the numerical precision of the attention mechanism to keep the generation quality.

Compatibility with Non-Causal Attention While Algorithm 1 illustrates the workflow of causal attention, our method is also compatible with non-causal attention. In the non-causal attention workflow, the low-bit computation covers both the lower and upper triangular regions of the attention matrix, excluding the diagonal window.

The structure of the workflow remains similar to that of the causal case, but the iteration range of Phase 1 differs: the tile index i spans two disjoint intervals. Specifically, tiles from 1 to $\lceil (n_Q B_M - T/2)/B_N \rceil$ corresponds to the lower triangular region, and tiles from $\lceil (n_Q B_M + T/2)/B_N \rceil$ to $\lceil n_K B_N / B_N \rceil$ covers the upper triangular. During this

phase, the attention weight $\mathbf{S}_i$ is computed using the low-precision copies of the query and key matrices. In Phase 2, tokens located within the diagonal window $[\lceil (n_Q B_M - T/2)/B_N \rceil, \lceil (n_Q B_M + T/2)/B_N \rceil]$ are reprocessed using high-precision copies of the inputs. The results are then accumulated into $\mathbf{O}_i$ computed in Phase 1. Once all key/value tiles have been iterated over, the final attention output is calculated by $\mathrm{diag}(\boldsymbol{l}_i)^{-1}\mathbf{O}_i$.

This partitioning is valid due to the equivalence under column-wise transformations in matrix multiplication. Specifically, the dot-product attention is a linear operation with respect to the key-value axis, and the output remains mathematically correct as long as all tiles of the attention matrix are covered and accumulated appropriately. Hence, splitting the computation into lower, upper triangular and diagonal regions does not compromise correctness, provided the transformations are applied consistently across tiles.

5.3. Dual MXFP Quantization Kernel Fusion

In this section, we describe the fused mixed-precision quantization kernel used in DMA. The goal is to convert the input tensor into both low-bit and high-bit MXFP representations within a single fused pipeline, so that the subsequent mixed-precision attention kernel can directly consume the quantized outputs without launching another kernels. The overall procedure is summarized in Algorithm 2.

Pre-process softmax scale (Step 1). Before quantization, we first apply the standard softmax scaling factor to the query tensor. Specifically, when the input is Q, we multiply it by $\log_2 e/\sqrt{D}$, where D is the head dimension. This incorporates the softmax normalization factor into the quantized computation in advance, so that the subsequent $QK^\top$ accumulation is already aligned with the scaled attention score formulation. Since our kernel computes the matrix product in base-2 arithmetic, folding this factor into the query in advance avoids an extra scaling step after accumulation and simplifies the fused implementation.

Quantization scale (Step 2). We fuse quantization with MXFP number format conversion at the kernel level. Following the two-level scaling strategy in SageAttention3, we note that the shared scale in NVFP4 uses the FP8 (E4M3) format, which ranges from $[-448, 448]$, while each element is represented in FP4, with a dynamic range of $[-6, 6]$. Therefore, the representable range of NVFP4 is the product of these two bounds. To fully utilize the available dynamic range and accommodate potential outliers without resorting to clipping, we compute the quantization scale in Step 2 (line 4) of Algorithm 2, scaling the original tensor into this representable range prior to quantization.

Algorithm 2 Fused Mixed-Precision Quantization

Input: $\mathbf{X} \in \mathbb{R}^{B\times D}$: input FP16 tensor. B: size of a tile in sequence dimension, D: size of head dimension. V_1, V_2: sizes of a block for low-/high-precision MXFP formats. l_1, l_2, u_1, u_2: lower/upper bound for both elements formats. $e^{\max}$: the exponent of the largest normal number of the element format.

Step 1: Pre-process softmax scale

1: **if** is Query **then**
2: $\quad \mathbf{X}_{\text{sm-scaled}} \leftarrow \mathbf{X} \cdot \frac{\log_2 e}{\sqrt{D}}$
3: **end if**

Step 2: Compute the quantization scale

4: $\mathbf{S}_q \leftarrow \max_D(|\mathbf{X}_{\text{sm-scaled}}|)/(448 \times 6)$
5: $\mathbf{X}_{\text{scaled}} \leftarrow \mathbf{X}_{\text{sm-scaled}}/\mathbf{S}_q$

Step 3: Compute shared scale for low precision format

6: $\mathbf{X}'_{\text{scaled}} \leftarrow \mathbf{X}_{\text{scaled}}$ ▷ reshape to $[B, D//V_1, V_1]$
7: $\mathbf{S}_{\text{FP4}} \leftarrow \max_D(|\mathbf{X}'_{\text{scaled}}|)/u_1$
8: $\mathbf{X}_{\text{clamped}} \leftarrow \text{clamp}(\mathbf{X}'_{\text{scaled}}/\mathbf{S}_{\text{FP4}}, l_1, u_1)$

Step 4: Encode $\mathbf{X}_{\text{clamped}}$ to E2M1

9: $\mathbf{X}_{\text{FP4}} \leftarrow$ Algorithm 3 $(\mathbf{X}_{\text{clamped}})$

Step 5: Pack two FP4 into one UINT8 along D

10: **if** pack along last dimension **then**
11: $\quad \mathbf{X}'_{\text{FP4}} \leftarrow \mathbf{X}_{\text{FP4}}$ ▷ reshape to $[B, (D+1)//2, 2]$
12: $\quad \mathbf{L}, \mathbf{H} \leftarrow \mathbf{X}'_{\text{FP4}}[:, :, 0], \mathbf{X}'_{\text{FP4}}[:, :, 1]$
13: $\quad \mathbf{X}_{\text{packed}} \leftarrow (\mathbf{H} \ll 4) \mid \mathbf{L}$
14: **end if**

Step 6: Compute shared scale for high precision format

15: $\mathbf{X}''_{\text{scaled}} \leftarrow \mathbf{X}_{\text{scaled}}$ ▷ reshape to $[B, D//V_2, V_2]$
16: $\mathbf{S}_{\text{shared}} \leftarrow \lfloor \log_2(\max_D |\mathbf{X}''_{\text{scaled}}|) \rfloor - e^{\max}$
17: $\mathbf{X}_{\text{FP8}} \leftarrow \text{clamp}(\mathbf{X}''_{scaled}/2^{\mathbf{S}_{\text{shared}}}, l_2, u_2)$

Step 7: Convert shared scale into E8M0

18: $\mathbf{S}_{\text{FP8}} \leftarrow \text{clamp}(\mathbf{S}_{\text{shared}} + 127, 0, 254)$

Output: Packed FP4 tensor $\mathbf{X}_{\text{packed}}$, FP8 tensor $\mathbf{X}_{\text{FP8}}$, shared scale of NVFP4 $\mathbf{S}_{\text{FP4}}$, shared scale of MXFP8 $\mathbf{S}_{\text{FP8}}$, quantization scale $\mathbf{S}_q$

Compute shared scale (Step 3 and Step 6). We first reshape the scaled input tensor along the packing dimension into a new matrix, where each row groups the V_1 or V_2 elements that share a common scale in the MXFP format. For NVFP4, the shared scale is computed directly as the absolute maximum of each group. For MXFP4 and MXFP8 formats, however, the scale is represented in integer E8M0 format. To maximize the representable range, we need to normalize the input exponent so that the largest exponent in the data aligns with the maximum representable exponent (denoted as $e^{\max}$) of the low-bit element format. Consequently, the shared exponent $\mathbf{S}_{\text{shared}}$ stores this offset from the input's exponent to $e^{\max}$. In E5M2, the maximum exponent $e^{\max}$ is 15 (i.e., $(11110)_2 = 30$ with a bias of 15, excluding reserved patterns with all 1 bits to represent in-

Algorithm 3 Encode FP16 tensor into E2M1 format

Input: $\mathbf{X} \in \mathbb{R}_{\text{FP16}}^{B\times D} \in [-6.0, 6.0]$: clamped FP16 tensor.

Step 4.1: Extract signbit

1: $\mathbf{S} \leftarrow \text{sign}(\mathbf{X})$

Step 4.2: Compute 2-bit exponent

2: $\mathbf{X}_{\text{abs}} \leftarrow |\mathbf{X}|$

3: $\mathbf{E} \leftarrow \mathbb{I}[\mathbf{X}_{\text{abs}} \geq 1] + \mathbb{I}[\mathbf{X}_{\text{abs}} \geq 2] + \mathbb{I}[\mathbf{X}_{\text{abs}} \geq 4]$

Step 4.3: Compute 1-bit mantissa

4: $bias \leftarrow 1$

5: $\mathbf{X}_{\text{norm}} \leftarrow \mathbf{X}_{\text{abs}}/2^{\mathbf{E}-bias}$

6: $\mathbf{M} \leftarrow \mathbb{I}[\mathbf{E} = 0]\cdot\mathbb{I}[\mathbf{X}_{\text{norm}} > 0.25] + \mathbb{I}[\mathbf{E} \neq 0]\cdot\mathbb{I}[\mathbf{X}_{\text{norm}} > 1.25]$

Step 4.4: Assemble quantized FP4 integer

7: $\mathbf{X}_{\text{FP4}} \leftarrow (\mathbf{S} \ll 3) \mid (\mathbf{E} \ll 1) \mid \mathbf{M}$

Output: Quantized tensor $\mathbf{X}_{\text{FP4}} \in \mathbb{R}_{\text{FP4}}^{B\times D}$

finite and NaN). In E4M3, $e^{\max} = 8$ (i.e., $(1111)_2 = 15$ with a bias of 7). Notably, E4M3 does not strictly follow IEEE-754, its normal numbers allows exponent bits set to all 1. The division of exponent shared scale allows full utilization of the exponent range within the limited bit budget of MXFP format.

Encoding of FP4 format (Step 4). Algorithm 3 shows the steps to quantize an FP16 tensor into E2M1 format (1-bit sign, 2-bit exponent, 1-bit mantissa). We begin by extracting the sign bit (Step 4.1). The 2-bit exponent is determined by thresholding the absolute value of the input against $\{1, 2, 4\}$, yielding exponent values in $\{0, 1, 2, 3\}$, which can be represented by 2 bits (Step 4.2). We employ the Kronecker delta function $\mathbb{I}[\cdot]$ in line 3 and 6, which returns 1 if the condition is true and 0 otherwise. In this way, subnormal values are assigned $E = 0$. To compute mantissas (Step 4.3), We normalize the input by the implied scale factor and assign the mantissa bit to 1 if $\mathbf{X}_{\text{norm}}$ exceeds 0.25 (the midpoint of 0 and 0.5). For normal values ($E \neq 0$), the mantissa is set to 1 if $\mathbf{X}_{\text{norm}}$ exceeds the midpoint of the two representable values under that exponent. For example, when $E = 3$, E2M1 can represent 4 and 6, which correspond to normalized values of 1 and 1.5, making 1.25 the comparison threshold for mantissa. To implement `roundTiesToEven` according to the standard, we prefer rounding to even mantissas (i.e., $M = 0$) in tie-breaking scenarios. For example, for input value 5, we prefer rounding to 4 to have mantissa bit as 0, rather than 6. So the comparison to midpoint threshold should be greater but not equal. In final, step 4 consrtucts the 4-bit representation using bitwise shift and OR operations.

Packing (Step 5). We encode two FP4 values into a single byte, assigning the value with the higher index to the most significant 4 bits and the other to the least significant 4 bits. This compact representation improves memory bandwidth utilization.

Scale Conversion into E8M0 (Step 7). The shared scale for MXFP8 should be in E8M0 (unsigned 8-bit integer) format according to the official document of MXFP computing. Therefore, we add 127 before clamping it into 0 to 254, ensuring it lies within the valid exponent range of E8.

6. Experiments

We conduct a comprehensive evaluation of the proposed DMA operator in terms of both accuracy and efficiency. We report results across a range of tasks and datasets to assess the overall performance. In addition, we perform detailed ablation studies to analyze the impact of key design choices, including numerical precision, quantization granularity, diagonal window size, and kernel fusion strategies.

6.1. Settings

Models and Datasets. We evaluate our method using LLaMA-3.1-8B and LLaMA-3.2-3B. Performance is measured on LongBench [2], which focuses on long-context understanding with sequence lengths ranging from 2.5K to 30K tokens. We report results across various subtasks to assess general long-context capabilities. For all reported metrics, higher values indicate better performance.

Implementation. We implement DMA in Triton. We compare our method with the native attention kernel, which refers to the SDPA kernel originally supported in `torch` and computed in BF16 format [11]. All experiments are conducted on a single NVIDIA B200 GPU.

6.2. Accuracy

Performance on LongBench We evaluate our method on LongBench to assess long-context language understanding performance. Table 3 shows that our method improves the average score over the native attention baseline on both LLaMA3.1-8B and LLaMA3.2-3B. Specifically, for LLaMA3.1-8B, the average score improves from 44.11 to 46.43. For LLaMA3.2-3B, the average score improves from 35.84 to 37.20. The gains are broad across many tasks, including `repobench-p`, `samsum`, `trec`, and `triviaqa`. In particular, `repobench-p` improves from 42.23 to 54.00 on LLaMA3.1-8B and from 44.14 to 50.15 on LLaMA3.2-3B. The average results across both models indicate that our method preserves, and in most cases improves, long-context accuracy compared with the native implementation.

Table 3. Comparison of attention implementations on LLaMA3.2-3B and LLaMA3.1-8B. "Native" refers to SDPA implementation supported by PyTorch.

Task	LLaMA3.1-8B Native	LLaMA3.1-8B Ours	LLaMA3.2-3B Native	LLaMA3.2-3B Ours
2wikimqa	39.15	32.54	30.75	36.33
dureader	28.40	32.93	28.11	33.04
gov_report	33.87	34.89	32.26	33.04
hotpotqa	49.89	50.49	46.18	48.59
lcc	49.46	56.96	42.98	45.54
lsht	41.50	44.50	26.75	30.75
multi_news	26.16	27.07	25.56	21.78
multifieldqa_en	52.26	52.65	46.84	47.87
multifieldqa_zh	57.57	58.81	48.06	53.62
musique	25.82	26.81	20.35	25.77
narrativeqa	25.28	26.70	19.76	25.44
passage_count	4.02	6.22	3.40	1.50
passage_retrieval_en	99.00	96.00	80.00	37.00
passage_retrieval_zh	92.14	85.50	8.25	10.50
qasper	42.64	43.05	32.13	39.62
qmsum	23.47	24.67	22.52	23.63
repobench-p	42.23	54.00	44.14	50.15
samsum	36.60	43.99	34.79	42.74
trec	65.00	71.50	62.50	69.50
triviaqa	77.01	87.89	83.68	88.13
vcsum	14.80	17.92	13.65	16.69
Avg.	44.11	**46.43**	35.84	**37.20**

6.3. Efficiency

Table 4 reports the latency breakdown of different formats and block-scale configurations. We compare our implementation with several fixed-format baselines, including MXFP4, NVFP4, and MXFP8. For each setting, we report the attention time, quantization overhead, and total runtime.

Among all evaluated settings, our configuration with diagonal and sink sizes set to 128 achieves the lowest total latency, at 7.776 ms. This is lower than MXFP4 (12.980 ms), NVFP4 (13.404 ms), and MXFP8 (16.771 ms). In particular, the main reduction comes from the attention kernel time, which is 7.110 ms in our 128/128 setting, compared with 12.491 ms, 12.941 ms, and 16.480 ms for the three baselines, respectively. We also evaluate a larger block-scale configuration with diagonal and sink sizes set to 256. In this case, the total latency increases to 15.720 ms. Compared with the 128/128 setting, this suggests that a larger block size is less efficient in our current implementation.

6.4. Ablation Study

Mixed-Precision Block Tile Sizes. We conduct ablation experiments to evaluate the impact of different block tile sizes on the similarity between the quantized attention matrix and its full-precision counterpart. As shown in Tab. 5, we report the similarity before and after quantization to show the representation errors under varying diagonal and sink tile configurations, including Cosine Similarity, Relative L1 Distance, Root Mean Square Error (RMSE), and Peak Signal-to-Noise Ratio (PSNR). The $\text{Bit}_{\text{high}}\%$ column denotes the percentage of values in the attention matrix that compute in high-precision.

We observe that increasing the tile size from 128 to 512 improves similarity metrics marginally. But it leads to a significant degradation in throughput due to reduced parallelism and slower computation as shown in Tab. 4. Based on this trade-off, we use the 128/128 configuration as the default setting in the following experiments.

Table 4. Latency breakdown of different block-scale types and configurations. "MP Size" denotes the mixed-precision block size used for the higher-bit diagonal and sink blocks.

Format	MP Size	Attn (ms)	Quant (ms)	Total (ms)
MXFP4	–	12.491	0.242	12.980
NVFP4	–	12.941	0.204	13.404
MXFP8	–	16.480	0.044	16.771
Ours	**128**	**7.110**	**0.382**	**7.776**
Ours	256	15.056	0.382	15.720

Table 5. Similarity metrics under different token numbers for diagonal and sink windows.

Diag.	Sink	Bit_{high} (%)	Cos Sim ↑	Rel. L1 ↓	RMSE ↓	PSNR ↑
-	-	0.0	0.778	0.620	0.065	43.715
-	-	100.0	0.819	0.547	0.059	44.568
0	128	1.15	0.781	0.780	0.072	42.817
128	0	1.15	0.782	0.644	0.066	43.635
128	**128**	**2.30**	**0.822**	**0.539**	**0.059**	**44.657**
512	512	9.22	0.826	0.542	0.058	44.731
2048	2048	36.87	0.852	0.521	0.054	45.352

Table 6. Ablation study of kernel fusion components to analyze their effect on throughput (TOPS) and latency. **Encode**: encoding FP16 to FP4/FP8 MX format. **Pack**: packing two FP4 values to one UINT8. **Scale Cvt.**: converting microscaling scalar to E8M0 format. **MP**: fusing quantization for mixed bitwidth to a single kernel.

Encode	Pack	Scale Cvt.	MP	$L = 2\text{k}$ (μs)	$L = 8\text{k}$ (μs)
✗	✗	✗	✗	7262.41	22628.96
✓	✗	✗	✗	802.90	1113.77
✓	✓	✗	✗	740.64	942.67
✓	✓	✓	✗	179.97	299.69
✓	✓	✓	✓	**97.87**	**282.46**

Kernel Fusion We evaluate the contribution of each kernel fusion component in Tab. 6. The fully unfused baseline, which executes all quantization-related steps separately, in-

Table 7. Latency breakdown of the non-fused MX encoding pipeline and our fused implementation.

Operator	Time	Time (%)
Not fused	18.320 ms	–
- Element encoding	17.742 ms	100.0%
MinOps	2.110 ms	11.89%
ArgMinOps	2.054 ms	11.58%
Direct_Copy	1.256 ms	7.08%
CompareEq	890.18 us	5.02%
AddOps	810.56 us	4.57%
MulFunctor	755.10 us	4.26%
Memcpy / Memset	206.85 us	1.17%
- Element packing	81.22 us	100.0%
BitwiseOr	23.74 us	29.24%
lshift	22.63 us	27.86%
- Scalar Convert	521.47 us	100.0%
IndexOps	35.75 us	6.85%
DeviceSelectSweep	25.73 us	4.93%
Write_Indices	20.42 us	3.91%
Direct_Copy	17.12 us	3.28%
Memcpy	13.57 us	2.60%
Kernel Fusion (Ours)	**116.480 us**	–

Table 8. Performance comparison under different quantization granularities. Latency are reported in milliseconds.

Granu.	Latency ↓	Cos Sim ↑	Rel. L1 ↓	RMSE ↓	PSNR ↑
Per-Tensor	6.276 ms	0.732	0.560	0.067	43.479
Per-Block	6.366 ms	0.736	0.558	0.067	43.531
Per-Token	7.131 ms	0.822	0.539	0.059	44.657

curs a latency of 7262.41 μs for sequence length $L = 2$k and 22628.96 μs for $L = 8$k. When only in-kernel FP16-to-MX encoding is enabled, the latency is reduced to 802.90 μs for $L = 2$k and 1113.77 μs for $L = 8$k. After further enabling FP4 packing, which packs two FP4 values into one `uint8`, the latency is further reduced to 740.64 μs for $L = 2$k and 942.67 μs for $L = 8$k. When scale conversion of microscaling factors to the `e8m0` format is also fused into the kernel, the latency drops substantially to 179.97 μs for $L = 2$k and 299.69 μs for $L = 8$k. Finally, after integrating mixed-precision quantization into a single fused kernel, denoted as **MP**, the latency is further reduced to 97.87 μs for $L = 2$k and 282.46 μs for $L = 8$k, achieving the best overall performance. Compared with the fully unfused baseline, this corresponds to a 74.2× speedup for $L = 2$k and an 80.1× speedup for $L = 8$k.

Overall, these results demonstrate that each fusion component contributes to latency reduction, and a fully fused kernel is critical for achieving an efficient mixed-precision attention.

Quantization Granularity. We further investigate the impact of different quantization granularities on both latency and kernel output precision. As shown in Tab. 8, we evaluate four granularity settings: Per-Tensor, Per-Block, and Per-Token. Latency is measured using the 5 warmups and average of 10 runs. The results show that finer granularity, such as Per-Token, achieves the highest output similarity before and after quantization, as indicated by higher Cosine Similarity (0.822), lower RMSE (0.059), and higher PSNR (44.657), but it also incurs the highest latency (7.131 ms). In contrast, Per-Tensor and Per-Block offer lower latency but exhibit larger quantization errors. Therefore, the choice of quantization granularity depends on the task requirement, specifically whether latency or output quality is prioritized.

7. Conclusion

In this paper, we present DMA, a diagonal-tiled mixed-precision attention method for low-bit MXFP attention computation for efficient LLM inference. DMA improves the accuracy of low-bit attention by preserving more sensitive diagonal regions in higher precision, while keeping the remaining regions in efficient low-bit formats. Combined with a fused dual-MXFP quantization kernel, our design reduces quantization preprocessing overhead and maintains practical efficiency. Experimental results show that DMA provides a better balance between accuracy and latency.

Limitation

Our current study is limited to a small set of model sizes and benchmark settings, and the evaluation is mainly focused on long-context text workloads. In particular, we mainly validate DMA on text-based tasks and do not extend the experiments on vision or vision-language settings. In addition, the mixed-precision tiling policy is not validated on extremely long sequence lengths, diverse hardware settings, or other model architectures and attention variants. Extensive experiments and validations are important directions for our future work.

References

[1] Eduardo Alvarez, Omri Almog, Eric Chung, Simon Layton, Dusan Stosic, Ronny Krashinsky, and Kyle Aubrey. Introducing nvfp4 for efficient and accurate low-precision inference. https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference, 2025. NVIDIA Technical Blog. 1

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding, 2024. 6

[3] Shimao Chen, Zirui Liu, Zhiying Wu, Ce Zheng, Peizhuang Cong, Zihan Jiang, Yuhan Wu, Lei Su, and Tong Yang. Int-flashattention: Enabling flash attention for int8 quantization, 2024. 2

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. 2

[5] Tim Dettmers, M. Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *arXiv.org*, 2022. 1

[6] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot, 2023. 1

[7] Hao Kang, Srikant Bharadwaj, James Hensman, Tushar Krishna, Victor Ruhle, and Saravan Rajmohan. Turboattention: Efficient attention approximation for high throughputs llms, 2024. 2

[8] NVIDIA. Nvidia rtx blackwell gpu architecture. https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf, 2025. Version 1.1, NVIDIA Technical Whitepaper. 1, 2

[9] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023. 2

[10] Yutao Sun, Zhenyu Li, Yike Zhang, Tengyu Pan, Bowen Dong, Yuyi Guo, and Jianyong Wang. Efficient attention mechanisms for large language models: A survey, 2026. 1

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 6

[12] Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization, 2025. 3

[13] Jintao Zhang, Jia Wei, Haofeng Huang, Pengle Zhang, Jun Zhu, and Jianfei Chen. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration, 2025. 2

[14] Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. *arXiv preprint arXiv:2502.18137*, 2025. 2

[15] Jintao Zhang, Jia Wei, Pengle Zhang, Xiaoming Xu, Haofeng Huang, Haoxu Wang, Kai Jiang, Jianfei Chen, and Jun Zhu. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training, 2026. 1