M^2XFP: A Metadata-Augmented Microscaling Data Format for Efficient Low-bit Quantization

Weiming Hu
weiminghu@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China

Zihan Zhang
tiancaizhangdaxian@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Haoyan Zhang
h.y.zhang-zdy@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China

Chen Zhang*
chenzhang.sjtu@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Cong Guo
guocong@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Yu Feng
y-feng@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Tianchi Hu
hutianchi1@huawei.com
Computing Product Line, Huawei
Shanghai, China

Guanglin Li
liguanglin10@huawei.com
Computing Product Line, Huawei
Shanghai, China

Guipeng Hu
huguipeng@huawei.com
Computing Product Line, Huawei
Shanghai, China

Junsong Wang
junsongwang@huawei.com
Computing Product Line, Huawei
Beijing, China

Jingwen Leng*
leng-jw@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China

Abstract

Existing low-bit Microscaling (MX) formats, such as MXFP4, often suffer from substantial accuracy degradation due to the use of a shared scaling factor with the Power-of-Two format. In this work, we explore strategies that introduce minimal metadata to recover accuracy lost during quantization while maintaining high bit efficiency across a wide range of large language models. We propose a complete algorithm-hardware co-design based on flexible metadata, featuring an online quantization with simple encoding. To support the proposed method efficiently, we implement a lightweight hardware unit and integrate it into the accelerator. Evaluation results demonstrate that our method substantially narrows the accuracy gap, achieving on average a 70.63% reduction in accuracy loss compared to MXFP4 and a 37.30% reduction relative to the latest NVFP4 on LLM benchmarks. Furthermore, our design delivers up to 1.91× speedup and 1.75× energy savings over state-of-the-art accelerators. Our code is available at https://github.com/SJTU-ReArch-Group/M2XFP_ASPLOS26.

CCS Concepts: • **Computer systems organization** → **Single instruction, multiple data**; **Systolic arrays**; **Neural networks**.

Keywords: Low-bit Quantization; Microscaling Data Formats; Hardware Acceleration

ACM Reference Format:
Weiming Hu, Zihan Zhang, Haoyan Zhang, Chen Zhang, Cong Guo, Yu Feng, Tianchi Hu, Guanglin Li, Guipeng Hu, Junsong Wang, and Jingwen Leng. 2026. M^2XFP: A Metadata-Augmented Microscaling Data Format for Efficient Low-bit Quantization. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '26), March 22–26, 2026, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3779212.3790185

1 Introduction

Large language models (LLMs) have grown rapidly in scale and capability, with model size emerging as a primary driver of accuracy and generalization. State-of-the-art deployments now involve hundreds of billions of parameters, such as

[*] Corresponding authors.

ASPLOS '26, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-2359-9/2026/03
https://doi.org/10.1145/3779212.3790185

LLaMA-3.1 [16], which contains up to 405 billion parameters. Storing these models in standard BF16 precision alone requires terabytes of main memory, far exceeding the capacity of commodity accelerators. The resulting memory and compute demands place tremendous stress on both cloud-scale and device-level systems, motivating aggressive model compression techniques [2, 19, 20, 23–25, 28, 44, 62]. Among these, low-bit quantization has emerged as a leading approach for reducing memory footprint, bandwidth consumption, and energy while preserving model quality, making it critical for the continued scaling of LLMs.

Recent advances in low-bit quantization have led to the adoption of Microscaling (MX) formats [56], which employ block-level shared scaling factors to enable fine-grained quantization. MX formats, such as MXFP4, have been widely adopted by industry and are natively supported in commercial accelerators, including NVIDIA's B200 [51], AMD's MI300 [61], and Microsoft's Maia 100 [67]. By exploiting shared exponents and streamlined dequantization, these formats deliver high throughput with minimal hardware overhead. However, accuracy degradation remains severe at 4-bit precision: the coarse resolution of power-of-two (E8M0) scaling misaligns with the local maximum, leading to significant rounding error, while more precise FP8 scaling (e.g., NVFP4) narrows dynamic range and requires additional rescaling[34]. Other attempts, such as custom data types [5, 15, 29, 32, 54], offer expressiveness but incur prohibitive hardware cost, especially for dynamic activations.

These limitations highlight a critical research gap: while scaling factor design has largely converged and data type innovations face scalability bottlenecks, the metadata axis remains relatively underexplored. Metadata, in principle, provides a flexible way to encode auxiliary precision or range information without altering the core data path, thereby enhancing quantization fidelity at low cost. Yet existing approaches remain fragmented. Outlier-oriented schemes (e.g., OliVe [27]) improve accuracy in tensor-wise settings but break down in group-wise MX formats, while structural metadata (e.g., MicroScopiQ [56]) incurs excessive overhead, often exceeding 40 bits per block. Consequently, MX formats today either sacrifice accuracy for efficiency or burden hardware with metadata complexity, leaving a wide unexplored design space. This motivates the central question of our work: *Can lightweight, principled metadata augmentation reconcile MX's efficiency with the accuracy demands of 4-bit LLM quantization?* Our exploration therefore focuses on the metadata axis as the primary remaining degree of freedom to close the 4-bit accuracy gap.

To answer this, we propose M^{2}XFP (Metadata-Augmented Microscaling Format), an algorithm–hardware co-design framework that systematically explores metadata allocation strategies. Our key insight is that metadata can serve as extra mantissa or exponent bits, enabling distinct trade-offs between precision refinement and range extension. Through a comprehensive design space exploration, we uncover a fundamental asymmetry: element-level metadata is most effective for dynamic activations, where lightweight, real-time encoding is essential, while subgroup-level metadata combined with scale search best serves static weights, where offline optimization is feasible. Building on this observation, we introduce a hybrid metadata scheme that applies element-level encoding to activations and subgroup-level encoding to weights. The resulting format improves bit efficiency with only 0.25 bits of metadata per element, delivering near-FP16 accuracy at effective 4.5-bit precision. We further design lightweight hardware support, integrated into systolic arrays with minimal extensions, that enables real-time metadata handling without disrupting the GEMM pipeline.

This paper makes the following contributions:

- We introduce a taxonomy of MX design dimensions (scaling factor, data type, metadata). Unlike prior work that primarily varies scaling factors or base data types, we perform an EBW-guided design space exploration along the under-explored metadata axis, covering both element-level and subgroup-level metadata under fixed and adaptive shared scales.
- We identify an asymmetric behavior between weights and activations and propose M^{2}XFP, a hybrid metadata-augmented MX format that uses element-level extra mantissa for activations and subgroup-level mantissa refinement with adaptive shared scales for weights.
- We design a hardware-efficient accelerator integration including a top-1 decode unit, an augmented FP4×FP4 PE, and a streaming quantization engine, and show that M^{2}XFP outperforms state-of-the-art MX accelerators in both accuracy and performance/energy at negligible area cost.

2 Background

2.1 Model Quantization

Quantization [2, 5, 15, 20, 26, 27, 39, 43, 44, 46, 58, 64] is a widely used technique for improving computational and memory efficiency by representing parameters with fewer bits. The standard approach maps a full-precision tensor $\mathbf{X} \in \mathbb{R}^n$ to a low-precision grid via a single affine transformation. Formally, for a target integer or floating point type with k-bit codes having dynamic range $[-Q_{\max}, Q_{\max}]$, each element is quantized as

$$\tilde{x}_i = \mathrm{round}\left(\frac{x_i}{s}\right), \quad s = \frac{\max(|\mathbf{X}|)}{Q_{\max}}, \tag{1}$$

where s is the per-tensor scaling factor and $\tilde{x}_i$ is stored as a k-bit integer or floating-point value.

Outliers are the primary cause of quantization error. To mitigate the impact of outliers, recent studies [11, 20, 44, 72] reduce the quantization granularity from the tensor or channel level to finer groups. This approach, known as group-wise

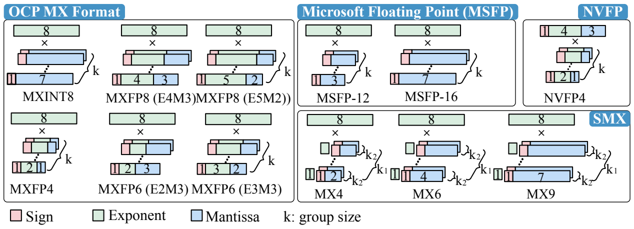


Figure 1. Microscaling data format.

quantization, partitions the tensor's weights into small, fixed-size blocks (e.g., 64 or 128 values). Each block is quantized independently with its own unique scaling factor, effectively isolating the impact of any outliers within that small region.

2.2 Microscaling Data Format

Microscaling (MX) is an emerging low-bit data format widely supported by existing hardware [51, 61, 67], and has been extensively studied and applied in recent works [9, 10, 18, 22, 36, 37, 40, 41, 48–50, 54, 59, 68, 70]. In this section, we introduce the standard MX format along with several of its variants. Notably, the MX format naturally embraces group-wise quantization.

Open Compute Project (OCP) Microscaling. Microscaling (MX) is a block floating-point format defined by the Open Compute Project (OCP)[56]. As shown in Fig. 1, k scalar elements share a common 8-bit scale factor. Unlike traditional group-wise quantization, where the scaling factor is FP16, the MX format restricts its scaling factor to the E8M0 format, a power-of-two representation with 8 exponent and 0 mantissa bits, making it particularly hardware-friendly for both quantization and dequantization processes.

The shared scale is derived from the block maximum $x_{\max} = \max_i |V_i|$. Following the OCP specification [56], the exponent of the shared scale is computed as $S = 2^{\lfloor \log_2(x_{\max}/P) \rfloor}$, where P is the largest power-of-two representable in the target format (e.g., $P = 4$ for FP4). Recent works propose alternatives such as using $S = 2^{\lceil \log_2(x_{\max}/M) \rceil}$ instead to reduce clipping [50], where M is the maximum representable value (e.g., $M = 6$ for FP4), or incorporating rounding strategies like $S = 2^{\lfloor \log_2(\text{Round}(x_{\max})/P) \rfloor}$ [68] to reduce systematic bias in scale selection. In this paper, we adopt the OCP-compliant floor-based method, and we will compare different scale calculations later.

The MX format quantization process can be simplified as a shift-and-rounding operation. For each element, its exponent is reduced by the shared exponent, which corresponds to a shift operation. Subsequently, the mantissa is rounded. The MX format dequantization process is seamlessly integrated into the General Matrix Multiplication (GEMM) operation in the latest modern GPUs [51]. Compared to conventional group-wise quantization, the dequantization process in MX format is more efficient and hardware-friendly.

Variants of the Microscaling Format. Several variants of the MX format exist, all sharing a common feature: a shared scaling factor [12, 13]. The concept of block floating-point (BFP), also known as Microsoft Floating Point (MSFP) [12], was introduced by the Brain Project. Fig. 1 illustrates the MSFP-12 and MSFP-16 formats, where the numbers 12 and 16 refer to the combined bit widths of the scalar element and the shared scaling factor.

Microsoft and Meta have proposed Shared Microexponents (denoted as SMX in this paper) [13], which is a novel 2-level shared MX format. The key distinction in this variant is that k_2 neighboring elements within share a 1-bit exponent, in addition to the 8-bit shared scaling factor by k_1 elements in a group. Typically, k_1 is 16 and k_2 is 2. The SMX family includes SMX4, SMX6, and SMX9, with differences in the mantissa bit width. The number in the SMX format name corresponds to the combined bit width of the sign, shared exponent, and mantissa.

Recently, NVIDIA introduced NVFP, replacing the E8M0 scaling factor with the FP8 (E4M3) scaling factor. While FP8 scaling is more precise than E8M0, it has a reduced range, as the 4-bit exponent cannot cover the range of FP16. To compensate for this reduced range, NVIDIA proposes a tensor-level scaling factor to adjust the original tensor's distribution, making the FP8 scale factors more practical. This adjustment helps reduce quantization error by enhancing the precision of the scaling factor. The 5th-generation tensor cores in NVIDIA's Blackwell architecture [51] support both MXFP4 and NVFP4.

Key Takeaway. Model quantization has evolved from coarse tensor-level schemes to fine-grained block-level formats, with Microscaling (MX) becoming the de facto hardware standard. MX achieves high throughput by exploiting shared power-of-two scaling and streamlined dequantization, and it has been widely adopted in commercial accelerators. However, this very reliance on a single shared scaling factor per block becomes a critical accuracy bottleneck at 4-bit precision, especially for LLM workloads where outliers dominate local dynamic ranges. Existing MX variants, e.g., MSFP, SMX, and NVFP, partially alleviate this issue but remain constrained by the same structural limitation, leading to either bit inefficiency or insufficient fidelity.

This gap motivates a deeper investigation into the design space of MX quantization, particularly exploring whether lightweight metadata augmentation can bridge the trade-off between bit efficiency and model accuracy. The next section analyzes the root causes of quantization error in MX formats and categorizes recent architectural optimizations, laying the groundwork for our proposed M^{2}XFP design.

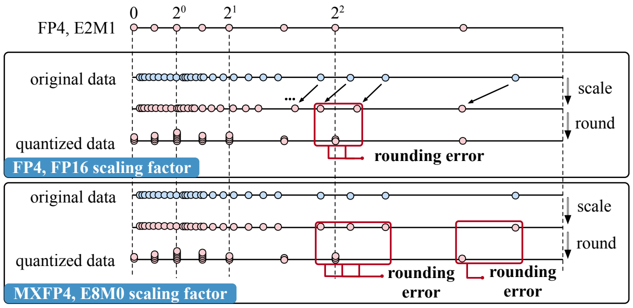


Figure 2. FP4 quantization: A comparison of FP16 and E8M0 scaling factors.

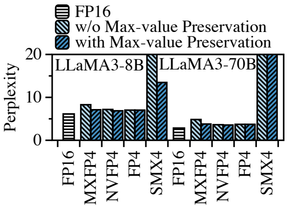


Figure 3. Perplexity of 4-bit quantization on LLaMA3, retaining the group-wise maximum in FP16 significantly enhances MXFP4.

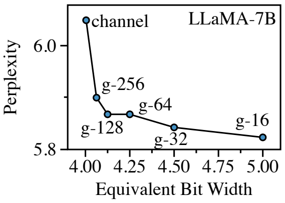


Figure 4. Perplexity decreases with increasing equivalent bit width (EBW), but the improvement diminishes beyond g-32 despite larger bit wdiths.

3 Motivation

In this section, we first analyze the root causes behind the significant quantization error observed in low-bit MX formats. We then summarize recent architectural optimizations designed to improve quantization performance in Sec. 3.2. Based on this, we categorize several optimization settings and evaluate them across different LLMs to identify the most effective configuration for MXFP.

3.1 Analysis of MX Quantization Error

Low-bit MX formats suffer from significant accuracy degradation because their shared power-of-two scaling *cannot* precisely align with block maximum [40]. As illustrated in Fig. 2, FP16-based scaling maps the maximum element of a group tightly to the FP4 maximum point, minimizing quantization error. In contrast, MX's E8M0 scaling only provides coarse power-of-two steps. When the group maximum falls between two exponent bins, the misalignment produces large rounding errors on the dominant value itself, which then propagates to the entire block.

We empirically validate this phenomenon by quantizing several LLMs with FP4, MXFP4, NVFP4, and SMX4, as shown in Fig. 3. Both MXFP4 and SMX4 exhibit pronounced perplexity degradation. SMX4 performs especially poorly due to the additional shared 1-bit exponent among neighboring elements, which amplifies errors when their magnitudes differ. Crucially, we find that simply preserving the maximum element of the block in FP16 precision drastically reduces MXFP4's perplexity, nearly matching FP4 and NVFP4. This experiment confirms that the mishandling of block maximum is the primary weakness of MX quantization.

3.2 A Taxonomy of Quantization Design Dimensions

To further identify promising solutions, we decompose recent architectural innovations into three design dimensions: the scaling factor, the data type, and metadata.

3.2.1 Scaling Factor: Converging Toward Group-Level E8M0/FP8.

The scaling factor determines how local dynamic ranges are represented. Early schemes adopted coarse per-tensor or per-channel scaling [15, 20], which are simple but highly sensitive to outliers. Subsequent designs refined granularity to group-level scaling, partitioning tensors into blocks (e.g., 32 or 64 elements), each with its own shared scale. This approach, now embodied in OCP's MX specification [56], isolates local outliers and has become the industry standard. In terms of numerical format, the field has largely converged on two options: (1) **E8M0** (power-of-two scaling): extremely hardware-friendly due to its shift-only implementation, but coarse resolution leads to misalignment with block maximum (Sec. 3.1). (2) **FP8** (E4M3): higher precision, as adopted in NVIDIA Blackwell [51], but with limited exponent range, requiring an additional tensor-level rescale for stability.

As illustrated in Fig. 4, our experiments show diminishing returns when simply reducing group size (e.g., from 32 to 16), while equivalent bit width (defined as per-element bits plus amortized scale bits) rises noticeably due to more scales per tensor, and accuracy gains quickly plateau.

In addition to scaling factor granularity, the data type of the scaling factor has also been explored in recent years to reduce storage overhead. Tbl. 1 summarizes recent designs adopting E8M0 or FP8 formats. Overall, recent hardware and system designs converge on E8M0 and FP8 for scaling factors, suggesting that this dimension offers little room for breakthrough improvements.

3.2.2 Data Type: Expressive but Hardware-Prohibitive.

Another line of work seeks to redesign the base data type to better match tensor distributions. This has led to a rich design space of specialized numerical formats, including custom types like Flint in ANT [29], non-uniform types in M-ANT [32], and the selectable 'dialects' in BlockDialect [34]. While these methods provide strong representational flexibility, they face two fundamental limitations: (1) Low efficiency for dynamic tensors. Most designs target weights that are static and can afford offline type selection. Applying them to activations, which are generated dynamically during inference, requires costly runtime decisions. (2) Decoder

Table 1. The features of DNN accelerators across different scaling factors, data types, and metadata designs are summarized. In the Scaling Factor column, 'Granularity' refers to the quantization granularity. In the Data Type column, a dash ('-') indicates that the architecture supports only a single data type.

Architecture	Scaling Factor		Data Type		Metadata	
	Granularity	Format	Granularity	Format	Granularity	Content
OliVe [27]	Tensor/Channel	FP16	Tensor/Channel	INT4, Flint4	Pair	Outlier-victim pair
ANT [29]	Tensor/Channel	FP16	Tensor/Channel	INT4, Flint4, PoT4	Tensor/Channel	2-bit index
Tender [39]	Channel	FP16	-	INT4	Channel	12-bit index data
MANT [32]	Group-64	FP16	Group-64	16 data types	Group-64	8-bit coefficient a
BitMod [5]	Group-128	FP16	Group-128	FP4+special value	Group-128	2-bit index
MXFP [56]	Group-32	E8M0	-	FP4	-	-
SMX [13]	Group-16	E8M0	-	INT3 (SMX4)	Pair	1-bit exponent
NVFP [51]	Group-16	FP8 (E4M3)	-	FP4	-	-
MicroScopiQ [54]	Group-128	E8M0	Group-128	FP4+INT4	Block	24-bit permutation list, 16-bit identifier, and 8-bit MXScale, depends on μblock
BBAL [31]	Group-32	E5M0	-	INT3	Element	1-bit flag
BlockDialect [34]	Group-32	E5M0	Group-32	16 dialects	Group-32	4-bit index
MX+ [40]	Group-32	E8M0	Group-32	FP4	Group-32	5-bit index and 3-bit reserved

complexity. Supporting multiple custom data types demands numerous decoders and format converters in hardware, significantly inflating area, latency, and energy.

Thus, although novel data types are intellectually appealing, they pose significant challenges for deployment in low-latency, high-throughput accelerators, particularly for activation quantization, where runtime overhead is prohibitive.

3.2.3 Metadata: A Flexible Yet Underutilized Design Axis.

Beyond scaling factors and data types, recent accelerators have begun exploring metadata [27, 39, 54], which apply small auxiliary bits that encode side information. Metadata can enhance accuracy without fundamentally altering the base data path, making it a lightweight yet versatile design axis. We identify three representative roles: (1) *Critical-value precision allocation.* Approaches such as OliVe [27] use "outlier-victim pairs" to assign extra bits to extreme values, while MicroScopiQ [54] allocates different bit-widths to inlier and outlier blocks. (2) *Range refinement.* SMX [13] attaches a 1-bit secondary exponent to value pairs, and BBAL [31] uses a 1-bit flag to shift exponents, both aiming to expand local dynamic range. (3) *Format or structure control.* ANT [29] and BlockDialect [34] employ metadata as indices for selecting numerical types, while Tender [39] uses metadata to store indices to hint extra operations.

Despite their promise, existing metadata schemes remain fragmented and bit-inefficient. First, many focus on a single error source (e.g., outliers) but fail to address systemic quantization loss from block maximum. Second, others improve accuracy but at excessive control overhead (e.g., MicroScopiQ introduces 40+ bits of structural metadata per block). Last but not least, most works lack a principled framework for where and how to allocate metadata, especially for activations where both latency and hardware overhead are critical.

3.3 Takeaway: Metadata as the Key Lever

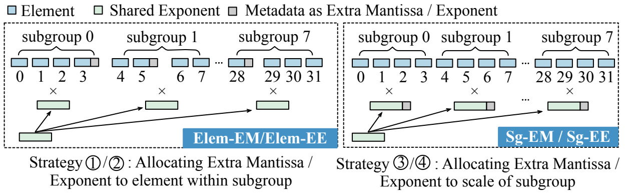
The analysis above highlights that scaling factor design has already converged, and data-type innovations face prohibitive hardware overheads for dynamic tensors. In contrast, metadata offers a flexible and underexplored design axis. Our findings indicate that properly allocating a small number of metadata bits (e.g., to preserve or enhance critical elements) can directly target the dominant error source in MX quantization. *In summary, metadata remains the most underutilized yet most promising lever to close the gap between MX's hardware efficiency and the accuracy demands of LLM quantization.*

4 M^{2}XFP Analysis and Design

In this section, we present the analysis and design of M^{2}XFP, driven by an extensive encoding design space exploration (DSE) of metadata strategies. We first establish a unified framework for systematically reasoning about subgroup-level metadata allocation, then analyze Pareto trade-offs between accuracy and bit efficiency. Based on these insights, we derive a hybrid design tailored to the distinct characteristics of weights and activations, and finally detail the hardware-friendly quantization and encoding process.

4.1 Framework for Design Space Exploration

As discussed in Sec. 3.2, metadata is the most flexible axis for improving MX quantization. To capture its full potential, we introduce a subgroup-centric framework that generalizes existing MX variants into a common design space. Specifically, a group of size k is divided into N contiguous subgroups, enabling localized metadata allocation. For instance, SMX can be interpreted as a group of 16 with subgroups of 2, each augmented by a 1-bit local exponent.



Strategy ①/② : Allocating Extra Mantissa / Exponent to element within subgroup

Strategy ③/④ : Allocating Extra Mantissa / Exponent to scale of subgroup

Figure 5. MX format with subgroup-level metadata. The figure contrasts two strategies for allocating extra bits: (1) Elem-EM/EE extends individual elements within subgroups; (2) Sg-EM/EE augments the subgroup scale.

This unified abstraction allows us to organize metadata strategies along two orthogonal axes, as illustrated in Fig. 5: (1) Precision vs. Range Enhancement: metadata can extend mantissa bits to refine precision or adjust exponent bits to expand dynamic range. (2) Element- vs. Subgroup-level Application: metadata can be applied to the most critical element within a subgroup or to the shared subgroup scale.

Focusing on hardware-feasible operations, we restrict metadata to mantissa or exponent augmentation, yielding four representative strategies:

- Elem-EM (Element-level Extra Mantissa): use metadata as extra mantissa bits to a single element within each subgroup;
- Elem-EE (Element-level Extra Exponent): use metadata to provide an exponent offset to a single element;
- Sg-EM (Subgroup-level Extra Mantissa): enhance the precision of the subgroup scale factor, conceptually similar to NVFP4 but at finer granularity;
- Sg-EE (Subgroup-level Extra Exponent): encode a subgroup's exponent to improve local dynamic range, analogous to the SMX concept.

While metadata can be applied in many ways, its impact fundamentally depends on how it interacts with the group's shared scaling factor (shared scale for short). Some strategies refine local precision without altering the global scale, whereas others allow metadata to reshape the scale itself. To clearly capture this distinction, we further define two allocation modes: (i) **fixed shared scale**: metadata locally refines elements or subgroups while leaving the group's shared scale unchanged, acting as a lightweight precision or range enhancement, and (ii) **adaptive shared scale**: metadata also influences the choice of shared scale, enabling adaptive selection of scaling factors that minimize quantization error. For instance, the fixed mode derives the shared scale strictly from the block maximum (i.e., $E = \lfloor \log_2(\text{amax}/P) \rfloor$), whereas the adaptive mode performs an MSE-based search over candidate exponents (e.g., $E, E \pm 1$) to jointly optimize the scale and metadata.

These two modes provide complementary perspectives. Fixed shared scale isolates the immediate effect of metadata bits with negligible complexity, while adaptive shared scale leverages metadata to globally rebalance quantization error. Taken together, they span the full metadata design space for MX quantization and establish a principled basis for the Pareto analysis in Sec. 4.2.

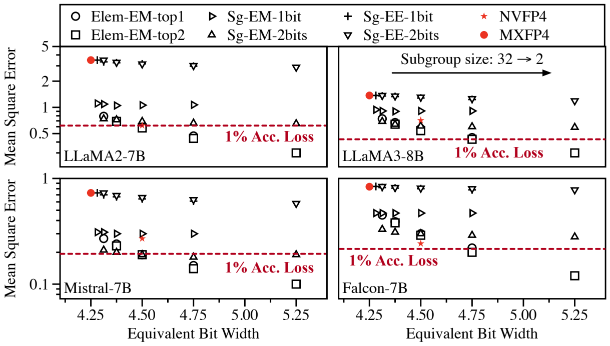


Figure 6. Encoding design space exploration of Elem-EM, Sg-EM, and Sg-EE under fixed shared scale. Elem-EM achieves the lowest MSE at 4.5-4.75 EBW.

4.2 Pareto-Optimal Analysis of Metadata Strategies

Building on the unified framework, we now evaluate the effectiveness of different metadata strategies under both fixed shared scale and adaptive shared scale modes. This analysis provides a principled way to characterize the trade-off between accuracy and bit efficiency, and identifies the Pareto-optimal configurations that best close the gap between MXFP4 efficiency and FP16 fidelity.

4.2.1 Evaluation Method. We quantify accuracy using mean squared error (MSE) relative to FP16. Specifically, the MSE is computed between the outputs of the quantized model, where both weights and activations are quantized, and those of the FP16 baseline, using the same input text. We also normalize the storage cost using equivalent bit width (EBW), which incorporates element bits, shared scale, and metadata overhead, as shown in Eq. 2.

$$\text{EBW} = \frac{(k \times B_{\text{elem}}) + B_{\text{meta}} + B_{\text{scale}}}{k} = B_{\text{elem}} + \frac{B_{\text{meta}} + B_{\text{scale}}}{k} \tag{2}$$

Here, k is the group size, B_{elem} the base data bit-width (e.g., 4 for FP4), and B_{meta} the metadata bits for the group. This metric allows fair comparisons across strategies by measuring the effective precision delivered per bit.

To ensure fairness, the group size is fixed at 32, while subgroup size is varied to adjust EBW. For element-level strategies (Elem-EM), we assign 2 bits of mantissa metadata per element and evaluate both top-1 and top-2 allocations, but as Sec. 3.1 shows that exponent offsets cannot alleviate block-maximum errors, we omit Elem-EE. Here, top-1/top-2 denote the largest one or two values (by absolute magnitude) within each subgroup. For subgroup-level strategies (Sg-EM/EE), we allocate 1–2 bits of mantissa or exponent

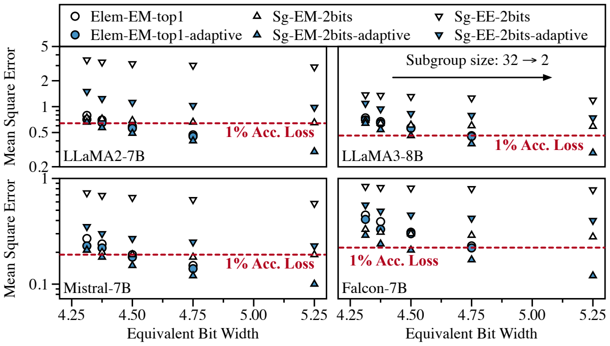


Figure 7. Impact of adaptive shared scale on Elem-EM and Sg-EM. Optimizing rounding direction enables Sg-EM-search to outperform Elem-EM-search at 4.5-4.75 EBW.

metadata to refine the shared scale. These configurations form the basis of the design space explored in Figs. 5–6 across LLaMA2-7B, LLaMA3-8B, Falcon-7B, and Mistral-7B. Experiments fix group size at 32 while varying subgroups to control EBW.

4.2.2 Fixed Shared Scale Result (Fig. 6). Under a fixed shared scale, Elem-EM consistently dominates, achieving the lowest MSE in the 4.5–4.75 EBW range across all models. Top-1 and top-2 assignments yield nearly identical results, indicating that capturing only the maximum element per subgroup suffices. Sg-EM becomes competitive only at lower EBW (≤ 4.375), while Sg-EE shows negligible or marginal improvements over MXFP4 regardless of bit allocation. These results confirm that subgroup-level range expansion cannot address the dominant error source—block maximum misalignment. Notably, the red dashed line in Fig. 6 marks the 1% accuracy-loss threshold: Elem-EM reaches this target with 4.6 bits on LLaMA2-7B, Falcon-7B, and Mistral-7B, and 4.75 bits on LLaMA3-8B, whereas Sg-EM requires ≥ 5.25 bits, and Sg-EE fails to meet the threshold entirely.

4.2.3 Adatpive Shared Scale Result (Fig. 7). When adaptive shared scale is enabled, the Pareto frontier shifts. By adaptively selecting the shared scale in conjunction with metadata, Sg-EM-2bit surpasses Elem-EM in the critical 4.5 to 4.75 EBW region, achieving lower MSE with minimal overhead. Elem-EM still performs strongly, but no longer dominates. Sg-EE also benefits from adaptive shared scale, yet remains far less efficient than either Elem-EM or Sg-EM. Overall, the performance ranking becomes: Sg-EM-adaptive > Elem-EM-adaptive > Elem-EM > Sg-EM > Sg-EE-adaptive > Sg-EE.

4.2.4 Key Takeaway. This Pareto analysis reveals a crucial asymmetry: element-level metadata is superior under a fixed shared scale due to its ability to capture dominant outliers without global adjustments, while subgroup-level metadata becomes preferable once an adaptive shared scale is incorporated. It leverages shared scale optimization to rebalance error across the block, which is quantitatively confirmed by the consistent MSE reduction for Sg-EM and Sg-EE in Fig. 7 (blue markers). These complementary behaviors directly motivate the hybrid M^{2}XFP design in Sec. 4.3, which assigns Sg-EM to static weights and Elem-EM to dynamic activations.

4.3 M^{2}XFP Design: A Hybrid Strategy

The Pareto analysis highlights an important asymmetry: element-level metadata (Elem-EM) is the most effective under a fixed shared scale, while subgroup-level metadata (Sg-EM) becomes superior once an adaptive shared scale is incorporated. This observation naturally suggests that a single uniform strategy cannot simultaneously optimize for both weights and activations, which differ in their statistical properties and quantization requirements.

Weights vs. Activations. Weights are static and can be quantized offline, allowing sufficient time for adaptive optimization to identify the optimal subgroup-level refinement. In contrast, activations are generated dynamically during inference, where latency constraints demand lightweight, deterministic quantization. This constraint forces activations to adopt a bit-efficient strategy under the fixed shared scale mode. As a result, weights benefit more from subgroup-level metadata with adaptive shared scale (Sg-EM-2bits-adaptive), while activations benefit from element-level metadata that directly preserves the most influential values (Elem-EM-top1).

Hybrid Strategy. M^{2}XFP adopts a hybrid design that assigns: (1) Weights apply Sg-EM-2bit format, enabling fine-grained subgroup-scale refinement through offline adaptive optimization, thereby improving bit efficiency while maintaining fidelity. (2) Activations apply Elem-EM-top1 format, which captures outliers within each subgroup in real time with minimal routing overhead.

This division of labor leverages the strengths of both strategies while respecting the distinct hardware and workload constraints of weights and activations. Since Elem-EM-top1 and top2 show nearly identical accuracy, we adopt top1 for its simpler implementation and lower metadata routing complexity. As a result, M^{2}XFP achieves near-FP16 accuracy at an effective precision of 4.5 bits. This hybrid strategy establishes a balanced trade-off between hardware cost, quantization fidelity, and runtime efficiency, providing the foundation for hardware-friendly quantization and encoding.

4.4 Quantization and Encoding Process

In this section, we introduce the quantization and encoding process for activation with Elem-EM and weight with Sg-EM.

4.4.1 Activation Quantization with Elem-EM. The online quantization process for M^{2}XFP is designed to be efficient, as detailed in Alg. 1 and illustrated in Fig. 8, with subgroup size 4 as an example. For each incoming activation

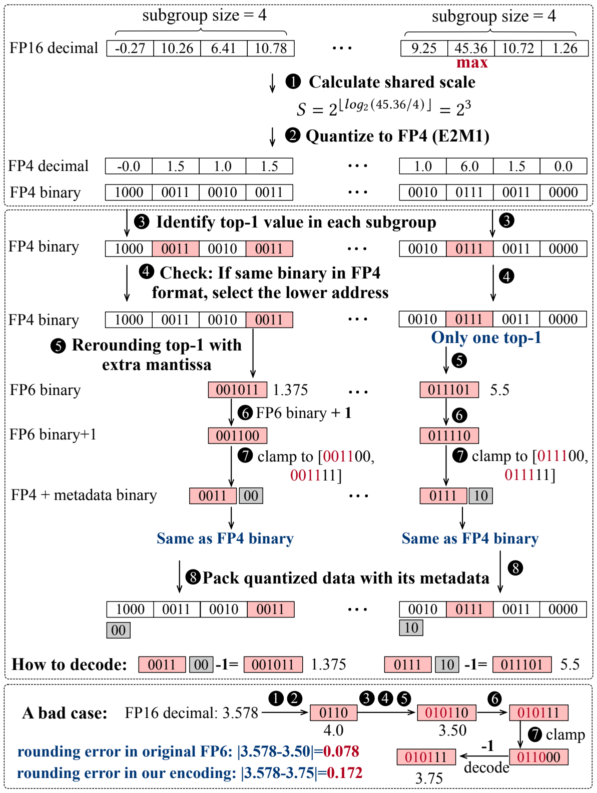


Figure 8. Quantization process to M^{2}XFP data format.

group, the group-level maximum absolute value is first determined to compute the shared scale factor (Step ❶). All elements in the group are then quantized into a baseline 4-bit E2M1 representation (Step ❷).

Given that the top1 maximum value within each subgroup must also be identified during decoding, we perform the selection in the 4-bit quantized format (FP4-E2M1) (Step ❸). In cases where multiple elements share the same maximum quantized value (i.e., different in FP16 but identical in FP4), M^{2}XFP selects the element with the lowest memory address as the unique identification (Step ❹). The original high-precision value of that identified top1 is then quantized to generate an FP6 value (Step ❺).

Encoding Strategy for FP6 Values. We identified a critical issue when directly replacing the FP4 value of the top1 element with its FP6 value: since the high 4 bits of the FP6 value are not necessarily identical to the original FP4 value, the top1 element may no longer remain the maximum after this replacement. To address this, we developed an improved encoding strategy.

Since quantization maps values to their nearest low-bit representation, a value quantized to a specific FP4 value x has only five potential corresponding values when quantized to FP6. For example, if a value is quantized to 4 in FP4, it must fall within the range (3.5, 5]. Thus, it can only be quantized to one of 5 possible FP6 values: 3.5, 3.75, 4, 4.5, or 5.

Algorithm 1 The M^{2}XFP Quantization Process

```
Input: High-precision data group $\mathbf{X}_{\text{FP16}}$ of size $k$.
Output: Final MXFP4 $\mathbf{X}_{\text{FP4}}$ and metadata $\mathbf{X}_{\text{meta}}$.
❶ Step 1: Calculate Shared Scale
$x_{max} \leftarrow$ find maximum absolute value in $\mathbf{X}_{\text{FP16}}$
$S \leftarrow 2^{\lfloor \log_2(\text{amax/FP4\_max\_pow2}) \rfloor}$
❷ Step 2: Quantize to FP4 (E2M1)
$\mathbf{X}_{\text{FP4}} \leftarrow \text{quantize\_to\_E2M1}(\mathbf{X}_{\text{FP16}}, S)$
For each subgroup:
for each subgroup $\mathbf{x}_{\text{FP4}}$ in $\mathbf{X}_{\text{FP4}}$ do
❸❹ Step 3 & 4: Identify the top-1 in subgroup, resolving duplicates by selecting the lowest index
    $\mathbf{x}_{\text{FP4\_abs}} \leftarrow abs(\mathbf{x}_{\text{FP4}})$
    $v_{max} \leftarrow \max(\mathbf{x}_{\text{FP4\_abs}})$    ▷ Get max in subgroup
    $C_{idx} \leftarrow \{j \mid |\mathbf{x}_{\text{FP4\_abs}}[j]| = v_{max}\}$ ▷ Get all candidate
    $idx \leftarrow \min(C_{idx})$    ▷ Select lowest index
❺ Step 5: Quantize top-1 to FP6(E2M3)
    $x_{\text{orig}} \leftarrow \mathbf{X}_{\text{FP16}}[\text{idx}_{\text{top1}}]$    ▷ Original value
    $x_{\text{FP6}} \leftarrow \text{Quantize}(x_{\text{orig}}, \text{E2M3}, S)$
❻ Step 6: Add bias for encoding
    fp6_bits $\leftarrow$ FloatToBits($|x_{\text{FP6}}|$)    ▷ 6-bit information
    fp4_bits $\leftarrow$ FloatToBits($|\mathbf{x}_{\text{FP4}}[\text{idx}_{\text{top1}}]|$)    ▷ 4-bit information
    encoded $\leftarrow$ fp6_bits + 1    ▷ Add bias in binary
❼ Step 7: Clamp to keep FP6 high 4 bits same as FP4
    range_min $\leftarrow$ fp4_bits$\underline{00}$    ▷ The minimum binary value with the same high 4 bits
    range_max $\leftarrow$ fp4_bits$\underline{11}$    ▷ The maximum binary value with the same high 4 bits
    clamp $\leftarrow$ Clamp(encoded, range_min, range_max)
    $\mathbf{x}_{\text{meta}} \leftarrow$ Get2BitsLow(clamp)    ▷ Extract 2-bit metadata
❽ Step 8: Pack quantized data with metadata
    Append $\mathbf{x}_{\text{FP4}}$ to $\mathbf{X}_{\text{FP4}}$ and $\mathbf{x}_{\text{meta}}$ to $\mathbf{X}_{\text{meta}}$
end for
return $\mathbf{X}_{\text{FP4}}$, $\mathbf{X}_{\text{meta}}$
```

Based on this observation, we can represent the FP6 value using a bias relative to the FP4 value. Centered at 4.0, the theoretical bias range is -2, -1, 0, 1, 2, corresponding to the FP6 candidates 3.5, 3.75, 4.0, 4.5, 5.0. However, for data alignment purposes, we clamp this bias to -1, 0, 1, 2, which introduces only minor rounding errors (in our example, rounding error occur only when a value is greater than 3.5 but less than 3.625). We give a case to indicate such a rounding error at the bottom of Fig. 8. The additional rounding error introduced by our method has a negligible impact. Perplexity results show that the maximum deviation between results on common large language models with and without this rounding error is only 0.02, indicating a minimal effect on performance.

Encoding Procedure. To implement this encoding, we first add a bias of 1 to the FP6 binary value, then clamp

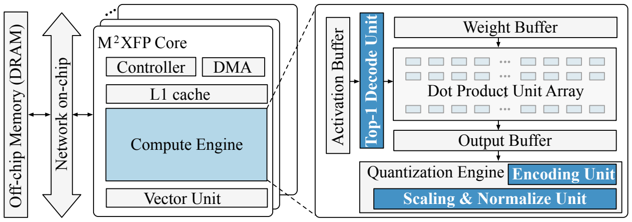


Figure 9. Architecture Overview.

the result to ensure the high bits remain consistent with the original FP4 value. The lower 2 bits serve as metadata representing the additional mantissa precision.(Step ❻ & ❼)

Finally, the definitive 4-bit values for the subgroup and the final 2-bit extra mantissa metadata are packed to form the final M^{2}XFP representation (Step ❽). This packed data is organized in a hardware-friendly memory layout where the metadata for all subgroups is gathered into a single, contiguous block, followed by the 4-bit data elements.

4.4.2 Weight Quantization with Sg-EM.

Weight quantization is simpler than activation quantization. Each subgroup uses a 2-bit extra mantissa to refine the shared scale $S = 2^E$, giving candidates $\{1.0, 1.25, 1.5, 1.75\} \cdot S$. The search space for each subgroup is:

$$\mathcal{S} = \{(1 + \tfrac{k}{4}) \cdot 2^E \mid k \in \{0, 1, 2, 3\}\} \tag{3}$$

When the adaptive shared scale is enabled, the whole group scale can be adjusted with a bias $b \in \{-1, 0, 1\}$ applied to the exponent. Notably, this bias requires no additional storage bits as it can be directly absorbed into the stored scale value. The optimal parameters are chosen via hierarchical MSE minimization:

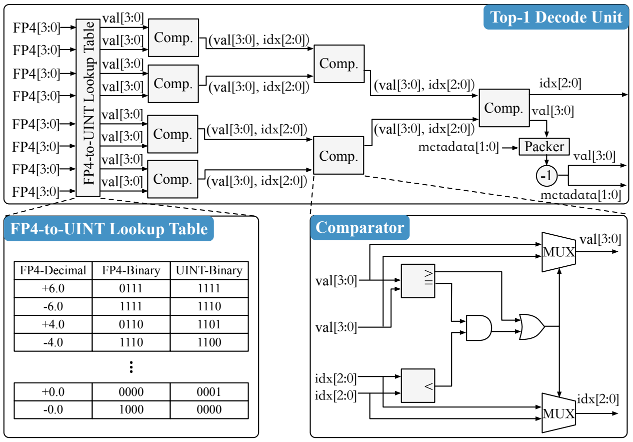
$$b^*, \{k_i^*\} = \arg \min_{b \in \{-1,0,1\}} \sum_{i \in \mathrm{sg}} \left\| \hat{W}_{k_i^*, b} - W_i \right\|_2^2 \tag{4}$$

where $k_i^* = \arg\min_{k \in \{0,1,2,3\}} \left\| \hat{W}_{k,b} - W_i \right\|_2^2$, W_i represents the original weights in subgroup i, and $\hat{W}_{k,b}$ denotes the weights quantized and dequantized using scale $(1 + \frac{k}{4}) \cdot 2^{E+b}$.The optimization first finds the optimal mantissa refinement k for each subgroup given a bias b, then selects the best group-level exponent bias b.

Concretely, for an 8-element subgroup, exploring all exponent and mantissa combinations requires evaluating up to 12 candidate scales, amounting to roughly $3 \times 8 \times 12 = 288$ FLOPs plus comparisons per subgroup. This overhead is acceptable for offline weight calibration but prohibitive for runtime activation quantization.

5 Architecture

We now describe the architectural extensions required to efficiently support M^{2}XFP. The design goal is to retain the high throughput and memory efficiency of systolic-array accelerators while introducing minimal logic to handle metadata and outlier refinement.Fig. 9 illustrates the overall compute core, which integrates M^{2}XFP into a conventional systolic array pipeline with lightweight modifications.



FP4-Decimal	FP4-Binary	UINT-Binary
+6.0	0111	1111
-6.0	1111	1110
+4.0	0110	1101
-4.0	1110	1100
⋮		
+0.0	0000	0001
-0.0	1000	0000

Figure 10. Microarchitecture of the Top-1 Decode Unit, consisting of an FP4-to-UINT lookup table, a three-level comparator tree, and supporting logic.

5.1 Architecture Overview

The baseline core includes standard components such as L1 cache, vector units, and on-chip buffers. M^{2}XFP introduces three specialized units to support its hybrid quantization format:(1) **Top-1 Decode Unit** identifies the maximum element in each subgroup and forwards its metadata to the processing elements (PEs). (2) **Augmented PE Tile** executes FP4 × FP4 multiply–accumulate (MAC) operations while incorporating additional mantissa and subgroup-scale refinements. (3) **Quantization Engine** performs online Elem-EM quantization of activations following the procedure in Sec. 4.4. These units are strategically inserted between the input buffers and the systolic array to ensure seamless integration with existing GEMM pipelines.

5.2 Memory Organization

M^{2}XFP preserves memory alignment by storing elements, scaling factors, and metadata in fixed-length fields. Each group consists of three separately organized streams: a 128-bit block of packed 4-bit elements, an 8-bit shared scale, and 8-bit metadata. These components are stored in contiguous memory regions, where elements in one continuous space, scale factors in another, and metadata in a third. This separation not only guarantees alignment but also simplifies indexing and parallel access. Upon loading into on-chip buffers, a dispatch unit delivers the scale factor, metadata, and elements to the decode unit and PE array. This layout ensures that metadata handling introduces no fragmentation or misalignment overhead compared to baseline MXFP.

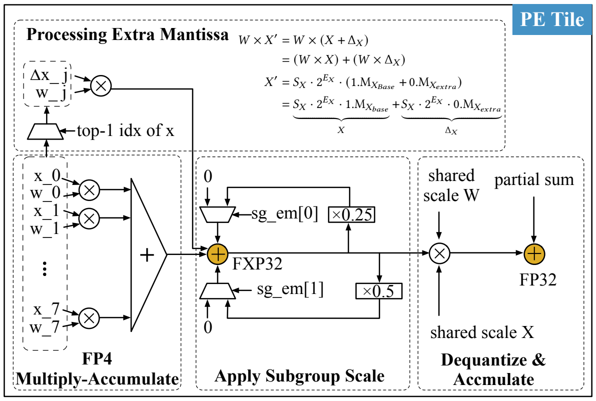

PE Tile
Processing Extra Mantissa
$W \times X' = W \times (X + \Delta_X)$
$= (W \times X) + (W \times \Delta_X)$
$X' = S_X \cdot 2^{E_X} \cdot (1.M_{X_{Base}} + 0.M_{X_{extra}})$
$= \underbrace{S_X \cdot 2^{E_X} \cdot 1.M_{X_{base}}}_{X} + \underbrace{S_X \cdot 2^{E_X} \cdot 0.M_{X_{extra}}}_{\Delta_X}$
Δx_j
w_j
top-1 idx of x
x_0
w_0
x_1
w_1
x_7
w_7
+
FP4
Multiply-Accumulate
0
sg_em[0]
×0.25
FXP32
sg_em[1]
×0.5
0
Apply Subgroup Scale
shared scale W
partial sum
FP32
shared scale X
Dequantize &
Acmculate

Figure 11. Microarchitecture of the processing element.

5.3 Decode Unit

The decode unit preprocesses input subgroups before computation. As shown in Fig. 10, each FP4 element is first mapped to an unsigned integer using a compact 16-entry lookup table, enabling monotonic comparisons. A three-level comparator tree then identifies the unique top-1 element per subgroup. In case of ties, the lowest index is chosen, ensuring deterministic results.

The selected index and metadata are then packed and forwarded to the PE array. This logic is lightweight, comprising only a small LUT, comparators, and a multiplexer, yet crucial for enabling the element-level refinement in M^{2}XFP without disrupting the systolic pipeline.

5.4 M^{2}XFP Processing Element

The PE tile is the key microarchitectural extension of M^{2}XFP, as is shown in Fig. 11. Each PE integrates a baseline FP4-FP4 MAC datapath, augmented with logic for metadata refinement.

Baseline FP4 MAC. The main datapath implements a conventional FP4 × FP4 MAC pipeline that processes the majority of subgroup elements. Eight parallel multipliers and an adder tree process subgroup elements using FP4 encodings, producing partial sums stored in a 32-bit fixed-point register. This provides sufficient dynamic range while facilitating efficient downstream scaling.

Extra Mantissa Processing. To efficiently support elements with extended mantissa, we exploit the distributive property of multiplication. Given an element X' with a 2-bit mantissa extension, it can be decomposed as $X' = X + \Delta X$, where X is the FP4 baseline value and ΔX is a small correction term. Accordingly, the product expands to $W \times X' = W \times X + W \times \Delta X$. The term $W \times X$ is handled by the standard FP4 MAC pipeline, while the correction term $W \times \Delta X$ is computed in a lightweight auxiliary MAC unit and then accumulated with the baseline result. As illustrated in Fig. 11, the baseline component is represented as $S_X \cdot 2^{E_X} \cdot 1.M_{X_{base}}$, whereas ΔX corresponds to $S_X \cdot 2^{E_X} \cdot 0.M_{X_{meta}}$. The hidden bit of ΔX is set to zero, preserving compatibility with FP4 hardware and avoiding datapath disruption.

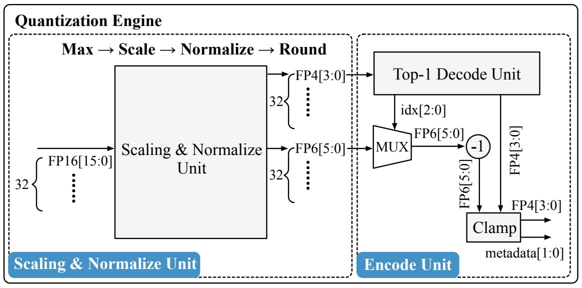


Figure 12. The quantization engine that contains scaling & normalization unit for quantization and encode unit to pack data to M^{2}XFP format.

Subgroup Scale Refinement. After intra-subgroup accumulation, each partial sum P_i is adjusted by a subgroup-level scale determined by its 2-bit Subgroup Extra Mantissa (sg_em). The sg_em encodes fractional mantissa extensions corresponding to multipliers of 1.0, 1.25, 1.5, and 1.75 for codes 00, 01, 10, and 11, respectively. These scaling factors can be efficiently realized using lightweight shift-and-add operations: $0.25P$ corresponds to a 2-bit right shift, $0.5P$ corresponds to a 1-bit right shift, and $0.75P$ is implemented by combining them ($0.5P + 0.25P$). As the fixed-point datapath already provides sufficient range, no costly multipliers are needed, and the process incurs only minor hardware overhead. The entire scaling procedure is illustrated in Fig. 11.

Dequantize and Accumulate. Finally, the scaled partial sums are dequantized into FP32 and accumulated across all subgroups before being written into the output buffer. The final group output is obtained by summing all subgroup results and applying the shared scale. For MX formats with an E8M0 shared scale, this dequantization is particularly lightweight: instead of full floating-point multiplications, it reduces to simple exponent alignment.

5.5 Quantization Engine.

As shown in Fig. 12, the quantization engine is a two-stage pipeline responsible for online encoding of activations. The first stage computes the group-level scale and generates FP4/FP6 candidates; the second stage identifies the top-1 element per subgroup, applies the bias-clamp encoding, and packs the resulting FP4 data with 2-bit metadata. The entire process is deterministic and streaming-friendly, enabling real-time quantization without stalling the systolic array.

6 Evaluation

6.1 Experimental Setup

Models and Benchmarks. We evaluate our M^{2}XFP across diverse workloads to demonstrate its generality. For large language models (LLMs), we test on LLaMA-2 [63] (7B), LLaMA-3 [16] (8B, 70B), OPT [71] (6.7B), Mistral [35] (7B), and Falcon [1] (7B), covering 7B-70B parameters. LLM benchmarks include Wikitext v2 and common sense QA tasks such as Arc-challenge, Arc-easy, HellaSwag, PIQA, WinoGrande, and BoolQ [4, 6, 7, 57, 69]. To further show robustness on reasoning, we evaluate reasoning-oriented models like DeepSeek-R1-Distill-Qwen [14] (1.5B, 7B) on AIME, MATH-500, GSM8K, GPQA-Diamond, and LiveCodeBench [8, 33, 42, 55, 65].

Algorithm Implementation. We implement the M^{2}XFP quantization framework in PyTorch [52], enabling precise modeling of both M^{2}XFP and baseline formats. Evaluation is conducted using lm-evaluation-harness [21]. Our MXFP4 baseline follows the OCP standard with group size 32; NVFP4 adopts group size 16; SMX also uses group size 16 with subgroup size 2. For M^{2}XFP, we configure a shared E8M0 scaling factor with group size 32 and subgroup size 8. This configuration is empirically validated in Sec. 4.2 as a near-Pareto-optimal trade-off between granularity and overhead, while matching the group-size choices of existing MX-capable hardware [51, 61].

Algorithm Baselines. We evaluate M^{2}XFP against MXFP4, SMX, and NVFP4, the quantization formats supported by existing hardware [51, 61, 67]. We further examine the benefits of enhancing NVFP4 with our proposed metadata augmentation.

Accelerator Implementation. We extend the open-source, cycle-level simulator DNNWeaver [60] to model our accelerator. The augmented components, decode unit, processing elements (PEs), and quantization engine, are implemented in Verilog and synthesized using Synopsys Design Compiler with the TSMC 28 nm standard cell library at 500 MHz, providing power and area estimates. On-chip buffer power and area are modeled with CACTI v7 [3].

Accelerator Baselines. We evaluate the performance and energy of M^{2}XFP against representative accelerator baselines. Our primary comparison is with MicroScopiQ, a state-of-the-art (SOTA) MX-based accelerator that partitions weights into inlier and outlier blocks, applying hybrid MX quantization to weights and MXINT to activation. To a comprehensive evaluation, we adapt non-MX accelerators (ANT, M-ANT, OliVe) to support fine-grained MX quantization, denoted MX-ANT, MX-M-ANT, and MX-OliVe. We also include BlockDialect, an SOTA algorithm-architecture co-design approach, with perplexity in Sec. 6.2.

For fairness, all accelerators are configured with 32×32 PEs supporting 4-bit multiplications, ensuring differences arise from architectural and algorithmic design.

6.2 Large Language Model Evaluation

Compared to Existing Data Types. We first evaluate accuracy on LLMs, with results summarized in Tbl. 2, comparing M^{2}XFP against several hardware-supported data types.

Table 2. Zero-shot evaluation results on five benchmarks: Arc-e (Arc-Easy), Arc-c (Arc-Challenge), Hella. (HellaSwag), PiQA, and Wino. (Winogrande). Group / subgroup sizes — MXFP4: 32 / 32, SMX: 16 / 2, M^{2}XFP: 32 / 8.

Method	Arc-e	Arc-c	Hella.	PiQA	Wino.	BoolQ	Avg.
	LLaMA2-7B						
FP16	74.58	46.25	75.99	79.11	69.06	77.71	70.45
SMX4	26.43	27.05	26.13	49.40	49.80	38.93	36.29
MXFP4	66.84	41.47	70.49	76.61	64.01	72.51	65.32
NVFP4	73.11	**44.88**	74.62	**78.13**	67.88	74.22	68.81
M^{2}XFP	**73.32**	44.37	**74.64**	77.58	**68.27**	**76.97**	**69.19**
	LLaMA3-8B						
FP16	77.49	53.33	79.15	80.85	72.53	81.28	74.11
SMX4	25.00	27.13	26.03	50.18	48.86	40.67	36.31
MXFP4	71.42	46.08	73.53	77.48	68.19	72.84	68.26
NVFP4	72.98	48.55	76.08	78.40	**72.14**	75.96	70.69
M^{2}XFP	**74.58**	**49.57**	**77.23**	**79.54**	70.96	**79.20**	**71.85**
	Mistral-7B-v0.3						
FP16	78.24	52.13	80.46	82.26	73.8	82.14	74.84
SMX4	26.39	27.22	25.69	49.18	49.33	40.06	36.31
MXFP4	74.03	46.67	75.87	78.94	69.06	73.49	69.68
NVFP4	76.47	49.23	78.13	**81.56**	70.64	78.07	72.35
M^{2}XFP	**76.64**	**50.85**	**79.76**	80.74	**71.27**	**82.45**	**73.62**

Table 3. Perplexity on the Wikitext dataset for M^{2}XFP and baseline accelerators (lower is better).

Method	LLaMA2	LLaMA3	LLaMA3	OPT	Mistral	Falcon
	7B	8B	70B	6.7B	7B	7B
FP16	5.47	6.14	2.85	10.86	5.32	6.59
MXFP4	7.15	8.30	4.84	19.21	6.56	7.59
MX-ANT	6.30	8.22	4.65	12.76	6.04	7.35
MX-M-ANT	6.12	7.83	4.54	12.45	5.89	7.32
MX-OliVe	7.46	11.33	6.84	36.80	6.77	8.40
MicroScopiQ	6.24	8.33	4.75	12.65	6.00	7.45
BlockDialect	5.84	7.05	3.76	**11.31**	5.65	6.94
M^{2}XFP	**5.77**	**6.84**	**3.56**	11.34	**5.58**	**6.88**

SMX4 shows severe degradation, with average accuracy loss exceeding 30% on 7B/8B models, making it impractical for 4-bit weight-activation quantization. MXFP4 is more stable, with an average loss of 5.38% on 7B/8B.

M^{2}XFP consistently outperforms MXFP4 across all model scales. Specifically, on 7B/8B models, the average accuracy loss is reduced to 1.58%, representing a 70.63% improvement over MXFP4. Compared with NVFP4 at the same effective bit-width (4.5 bits), M^{2}XFP also shows lower loss (1.58% vs. 2.52%), corresponding to an absolute accuracy gain of 0.94% (37.30% improvement). We note that NVFP4 achieves higher accuracy on certain tasks (e.g., WinoGrande on LLaMA3-8B), which demonstrates its effectiveness. However, when averaged across all benchmarks, M^{2}XFP consistently delivers superior overall accuracy. Other data types also benefit from adaptive shared scale search, but these gains do not change the overall trends in Tbl. 2.

Compared to Baseline Accelerators. We evaluate perplexity on Wikitext v2 across several LLMs, comparing M^{2}XFP with MX-ANT, MX-M-ANT, MX-OliVe, MicroScopiQ, and

Table 4. Evaluation of reasoning tasks on DeepSeek-R1-Distill-Qwen: MXFP4 vs. M^{2}XFP.

Method	AIME-90	MATH-500	GSM8K	GPQA	LiveCodeBench	Avg.
	DeepSeek-R1-Distill-Qwen-1.5B					
FP16	21.11	85.4	84.76	36.36	17.54	49.03
MXFP4	7.78	66.6	69.37	31.82	8.96	36.91
M^{2}XFP	18.89	80.2	79.83	32.83	10.45	44.44
	DeepSeek-R1-Distill-Qwen-7B					
FP16	45.56	93.80	90.83	50.51	35.82	63.30
MXFP4	26.67	89.60	88.40	46.97	28.36	56.00
M^{2}XFP	40.00	93.80	90.83	52.02	32.40	61.81

Table 5. The area and power of core components and buffers for M^{2}XFP using a 28nm process.

Component	Number	Area(mm^2)	Power(mW)
PE Tile (2140.12μm^2)	128	0.2739	27.021
Top-1 Decode Unit (82.91μm^2)	4	0.0003	0.064
Quantization Engine (2451.47μm^2)	1	0.0024	0.663
Buffer (324KB)	1	0.7740	176.268
Total		1.051	204.02

BlockDialect, all under W4A4 quantization with group size 32 and an E8M0 shared scaling factor.

M^{2}XFP achieves the lowest perplexity on all models except OPT-6.7B, where BlockDialect is better by only 0.03. MX-ANT and MX-M-ANT improve over MXFP4 by adapting weight types, but extending to activations is limited by costly online search. BlockDialect addresses this with efficient real-time decision, yielding larger gains over MXFP4. MX-OliVe, though effective tensor-wise, underperforms MXFP4 in group-wise due to its 'outlier-victim' encoding that sacrifices neighbors. MicroScopiQ shows that such neighboring outliers frequently occur in LLMs, and adopts a block-level scheme for better balance. However, its reliance on naive MXINT activation quantization leads to suboptimal W4A4 perplexity.

Reasoning Tasks. We evaluate M^{2}XFP on complex reasoning benchmarks using DeepSeek-R1-Distill-Qwen. Prior work [45] shows that MXFP4 severely degrades reasoning ability, making LLMs nearly incapable of handling advanced math or coding tasks. Our results in Tbl. 4 confirm this: MXFP4 causes a 12.12% accuracy drop on DeepSeek-R1-Distill-Qwen-1.5B. M^{2}XFP can recover the average accuracy loss to 4.59%. Moreover, M^{2}XFP scales robustly to 7B reasoning models, maintaining reliable performance across sizes.

6.3 Performance, Area, and Energy

Tbl. 5 presents the component breakdown of M^{2}XFP. A 32 × 32 systolic array is modeled with four top-1 decode units, each handling eight 4-bit inputs. Together with the quantization engine, these account for only 0.26% of area and 0.36% of power overhead in all components, reflecting the low overhead of MX quantization in E8M0 format. To quantify the hardware overhead across data formats, we synthesized MXFP4, NVFP4, and M^{2}XFP PE tile using the same 28nm flow. The resulting PE tile areas are 2057.6μm^2 (MXFP4), 2104.7μm^2 (NVFP4, +2.3%), and 2140.1μm^2 (M^{2}XFP, +4.0%), showing that M^{2}XFP remains in the same cost range as existing MX-based formats and introduces only modest additional area. The design includes 324 KB of buffer: 144 KB each for activations and weights, plus 36 KB for outputs with scaling factors and metadata. It is worth noting that the buffer size also incorporates storage for scaling factors and metadata.

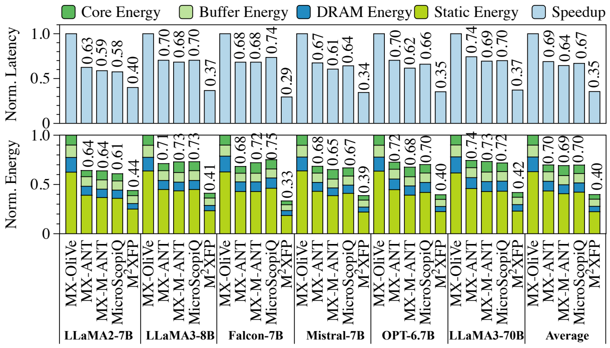


Figure 13. The normalized latency and energy comparison between M^{2}XFP and baseline accelerators.

Table 6. Wikitext perplexity of NVFP4 and NVFP4 with Elem-EM and Sg-EM metadata (lower is better).

Method	LLaMA2 7B	LLaMA3 8B	LLaMA3 70B	OPT 6.7B	Mistral 7B	Falcon 7B
FP16	5.47	6.14	2.85	10.86	5.32	6.59
NVFP4	5.81	7.18	3.63	11.46	5.76	6.90
M^2-NVFP4	5.77	6.85	3.57	11.32	5.58	6.88

Table 7. Comparison with several algorithm schemes. The dataset is Wikitext and lll group size is 32.

Method Data Type	QuaRot INT4	DuQuant INT4	MR-GPTQ FP4	M^{2}XFP FP4	MR-GPTQ-M^{2}XFP FP4
LLaMA2-7B	5.84	6.28	5.97	5.77	5.73
LLaMA3-8B	7.13	7.90	7.17	6.84	6.84

Fig. 13 compares performance and energy against MX-based baselines under identical systolic array sizes, differing only in decoder, encoder, or PE design. To match accuracy, the baselines require quantizing some tensors to 8 bits, which contributes a lot to their higher latency and energy consumption. In particular, MX-OliVe falls back to 8-bit quantization for more than 50% of tensors, resulting in a large performance gap compared with M^{2}XFP. MX-ANT, MX-M-ANT, and MicroScopiQ achieve similar performance, but MX-M-ANT consumes extra core energy from shift-and-accumulate operations, while MicroScopiQ expends more in its ReCoN unit for outlier processing. Overall, M^{2}XFP achieves on average 1.91× speedup and 1.75× energy reduction compared to the state-of-the-art MX accelerator MicroScopiQ.

Table 8. Wikitext perplexity of different methods to calculate the shared scale for MXFP4.

Models	LLaMA2-7B		LLaMA3-8B	
	MXFP4	M^{2}XFP4	MXFP4	M^{2}XFP4
floor	7.15	**5.77**	8.30	**6.84**
ceil/RTNE	**6.21**	5.80	**7.97**	6.96
RTN1	9.21	5.79	9.34	6.87
RTN2	6.26	5.81	8.08	7.01

6.4 Analysis and Discussion

Applying on NVFP4. M^{2}XFP is a general design that can also extend to formats such as NVFP4. By integrating Sg-EM for weights and Elem-EM for activations, we construct M^2-NVFP4 (Tbl. 6), which yields lower perplexity than the original NVFP4. However, since NVFP4 uses group size 16, the added metadata raises its effective bit-width from 4.5 to 5 bits. Therefore, NVFP4 may benefit from further exploration in our encoding design framework to identify more bit-efficient designs.

Impact of Shared Scale Calculation. Different ways of computing the shared scale can affect quantization error and accuracy [9, 50, 68]. We evaluate five strategies for computing the shared scale $S = 2^E$ from the block maximum amax in MX-style quantization. The first rule, *floor*, follows the OCP [56] specification and sets $E = \lfloor \log_2(\text{amax}/P) \rfloor$, where P is the largest power-of-two value representable by the format (e.g., $P = 4$ for FP4). The second rule, *ceil*, instead normalizes by the maximum representable magnitude M and uses $E = \lceil \log_2(\text{amax}/M) \rceil$ (e.g., $M = 6$ for FP4). The third rule, *RTN1*, replaces the ceil operation with round-to-nearest, $E = \text{round}(\log_2(\text{amax}/M))$. The fourth rule, *RTN2*, uses round-to-nearest on the normalized block maximum with respect to the largest power-of-two representable value P, i.e., $E = \text{round}(\log_2(\text{amax}/P))$. Finally, the *RTNE* rule, introduced in prior work [68], rounds amax in value space, normalizes by P, and then applies floor to the logarithm, $E = \lfloor \log_2(\text{round}(\text{amax})/P) \rfloor$. Notably, for FP4 (where $P = 4$ and $M = 6$), RTNE and ceil produce identical exponents because $M = \frac{3}{2}P$, which ensures $\lfloor \log_2(\text{round}(a)/P) \rfloor = \lceil \log_2(a/M) \rceil$ for all block maxima a; thus, the two rules are equivalent in this format. In all cases, the shared scale is obtained as $S = 2^E$. Our previous experiments used only the floor rule, which corresponds to the OCP-recommended default configuration.

As shown in Tbl. 8, for MXFP4 the ceil/RTNE rule achieves the lowest perplexity, consistent with the MXFP8 recipe [50] (Appendix A.1), whereas RTN1 performs worse because it does not address the dominant block-maximum error and introduces additional nondeterminism. RTN2 is close to RTNE and ceil, but is slightly worse on average. Across all five shared-scale computation rules, M^{2}XFP consistently improves accuracy over the MXFP4 baseline, indicating that its gains are robust to the choice of scaling strategy.

Comparison with Algorithm Schemes. To demonstrate that the MXFP data format is competitive with recent algorithmic schemes [2, 17, 43], we add comparisons with DuQuant [43], QuaRot [2] (INT), and the MX-based MR-GPTQ [17]. Under the same group size, M^{2}XFP achieves lower perplexity. Since MR-GPTQ is an algorithmic scheme and orthogonal to M^{2}XFP, we also combine them; the joint gain is incremental but may improve with further tuning.

Extension to Attention and KV Cache. While Linear layers (handling Q/K/V/O projections) dominate latency (~83%) at typical sequence lengths of 4096, the Attention mechanism becomes significant at longer contexts, accounting for ~45% of latency at length 16384. Extending M^{2}XFP to the KV cache is therefore crucial for sustaining its benefits across varying sequence lengths and workloads. Furthermore, quantization strategy can be integrated with recent memory management systems [30, 38, 53, 66, 73] to further optimize memory usage and minimize data movement.

In practice, applying M^{2}XFP to the KV cache follows the same design principles as for Linear layers. In Attention, K/V are both right-hand operands in GEMM ($P = QK^T$, $O = PV$). Systems like KIVI [47] and VQ-LLM [46] adopt a lazy KV cache quantization policy, allowing adaptive shared scale search. Therefore, Sg-EM can be used for K and V, and Elem-EM for Q and P, which is compatible with M^{2}XFP architecture.

7 Conclusion

In this paper, we presented M^{2}XFP, a metadata-augmented microscaling (MX) data format that mitigates accuracy loss in 4-bit weight-activation quantization. We explored bit-efficient metadata allocation schemes, built a dedicated hardware unit for encoding support, and integrated it into a systolic array. M^{2}XFP reduces accuracy loss by 70.6% over MXFP4 and 37.3% over NVFP4, while achieving up to 1.91× performance and 1.75× energy gains, demonstrating the practicality of metadata-driven MX formats for future LLM accelerators.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) Grants (62222210, 62532006, and 62502305), Shanghai Qi Zhi Institute Innovation Program SQZ202316, and Natural Science Foundation of Shanghai Grants (25ZR1402275). The authors express their gratitude to the anonymous reviewers for their insightful feedback, which greatly contributed to improving this work. We also thank our shepherd for the ongoing support and guidance during the revision process. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

[1] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. The Falcon Series of Open Language Models. arXiv:2311.16867 [cs.CL] https://arxiv.org/abs/2311.16867

[2] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. QuaRot: outlier-free 4-bit inference in rotated LLMs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 3180, 28 pages.

[3] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (June 2017), 25 pages. doi:10.1145/3085572

[4] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

[5] Yuzong Chen, Ahmed F. AbouElhamayed, Xilai Dai, Yang Wang, Marta Andronic, George A. Constantinides, and Mohamed S. Abdelfattah. 2025. BitMoD: Bit-serial Mixture-of-Datatype LLM Acceleration. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1082–1097. doi:10.1109/HPCA61900.2025.00084

[6] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 2924–2936. doi:10.18653/v1/N19-1300

[7] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. arXiv:1803.05457 [cs.AI] https://arxiv.org/abs/1803.05457

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168 [cs.LG] https://arxiv.org/abs/2110.14168

[9] Jack Cook, Junxian Guo, Guangxuan Xiao, Yujun Lin, and Song Han. 2025. Four Over Six: More Accurate NVFP4 Quantization with Adaptive Block Scaling. arXiv:2512.02010 [cs.CL] https://arxiv.org/abs/2512.02010

[10] Stef Cuyckens, Xiaoling Yi, Nitish Satya Murthy, Chao Fang, and Marian Verhelst. 2025. Efficient Precision-Scalable Hardware for Microscaling (MX) Processing in Robotics Learning. arXiv:2505.22404 [cs.AR] https://arxiv.org/abs/2505.22404

[11] Steve Dai, Rangharajan Venkatesan, Haoxing Ren, Brian Zimmer, William J. Dally, and Brucek Khailany. 2021. VS-Quant: Per-vector Scaled Quantization for Accurate Low-Precision Neural Network Inference. arXiv:2102.04503 [cs.LG]

[12] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, Alessandro Forin, Haishan Zhu, Taesik Na, Prerak Patel, Shuai Che, Lok Chand Koppaka, XIA SONG, Subhojit Som, Kaustav Das, Saurabh T, Steve Reinhardt, Sitaram Lanka, Eric Chung, and Doug Burger. 2020. Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point. In *Advances in Neural Information Processing Systems*, Vol. 33. 10271–10281.

[13] Bita Darvish Rouhani, Ritchie Zhao, Venmugil Elango, Rasoul Shafipour, Mathew Hall, Maral Mesmakhosroshahi, Ankit More, Levi Melnick, Maximilian Golub, Girish Varatkar, Lai Shao, Gaurav Kolhe, Dimitry Melts, Jasmine Klar, Renee L'Heureux, Matt Perry, Doug Burger, Eric Chung, Zhaoxia (Summer) Deng, Sam Naghshineh, Jongsoo Park, and Maxim Naumov. 2023. With Shared Microexponents, A Little Shifting Goes a Long Way. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 83, 13 pages. doi:10.1145/3579371.3589351

[14] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[15] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG]

[16] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, , et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[17] Vage Egiazarian, Roberto L. Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2025. Bridging the Gap Between Promise and Performance for Microscaling FP4 Quantization. arXiv:2509.23202 [cs.LG] https://arxiv.org/abs/2509.23202

[18] Chao Fang, Man Shi, Robin Geens, Arne Symons, Zhongfeng Wang, and Marian Verhelst. 2025. Anda: Unlocking Efficient LLM Inference with a Variable-Length Grouped Activation Data Format . In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 1467–1481. doi:10.1109/HPCA61900.2025.00110

[19] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 10323–10337. https://proceedings.mlr.press/v202/frantar23a.html

[20] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs.LG]

[21] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. The Language Model Evaluation Harness. doi:10.5281/zenodo.12608602

[22] Minseong Gil, Dongho Ha, Simla Burcu Harma, Myung Kuk Yoon, Babak Falsafi, Won Woo Ro, and Yunho Oh. 2025. Avant-Garde: Empowering GPUs with Scaled Numeric Formats. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 153–165. doi:10.1145/3695053.3731100

[23] Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, and Minyi Guo. 2022. Transkimmer: Transformer Learns to Layer-wise Skim. *arXiv preprint arXiv:2205.07324* (2022).

[24] Yue Guan, Changming Yu, Yangjie Zhou, Jingwen Leng, Chao Li, and Minyi Guo. 2024. Fractal: Joint Multi-Level Sparse Pattern Tuning of Accuracy and Performance for DNN Pruning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA). New York, NY, USA, 416–430.

[25] Cong Guo, Bo Yang Hsueh, Jingwen Leng, Yuxian Qiu, Yue Guan, Zehuan Wang, Xiaoying Jia, Xipeng Li, Minyi Guo, and Yuhao Zhu. 2020. Accelerating sparse dnn models without hardware-support via tile-wise sparsity. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[26] Cong Guo, Yuxian Qiu, Jingwen Leng, Xiaotian Gao, Chen Zhang, Yunxin Liu, Fan Yang, Yuhao Zhu, and Minyi Guo. 2022. SQuant: On-the-Fly Data-Free Quantization via Diagonal Hessian Approximation. arXiv:2202.07471 [cs.LG] https://arxiv.org/abs/2202.07471

[27] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 15 pages. doi:10.1145/3579371.3589038

[28] Cong Guo, Fengchen Xue, Jingwen Leng, Yuxian Qiu, Yue Guan, Weihao Cui, Quan Chen, and Minyi Guo. 2024. Accelerating sparse dnns based on tiled gemm. *IEEE Trans. Comput.* (2024).

[29] Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2022. ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1414–1433. doi:10.1109/MICRO56248.2022.00095

[30] Cong Guo, Rui Zhang, Jiale Xu, Jingwen Leng, Zihan Liu, Ziyu Huang, Minyi Guo, Hao Wu, Shouren Zhao, Junping Zhao, and Ke Zhang. 2024. GMLake: Efficient and Transparent GPU Memory Defragmentation for Large-scale DNN Training with Virtual Memory Stitching *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 450–466. doi:10.1145/3620665.3640423

[31] Xiaomeng Han, Yuan Cheng, Jing Wang, Junyang Lu, Hui Wang, X. X. Zhang, Ning Xu, Dawei Yang, and Zhe Jiang. 2025. *BBAL: A Bidirectional Block Floating Point-Based Quantisation Accelerator for Large Language Models*. IEEE Press. https://doi.org/10.1109/DAC63849.2025.11132978

[32] Weiming Hu, Haoyan Zhang, Cong Guo, Yu Feng, Renyang Guan, Zhendong Hua, Zihan Liu, Yue Guan, Minyi Guo, and Jingwen Leng. 2025. M-ANT: Efficient Low-bit Group Quantization for LLMs via Mathematically Adaptive Numerical Type. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1112–1126. doi:10.1109/HPCA61900.2025.00086

[33] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv:2403.07974 [cs.SE] https://arxiv.org/abs/2403.07974

[34] Wonsuk Jang and Thierry Tambe. 2025. BlockDialect: Block-wise Fine-grained Mixed Format Quantization for Energy-Efficient LLM Inference. arXiv:2501.01144 [cs.CL] https://arxiv.org/abs/2501.01144

[35] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[36] Alireza Khodamoradi, Kristof Denolf, and Eric Dellinger. 2024. Error Diffusion: Post Training Quantization with Block-Scaled Number Formats for Neural Networks. arXiv:2410.11203 [cs.LG] https://arxiv.org/abs/2410.11203

[37] Jahyun Koo, Dahoon Park, Sangwoo Jung, and Jaeha Kung. 2024. OPAL: Outlier-Preserved Microscaling Quantization Accelerator for Generative Large Language Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 259, 6 pages. doi:10.1145/3649329.3657323

[38] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[39] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. 2024. Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization. arXiv:2406.12930 [cs.LG] https://arxiv.org/abs/2406.12930

[40] Jungi Lee, Junyong Park, Soohyun Cha, Jaehoon Cho, and Jaewoong Sim. 2025. MX+: Pushing the Limits of Microscaling Formats for Efficient Large Language Model Serving. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 869–883. doi:10.1145/3725843.3756118

[41] Janghwan Lee, Jiwoong Park, Jinseok Kim, Yongjik Kim, Jungju Oh, Jinwook Oh, and Jungwook Choi. 2025. AMXFP4: Taming Activation Outliers with Asymmetric Microscaling Floating-Point for 4-bit LLM Inference. arXiv:2411.09909 [cs.AI] https://arxiv.org/abs/2411.09909

[42] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050* (2023).

[43] Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. 2024. DuQuant: distributing outliers via dual transformation makes stronger quantized LLMs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 2786, 35 pages.

[44] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, Chuang Gan, and Song Han. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv:2306.00978 [cs.CL]

[45] Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan, and Lu Hou. 2025. Quantization Hurts Reasoning? An Empirical Study on Quantized Reasoning Models. arXiv:2504.04823 [cs.CL] https://arxiv.org/abs/2504.04823

[46] Zihan Liu, Xinhao Luo, Junxian Guo, Wentao Ni, Yangjie Zhou, Yue Guan, Cong Guo, Weihao Cui, Yu Feng, Minyi Guo, Yuhao Zhu, Minjia Zhang, Chen Jin, and Jingwen Leng. 2025. VQ-LLM: High-performance Code Generation for Vector Quantization Augmented LLM Inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1496–1509. doi:10.1109/HPCA61900.2025.00112

[47] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. *arXiv preprint arXiv:2402.02750* (2024).

[48] Yun-Chen Lo, Tse-Kuang Lee, and Ren-Shuo Liu. 2023. Block and Subword-Scaling Floating-Point (BSFP) : An Efficient Non-Uniform Quantization For Low Precision Inference. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=VWm4o4l3V9e

[49] Yun-Chen Lo and Ren-Shuo Liu. 2023. Bucket Getter: A Bucket-based Processing Engine for Low-bit Block Floating Point (BFP) DNNs. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 1002–1015. doi:10.1145/3613424.3614249

[50] Asit Mishra, Dusan Stosic, and Simon Layton. 2025. Recipes for Pretraining LLMs with MXFP8. arXiv:2506.08027 [cs.LG] https://arxiv.org/abs/2506.08027

[51] Nvidia. 2024. NVIDIA Blackwell Architecture Technical Brief. In *Technical report*. NVIDIA.

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[53] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. 2025. vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1133–1150. doi:10.1145/3669940.3707256

[54] Akshat Ramachandran, Souvik Kundu, and Tushar Krishna. 2025. MicroScopiQ: Accelerating Foundational Models through Outlier-Aware Microscaling Quantization. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1193–1209. doi:10.1145/3695053.3730989

[55] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *First Conference on Language Modeling*. https://openreview.net/forum?id=Ti67584b98

[56] Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Jasmine Klar, Eric Chung, Yuan Yu, Michael Schulte, Ralph Wittig, Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Summer Deng, Maxim Naumov, Paulius Micikevicius, Michael Siu, and Colin Verrilli. 2023. OCP Microscaling Formats (MX) Specification. https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf. Accessed: 2023-09-07.

[57] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: an adversarial winograd schema challenge at scale. *Commun. ACM* 64, 9 (Aug. 2021), 99–106. doi:10.1145/3474381

[58] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. arXiv:2308.13137 [cs.LG]

[59] Sayeh Sharify, Utkarsh Saxena, Zifei Xu, Wanzin Yazar, Ilya Soloveychik, and Xin Wang. 2024. Post Training Quantization of Large Language Models with Microscaling Formats. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop (Proceedings of Machine Learning Research, Vol. 262)*, Mehdi Rezagholizadeh, Peyman Passban, Soheila Samiee, Vahid Partovi Nia, Yu Cheng, Yue Deng, Qun Liu, and Boxing Chen (Eds.). PMLR, 241–258. https://proceedings.mlr.press/v262/sharify24a.html

[60] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 17, 12 pages.

[61] Alan Smith and Vamsi Alla. 2024. AMD Instinct MI300X Generative AI Accelerator and Platform Architecture. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. 1–22. doi:10.1109/HCS61935.2024.10664659

[62] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A Simple and Effective Pruning Approach for Large Language Models. arXiv:2306.11695 [cs.CL] https://arxiv.org/abs/2306.11695

[63] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[64] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks. arXiv:2402.04396 [cs.LG] https://arxiv.org/abs/2402.04396

[65] Hemish Veeraboina. 2023. *AIME Problem Set 1983-2024*. https://www.kaggle.com/datasets/hemishveeraboina/aime-problem-set-1983-2024

[66] Jiale Xu, Rui Zhang, Cong Guo, Weiming Hu, Zihan Liu, Feiyang Wu, Yu Feng, Shixuan Sun, Changxu Shao, Yuhong Guo, Junping Zhao, Ke Zhang, Minyi Guo, and Jingwen Leng. 2024. vTensor: Flexible Virtual Tensor Management for Efficient LLM Serving. arXiv:2407.15309 [cs.DC] https://arxiv.org/abs/2407.15309

[67] Sherry Xu and Chandru Ramakrishnan. 2024. Inside Maia 100. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. 1–17. doi:10.1109/HCS61935.2024.10665248

[68] Hanmei Yang, Summer Deng, Amit Nagpal, Maxim Naumov, Mohammad Janani, Tongping Liu, and Hui Guan. 2025. An Empirical Study of Microscaling Formats for Low-Precision LLM Training. In *2025 IEEE 32nd Symposium on Computer Arithmetic (ARITH)*. 1–8. doi:10.1109/ARITH64983.2025.00011

[69] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence?. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4791–4800. doi:10.18653/v1/P19-1472

[70] Jintao Zhang, Jia Wei, Pengle Zhang, Xiaoming Xu, Haofeng Huang, Haoxu Wang, Kai Jiang, Jun Zhu, and Jianfei Chen. 2025. SageAttention3: Microscaling FP4 Attention for Inference and An Exploration of 8-Bit Training. arXiv:2505.11594 [cs.LG] https://arxiv.org/abs/2505.11594

[71] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[72] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2023. Atom: Low-bit Quantization for Efficient and Accurate LLM Serving. arXiv:2310.19102 [cs.LG]

[73] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems* 37 (2024), 62557–62583.