PRESTE: Preserving Tiny Exponent Precision for Efficient Sub-8-bit LLM Inference and Fine-Tuning

Anonymous Author(s)

ABSTRACT

Fine-tuning large language models (LLMs) for specialized downstream tasks has become the prevailing paradigm in real-world deployments. Previous studies have explored 8-bit floating-point (FP8) quantization for neural network training. However, limited exponent range often results in precision loss and requires high-precision hardware accumulators to ensure resolution. We propose a novel quantization technique for fine-tuning and inference that preserves small-magnitude values, which would otherwise vanish under conventional quantization schemes. Additionally, we design a hardware architecture optimized for this format, enabling energy-efficient computation with reduced reliance on high-precision accumulators. Comprehensive evaluations across LLMs demonstrate that our approach achieves 2.22-3.55× higher energy efficiency and 1.67-2.26× higher performance compared to baseline methods during inference and fine-tuning.

KEYWORDS

Large Language Model, Fine-tuning, Quantization

ACM Reference Format:

Anonymous Author(s). 2025. PRESTE: Preserving Tiny Exponent Precision for Efficient Sub-8-bit LLM Inference and Fine-Tuning . In *63st ACM/IEEE Design Automation Conference (DAC '26), July 26-29, 2026, Long Beach, CA.* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Pre-trained large language models (LLMs) have demonstrated remarkable capabilities across various tasks, establishing themselves as foundational models [31]. Leveraging pre-trained LLMs that encode general knowledge and subsequently fine-tuning them on task-specific datasets for domain-specific tasks, has proven to be both time-efficient and performance-enhancing compared to training models from scratch [6, 22]. For applications from healthcare to personal assistants, access to sensitive datasets is indispensable to construct and deploy advanced models [26, 27]. Enabling fine-tuning and inference directly on edge devices allows locally generated data to be processed in situ, enhancing privacy and eliminating sensitive data transmission to centralized data centers [15, 34, 39]. Low-rank adaptation (LoRA) enables the fine-tuning of billion-parameter LLMs on a single GPU [7]. However, edge devices inherently suffer from strict computational and memory constraints, necessitating an efficient quantization method to make fine-tuning feasible.

DAC '26, July 26-29, 2026, Long Beach, CA

ACM ISBN XXX-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

Most existing quantization techniques, however, are optimized for inference workloads [5, 10, 13, 38]. These methods typically rely on calibration-based quantization, which cannot adequately handle the numerical variance encountered during backpropagation. Consequently, they are inadequate for enabling on-device fine-tuning. On the other hand, quantization from 32-bit floating-point (FP32) parameters to low-precision formats such as FP8 become prevalent, while maintaining gradient during backpropagation [2, 19, 21, 28, 33]. However, due to limited exponent bits, FP8 formats result in significant numerical errors and instability, especially during training. Techniques like bias shifting, which adjusts the exponent bias through heuristic searches and profiling, are inadequate in edge systems [19, 28]. An advanced method, Microscaling data formats (MX), applies group-level quantization with a shared factor to avoid heuristic scaling [23]. However, 8-bit MX floating-point (MXFP8) requires an FP32-level accumulation unit with FP8 multiplier for computation [20].

To overcome these limitations, we propose **PRESTE (Preserving Precision of Tiny Exponents)**, a quantization approach designed to retain all exponent values without loss, alongside a hardware architecture optimized for energy and area efficiency. Unlike traditional FP8 approaches which require FP32 accumulator, PRESTE's computation unit utilizes 4-bit integer (INT4) multiplier and INT21 accumulator, enabling area- and energy-efficient hardware implementation. We observe that, in many vectors, only a small subset of values have tiny exponents that are much smaller than the maximum exponent within the same vector. These small-magnitude values often vanish during quantization due to limited precision, leading to numerical errors during computation. By preserving values with these exponents, PRESTE minimizes quantization errors during computations. In addition, PRESTE efficiently supports a wide exponent range for activations and gradients, enabling to achieve Bfloat16 level accuracy. This capability alleviates the memory access overhead of high-precision gradients in backpropagation, making it feasible to fine-tune LLMs directly on edge environments.

In summary, our main contributions are as follows:

- We propose a quantization method that preserves the full range of exponent values without truncation. By preserving wide-range exponent elements during fine-tuning, such as activations and gradients, our method achieves the accuracy level of Bfloat16.
- We introduce a processing architecture to address the limitations of FP8 dot-product operations, which require high-precision hardware component. Our architecture minimizes hardware overhead with the 21-bit integer accumulator, while still covering the entire range of 8-bit exponents.
- We validate the PRESTE quantization method and hardware design through extensive evaluations, applying PRESTE to fine-tune and inference across various LLMs and datasets.

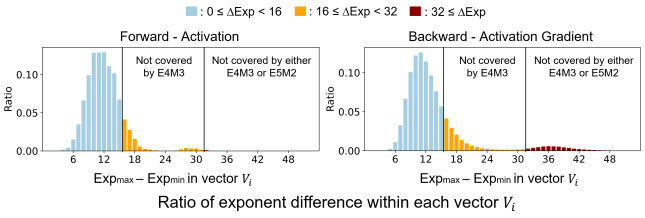


Figure 1: Distribution of the difference between the maximum and minimum exponents within 32 elements vector of activation and gradient tensors.

2 BACKGROUND AND CHALLENGES

2.1 Vanishing of Tiny Value During Training

Researchers have explored training DNNs using a hybrid FP8 format, which utilizes FP8-E4M3 (4-bit exponent, 3-bit mantissa) for forward propagation and FP8-E5M2 for backward propagation [21, 28, 33]. However, the FP8 format encounters limitations during forward and backward propagation due to its limited exponent range. The restriction necessitates additional techniques, such as heuristic bias searches guided by gradient statistics, to manage the dynamic range of values during training [19, 28]. Consequently, they are inadequate for enabling fine-tuning on edge environments.

The MXFP addresses the heuristic adjustment by sharing an 8-bit exponent across 32 elements. However, MXFP still suffers from significant quantization errors when a few elements in the vector have exponents that differ substantially from the maximum exponent. For instance, in MXFP8-E4M3, the 4-bit exponent supports reliable quantization only when the exponent difference within a vector is less than or equal to 15, as indicated by the blue bars in Fig. 1. When the exponent difference exceeds 15, accurate representation becomes impossible, causing affected elements to be quantized as a subnormal or zero. In Fig. 1, orange and red bars indicate the proportion of the vectors that cannot be covered by conventional MXFP8-E4M3 and MXFP8-E5M2, respectively. This quantization leads to distortions in dot-product computations, particularly when subnormal or zero values are multiplied by large counterpart elements.

Fig. 2 visualizes the relative errors in dot-product computations across various formats. The x-axis and y-axis represent the difference between the maximum and minimum exponents in each 32-element vector, A and B, respectively. The results demonstrate that an 8-bit exponent effectively captures large exponent variations within a vector, significantly reducing errors compared to FP8 and MXFP8 formats. Preserving tiny exponent precision prevents the truncation to subnormal or zero and ensures accurate computation. Experimental results for the impact of preserving tiny exponent will be detailed in Sec. 5. The key innovation of our proposed method lies in preserving the precision of an 8-bit exponent while utilizing only a 3-bit exponent representation.

2.2 FP8-based Dot-product Unit

As discussed earlier, effective fine-tuning requires both preserving wide exponent range to reduce quantization-induced errors and ensuring computational efficiency. In prior approaches, FP8-based arithmetic is used to general matrix multiplication (GEMM) operation during training [19, 28]. To improve computation efficiency, dot-product unit typically performs multiple multiplications

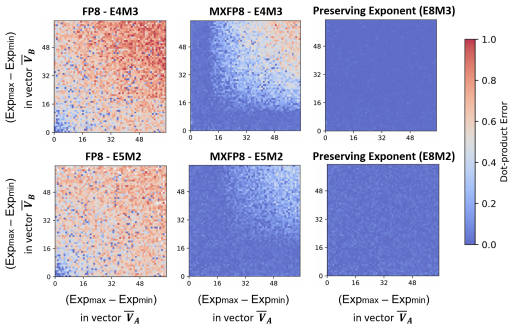
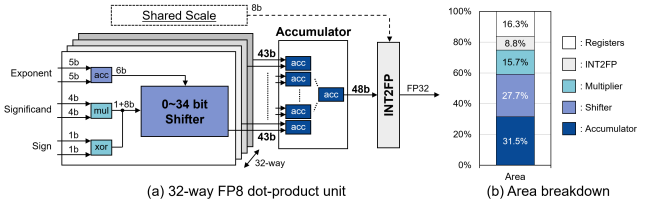


Figure 2: Visualization of the dot-product error for two vectors across various quantization formats. The x-axis and y-axis represent the difference between the maximum and minimum exponents in vectors A and B, respectively.



(a) 32-way FP8 dot-product unit (b) Area breakdown

Figure 3: (a) The block diagram of a 32-way dot product unit for FP8. (b) The area breakdown of dot-product unit.

in parallel and the results are aggregated through a single FP accumulator [11]. Fig. 3(a) illustrates a 32-way FP8 dot-product unit. This unit first aligns the 32 multiplication results based on their respective exponents and then accumulates them into a final FP32 output. As shown in Fig. 3(a), the MXFP8 utilizes the FP8 as the element format, and requires additional registers to store the shared scaling factor in dotted line, without modification of the existing hardware [18, 23].

The FP8 hardware supports both FP8-E5M2 and FP8-E4M3 [2, 33]. To handle a wide range of exponents, the compute unit requires 34-bit shifter for alignment and a 43-bit adder tree for accumulation. This design achieves FP32-level precision, but suffers from a significant hardware cost [9, 16]. We analyzed the area overhead of the shifter and accumulator in the MXFP8 dot-product unit. As shown in Fig. 3(b), the shifter accounts for 27.7% of the total area, while the accumulator occupies 31.5% and the pipeline registers contribute 16.3% of the area. These findings suggest that limiting the exponent range in the dot-product unit can reduce the size of the alignment shifter and accumulator, thereby improving hardware efficiency.

3 PRESTE FORMAT

3.1 Representing 8-bit exponents with 3-bit

In many vectors, the range between the maximum and minimum exponents can be large; however, we observe that only a small fraction of elements contribute to this range. Most elements have exponents close to the maximum in each vector, indicating that the exponent values are tightly clustered. This localized distribution suggests that reduced bit precision can be applied with minimal overhead. Fig. 4 presents the profiling results for the exponents of the query during fine-tuning of the LLaMA-2 7B model on the

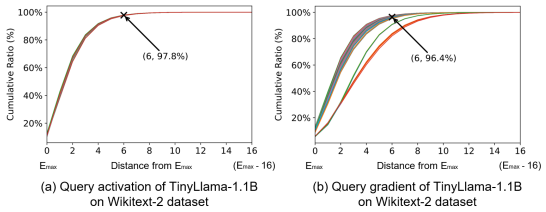


Figure 4: Cumulative ratio of exponent distances from maximum exponent within vector. E_{max} denotes maximum exponent.

WikiText-2 dataset using Bfloat16. The x-axis represents the difference between each element's exponent and the max exponent within its 32-element vector. As shown in Fig. 4, 97.8% of activation and 96.4% of gradient exhibit an exponent difference of 6 or less, indicating that most elements are clustered around the maximum exponent within their respective vectors. We refer to these elements as normal elements in the rest of the paper. In the proposed scheme, we leverage this property and represent the difference between most elements' exponents and the maximum exponent using just 3-bit. Elements with exponent difference of 7 or more from the maximum exponent in each vector are classified as tiny elements. We assign 8-bit exclusively for the tiny exponents to enable precise representation across a wide exponent range. This strategy allows for efficient usage of bit precision, with most exponents requiring only 3-bit. A detailed analysis of the exponent distribution during fine-tuning with various models on different datasets is provided in Section 5.2.

3.2 PRESTE Number Format Encoding

Fig. 5(a) illustrates an example of a typical Bfloat16-format vector. In the proposed PRESTE format, as shown in Fig. 5(b), the element with the maximum exponent in the Bfloat16 is set as the maximum exponent $E_{\max}$ for the entire vector, and the difference between maximum exponent $E_{\max}$ and each element's exponent is calculated. Depending on the magnitude of the difference, the exponent difference is expressed by the 3-bit align exponent for normal elements or 8-bit tiny exponent for tiny elements.

For example, in Fig. 5, if the maximum exponent in a vector is $E_{\max} = 141$, and the first element has an original exponent $\text{Exp}^1 = 139$, the 3-bit align exponent E^1_{align} becomes:

$$E^1_{\text{align}} = E_{\max} - \text{Exp}^1 = 141 - 139 = 2 = 010_{(2)}$$

On the other hand, the 31st element has an original exponent $\text{Exp}^{31} = 112$, and its aligned exponent E^{31}_{align} becomes larger than 6:

$$E^{31}_{\text{align}} = E_{\max} - \text{Exp}^{31} = 141 - 112 = 29.$$

In case $E^{31}_{\text{align}} > 6$, the 3-bit align exponent E^{31}_{align} is used as a tiny indicator with value of 7 ($=111_2$), and this element is represented using a separate 8-bit tiny exponent E_{tiny} with value:

$$E^{31}_{\text{tiny}} = \text{Exp}^{31} = 112.$$

In summary, PRESTE has 3-bit align exponents to indicate distance from max exponent. In cases where the difference exceeds 6, the original exponent is stored as an 8-bit tiny exponent. PRESTE format can be extended to different bit precision by assigning different number of bits to mantissa while using 3-bit align exponent

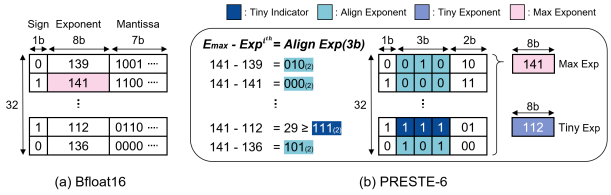


Figure 5: (a) The illustration of Bfloat16 data format. (b) The proposed PRESTE-6 data format with tiny and maximum exponents.

in common. For example, PRESTE-6 use 2-bit for mantissa and PRESTE-8 uses 4-bit for mantissa. Both of them use 1-bit for sign, and 3-bit for align exponents.

4 PRESTE ARCHITECTURE

4.1 Overall Architecture and Memory Layout

Fig. 6(b) illustrates the overall architecture of PRESTE. Every buffers serve as a scratchpad that are explicitly managed by controllers. The lane consists of compute array operating in weight stationary dataflow, scheduler, vector processing unit (VPU), and encoder. The scheduler in each lane feeds activations and weights to the compute array, ensuring correct timings for efficient computation. In each cycle, the compute array generates FP results, which are either processed by the VPU or encoded back into the PRESTE format by the encoder. The encoder supports quantization of floating-point vector into PRESTE vector with maximum exponent and tiny exponents. The final output vector consists of 32 PRESTE elements in reduced bit-precision, with a maximum exponent and tiny exponents. The 24-byte PRESTE vector and its corresponding 1-byte maximum exponent form the minimal unit of data in the layout as shown in Fig. 6(a). The 1K vectors are grouped into a PRESTE tile and stored in the local buffer. The tiny data tile contains tiny exponents of corresponding PRESTE tile. Both the PRESTE tile and the tiny tile are stored in the local buffer, and a controller orchestrates tile mapping across the memory hierarchy.

4.2 Dot-product Engine

Dot-product Engine. As discussed in Sec. 2.2, the MXFP8 dot-product unit requires a 34-bit shifter for alignment and a 43-bit accumulator, which results in significant area overhead. In contrast, PRESTE normal elements use align exponent range between 0 and 6. Therefore, the dot-product engine (DPE) for normal elements can align multiplication results using a 12-bit shifter and accumulate with 21-bit accumulator.

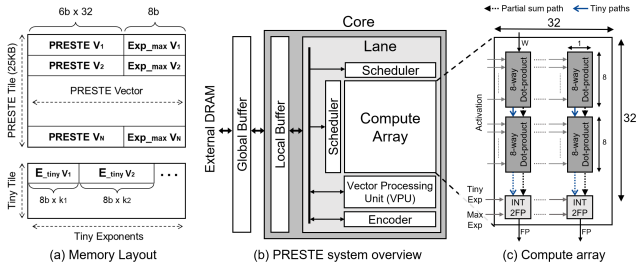
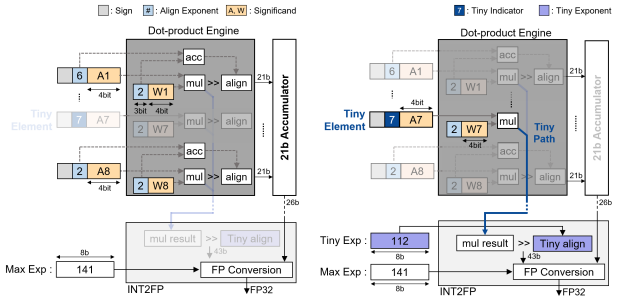


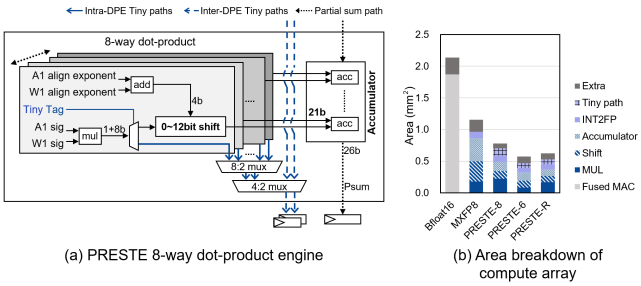
Figure 6: (a) The memory layout for PRESTE data tile and tiny data tile. (b) The overview of PRESTE system architecture. (c) The internal structure of compute array.



(a) Normal elements dot-product with 21b accumulator

(b) Tiny element with tiny path

Figure 7: Simultaneous dot-product operation for the normal (a) and tiny elements (b) within in same dot-product unit.



(a) PRESTE 8-way dot-product engine

(b) Area breakdown of compute array

Figure 8: (a) The detailed diagram of dot-product engine. (b) The area breakdown of compute array.

Fig. 7 illustrates how the DPE handles normal and tiny elements simultaneously. For normal elements, DPE aligns the multiplication results and accumulates them with 21-bit accumulator to generate a partial sum (Fig. 7(a)). When an element is identified as a tiny element, the multiplication result bypasses the alignment. Instead, the tiny result is routed through the dedicated tiny path to the INT2FP unit, which aligns the result based on its tiny exponent (Fig. 7(b)). After the alignment, FP conversion unit merges the tiny result with the normal partial sum to generate the final FP32 output. By eliminating intermediate truncation or rounding throughout the accumulation, PRESTE DPE maintains the FP32-level exponent range in the final output using 21-bit accumulator.

Intra-DPE and Inter-DPE Paths. The bypass mechanism using tiny paths for tiny results reduces the computation bit width required within the DPE. However, assigning a dedicated tiny path to each element in 32-element dot-product introduces significant routing complexity. To address this issue, we design two levels of tiny path: inter-DPE tiny paths and intra-DPE tiny paths [12]. Fig. 8(a) illustrates the microarchitecture of an 8-way DPE that includes two intra-DPE and two inter-DPE tiny paths. The multiplication result, either enters the shifter or routes through the intra-DPE tiny path based on the tiny tag. The dashed inter-DPE paths collect tiny results from both intra-DPE paths and upper DPE, forward them to the downstream DPE. As shown in Sec. 3.1, tiny elements typically appear only once per 32-element vector. This low occurrence rate allows us to use a small number of tiny paths, limiting overhead.

Limiting the Number of Tiny Elements. To prevent collisions with random number of tiny elements across inter-DPE tiny paths, a scheduler limits the number of tiny elements in each input vector before transmitting data to the compute array. As illustrated in Fig. 9, the scheduler ensures that each input vector contains no more than N tiny elements per DPEs with N inter-DPE tiny paths. Zero insertion is applied to fill the remaining positions in the sub-vectors, effectively mitigating inter-DPE tiny path collisions. The impact of this restriction is minimal because the occurrence of tiny elements exceeding the available tiny paths is rare. A detailed analysis of the performance impact due to the zero insertion caused by tiny elements is presented in Sec. 5.2.

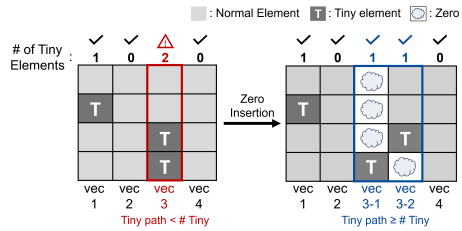


Figure 9: Zero insertion to limit the number of tiny elements in each vector.

4.3 Reconfigurable DPE

We also propose a minimally modified design to improve PRESTE-8 dot-product efficiency based on the existing PRESTE-6 DPE. This enhancement enables PRESTE-8-based GEMM operations in the transformer attention scores, supporting precise fine-tuning. In the PRESTE-6 format ⟨sign, 3-bit align exponent, 2-bit mantissa⟩, values are expressed as follows :

$$\langle s, (e_1, e_2, e_3), (m_1, m_2) \rangle$$

Before transmitting PRESTE-6 values into the compute array, we apply a 1-bit pre-align to the significand (including hidden bit) using the LSB of the 3-bit align exponent. The pre-align scheme converts 3-bit significand to a 4-bit pre-aligned significand, represented by:

$$\langle s, (e_1, e_2), [1.(m_1, m_2) \gg e_3] \rangle$$

More importantly, this design enables a multiplication in the PRESTE-8 format using 4 PRESTE-R computation units. This reconfigurability is useful to enhance the accuracy of LLMs by selectively applying PRESTE-8 format for multi-head attention computations during inference and fine-tuning, without requiring independent PRESTE-8 computational units. Since multi-head attention accounts for a relatively small proportion of the overall computation, using multiple PRESTE-R units to conditionally support PRESTE-8 does not incur noticeable degradation in overall computing performance. Fig. 8(b) shows the area breakdown for compute array consist of 32x32 MACs. PRESTE-R demonstrates high area efficiency, achieving area reduction of 3.50× and 1.89× compared to Bfloat16 and MXFP8, respectively.

5 EVALUATION

Model and Dataset. To validate the effectiveness of the PRESTE method, we evaluate various pre-trained models [1, 31, 32, 42]. We used the Alpaca dataset [30] for fine-tuning and evaluate CSQA scores [3, 24, 29, 40]. We also used WikiText-2 dataset [17] for fine-tuning and evaluate PPL using the Hugging Face evaluate [35].

Fine-tuning Hyperparameters. During fine-tuning, we applied LoRA across all quantization methods [4, 37] and quantized every tensors except weight gradient and optimizer states for LoRA adapters [8]. For a fair comparison, the same hyperparameters are used across quantization methods for each model. The fine-tuning sequence length is set to 1024, with LoRA rank to 32. A learning

Table 1: Fine-tuning results for different models on CSQA and perplexity across various quantization formats.

	Format	Bit		CSQA (Higher is better)				Perplexity (Lower is better)			
				TL-	SL-	L2-	SL2-	TL-	SL-	L2-	SL2-
		A	W	1.1B	3B	7B	12B	1.1B	3B	7B	12B
Vanilla	BF16	16	16	40.17	51.28	55.48	60.88	14.69	52.40	9.48	6.32
Fine-tuned	BF16	16	16	42.16	52.62	57.71	62.62	8.53	10.49	5.28	6.21
	MXFP-8	8	8	41.73	52.15	57.17	62.14	8.75	10.68	5.35	6.23
	PRESTE-8	**8**	**8**	**42.10**	**52.57**	**57.68**	**62.42**	**8.64**	**10.55**	**5.29**	**6.20**
	PRESTE-6	6	6	41.95	52.15	57.35	62.03	8.80	10.74	5.35	6.21
Vanilla	BF16	16	4	39.55	48.64	54.42	59.53	15.08	64.85	10.10	6.72
Fine-tuned	BF16	16	4	40.86	50.55	56.05	62.22	9.20	11.06	5.55	6.22
	MXFP-8	8	4	40.67	50.36	55.67	61.87	9.29	11.23	5.56	6.29
	PRESTE-8	**8**	**4**	**40.89**	**50.42**	**56.08**	**62.08**	**9.22**	**11.08**	**5.52**	**6.23**
	PRESTE-6	6	4	40.73	50.21	55.77	61.82	9.30	11.20	5.57	6.29

TinyLlama-1.1B (TL-1.1B), StableLM-3B (SL-3B), LLaMA-2-7B (L2-7B), StableLM-2-12B (SL2-12B)

MXFP-4 is used for 4-bit weight.

rate of either 1e-4 or 1e-5 is applied, and each model undergoes a total of 100 weight update steps.

Quantization Baseline. We compared our PRESTE quantization method with BF16 and MXFP, a state-of-the-art (SOTA) numeric format optimized for LLM training. Our evaluation focuses on approaches that do not rely on profiling-based heuristic adjustments, which are inadequate for edge device fine-tuning. We evaluated two scenarios for each model: Freezing weights in the same format as the activations or weights in the MXFP4 format. For both MXFP and PRESTE methods, we performed computations for the multi-head attention in MXFP-8 and PRESTE-8 formats, respectively. We also compared our scheme with SmoothQuant [36], OliVe [5] and Tender [13] for inference tasks. Although these methods are not compatible with training unlike our scheme, we still evaluate them to demonstrate the advantages of our PRESTE format in inference applications.

Accelerator Baseline. We compared the energy consumption and performance of BF16, MXFP, and PRESTE hardware. For MXFP, we used hardware optimized for dot-product operations [11, 16]. OliVe and Tender support MAC operations for custom number formats. Each PE supports 4-bit integer multiplication and 32-bit INT accumulation for 4-bit quantized data formats.

Hardware Implementation. All designs were implemented in Verilog and place and route were done using ICC2 with SRAM compiler on a 28nm process for area and power measurements. For architecture simulation, we integrate ScaleSim, LLMCompass for cycle-level simulation and DRAMsim3 for DRAM behavior, including tiny element overhead [14, 25, 41].

5.1 Accuracy Evaluation

Fine-tuning Accuracy. To demonstrate the effectiveness of our method in fine-tuning LLMs, we trained and evaluated various models on different datasets. The left side of Table 1 shows the results of fine-tuning each model on the Alpaca dataset, measuring their CSQA score. The right side of Table 1 shows the results of fine-tuning each model on the WikiText-2 dataset and measuring perplexity. PRESTE method with 8-bit precision consistently achieves better scores than MXFP at the same precision. Additionally, in most cases, the 6-bit PRESTE method outperforms 8-bit MXFP. This improvement stems from PRESTE's ability to preserve tiny exponents, ensuring precise exponent representation. By preserving exponents that are often compromised in conventional quantization methods, PRESTE significantly reduces errors in dot-product computations.

Table 2: Direct inference CSQA score on StableLM-3B across various quantization methods. The higher is better.

Method	Bit		StableLM-3B					CSQA
	A	W	Hella.	Arc_c	Arc_e	Wino.	OBQA.	Average
BF16	16	4*	52.61	37.37	65.32	61.48	26.40	48.64
SmoothQuant	8	4	29.01	21.93	37.31	58.99	15.40	32.53
OliVe	8	4	28.09	20.09	35.23	53.04	15.80	30.45
Tender	8	4	46.08	22.10	52.31	54.94	21.40	39.37
MXFP-8	8	4*	51.95	35.49	63.30	61.96	26.40	47.82
PRESTE-8	**8**	**4***	**52.57**	**36.69**	**64.86**	**62.90**	**26.20**	**48.64**
PRESTE-6	6	4*	52.08	36.01	64.86	61.17	25.80	47.98

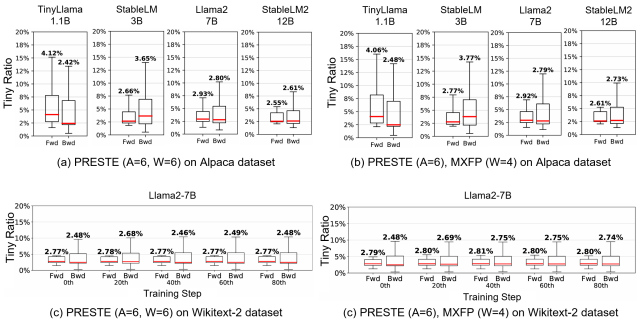
*MXFP4 is used for 4-bit weight.

Inference Accuracy. Table 2 presents detailed inference scores on pre-trained StableLM-3B across various quantization methods. The PRESTE method with 8-bit precision achieves better score than any other quantization methods at the same bit precision. Compared to inference-specialized formats, PRESTE demonstrates superior performance even on pre-trained models. This indicates that, in fine-tuning scenarios, PRESTE enables the deployment of more advanced downstream models with higher performance in edge environments.

5.2 Tiny Exponent Profiling

We intensively profiled the distribution of tiny elements of each tensor across LLM models with various datasets. All operand matrices including attention and linear modules are averaged for each matrix and are displayed as box plots. Fig. 10(a, b) provides an aggregated view of profiling data across different models with Alpaca dataset. The results indicate a broader distribution of tiny elements in the matrices associated with backward propagation. Fig. 10(c, d) shows profiling results with different training steps with WikiText-2 dataset. The average ratio of tiny elements is lower than 3% during fine-tuning of LLaMA-2 7B model on various datasets. Overall, the consistent proportion of tiny elements is observed, regardless of variations in data formats, datasets and models during fine-tuning. Based on these profiling results, we simulate the impact of tiny element distributions on the latency of GEMM operations in hardware including scheduling overhead.

Fig. 11 compares the latency of GEMM operations on 4096×4096 activation and weight matrices during the fine-tuning of the LLaMA-2 7B model. The distribution of tiny elements in both operand matrices was profiled using data generated during fine-tuning. Latency was simulated for cases where tiny elements comprised 2%, 3%, 5%, and 10% of a matrix. The simulations considered various number of



(a) PRESTE (A=6, W=6) on Alpaca dataset

(b) PRESTE (A=6), MXFP (W=4) on Alpaca dataset

(c) PRESTE (A=6, W=6) on Wikitext-2 dataset

(c) PRESTE (A=6), MXFP (W=4) on Wikitext-2 dataset

Figure 10: Each plot shows the ratio of tiny elements in tensors during fine-tuning process of models on different datasets.

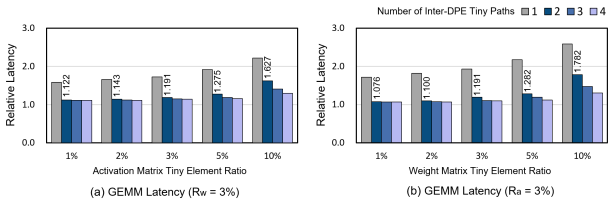


Figure 11: Relative latency of GEMM operation with various tiny elements ratio. R_w and R_a represent the ratio of tiny elements in weights and activations, respectively.

tiny paths for the first operand matrix, while the second operand matrix had a fixed 3% proportion of tiny elements and 2 tiny paths. The results indicate that with two tiny paths for each activation and weight matrices, the latency overhead for GEMM operations remains negligible, even as the proportion of tiny elements increases in the operand matrix. Based on these findings, PRESTE incorporates 2 tiny paths for each of activations and weights, ensuring efficient processing with minimal latency impact.

Table 3: Hardware Specification by Compute Type

	BF16	MXFP8	OliVe	Tender	PRESTE
Compute Array Area (mm^2)	**2.136**	**1.154**	**0.878**	**0.636**	**0.610**
Array / Lane	1	1	1	1	1
Lane / Core	2	3	4	5	5
Core Area (mm^2)	**5.368**	**5.139**	**5.329**	**5.171**	**5.126**
Common	2-channel DDR4 (16GB, 51.2GB/s) Global Buffer (4MB), Local Buffer (256KB), Core Frequency (500MHz)				

Table 4: Area and Power Breakdown of PRESTE Single Core

	Area (mm^2)	Power (mW)
Single Core	5.126	1274
Local Buffer	0.586 (11.4%)	0.2343pJ/bit
Lane	4.540	1274
Compute Array	3.049 (59.5%)	977 (76.7%)
Vector Processing Unit	1.278 (24.9%)	229 (17.9%)
Encoder + Scheduler	0.213 (4.2%)	69 (5.4%)

5.3 Hardware Evaluation

Hardware Configuration. Table 3 summarizes the hardware configurations based on different compute types. PRESTE demonstrates high area efficiency in compute array, achieving area reductions of 3.51× and 1.89× compared to Bfloat16 and MXFP8, respectively. OliVe [5], which is designed for inference, is 1.44× larger than PRESTE due to its 32-bit integer accumulation unit in each PE. Therefore, to ensure a fair comparison under the same core area constraints, the number of lanes is adjusted accordingly across configurations. We also considered additional overhead introduced by other components such as the vector unit, scheduler, and encoder. Table 4 presents the area and power breakdown of each component in PRESTE single core.

Analysis on Fine-tuning. We analyze area efficiency and energy efficiency to run the fine-tuning across various models and numerical formats, with sequence lengths set to 1024 and batch sizes to 4. Fig. 12 illustrates the TOPS/mm^2 and TOPS/W for forward and backward propagation, normalized to Bfloat16 baseline for each model. The PRESTE configuration includes 2.5× more number of compute unit than Bfloat16 configuration and 1.67× more unit than MXFP, resulting in higher throughput. Moreover, decreased bit precision of PRESTE minimizes off-chip DRAM access, thereby lowering energy

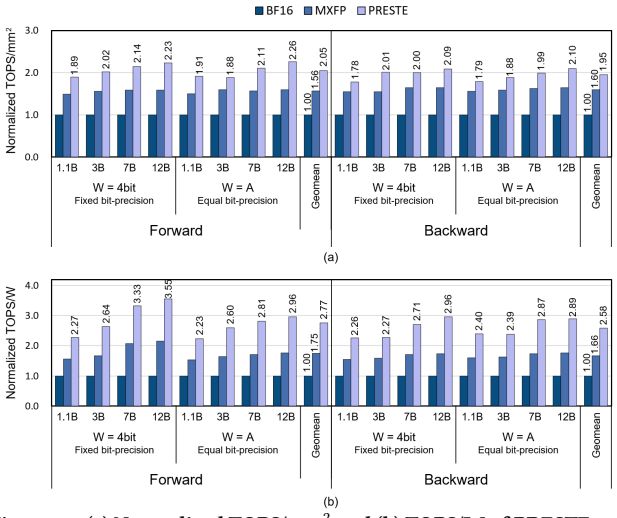


Figure 12: (a) Normalized TOPS/mm^2 and (b) TOPS/W of PRESTE and baseline designs during fine-tuning.

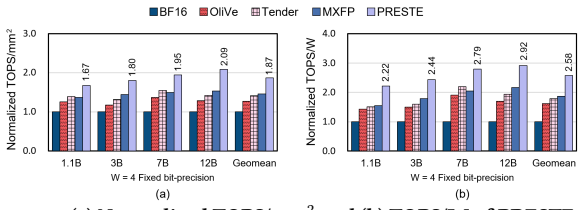


Figure 13: (a) Normalized TOPS/mm^2 and (b) TOPS/W of PRESTE and baseline designs during inference.

consumption. Overall, PRESTE achieves a 2.77× TOPS/W improvement for the forward propagation, and 2.58× TOPS/W improvement for the backward propagation.

Analysis on Inference. We also analyze area efficiency and energy efficiency during the inference, using a batch size of 8 with 1024 input tokens for each model. The evaluation of area efficiency and energy efficiency is based on the time taken to generate the 32th token. During inference, PRESTE achieves 1.28× higher TOPS/mm^2 than Tender, which uses multiple 4-bit MACs to compute a single 8-bit format, as shown in Fig. 13. Unlike baseline accelerators such as Tender and OliVe, which support inference only, PRESTE supports both inference and fine-tuning while outperforming them in inference efficiency.

6 CONCLUSION

In this paper, we propose PRESTE, a novel quantization approach designed to preserve the precision of small exponents with the goal of enhancing both inference and fine-tuning accuracy. The key idea is to preserve exponents smaller than threshold that are often truncated during the quantization process, thereby reducing the error in dot-product computations and improving overall accuracy. PRESTE-based LLM models show inference and fine-tuning performance comparable to the Bfloat16-based LLM models. Additionally, we propose PRESTE hardware that reduces the overhead associated with alignments and accumulators in the conventional FP dot-product operations. PRESTE hardware achieves area efficiency improvement of 1.67-2.26× and energy efficiency improvement of 2.22-3.55× compared to Bfloat16 hardware for inference and fine-tuning process.

REFERENCES

[1] Marco Bellagente, Jonathan Tow, Dakota Mahan, Duy Phung, Maksym Zhuravinskyi, Reshinth Adithyan, James Baicoianu, Ben Brooks, Nathan Cooper, Ashish Datta, et al. 2024. Stable LM 2 1.6 B Technical Report. *arXiv preprint arXiv:2402.17834* (2024).

[2] Jack Choquette. 2023. Nvidia hopper h100 gpu: Scaling performance. *IEEE Micro* 43, 3, 9–17.

[3] 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 preprint arXiv:1803.05457* (2018).

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems* 36 (2024).

[5] 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*. 1–15.

[6] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*. PMLR, 2790–2799.

[7] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.

[8] Hyesung Jeon, Yulhwa Kim, and Jae-joon Kim. 2024. L4q: Parameter efficient quantization-aware training on large language models via lora-wise lsq. *arXiv preprint arXiv:2402.04902* (2024).

[9] Himanshu Kaul, Mark Anders, Sanu Mathew, Seongjong Kim, and Ram Krishnamurthy. 2019. Optimized fused floating-point many-term dot-product hardware for machine learning accelerators. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 84–87.

[10] 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*. 1–6.

[11] Tomás Lang and Javier D Bruguera. 2002. Floating-point fused multiply-add with reduced latency. In *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 145–150.

[12] Jehun Lee and Jae-Joon Kim. 2025. Integer Unit-Based Outlier-Aware LLM Accelerator Preserving Numerical Accuracy of FP-FP GEMM. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–7.

[13] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. 2024. Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization. *arXiv preprint arXiv:2406.12930* (2024).

[14] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109.

[15] Chang Liu and Jun Zhao. 2024. Resource allocation for stable llm training in mobile edge computing. In *Proceedings of the Twenty-fifth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. 81–90.

[16] David R Lutz, Anisha Saini, Mairin Kroes, Thomas Elmer, and Harsha Valsaraju. 2024. Fused FP8 4-Way Dot Product With Scaling and FP32 Accumulation. In *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. IEEE, 40–47.

[17] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]

[18] Paulius Micikevicius, Stuart Oberman, Pradeep Dubey, Marius Cornea, Andres Rodriguez, Ian Bratt, Richard Grisenthwaite, Norm Jouppi, Chiachen Chou, Amber Huffman, et al. 2023. OCP 8-bit Floating Point Specification (OFP8). *Open Compute Project* (2023).

[19] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. 2022. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*.

[20] Microsoft. 2024. MX Pytorch Emulation Library. https://github.com/microsoft/microxcaling

[21] Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. 2023. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*.

[22] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

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

[24] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM* 64, 9 (2021), 99–106.

[25] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*.

[26] Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. 2024. Privacylens: Evaluating privacy norm awareness of language models in action. *Advances in Neural Information Processing Systems* 37 (2024), 89373–89407.

[27] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2023. Large language models encode clinical knowledge. *Nature* 620, 7972 (2023), 172–180.

[28] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. 2019. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. *Advances in neural information processing systems* 32.

[29] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937* (2018).

[30] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

[31] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

[32] Jonathan Tow, Marco Bellagente, Dakota Mahan, and Carlos Riquelme. 2024. StableLM 3B 4E1T. https://huggingface.co/stabilityai/stablelm-3b-4e1t

[33] Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Wang, Sanchari Sen, Jintao Zhang, Ankur Agrawal, Monodeep Kar, Shubham Jain, Alberto Mannari, Hoang Tran, et al. 2021. RaPiD: AI accelerator for ultra-low precision training and inference. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 153–166.

[34] Herbert Woisetschläger, Alexander Erben, Shiqiang Wang, Ruben Mayer, and Hans-Arno Jacobsen. 2024. Federated fine-tuning of llms on the very edge: The good, the bad, the ugly. In *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning*. 39–50.

[35] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 38–45.

[36] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*. PMLR, 38087–38099.

[37] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. 2023. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*.

[38] Ning Yang, Zongwu Wang, Qingxiao Sun, Liqiang Lu, and Fangxin Liu. 2025. PISA: Efficient Precision-Slice Framework for LLMs with Adaptive Numerical Type. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.

[39] Zhongzhi Yu, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. 2024. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[40] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).

[41] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. 2024. Llmcompass: Enabling efficient hardware design for large language model inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1080–1096.

[42] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385* (2024).