

FP8-Flow-MoE: A Casting-Free FP8 Recipe without Double Quantization Error

Fengjuan Wang, Zhiyi Su, Xingzhu Hu, Cheng Wang, Mou Sun[†]

Zhejiang Lab

Abstract

Training large Mixture-of-Experts (MoE) models remains computationally prohibitive due to their extreme compute and memory demands. Although low-precision training promises to accelerate computation and reduce memory footprint, existing implementations still rely on BF16-dominated dataflows with frequent quantize-dequantize (Q/DQ) conversions. These redundant casts erode much of FP8’s theoretical efficiency. However, naively removing these casts by keeping dataflows entirely in FP8 introduces double quantization error: tensors quantized along different dimensions accumulate inconsistent scaling factors, degrading numerical stability.

We propose FP8-Flow-MoE, an FP8 training recipe featuring a quantization-consistent FP8-centric dataflow with a scaling-aware transpose and fused FP8 operators that streamline computation and eliminate explicit cast operations from 12 to 2. Evaluations on a 671B-parameter MoE model demonstrate up to 21% higher throughput and 16.5 GB lower memory usage per GPU compared to BF16 and naïve FP8 baselines, while maintaining stable convergence. We provide a plug-and-play FP8 recipe compatible with TransformerEngine and Megatron-LM, which will be open-sourced soon.

1 Introduction

The development of large language models (LLMs) has witnessed remarkable progress in recent years, with model scales growing to hundreds of billions and even trillions of parameters Brown et al. (2020); Zhang et al. (2022); Meta AI Team (2023). These massive models have demonstrated unprecedented capabilities across natural language understanding, reasoning, and knowledge-intensive tasks OpenAI (2023); Bubeck et al. (2023). However, training such large-scale models poses significant computational challenges, and the enormous cost has become a de facto barrier for many research institutions and even large industrial labs.

To mitigate this, the Mixture-of-Experts (MoE) architecture Lepikhin et al. (2021); Fedus et al. (2022); Rajbhandari et al. (2022) has been introduced to increase model capacity without proportionally increasing computation, activating only a subset of experts per token. While MoE significantly reduces the overall training cost compared to dense transformers, its large communication footprint and memory demand still make training at scale highly expensive—motivating the exploration of low-precision formats such as FP8 to further improve efficiency.

Low-precision computation has emerged as a promising direction to address this challenge. Compared to the mainstream BF16-based Automatic Mixed Precision (AMP) scheme, adopting the FP8 format can theoret-

ically double computation throughput and halve communication latency Micikevicius et al. (2022). Since the introduction of NVIDIA’s Hopper architecture with native FP8 Tensor Core support NVIDIA Corporation (2022), FP8 training has drawn increasing attention. DeepSeek first demonstrated the feasibility of FP8 precision at massive scale in its V3 Mixture-of-Experts (MoE) model DeepSeek-AI Team (2024), and subsequently open-sourced *DeepGEMM* DeepSeek-AI Team (2025b) and *DeepEP* DeepSeek-AI Team (2025a)—two critical kernels for FP8 MoE efficiency—enabling high-throughput grouped GEMM computation and low-latency cross-node all-to-all communication.

Despite these advances, a complete training recipe designed around an FP8 dataflow has been missing. Naively replacing BF16 kernels with their FP8 counterparts introduces **excessive casting overheads**—each GEMM boundary typically involves multiple quantize-dequantize (Q/DQ) conversions DeepSeek-AI Team (2024); NVIDIA Corporation (2023); Zhao et al. (2025). As a result, the realized throughput of current FP8 systems often falls short of their theoretical potential, sometimes even lagging behind optimized BF16 baselines. Moreover, without a properly designed recipe and careful memory management, FP8 kernels may incur extra activation copies, undermining the expected memory savings.

However, simply removing these casts is not a viable

[†] Correspondence to: Mou Sun <123ssssmm@gmail.com>.

solution. A straightforward strategy is to keep operator inputs and outputs directly in FP8—bypassing BF16 storage and boundary casting—to avoid Q/DQ overhead. While this approach reduces conversions, it introduces **double quantization error**: tensors quantized along different dimensions (e.g., row-wise before GEMM and column-wise afterward) accumulate inconsistent scaling factors, leading to directional precision loss and potential instability Dettmers et al. (2022).

To address these challenges, we propose **FP8-Flow-MoE**, a systematic and practical FP8 training recipe centered on an end-to-end FP8 dataflow. Our key contributions are summarized as follows:

1. **Avoidance of double quantization error in FP8 dataflows.** We eliminate quantization inconsistency by introducing a **scaling-aware transpose** operator that directly converts FP8 tensors between row- and column-wise layouts while preserving scaling consistency. This design removes the need for dequantize–requantize cycles across operators, effectively avoiding double quantization error in FP8 dataflows.
2. **Casting-free FP8 recipe with a native FP8 kernel suite.** We present the FP8-Flow-MoE recipe, where the entire MoE stage operates in FP8 with no explicit casting except at the entry point. Compared to a drop-in kernel replacement approach, the number of cast operations is reduced from 12 to 2, marking a paradigm shift from BF16-centric to FP8-centric computation. We further provide a suite of native FP8 operators that make the recipe feasible and efficient in practice. Several of these kernels have already been merged into upstream repositories (e.g., NVIDIA/TransformerEngine NVIDIA Corporation (2023)), while others will be released as part of our open-source implementation.
3. **Comprehensive analysis of FP8 training at system scale.** We conduct an in-depth study comparing FP8 and BF16 across MoE components, analyzing both kernel-level and quantization-related effects. FP8-Flow-MoE achieves up to 21% throughput improvement and 7.4 GB memory reduction per GPU compared to the BF16 baseline, while naive FP8 kernel replacement yields only an 3% gain with negligible or even negative memory savings.

The remainder of this paper is organized as follows. Section 2 reviews related work on low-precision training techniques for LLMs and the architectural design of Mixture-of-Experts models. Section 3 details how the scaling-aware transpose operator eliminates double-quantization error and enables a casting-free FP8 dataflow, supported by a suite of high-performance FP8 kernels

integrated into FP8-Flow-MoE. Section 4 compares FP8-Flow-MoE, BF16, and naïve FP8 on a 671B DeepSeek-V3 model for compute, memory, and communication efficiency, and validates convergence on a 16B DeepSeek-V2-lite model trained with 200B tokens. Finally, Section 5 concludes the paper and discusses potential directions for extending FP8-Flow-MoE to other low-precision formats and representations.

2 Background and Related Work

2.1 Low-Precision Training and FP8 Quantization

Low-precision computation has become a key enabler for efficient large-scale model training. Early FP16 and BF16 mixed-precision schemes significantly reduced training cost while maintaining convergence Micikevicius et al. (2018); Wang et al. (2018). Recently, the FP8 format has gained attention for its ability to double arithmetic throughput on modern accelerators NVIDIA Corporation (2022); Micikevicius et al. (2022); DeepSeek-AI Team (2024). FP8 generally follows IEEE 754 conventions but defines multiple variants, most notably E4M3 and E5M2, which balance precision and dynamic range differently Wang et al. (2018). E4M3 offers finer precision, while E5M2 provides a wider numeric range. Another variant, UE8M0, encodes powers of two and is typically used for scaling factors rather than activations.

Converting higher-precision tensors (e.g., FP32 or BF16) into FP8 requires quantization. The scaling strategy defines the quantization recipe and directly impacts stability and accuracy. Early works relied on per-tensor scaling Micikevicius et al. (2018), which often underutilizes FP8’s range when tensor values span several magnitudes. Later studies proposed finer-grained scaling, such as per-block or per-channel quantization Peng et al. (2023), which preserve accuracy for small-magnitude values while maintaining efficiency. These techniques have become standard in recent FP8 training systems.

2.2 FP8 Training Systems and MoE Architectures

The adoption of FP8 was accelerated by both hardware and software advances. NVIDIA’s Hopper architecture introduced native FP8 Tensor Core support NVIDIA Corporation (2022), while TransformerEngine provided open-source operator-level support for FP8 quantization NVIDIA Corporation (2023). Early large-scale studies, including simulated FP8 GPT-3 training Micikevicius et al. (2022) and Inflection-2 Inflection AI (2023), demonstrated that FP8 can maintain convergence comparable to BF16. DeepSeek-V3 DeepSeek-AI Team (2024) further validated FP8 training at trillion-parameter scale and released two key libraries—DeepGEMM DeepSeek-AI Team (2025b) for grouped-GEMM and DeepEP DeepSeek-AI Team (2025a) for cross-node communication—establishing a strong FP8 baseline for Mixture-of-Experts (MoE) models.

However, these implementations still follow BF16 dominated dataflows, requiring frequent quantize-dequantize (Q/DQ) conversions across GEMM boundaries Zhao et al. (2025). Such redundant casting limits realized FP8 performance and increases memory traffic. Conversely, removing these casts introduces double quantization error, as tensors quantized along different dimensions (row-wise vs. column-wise) accumulate inconsistent scaling factors Dettmers et al. (2022). This trade-off between computational efficiency and numerical stability remains a major challenge in current FP8 systems.

2.3 System-Level Challenges in Mega-Scale MoE Training

The rapid scaling of large language models (LLMs) Brown et al. (2020); Meta AI Team (2023) has pushed training infrastructure to its limits. Parallelization strategies such as tensor, pipeline, and data parallelism Rajbhandari et al. (2022), together with the Mixture-of-Experts (MoE) architecture Lepikhin et al. (2021); Fedus et al. (2022), have become essential for trillion-parameter training. MoE reduces active computation by routing each token to a small subset of experts, achieving high model capacity without proportional cost. However, the sparse and irregular computation pattern introduces new bottlenecks: all-to-all expert communication, activation padding, and dynamic routing overheads Rajbhandari et al. (2022); DeepSeek-AI Team (2024). Integrating FP8 precision into such workflows is particularly challenging, as each communication or expert boundary typically requires additional quantization and data reformatting. These factors fragment computation into many small kernels and diminish arithmetic intensity, preventing FP8’s theoretical benefits from being fully realized in end-to-end MoE training.

These challenges motivate our FP8-Flow-MoE design—a quantization-consistent, casting-free FP8 dataflow that eliminates redundant Q/DQ operations and maintains numerical stability throughout the MoE pipeline.

3 Method

There have been extensive studies on applying FP8 quantization in transformer-based model training, particularly, dense models such as BERT, GPT, etc., achieving significant performance gain with negligible accuracy loss. This contribution focus on another archetype of models, Mixture of Experts (MOE) models, which has emerged as the most performant LLM because its high capability and reduced resource required for inference, i.e. only small parts of all experts are activated.

3.1 Scaling-aware transpose

One of the key challenges in fully adopting FP8 precision in the MoE module is the two different quantization layouts required by the *grouped linear* operations. Specifically, the activations (and their gradients) are consumed in a row-wise quantized format (referred to as *Fprop* and

Dgrad in Figure 2), while the weight-gradient computation requires column-wise quantized inputs (*Wgrad*). However, the preceding all-to-all communication (*dispatch*) typically transmits only one format, usually the row-wise quantized data, to leverage the reduced-communication advance brought by using a low-precision format.

As a consequence, when the grouped linear layer subsequently needs column-wise quantized data, a naive implementation must first dequantize → transpose → requantize, introducing two rounds of (de)quantization. This process leads to what we call the double quantization error, whose root cause is that 1D per-token quantization uses scaling factors computed over contiguous segments (e.g., 128 elements per scale), and the scales differ across the two layouts. Since FP8 quantization maps scaled values onto a discrete representable grid determined by the exponent and mantissa bits, performing two independent quantizations with different scaling factors effectively remaps values twice to non-overlapping discrete sets. Formally, the double quantization error is defined by

$$E = Q_{\text{col}}(D(Q_{\text{row}}(X)) - Q_{\text{col}}(X)) \quad (1)$$

where $Q_{\text{row}}(\cdot)$, $Q_{\text{col}}(\cdot)$ are the row-wise and column-wise quantization operator and $D(\cdot)$ is the de-quantization operator. Both row-wise and column-wise quantization are performed in a per-tile fashion, with 128 continuous elements in each tile. in other words, the scaling factor is calculated as

$$s = \frac{\max_{0 \leq i < 128} \|x_i\|}{448} \quad (2)$$

where 448 is the maximum representable number of FP8_E4M3 format. After scaling, a quantization operator essentially maps x_i into the nearest discrete value

$$Q_{\text{row}}(x_i) = \text{round}\left(\frac{x_i}{s}\right) \quad (3)$$

A de-quantization operator scales the FP8 data back to its original range by multiplying the scaling factor, but the rounding error remains,

$$D(Q_{\text{row}}(x_i)) = \text{round}\left(\frac{x_i}{s}\right) \cdot s \quad (4)$$

Following this representation, it is trivial that performing a second row-wise quantization do not introduced any extra error since the same 1x128 elements are again tiled for quantization, which gives same maximum value and eventually leaving s unchanged. An number is strictly mapped to the identical discrete value, or formally,

$$\begin{aligned}
Q_{row}(D(Q_{row}(x_i))) &= \text{round}\left(\frac{\text{round}\left(\frac{x_i}{s}\right) \cdot s}{s}\right) \cdot s \quad (5) \\
&= \text{round}(\text{round}\left(\frac{x_i}{s}\right)) \cdot s \quad (6) \\
&= \text{round}\left(\frac{x_i}{s}\right) \cdot s \quad (7) \\
&= Q_{row}(x_i) \quad (8)
\end{aligned}$$

where we assumed a deterministic rounding algorithm, e.g. round to nearest (RtN), is used and rounding the same number twice does not change its value.

However, in case that the second quantization is performed column-wise, its scaling factor will be calculated from the 128-element tile after transpose, marked as s' , which is generally not the same as s . We have

$$Q_{col}(D(Q_{row}(x_i))) = \text{round}\left(\frac{\text{round}(x_i/s) \cdot s}{s'}\right) \cdot s' \quad (9)$$

where the two rounding operators cannot be combined because they are applying to two different values. However, This error amplification can be mitigated if the scaling factor is constrained to powers of two, in which case, rescaling between two quantization domains (row-wise and column-wise) involves only adjusting exponent bits, provided that no overflow or underflow occurs. Formally, let us assume

$$s = 2^T, s' = 2^{T'}, \text{ where } T \in \mathbb{N}, T' \in \mathbb{N}.$$

At row-wise quantization, the value of an element is discretized to

$$Q_{row}(x) = (-1)^{SN} \cdot 2^{E-7} \cdot (1 + M/8) \cdot 2^T \quad (10)$$

where SN , E and M are the sign bit, exponent bits and matissa bits of the its FP8 coding respectively, 7 and 8 are constants derived from FP8_E4M3 definition. While applying column-wise quantization, the same value is discretized to a different representation,

$$Q_{col}(x) = (-1)^{SN'} \cdot 2^{E'-7} \cdot (1 + M'/8) \cdot 2^{T'} \quad (11)$$

Since both T and T' are natural numbers, there exist another natural number D , where $D = T' - T$. Notice that if we set

$$\begin{aligned}
SN' &= SN \\
E' &= E - D \\
M' &= M
\end{aligned}$$

, we have

$$Q_{col}(x) = (-1)^{SN} \cdot 2^{E-D-7} \cdot (1 + M/8) \cdot 2^{D+T} \quad (12)$$

$$= (-1)^{SN} \cdot 2^{E-D-7+D+T} \cdot (1 + M/8) \quad (13)$$

$$= (-1)^{SN} \cdot 2^{E-7+T} \cdot (1 + M/8) \quad (14)$$

$$= (-1)^{SN} \cdot 2^{E-7} \cdot (1 + M/8) \cdot 2^T \quad (15)$$

$$= Q_{row}(x) \quad (16)$$

$$(17)$$

Building on this insight, we further align all scaling factors within a transposed block to the largest one (to avoid overflow) and directly convert between layouts by exponent manipulation alone. In other words, we can transform row-wise quantized FP8 tensors into column-wise quantized ones without any dequantization or re-quantization, by simply modifying the exponent bits of the FP8 encodings. This principle underlies our Direct Transpose operator, as outlined in Algorithm 1.

Algorithm 1 Scaling-aware FP8 Transpose

Input: row-wise data $X_{i,j}^{row}$, scaling factor $S_{i,j}^{row}$
 Initialize column-wise data $X_{i,j}^{col} = X_{j,i}^{row}$
for each 128×128 block **do**
 $S_{max} = \max_{0 \leq i < 128} S_{i,j}^{row}$,
 $S_{i,j}^{col} = S_{max}$
for each element $X_{i,j}^{col}$ in block **do**
 calculate SN , E and M from $X_{i,j}^{col}$
 $k = \log_2(S_{max})/S_{i,j}^{col}$,
 $E_{new} = E - k$
 reassemble $X_{i,j}^{col}$ coding using SN , E_{new} and M
end for
end for

To validate the efficiency benefits of this operator, we conducted numerical experiments at various input dimension. For each, we compared two strategies to acquire the column-wise quantized data:

- (1) **Naive conversion:** dequantize \rightarrow transpose \rightarrow quantize method.
- (2) **Direct Transpose (ours):** exponent bits manipulation without (de)quantization.

The latency results are illustrated in Figure 1, it is clear that our Direct Transpose operator delivers 2 to 3 times speedup across all tensor shapes, confirming its effectiveness in computational efficiency.

3.2 Casting-free FP8 dataflows

The MoE architecture has emerged as an effective approach to scale model capacity without proportionally increasing the computational footprint. Within each MoE

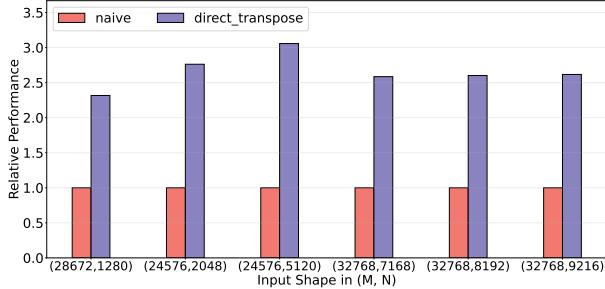


Fig. 1: Comparison of latency using different quantization strategies.

layer, a router dynamically assigns input tokens to a small subset of experts, where each expert typically consists of a two-layer feed-forward network. From a dataflow perspective, the computation of a single MoE layer can be decomposed into the following stages: routing, dispatch, permutation, expert computation, unpermutation, and combination. The expert computation itself usually includes two grouped linear projections separated by a nonlinearity (e.g., SwiGLU). In large-scale implementations, these projections are executed as grouped GEMM operations to exploit hardware efficiency across multiple experts.

Under conventional BF16-based training, the end-to-end dataflow involves multiple data format conversions across these stages. Tokens are first routed and dispatched through an all-to-all communication primitive, then permuted and packed into contiguous expert batches for efficient GEMM computation. Each expert executes the first grouped linear transformation, followed by activation, a second grouped linear transformation, and finally the unpermutation and combination steps to aggregate the outputs. This structure defines the core of the MoE layer’s computational graph.

When extending this flow to FP8 computation, existing frameworks primarily adopt conservative integration strategies. For instance, NVIDIA’s Transformer Engine implements a blockwise FP8 recipe, where quantization and computation in FP8 are confined strictly within the *grouped_linear* modules. This design allows FP8 acceleration in matrix multiplications (grouped GEMMs), achieving higher arithmetic throughput compared to BF16, but leaves all communication and data movement stages—particularly the dispatch all-to-all operations—in higher precision formats. As a result, while the GEMM kernels benefit from FP8 tensor cores, the overall MoE layer performance remains bounded by high-precision communication and frequent format conversions.

Another representative implementation is the DeepSeek-V3 training system, which demonstrated successful large-scale FP8 MoE training with state-of-the-art

performance. Although the complete training framework was not fully open-sourced, the available descriptions and accompanying releases—specifically DeepGEMM and DeepEP—suggest that both expert computation and inter-expert communication were accelerated using customized, high-performance FP8 kernels. However, the overall dataflow still incurs substantial quantization overhead. Our analysis shows that such an approach introduces up to twelve quantization/dequantization operations within a single MoE forward-backward pass. Moreover, in the weight gradient computation (wgrad) stage, FP8 activations are typically quantized row-wise, dequantized, and then re-quantized in a column-wise manner. Since these are per-token quantization schemes with their scaling happens along a particular direction, the repeated quantization steps can amplify rounding and scaling errors (“double quantization error”), negatively affecting numerical stability and convergence.

To fully exploit the computational advantages of FP8 while maintaining numerical stability, we design an FP8-centric dataflow, referred to as FP8-Flow-MoE, that minimizes format conversions and preserves FP8 representation across the entire MoE expert path whenever possible. In this design, both the input activations and output gradients remain in FP8 throughout forward and backward propagation, except for two specific boundaries:

1. between the output of the first *grouped_linear* and the activation function, and
2. between the second *grouped_linear* and the combination/dispatch operation in the backward pass.

These two exceptions correspond to fundamental numerical challenges for FP8 arithmetic. The first involves reduction operations (e.g., summations) that are highly susceptible to overflow under limited exponent range; the second involves nonlinear transformations (e.g., GELU or ReLU) that can amplify small quantization errors. Since these two computations are adjacent in the graph, we retain BF16 precision locally to ensure stability, while keeping all other tensors strictly in FP8 format. This design yields a near-continuous FP8 dataflow and substantially reduces quantization boundaries compared to prior FP8-in-MoE implementations.

3.3 High-Performance FP8 Kernels

To make the proposed FP8-centric computation flow practical and efficient, we further develop a comprehensive set of supporting operators specifically tailored to the data dependencies and numerical patterns within the MoE layer. These operators address key challenges that arise when applying FP8 to large-scale MoE training, including cumulative rounding and scaling errors, redundant data conversions, and schedule inefficiencies caused by fractured kernels. Together, they enable a seamless and

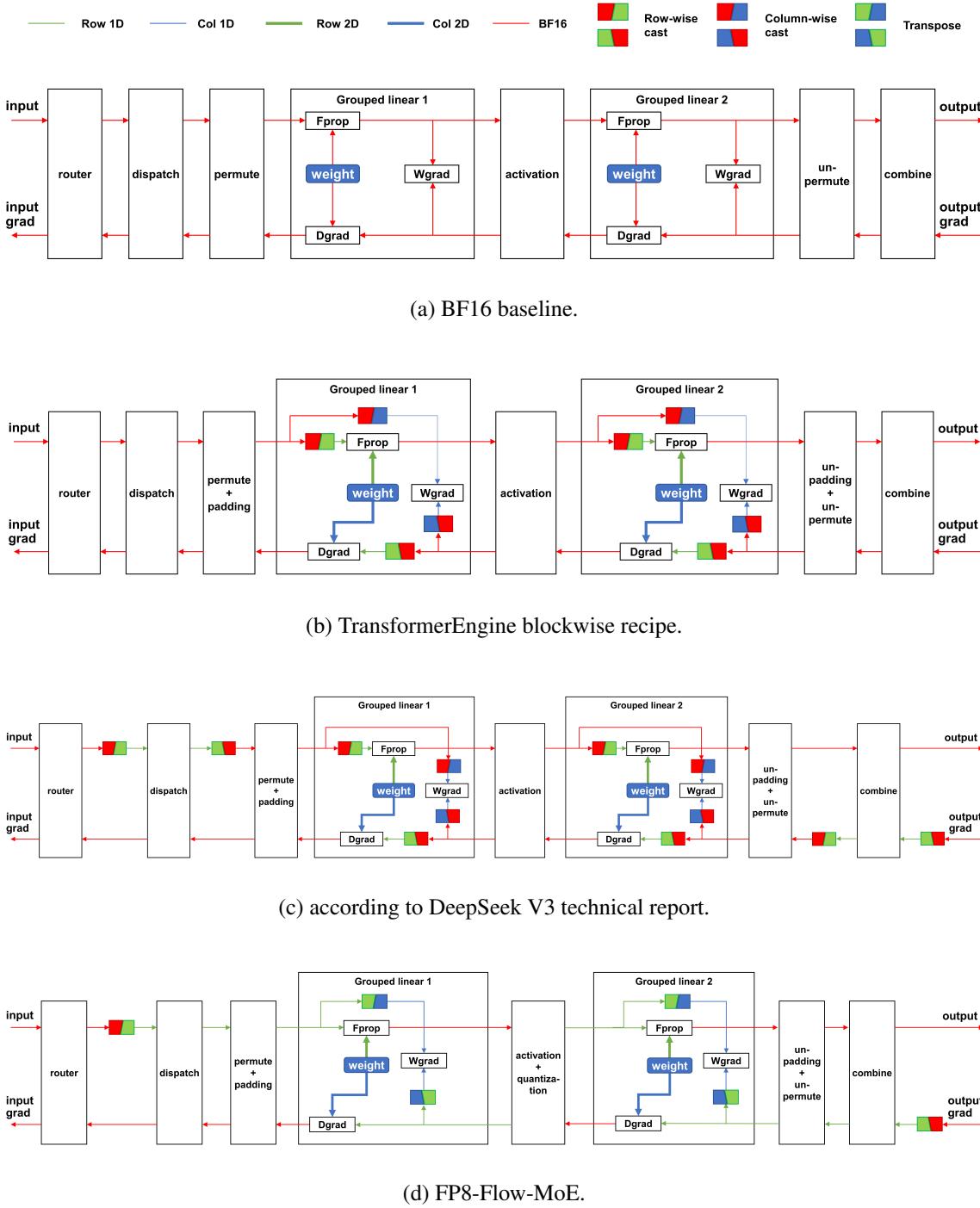


Fig. 2: The computational graph of an MoE module

high-throughput execution of the FP8-Flow-MoE, ensuring that the benefits of FP8 tensor cores extend beyond isolated kernels to the full end-to-end training pipeline.

To evaluate the effectiveness of these FP8 operators, we conduct a series of unit-level benchmarks that isolate each kernel’s performance and numerical behavior. The test inputs are generated through simulation, and their tensor shapes are chosen to reflect the real-world configurations of representative MoE models—DeepSeek V2-Lite, DeepSeek V2, and DeepSeek V3—under different expert parallelism (EP) settings. Specifically, These microbenchmarks allow us to directly measure raw kernel throughput, demonstrating how each operator contributes to the overall efficiency of the FP8-Flow-MoE system.

3.3.1 Fused Permute and Padding

In FP8 data pipeline, a *padding* step is introduced to satisfy the shape constraint of FP8 GEMM kernels—each expert’s input must contain a multiple of 16 entries to fully utilize tensor cores. Before this, a *permute* operation reorganizes dispatched tokens so that samples belonging to the same expert are contiguous in memory. Executing these two light-weight data movement kernels separately incurs redundant HBM accesses and kernel launches. Since both operations are element-wise and independent, they can be naturally fused into a single kernel.

Based on these observations, we design a Fused Permute+Padding operator that directly performs expert-wise reordering and shape alignment in one pass. The fusion is implemented through a thread-block mapping scheme that dynamically computes the target offset in the padded layout while streaming reordered input elements from global memory. Similarly, in the backward propagation phase, we apply the same fusion strategy to combine unpermute and unpadding operations into a single kernel. This symmetric fusion ensures consistent data handling between forward and backward flows.

Figure 4 reports the performance comparison between the fused and unfused implementations under both BF16 and FP8 precision settings. Across different tensor sizes corresponding to typical MoE workloads, the fused version demonstrates a consistent acceleration. For the forward permute+padding case, we observe up to 1.7 \times speedup, while for the backward unpermute+unpadding kernel, the improvement reaches as high as 6.6 \times on large-scale configurations.

3.3.2 Fused SwiGLU and Quantization

The motivation for designing an FP8-centric training flow originates from a key performance observation: naively inserting FP8 kernels (e.g., high-performance GEMM and communication) into a BF16 computation graph introduces frequent quantization and dequantization operations, which can substantially offset the benefits of FP8 acceleration.

To illustrate this issue, we analyze FP8 communica-

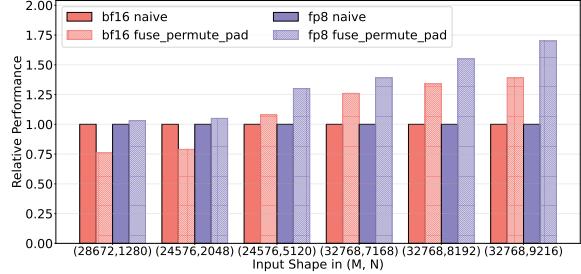


Fig. 3: Performance comparison between fused and separate implementation of permute and padding kernels.

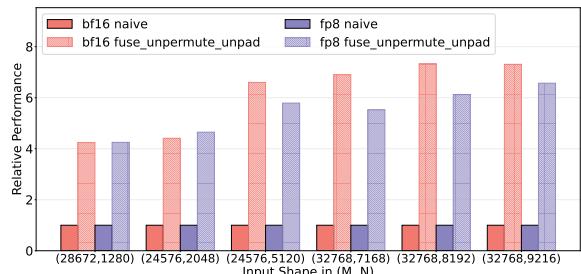


Fig. 4: Performance comparison between fused and separate implementation of un-permute and un-padding kernels.

tion using the DeepEP library released by DeepSeek. In its default usage, quantization and dequantization are performed immediately before and after the communication. Table 1 reports the measured latency under three representative data scales (corresponding to DeepSeek V2-Lite, V2, and V3) and three expert-parallel (EP) degrees (8, 16, 32). The results reveal two key findings: for smaller-scale workloads, quantization and dequantization each consume a similar amount of time as the communication kernel, nearly eliminating any performance gain; While the communication cost grows roughly linearly with workloads, the (de)quantization cost remains constant, however, such data conversion leads to a reduction of end-to-end FP8 speedup from 1.6 \times to 1.4 \times .

It is worth noting that although FP8 communication theoretically halves the data transfer size, in practice, the need to transmit both the FP8 tensor and its corresponding scaling factors doubles the number of data buffers and synchronizations, limiting achievable speedup to around 1.6 \times . Nevertheless, a single pair of quantization and dequantization operators reduces the performance gain of FP8 communication by roughly one third (from 0.6 \times to 0.4 \times improvement). Considering that a typical MoE forward or backward pass involves around three such pairs (see Figure 2(c)), the benefit of FP8 kernels can be almost completely neutralized.

To mitigate this issue, FP8-Flow-MoE integrates quantization directly into surrounding compute kernels when-

(M,N,EP)	BF16 (ms)	FP8 TIME (ms)			SPEEDUP	
		Q/D	COMM	ALL	COMM	ALL
(24576,2048,8)	0.537	0.127/0.084	0.325	0.535	1.65x	1.00x
(24576,5120,8)	0.785	0.087/0.089	0.526	0.703	1.49x	1.12x
(32768,7168,8)	1.276	0.086/0.089	0.905	1.080	1.41x	1.18x
(24576, 2048, 16)	1.224	0.091/0.083	1.176	1.350	1.04x	0.91x
(24576,5120,16)	2.213	0.082/0.082	1.400	1.564	1.58x	1.42x
(32768,7168,16)	2.934	0.084/0.092	1.847	2.023	1.59x	1.45x
(24576,2048,32)	3.005	0.094/0.083	2.740	2.918	1.10x	1.03x
(24576,5120,32)	5.003	0.082/0.081	2.868	3.031	1.74x	1.65x
(32768,7168,32)	7.327	0.082/0.082	4.319	4.483	1.70x	1.63x

Table 1: Communication Performance with Speedup

ever possible. In particular, we fuse the SwiGLU activation and quantization steps into a single kernel. Figure 5 presents the results of our unit tests across several representative tensor shapes. The fused implementation achieves nearly identical latency to the original standalone SwiGLU operator in NVIDIA’s Megatron framework, while seamlessly producing FP8-quantized outputs for subsequent computation. This demonstrates that FP8-Flow-MoE can preserve the numerical efficiency of mixed precision while removing the quantization bottleneck that previously limited end-to-end performance.

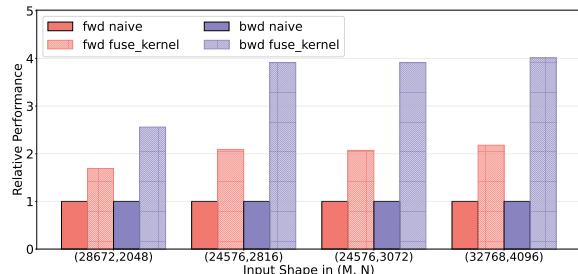


Fig. 5: Performance of fused SwiGLU and quantization kernel.

4 Experiments

We evaluate FP8-Flow-MoE from two perspectives: numerical convergence and training efficiency. All experiments are conducted on a 32-node NVIDIA Hopper cluster, where each node is equipped with eight GPUs (80 GB memory per GPU). Expert parallelism (EP) is scaled across nodes to assess training efficiency and scalability under realistic large-model workloads.

4.1 Convergence Validation

To verify that FP8-Flow-MoE maintains numerical stability and convergence behavior, we train a DeepSeek-V2-Lite model (16 B parameters) from scratch under two precision settings: BF16 and FP8-Flow-MoE. Both models are trained for 200 B tokens, using identical optimization hyperparameters, learning rate schedule, data ordering, and parallelism configurations.

As shown in Fig. 6, the two loss curves are nearly in-

distinguishable across the entire training trajectory—from the early warm-up phase through the stabilized loss regime. The FP8-Flow-MoE model shows no sign of divergence, gradient underflow, or numerical drift, demonstrating that our fp8-centric dataflow does not compromise convergence fidelity even under extended-scale workloads. This observation confirms that the scaling-aware transpose and fused operators in FP8-Flow-MoE preserve the numerical dynamics of BF16 training.

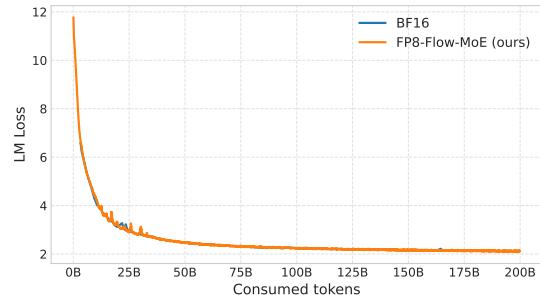


Fig. 6: Training loss comparison across 200 billion tokens

4.2 Efficiency Evaluation

We next benchmark end-to-end training efficiency across multiple precision and dataflow configurations. Specifically, we compare the following three representative configurations: **(1) BF16**: baseline configuration using standard FP32-BF16 mixed precision. **(2) Blockwise**: TE-style blockwise recipe with FP8 grouped GEMM kernels. **(3) FP8-Flow-MoE (ours)**: FP8-centric dataflow with fused operators and minimal quantization overhead.

All experiments are conducted on the DeepSeek-V3 (671B) model to validate the scalability and robustness of FP8-Flow-MoE under realistic large-scale training conditions. Models are trained using standard expert and pipeline parallelism configurations (EP/PP = 8/32, 16/16, 32/8) and evaluated under two activation checkpointing (AC) strategies: $AC = full$, which applies full checkpointing to all modules; and $AC = sel$ (+MoE expert), which selectively checkpoints the MoE layer while excluding experts to evaluate FP8 activation compression. The latter represents the most memory-efficient configuration compatible with the 1F1B-overlap schedule in Megatron, as checkpointing the entire MoE layer would otherwise cause OOM across all recipes. To isolate the effect of computation efficiency from communication overlap, all end-to-end experiments are performed without computation–communication overlap.

Throughput Analysis. Table 2 and 3 summarizes the measured throughput (TGS, tokens/GPU/s) and peak memory (Mem, GB) under three EP settings. Across all activation checkpointing modes, FP8-Flow-MoE consistently outperforms the BF16 baseline, achieving +6 % (EP8), +8 % (EP16), and +16 % (EP32) throughput im-

provements. Compared with the Blockwise recipe, FP8-Flow-MoE further improves throughput by 3 % (EP8), 8 % (EP16), and up to 21 % (EP32). The performance gap widens at higher EP levels, where the cumulative impact of reduced quantization overhead and kernel fusion becomes more significant.

Memory Efficiency. Under $AC = sel (+MoE\ expert)$ at EP8, FP8-Flow-MoE reduces peak memory by approximately 8 GB vs. BF16 and 16.5 GB vs. Blockwise, directly benefiting from the FP8 checkpoint compression mechanism, which stores intermediate activations in FP8 instead of BF16. At the largest scale of EP = 32, both the BF16 and Blockwise baselines encounter out-of-memory (OOM) errors, while FP8-Flow-MoE remains stable, clearly demonstrating its superior scalability and memory efficiency.

Method	EP8		EP16		EP32	
	TGS	Mem	TGS	Mem	TGS	Mem
BF16	1,109	39	939	36	671	43
Blockwise	1,146	37	938	41	644	51
FP8-Flow-MoE	1,176	37	1,012	39	779	49

Table 2: Throughput analysis under **AC=full**

Method	EP8		EP16		EP32	
	TGS	Mem	TGS	Mem	TGS	Mem
BF16	1,178	64	1,055	71	OOM	OOM
Blockwise	1,178	73	1,031	77	OOM	OOM
FP8-Flow-MoE	1,193	56	1,111	66	912	75

Table 3: Throughput analysis under **AC=sel (+MoE expert)**

4.3 Discussion and Insights

The experimental results collectively highlight three key findings:

Persistent FP8 dataflow matters. Simply replacing GEMM or communication kernels with FP8 counterparts yields limited benefit, since quantization/dequantization costs dominate at small batch sizes or high communication frequency. The proposed scaling-aware transpose and fused operator design eliminates redundant precision transitions, directly improving kernel launch efficiency and memory bandwidth utilization.

Scaling amplifies FP8-Flow-MoE’s gains. As EP increases, communication and quantization overheads compound in baseline methods, whereas FP8-Flow-MoE’s low-precision persistence minimizes these costs, resulting in up to $1.2\times$ end-to-end acceleration at large expert parallelism.

Practical training stability. Despite adopting lower precision for a larger portion of the training pipeline than

baseline implementations, FP8-Flow-MoE preserves identical convergence behavior and robustness across hundreds of billions of tokens. This stability stems from our insight of potential double quantization errors at transpose boundaries. By introducing a scaling-aware transpose, FP8-Flow-MoE effectively eliminates this redundant quantization while streamlining data movement, yielding both improved numerical fidelity and higher efficiency.

5 Conclusions

This study identifies two additional factors that play critical roles in achieving high end-to-end efficiency for FP8 precision in large-scale Mixture-of-Experts (MoE) training.

First, maintaining data in low precision throughout the computation flow minimizes redundant conversions and alleviates memory bandwidth pressure. A representative example is our FP8 transpose operator, which directly transforms row-wise to column-wise quantized tensors without reverting to higher precision—achieving high data throughput while avoiding double quantization error.

Second, FP8 precision inevitably fragments the computation graph by introducing small operators such as quantization and padding, leading to excessive kernel launches and memory traffic. By fusing lightweight operations and developing custom high-performance implementations, FP8-Flow-MoE effectively eliminates this structural inefficiency.

In summary, FP8-Flow-MoE establishes the first FP8-centric training paradigm for MoE models, maintaining convergence parity with BF16 while achieving up to 21% higher throughput and 16.5 GB lower peak memory per GPU compared with BF16 workflows and naïve FP8 kernel replacements. These results demonstrate the feasibility of fully low-precision MoE training and open new opportunities for scalable, energy-efficient foundation model development.

References

- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Tulio Ribeiro, M., and Zhang, Y. Sparks of Artificial General Intelligence: Early experiments with GPT-4. *arXiv e-prints*, art. arXiv:2303.12712, March 2023. doi: 10.48550/arXiv.2303.12712.
- DeepSeek-AI Team. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- DeepSeek-AI Team. Deepep: an efficient expert-parallel communication library. <https://github.com/deepseek-ai/DeepEP>, 2025a.
- DeepSeek-AI Team. Deepgemm: clean and efficient fp8 gemm kernels with fine-grained scaling. <https://github.com/deepseek-ai/DeepGEMM>, 2025b.
- Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30318–30332, 2022.
- Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.
- Inflection AI. Inflection-2: The next step up. <https://inflection.ai/inflection-2>, 2023.
- Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=qrwe7XHTmYb>.
- Meta AI Team. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv e-prints*, art. arXiv:2307.09288, July 2023. doi: 10.48550/arXiv.2307.09288.
- Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. *International Conference on Learning Representations (ICLR)*, 2018.
- Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., Mellemundi, N., Oberman, S., Shoeybi, M., Siu, M., and Wu, H. FP8 Formats for Deep Learning. *arXiv e-prints*, art. arXiv:2209.05433, September 2022. doi: 10.48550/arXiv.2209.05433.
- NVIDIA Corporation. Nvidia hopper architecture whitepaper. *Technical Whitepaper*, 2022.
- NVIDIA Corporation. Nvidia transformer engine documentation. <https://github.com/NVIDIA/TransformerEngine>, 2023.
- OpenAI. Gpt-4 technical report. *arXiv e-prints*, art. arXiv:2303.08774, March 2023. doi: 10.48550/arXiv.2303.08774.
- Peng, H., Wu, K., Wei, Y., Zhao, G., Yang, Y., Liu, Z., Xiong, Y., Yang, Z., Ni, B., Hu, J., et al. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*, 2023.
- Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18332–18346. PMLR, 17–23 Jul 2022.
- Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31, 2018.
- Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Singh Koura, P., Sridhar, A., Wang, T., and Zettlemoyer, L. OPT: Open Pre-trained Transformer Language Models. *arXiv e-prints*, art. arXiv:2205.01068, May 2022. doi: 10.48550/arXiv.2205.01068.
- Zhao, C., Deng, C., Ruan, C., Dai, D., Gao, H., Li, J., Zhang, L., Huang, P., Zhou, S., Ma, S., Liang, W., He, Y., Wang, Y., Liu, Y., and Wei, Y. Insights into deepseek-v3: Scaling challenges and reflections on hardware for ai architectures. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture, ISCA ’25*, pp. 1731–1745, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712616. doi: 10.1145/3695053.3731412. URL <https://doi.org/10.1145/3695053.3731412>.