Layout and Fusion Trade-offs for Mixture-of-Experts Inference under Single-Node Tensor Parallelism

June Yong Yang [*1] **Inhyuk Cho** [*1] **Taehyeon Kim** [1] **Yu Jin Kim** [1] **Moontae Lee** [1]

Abstract

Mixture-of-Experts (MoE) LLMs are increasingly deployed on a single multi-GPU node with tensor parallelism, a practical regime for private and on-premise deployments with bounded user populations, but one that has received less attention than datacenter-scale expert parallelism. This paper argues that this setting is not simply a smaller version of expert-parallel inference. The expert-major pipeline inherited from expert-parallel backends reshapes the bottlenecks of an MoE feed-forward layer under tensor parallelism: turning the pre-permute to an expert-major layout into a dominant cost at large token counts, making down-projection grouped GEMMs unusually thin, and creating a reduction chain between top-k combine and the tensor-parallel AllReduce. Through controlled kernel-level experiments on the MoE-FFN module, we find that the best layout flips with workload shape. Token-major variants win in decode-like regimes, while in-flight-permute variants with persistent down projection win in prefill-like regimes. Across representative MoE shapes, the best variant improves MoE FFN latency by up to nearly **2×** over the conventional expert-major baseline (pre-permute + expert-major up/down). Recent serving systems separate prefill and decode at the request and batch level. Our findings suggest that this separation should extend down to the MoE kernel layout itself, calling not for a single fused MoE kernel but for a workload-conditioned layout policy within the MoE FFN.

1. Introduction

Mixture-of-Experts (MoE) architectures (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022; Du et al., 2022) have dominated the open-weight frontier: DeepSeek-V3, Qwen3-235B-A22B, gpt-oss, and GLM-5 (Liu et al., 2024; Yang et al., 2025; Agarwal et al., 2025; Zeng et al., 2026). Activating only a small subset of experts per token improves the compute-to-parameter ratio, but makes inference more complex than dense-model serving. An MoE layer requires routing, token permutation, grouped expert GEMMs, top-k combination, and reduction across devices (Rajbhandari et al., 2022; Hwang et al., 2023; He et al., 2022).

MoE inference systems are typically optimized for one of two regimes. On-device and edge inference, characterized by very small batch sizes, leverages sparse weight loading (Yi et al., 2025) and aggressive compression (Frantar & Alistarh, 2024) to alleviate the memory-bandwidth bottleneck. Datacenter-scale serving targets high concurrency across many GPUs and nodes, using expert parallelism (EP) (Rajbhandari et al., 2022; He et al., 2022; Hwang et al., 2023) and prefill/decode disaggregation (Zhong et al., 2024; Patel et al., 2024) to maximize throughput. On high performance inference engines (Kwon et al., 2023; Zheng et al., 2024), recent MoE kernel backend libraries such as FlashInfer (Ye et al., 2025) and DeepGEMM (DeepSeek-AI, 2025) build around expert-major grouped GEMMs. Inputs are pre-permuted into expert-major order before up/gate, fed to expert-major (EE) grouped GEMMs for both up/gate and down, and reduced through a separate top-k combine. This pattern fits expert parallelism naturally, since the inter-rank all-to-all already arranges tokens by destination expert.

Between these extremes lies a practical but less-studied regime, serving large MoE models on a single multi-GPU node. This setting arises in private, on-premise, or compliance-sensitive deployments, where external serverless APIs may be impractical due to privacy, data residency, auditability, or procurement constraints. Such deployments must co-serve a mix of short decode-heavy chat with long prefill-heavy requests on the same hardware node, without relying on cross-node prefill/decode disaggregation. We will argue that this demand should propagate into MoE kernel layout design itself.

In such practice, MoE models are deployed with tensor parallelism (TP) (Shoeybi et al., 2019) because it reuses dense-

[*]Equal contribution [1]LG AI Research, Seoul, South Korea. Correspondence to: Moontae Lee <moontae.lee@lgresearch.ai>.

AdaptFM: Resource-Adaptive Foundation Model Inference Workshop at ICML 2026, Seoul, South Korea.

model serving infrastructure, relies on intra-node NVLink rather than cross-node fabric, and avoids device-level expert imbalance at low concurrency. TP-MoE is mechanically distinct from EP-MoE because it shards the expert intermediate dimension I across ranks rather than the expert set E across nodes. This shifts where the bottlenecks live.

First, expert-major permutation must move *full*-hidden activations before the column-parallel up/gate-projection shrinks the hidden dimensions. Second, TP Feed-Forward Network (FFN) decomposition makes row-parallel down-projection grouped GEMMs thin, since each rank reduces over only its local shard of the expert intermediate dimension, and MoE FFNs often already have smaller intermediate-to-hidden ratios than dense FFNs. Third, top-k expert outputs must be combined locally and then reduced across TP ranks, creating an adjacent pair of reductions that opens a fusion opportunity but also a synchronization risk.

These effects make single-node TP-MoE a distinct operating point, not a smaller version of datacenter EP. We ask how the MoE FFN kernel layout should be chosen given (i) thin TP down-projection GEMMs, (ii) the cost of full-hidden permutation before up/gate, and (iii) the combine plus AllReduce reduction chain, and how the answer changes across decode- and prefill-like workloads. We implement MoE FFN kernel variants in Triton and evaluate them under representative tensor-parallel configurations on a single 8×H200 node. **Our goal is not a new MoE architecture, router, or scheduler. We provide a controlled study of the layout and fusion trade-offs that determine single-node TP-MoE inference performance, and we argue for a workload-conditioned layout policy.**

We find that the optimal layout *flips* with workload shape. A token-major variant (TT) wins decode-like regimes with token count $T \leq 128$, while an in-flight-permute variant with persistent down projection (TE-pers) wins prefill-like regimes with $T \geq 4096$. Across representative MoE shapes, the best variant beats the FlashInfer-like EE baseline (pre-permute + expert-major up/down) by up to nearly 2× in MoE FFN latency at T=16384. The implication is that prefill/decode disaggregation should extend below the request scheduler to the MoE kernel layout itself.

Our contributions are as follows.

- We identify three TP-specific MoE FFN bottlenecks: thin down-projection GEMMs from sharding the expert intermediate dimension across TP ranks, full-hidden permute before column-parallel up/gate, and top-k combine adjacent to TP AllReduce.
- We benchmark persistent grouped GEMM (DeepSeek-AI, 2025), token-major up/gate, fused in-flight permute (vLLM Project, 2024), and combine/AllReduce overlap (Ye et al., 2025) against the EE baseline.
- We show a decode/prefill crossover (between T=128 and T=4096 on Qwen3-235B-A22B at TP=8) and argue for a workload-conditioned layout policy.

2. Background

2.1. Mixture-of-Experts

A Mixture-of-Experts (MoE) layer replaces the dense feed-forward network (FFN) in a Transformer (Vaswani et al., 2017) block with a set of expert FFNs and a learned router. Given input hidden states $X \in \mathbb{R}^{T \times H}$, where T is the number of tokens, H is the hidden dimension, and the experts have an intermediate dimension I, the router computes a score over E experts for each token. Each token is assigned to its top-k experts, producing expert indices and routing weights. The selected experts independently process the token, and their outputs are weighted and combined back into the original token order.

A typical MoE FFN forward pass consists of the following stages:

$$\begin{aligned} &\text{router} \rightarrow \text{top-}k \rightarrow \text{permute} \\ &\qquad \rightarrow \text{up/gate} \rightarrow \text{act.} \rightarrow \text{down} \rightarrow \text{combine}. \end{aligned} \tag{1}$$

We use *permute* for the within-rank layout conversion that reorders selected token activations into a layout suitable for expert computation, and reserve *dispatch* for the inter-rank all-to-all that EP backends use. The final combine stage includes scattering expert outputs back to token-major order and performing top-k reduction. Many implementations choose an expert-major layout, in which tokens assigned to the same expert are *contiguous*. This fits grouped GEMM kernels naturally, since each expert is one GEMM in the group.

For each expert e, the FFN computation typically follows the gated form

$$Z_e = \left[\phi\left(X_e W_{\text{gate}}^{(e)}\right) \odot \left(X_e W_{\text{up}}^{(e)}\right)\right] W_{\text{down}}^{(e)}, \tag{2}$$

where X_e denotes the tokens routed to expert e, ϕ is an activation such as SiLU or GELU, and $\odot$ denotes elementwise multiplication. The expert weights have shapes $W_{\text{up}}^{(e)}, W_{\text{gate}}^{(e)} \in \mathbb{R}^{H \times I}$ and $W_{\text{down}}^{(e)} \in \mathbb{R}^{I \times H}$. After the down projection, outputs are scattered back to their original token positions and combined using the router weights. For top-k routing with $k > 1$, multiple expert outputs may contribute to the same token and must be reduced.

Although the MoE FFN is GEMM-dominated in floating-point operations, its runtime is determined by more than GEMM throughput alone. Routing, permutation, grouped

GEMM scheduling, activation, optional quantization, scatter, and top-k reduction all enter the critical path. For serving, T at the FFN layer varies widely with workload phase even for a single deployment. Decode steps batch many requests' single new tokens together and produce moderate T at the FFN (e.g., $T \in \{8, 128\}$ at low or mid concurrency), while prefill processes batched prompt tokens together, producing large T. In both cases, the average load per expert is approximately Tk/E under balanced routing, so per-expert grouped-GEMM tiles remain small even when T is large (§3). We therefore parametrize layout choices by T rather than by the request phase.

2.2. Tensor Parallelism

Tensor parallelism (TP) (Shoeybi et al., 2019) partitions Transformer weights and intermediate activations across GPUs and is the standard choice for single-node dense-model serving on intra-node NVLink. We apply the same column-/row-parallel FFN decomposition to MoE layers, study plain TP without sequence parallelism, and shard each expert's intermediate dimension across all P ranks rather than assigning whole experts to ranks (the expert-sliced alternative would forfeit the dense-serving infrastructure reuse that motivates this regime).

Let P denote the TP degree. The up and gate projections are column-parallel: each rank owns a shard of the expert intermediate dimension and computes

$$U_{e,p} = X_e W^{(e)}_{\mathrm{up},p}, \tag{3}$$

$$G_{e,p} = X_e W^{(e)}_{\mathrm{gate},p}, \tag{4}$$

$$A_{e,p} = \phi(G_{e,p}) \odot U_{e,p}, \tag{5}$$

where $p \in \{1, \ldots, P\}$ indexes the TP rank. Since the intermediate dimension is partitioned, no TP reduction is required immediately after the up/gate projection.

The down projection is row-parallel. Each rank consumes its local intermediate shard and produces a partial hidden output:

$$Z_{e,p} = A_{e,p} W^{(e)}_{\mathrm{down},p}. \tag{6}$$

The partial outputs are then summed across TP ranks, typically using an AllReduce:

$$Z_e = \sum_{p=1}^{P} Z_{e,p}. \tag{7}$$

In MoE layers, this TP reduction interacts with top-k combine, because multiple expert outputs may also be reduced into the same token position.

Three TP-specific consequences drive the analysis in §3. (i) The down GEMM reduces over $K{=}I/P$, so the grouped GEMM of each rank is thinner than its EP-MoE counterpart. (ii) Any pre-permute into expert-major order moves full-hidden (H-wide) activations before the column-parallel up/gate has shrunk them. (iii) The local top-k combine and the cross-rank AllReduce form an adjacent pair of reductions whose fusion is an opportunity at small T but a synchronization risk at large T.

3. Bottlenecks and Trade-Offs in TP MoE

Tensor parallelism changes not only how MoE computation is distributed across GPUs, but also where the main bottlenecks appear. The intermediate dimension of expert projections is split across ranks, making some expert GEMMs thinner. Expert-major dispatch can move full-hidden activations before the column-parallel up/gate projection reduces the intermediate dimension. Finally, top-k combine is followed by a TP collective, creating a sequence of local and cross-rank reductions. These effects make single-node TP MoE serving a distinct optimization problem: the main performance question is not only how fast each expert GEMM is in isolation, but how layout movement, grouped GEMM shape, local reduction, and tensor-parallel communication interact across decode- and prefill-like workloads.

3.1. Layout Conversion

Many MoE implementations use an expert-major layout for grouped GEMM. After routing, selected token activations are rearranged so that tokens assigned to the same expert are contiguous. The conventional up/gate path is

$$\begin{aligned}\text{token-major input} &\rightarrow \text{expert-major dispatch} \\ &\rightarrow \text{gate/up grouped GEMM}.\end{aligned} \tag{8}$$

This layout is convenient because each expert can be processed as one GEMM in a grouped GEMM kernel. However, under TP, this dispatch occurs before the column-parallel up/gate projection reduces the intermediate activation size. Therefore, the dispatch can move full-hidden activations of dimension H. With top-k routing, the same input token may also be copied multiple times, once for each selected expert.

This creates a memory-bound bottleneck that can dominate the up/gate path, particularly when T is small or moderate. In such cases, the grouped GEMM may not be large enough to amortize the cost of full-hidden dispatch. The effect becomes more pronounced as hidden size or top-k increases.

An alternative is to keep the input token-major and avoid the full-hidden dispatch. We distinguish two such variants by the (input, intermediate) layout pair for the up/gate kernel: a **TT** kernel reads token-major input and writes a token-major intermediate, so the down projection also operates on token-major data; a **TE** kernel reads token-major input but writes an expert-major intermediate via an in-flight permute fused into up/gate, feeding an EE down projection without further

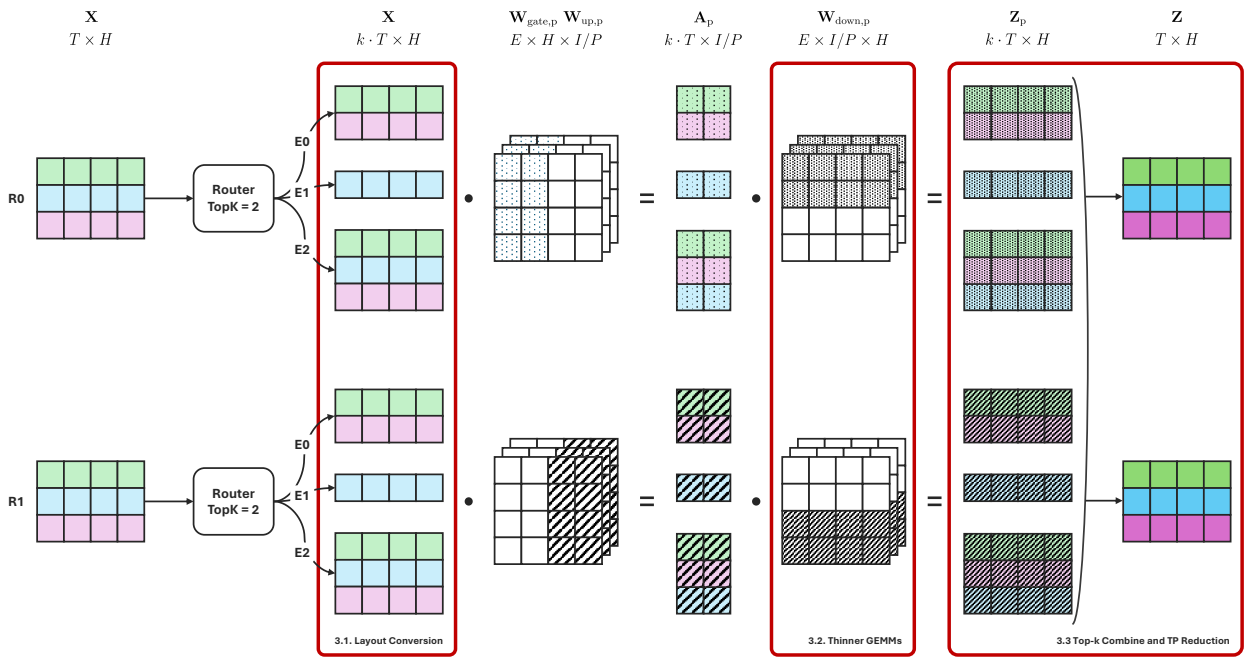

Figure 1. Overview of bottlenecks and kernel trade-offs in TP MoE FFN layers. The figure summarizes the three main bottlenecks studied in §3: full-hidden expert-major dispatch before up/gate projection, TP-induced thin down projections, and the reduction chain formed by top-k combine and TP AllReduce. The left path shows that the conventional expert-major pipeline permutes H-wide token activations before the column-parallel up/gate projection. The middle path shows that the row-parallel down projection reduces over only $K = I/P$ on each TP rank. The right path shows that local top-k combine is immediately followed by a cross-rank TP reduction.

movement. Together with the conventional **EE** path (pre-permute, used by the FlashInfer-like baseline), these three layouts span the principal trade-off: EE pays full-hidden dispatch but enjoys regular grouped GEMMs throughout; TE avoids the dispatch while preserving expert-major locality for down; TT minimizes activation movement at the cost of a less regular down projection. We thus expect TT to favor small-T (decode-like) regimes where per-expert grouped-GEMM padding waste and routing-metadata overhead dominate, and TE to favor large-T (prefill-like) regimes where the avoided full-hidden permute and expert-major down locality both matter.

3.2. Thinner GEMMs

MoE expert computation already consists of many small or irregular GEMMs. If a layer processes T tokens, uses E experts, and routes each token to k experts, then the average number of token-expert pairs per expert is approximately Tk/E under balanced routing. In serving workloads, this number can be small, especially during decode. Routing skew further increases variance across experts, leaving some experts with very few tokens.

Tensor parallelism makes this problem more pronounced for the down projection. In the standard TP decomposition, the up and gate projections are column-parallel, while the down projection is row-parallel. Thus, each TP rank consumes only a $1/P$ shard of the expert intermediate dimension. For expert e on rank p, the local down projection has the form

$$Z_{e,p} = A_{e,p} W^{(e)}_{\text{down},p}, \tag{9}$$

where $A_{e,p}$ contains only the local intermediate shard. Compared with the unsharded expert down projection (as in EP), this GEMM has reduction dimension I/P rather than I, yielding a poorer aspect ratio. As P increases, each rank's down-projection GEMM becomes thinner.

This bottleneck is especially relevant in prefill-like workloads. Although the total number of tokens can be large, tokens are divided across experts, and TP further partitions the intermediate dimension. The resulting grouped GEMM may have insufficient work per expert tile to fully utilize tensor cores. Its performance can then be limited by pipeline depth, scheduling overhead, and memory traffic rather than peak floating-point throughput.

We evaluate persistent grouped GEMM (DeepSeek-AI, 2025) as a response to this bottleneck. A persistent kernel keeps work resident across multiple small expert GEMMs and can improve scheduling continuity for thin shapes. The trade-off is that persistence is not universally beneficial: it may provide little benefit when the token count is extremely small and non-GEMM overheads dominate, or when the GEMM is large enough for conventional grouped GEMM scheduling to reach high utilization. We therefore treat persistent down projection as a workload-dependent choice rather than a fixed optimization.

Table 1. MoE FFN latency breakdown at the canonical setting ($H = 4096$, per-rank $I = 256$, BF16) on a single 8×H200 node with TP=8. Latencies are in ms; bold marks the lowest latency in each numeric column within each token count. **Up (in/out)**: input and output activation layouts of the up/gate kernel (TM = token-major, EM = expert-major). **Down kernel**: down-projection grouped GEMM kernel, either standard (Std) or persistent (Pers). EE (baseline) corresponds to the dominant expert-major layout used in current MoE inference stacks (pre-permute + EE up/gate + EE down); TT and TE-pers are the decode- and prefill-optimized variants from §3. Stage latencies and Total exclude the final tensor-parallel AllReduce.

T	Variant	Up (in/out)	Down kernel	Align	Permute	Up/Gate	Act.	Down	Comb.	Total
8	EE (baseline)	EM/EM	Std	0.074	0.117	0.128	0.009	0.077	0.047	0.484
	EE-pers	EM/EM	Pers	0.217	0.148	0.155	0.009	0.098	0.070	0.737
	TT	TM/TM	Std	**0.033**	**0.008**	**0.110**	**0.006**	**0.055**	**0.009**	**0.249**
	TE-pers	TM/EM	Pers	0.071	**0.008**	0.112	0.009	0.066	0.036	0.335
128	EE (baseline)	EM/EM	Std	0.077	0.168	0.161	0.012	**0.074**	0.015	0.535
	EE-pers	EM/EM	Pers	0.075	0.169	**0.158**	0.012	0.088	0.015	0.546
	TT	TM/TM	Std	**0.034**	**0.008**	0.180	**0.007**	0.082	**0.012**	**0.352**
	TE-pers	TM/EM	Pers	0.070	**0.008**	0.187	0.012	0.085	0.015	0.408
4096	EE (baseline)	EM/EM	Std	0.089	0.634	0.289	0.037	0.226	0.100	1.406
	EE-pers	EM/EM	Pers	0.087	0.633	**0.284**	0.037	0.195	0.101	1.368
	TT	TM/TM	Std	**0.052**	**0.004**	0.334	**0.027**	0.273	**0.095**	0.810
	TE-pers	TM/EM	Pers	0.079	0.009	0.332	0.037	**0.194**	0.102	**0.785**
16384	EE (baseline)	EM/EM	Std	0.117	2.117	0.844	0.098	0.756	0.361	4.319
	EE-pers	EM/EM	Pers	0.116	2.111	**0.815**	0.098	**0.610**	0.361	4.136
	TT	TM/TM	Std	**0.109**	**0.003**	0.949	**0.089**	0.965	**0.346**	2.486
	TE-pers	TM/EM	Pers	0.113	**0.003**	0.957	0.099	0.633	0.361	**2.190**

3.3. Top-k Combine and TP Reduction

After the down projection, MoE outputs must be combined back into token-major order. With top-k routing, multiple expert outputs for the same token are weighted and summed. Under TP, each rank also produces only a partial hidden output, so the result must be reduced across ranks. A straightforward implementation performs

$$\begin{aligned} &\text{down} \rightarrow \text{materialize expert outputs} \\ &\quad \rightarrow \text{top-}k \text{ combine} \rightarrow \text{AllReduce}. \end{aligned} \tag{10}$$

This incurs extra HBM traffic, kernel launches, and serialization between local combine and TP communication.

A natural opportunity is to consider top-k combine and TP AllReduce together (Ye et al., 2025). Since they are adjacent reductions, one may split the output into chunks and overlap local combine for later chunks with communication for earlier chunks. In principle, this can hide part of the intra-node collective behind local reduction work. In practice, the benefit depends on chunk size, collective launch overhead, stream synchronization, and whether the runtime exposes enough independent work for overlap. We therefore evaluate combine–AllReduce overlap as a workload-dependent choice rather than assuming that fusion or chunking is always beneficial.

4. Experiments

4.1. Setup

We evaluate on a single node with 8 NVIDIA H200 SXM5 GPUs connected by NVLink. All kernels are implemented in Triton (Tillet et al., 2019), and tensor-parallel collective baselines use NCCL. For each kernel measurement, we run 10 warmup iterations followed by 50 timed iterations and report the average latency. We conduct experiments on the MoE shape of hidden dimension $H = 4096$ and $E = 128$ experts with top-$k = 8$. The main breakdown is reported at the canonical per-rank intermediate dimension $I = 256$ in BF16, which corresponds to typical (model, TP-degree) combinations at TP=8. For each measurement instance, token-expert assignments are generated randomly.

4.2. Latency Breakdown

Table 1 reports the stage-level breakdown at the canonical setting ($I = 256$, BF16) across T spanning decode- and prefill-like regimes. These totals cover the MoE FFN with local top-k combine and exclude the final TP AllReduce, which we study separately in Section 4.3. We compare the FlashInfer-style expert-major baseline (**EE**) against three variants: **EE-pers** (EE with persistent down), **TT** (token-major up/gate and down), and **TE-pers** (token-major up/gate with in-flight expert-major output and persistent down).

Table 1 shows a workload-dependent crossover. At $T \leq 128$, TT is fastest as it avoids expert-major pre-permutation and associated preprocessing, while the grouped GEMMs are

Table 2. BF16 MoE FFN latency *before* the final TP AllReduce and post-down reduction sensitivity across TP degrees for MoE shape (E=128, H=4096). Per-rank I is given by $I = I_{\text{model}}/\text{TP}$. The two panels are measured separately: the left panel reports the MoE FFN before the final TP AllReduce, while the right panel isolates the post-down fusion opportunity between top-k combine and TP AllReduce.

I_{model}	TP	variant	T=8	T=128	T=4096	T=16384
1024	4	TT	**0.249**	**0.352**	0.810	2.486
		TE-pers	0.335	0.408	**0.785**	**2.190**
	8	TT	**0.300**	**0.330**	0.639	1.778
		TE-pers	0.353	0.419	**0.621**	**1.532**
2048	4	TT	**0.378**	**0.541**	1.226	3.913
		TE-pers	0.385	0.607	**1.190**	**3.639**
	8	TT	**0.249**	**0.352**	0.810	2.486
		TE-pers	0.335	0.408	**0.785**	**2.190**

(a) Latency (ms) of the MoE FFN across TP configurations. Fastest is **bolded**.

I_{model}	TP	method	T=8	T=128	T=4096	T=16384
1024	4	FUSE	**0.045**	**0.040**	0.333	1.195
		NCCL	0.074	0.064	**0.263**	**0.908**
	8	FUSE	**0.049**	**0.049**	0.437	2.101
		NCCL	0.068	0.077	**0.284**	**0.858**
2048	4	FUSE	**0.045**	**0.040**	0.333	1.195
		NCCL	0.074	0.064	**0.263**	**0.908**
	8	FUSE	**0.049**	**0.049**	0.437	2.101
		NCCL	0.068	0.077	**0.284**	**0.858**

(b) Latency (ms) of the post-down reduction chain of top-k combine and TP AllReduce. Fastest is **bolded**.

too small for expert-major locality to compensate. At $T \geq 4096$, TE-pers becomes fastest: it still avoids the full-hidden pre-permute, but preserves expert-major locality for the thin down projection and benefits from persistent scheduling.

Although EE obtains a faster up/gate GEMM in isolation due to expert-major locality, this local gain is overwhelmed by the full-hidden pre-permute: at $T = 16384$, EE saves only 0.113 ms in up/gate relative to TE-pers, but pays an additional 2.114 ms in permutation. The up/gate and down projections therefore benefit from different layouts: up/gate has enough output width to sustain the pipeline, while the TP-sharded down projection is limited by its thin reduction dimension and benefits from persistent scheduling. At small T, this advantage does not materialize because the per-expert M dimension is too small to amortize expert-major scheduling and persistence overheads, making TT's simpler token-major path preferable. EE-pers further shows that improving the down projection alone is insufficient when pre-permutation dominates the critical path. Thus, the optimal MoE FFN layout flips from TT to TE-pers as the workload shifts from decode-like to prefill-like token counts.

4.3. Sensitivity Across TP Degrees

Tensor-parallel degree is the principal deployment-scale knob in single-node serving: increasing TP shrinks the per-rank intermediate dimension and extends the post-down AllReduce across more ranks. We reframe the per-rank I sweep on a (model I, TP) axis: a model with intermediate dimension I_{model} deployed at TP degree P has per-rank $I = I_{\text{model}}/P$. Table 2a reports the MoE FFN before the final TP AllReduce, while Table 2b reports the post-down top-k combine plus TP AllReduce sub-region for the same deployment scenarios. We denote our fused reduction path as **FUSE**; the **NCCL** baseline runs top-k combine and TP AllReduce sequentially. Since the post-down reduce chain depends only on (TP, T), rows at the same TP in Table 2b repeat values across model I.

We make two observations. First, the layout policy from Section 4.2 is robust across deployment scenarios: in every (model I, TP) row of Table 2a, TT wins at $T \leq 128$ and TE-pers at $T \geq 4096$, with the crossover always between these two values. Absolute latency scales with per-rank I as expected, but the choice of variant does not flip. Although Table 2a reports MoE FFN totals before the final TP AllReduce, the full stage breakdown in Appendix B exposes the thin-down-GEMM effect: at $T = 16384$, TE-pers reduces down-projection latency relative to TT by 47%, 34%, and 28% for per-rank $I = 128, 256, 512$, respectively. The same breakdown shows that the EE pre-permute remains nearly constant across these I values (2.11–2.14 ms), since it moves full-hidden activations rather than the sharded intermediate. Second, the post-down reduction choice is also workload-dependent: in Table 2b, FUSE is faster at the reported small-T points ($T = 8, 128$), while NCCL is faster at the reported large-T points ($T = 4096, 16384$). Thus, both layout choice and reduction-chain fusion expose a small-T versus large-T crossover. We report the reduction-chain experiment as a separate sub-region rather than including it in the MoE FFN totals.

5. Conclusion

Single-node TP-MoE inference presents unique bottlenecks unobserved in EP-MoE. TP turns full-hidden expert-major pre-permutation into a major cost, makes down projections unusually thin, and places top-k combine next to the TP AllReduce. Our study shows that these artifacts make the best MoE FFN layout workload-dependent: token-major execution is preferable in decode-like regimes, while in-flight permutation with persistent expert-major down projection is preferable in prefill-like regimes. Across representative MoE shapes, this policy consistently improves over the conventional expert-major baseline. These results suggest that prefill/decode specialization should extend below scheduling and batching to the MoE kernel layout itself.

Limitations

Our implementations intentionally avoid architecture-specific kernel optimizations such as Hopper warp specialization, Blackwell-specific `tcgen05` instructions, and hand-written multi-CTA MMA pipelines. Instead, we implement all variants in Triton to compare layout and fusion choices under a common programming and compilation stack. This means that our results should be interpreted as a study of relative design trade-offs rather than absolute peak performance. We expect many of the observed trends to remain relevant with more specialized kernels, but architecture-specific optimizations may shift absolute latencies and some crossover points. Our evaluation is also limited to MoE FFN kernels on a limited set of hyperparameters; evaluations on additional GPU architectures, expert counts, and top-k settings are left to future work.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agarwal, S., Ahmad, L., Ai, J., Altman, S., Applebaum, A., Arbus, E., Arora, R. K., Bai, Y., Baker, B., Bao, H., et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

DeepSeek-AI. DeepGEMM: Clean and efficient FP8 GEMM kernels with fine-grained scaling. https://github.com/deepseek-ai/DeepGEMM, 2025.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. GLaM: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 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.

FlashInfer Project. Fused MoE operators in FlashInfer. https://github.com/flashinfer-ai/flashinfer, 2024.

Frantar, E. and Alistarh, D. QMoE: Sub-1-bit compression of trillion parameter models. In Gibbons, P., Pekhimenko, G., and Sa, C. D. (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 439–451, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/c74b624843218d9b6713fcf299d6d5e4-Paper-Conference.pdf.

He, J., Qiu, J., Zeng, A., Yang, Z., Zhai, J., and Tang, J. FastMoE: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021.

He, J., Zhai, J., Antunes, T., Wang, H., Luo, F., Shi, S., and Li, Q. FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pp. 120–134, 2022.

Hwang, C., Cui, W., Xiong, Y., Yang, Z., Liu, Z., Hu, H., Wang, Z., Salas, R., Jose, J., Ram, P., et al. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of Machine Learning and Systems*, 2023. arXiv:2206.03382.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. arXiv:2006.16668.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

NVIDIA. TensorRT-LLM: A TensorRT toolbox for optimized large language model inference. https://github.com/NVIDIA/TensorRT-LLM, 2023.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative LLM inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

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 *International Conference on Machine Learning*, pp. 18332–18346. PMLR, 2022.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*,

2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Tillet, P., Kung, H. T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Triton and Contributors, P. Persistent grouped GEMM kernels. https://github.com/triton-lang/triton, 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

vLLM Project. Fused MoE triton kernels in vLLM. https://github.com/vllm-project/vllm, 2024.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., and Ceze, L. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, 2025. arXiv:2501.01005.

Yi, R., Guo, L., Wei, S., Zhou, A., Wang, S., and Xu, M. EdgeMoE: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 2025.

Zeng, A., Lv, X., Hou, Z., Du, Z., Zheng, Q., Chen, B., Yin, D., Ge, C., Huang, C., Xie, C., et al. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VqkAKQibpq.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.

A. Related Work

Mixture-of-Experts models. MoE layers use conditional computation to increase model capacity while activating only a small subset of parameters for each token. Early sparsely-gated MoE layers introduced this idea in neural networks (Shazeer et al., 2017), and later Transformer-based models such as GShard, Switch Transformer, and GLaM scaled it to large language models (Lepikhin et al., 2021; Fedus et al., 2022; Du et al., 2022). These works focus primarily on model scaling, routing, training stability, and quality–compute trade-offs. Our work assumes a trained MoE model and studies inference-time kernel and layout trade-offs.

MoE systems. Large-scale MoE systems have studied expert placement, expert parallelism, communication, and load balancing across many accelerators (He et al., 2021; 2022; Hwang et al., 2023; Rajbhandari et al., 2022). These systems target distributed MoE training or large-scale serving, where multi-node communication and expert load balancing are central concerns. In contrast, we study a single-node tensor-parallel regime, where the bottlenecks shift toward activation layout movement, thin grouped GEMMs, local combine, and intra-node collectives.

LLM serving and MoE kernels. Modern LLM serving systems optimize batching, memory management, parallelism, and GPU utilization for autoregressive inference (Kwon et al., 2023; NVIDIA, 2023). Recent inference stacks and kernel libraries also provide specialized fused MoE operators, including dispatch, grouped GEMM, activation, quantization, and combine kernels (FlashInfer Project, 2024; vLLM Project, 2024; Triton & Contributors, 2024). Our work is complementary: rather than proposing a single fused MoE kernel, we compare when different layout and fusion choices are beneficial under single-node tensor parallelism.

Positioning. Prior work establishes MoE as an effective sparse architecture and develops systems for scaling MoE across many devices. We focus on a narrower but practical setting: serving MoE models on one multi-GPU node with tensor parallelism. In this setting, the main trade-offs are not only expert placement and load balancing, but also full-hidden dispatch, TP-induced thin down projections, top-k combine, and tensor-parallel reduction.

B. Additional Stage Breakdowns

This section provides the stage-level breakdown behind the TP-degree sensitivity results in Section 4.3. We focus on the large-T setting, where prefill-like workloads expose both the full-hidden pre-permute cost and the thin down-projection GEMM. The breakdown separates these effects: the EE pre-permute remains nearly constant across per-rank intermediate dimensions, while the down-projection advantage of TE-pers over TT is largest for the thinnest per-rank I.

Table 3. BF16 stage-level MoE FFN latency breakdown at $T = 16384$ for $H = 4096$ and $E = 128$ across per-rank intermediate dimensions. Latencies are in ms.

Per-rank I	Variant	Align	Permute	Up/Gate	Act.	Down	Comb.	Total
128	EE (baseline)	0.1257	2.1124	0.4646	0.0956	0.5738	0.3606	3.7591
	EE-pers	0.1198	2.1124	0.4859	0.0957	0.3807	0.3609	3.5810
	TT	0.1071	0.0032	0.4958	0.0854	0.7149	0.3457	1.7780
	TE-pers	0.1192	0.0032	0.5478	0.0954	0.3786	0.3606	1.5317
256	EE (baseline)	0.1173	2.1173	0.8440	0.0978	0.7562	0.3609	4.3189
	EE-pers	0.1159	2.1109	0.8147	0.0979	0.6098	0.3611	4.1355
	TT	0.1090	0.0031	0.9491	0.0886	0.9648	0.3460	2.4856
	TE-pers	0.1127	0.0032	0.9565	0.0987	0.6325	0.3610	2.1898
512	EE (baseline)	0.1182	2.1410	1.6414	0.1319	1.1220	0.3599	5.5399
	EE-pers	0.1190	2.1505	1.5453	0.1323	0.9505	0.3613	5.2850
	TT	0.1070	0.0031	1.8686	0.1181	1.4464	0.3449	3.9127
	TE-pers	0.1166	0.0033	1.9617	0.1331	1.0360	0.3625	3.6388