ROUTEEP: ROUTING-AWARE EXPERT PARALLELISM FOR EFFICIENT MOE LLM DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

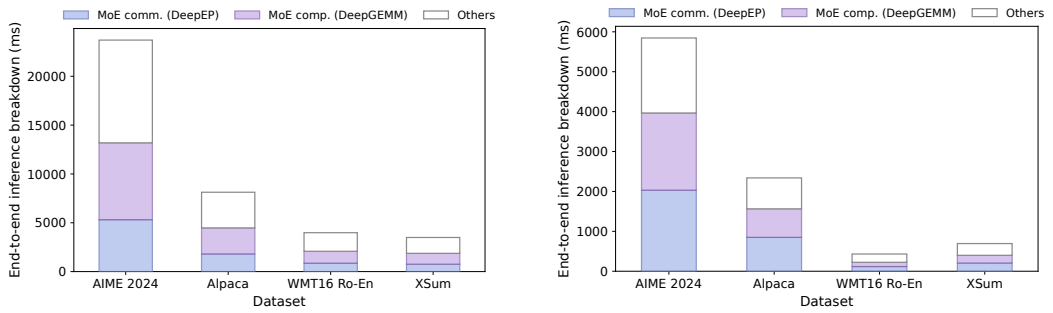
Mixture-of-Experts (MoE) architectures enable Large Language Models (LLMs) to scale their parameter capacity while activating only a small subset of experts for each token. However, dynamic expert routing in expert parallelism (EP) and expensive all-to-all collective communication incur substantial inter-device communication overhead during model inference. Such bottlenecks become more pronounced in decoding, where sparse expert activation at small active batch sizes limits computation–communication overlap and effective parallelism. In this paper, we present `RouteEP`, a routing-aware expert parallelism framework to reduce inter-device communication and improve the load balance of expert execution for LLM decoding. `RouteEP` introduces a decode-specific parallelism design tailored to the memory- and communication-bound nature of decoding through three coordinated contributions. (1) We propose a lead-anchored offline expert grouping algorithm that selects lead experts, builds expert groups around them, and assigns the remaining experts to maximize position-wise routing affinity under capacity constraints. (2) Based on these expert groups, we design a routing-aware expert parallelism scheme for LLM decoding that replaces the communication-intensive dispatch–combine EP pattern with rank-local partial expert computations followed by a global `all-reduce`. (3) We further integrate intra-expert sharding into this framework to increase parallelism within the limited set of activated experts. Experiments on a range of MoE LLMs, including Qwen, DeepSeek, and Mixtral models, show that `RouteEP` achieves 1.1–1.8$\times$ speedups over state-of-the-art inference frameworks.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities in programming assistance (Nam et al., 2024; Mu et al., 2024), machine translation (Xu et al., 2024; Lu et al., 2024), and domain-specific agents (Ge et al., 2023; Diao et al., 2025). As LLM parameter counts have grown (Epoch AI, 2026), the computational cost of dense models has made sparse Mixture-of-Experts (MoE) architectures increasingly attractive for expanding model capacity while limiting per-token computation. MoE architectures replace dense feed-forward blocks with banks of experts and use a learned router to select a small subset for each token (Shazeer et al., 2017). This approach underlies prominent LLM families, including Mixtral, Qwen, and DeepSeek (Jiang et al., 2024a; Dai et al., 2024).

To achieve efficient LLM inference on multi-GPU systems, expert parallelism (EP) distributes experts across ranks and adaptively routes tokens to the ranks that own the selected experts (Rajbhandari et al., 2022; Hwang et al., 2023). Modern LLM serving frameworks such as vLLM (Kwon, 2025) and SGLang (Zheng et al., 2024) provide efficient request scheduling, KV-cache management, and multi-dimensional parallelism to improve inference efficiency. Moreover, prefill–decode disaggregation schemes (Zhong et al., 2024; Patel et al., 2024; Qin et al., 2026) deploy the typically compute-bound prefill and memory-bound decode phases on separate GPU nodes or clusters. More recent studies further explore fine-grained attention–expert disaggregation (Zhu et al., 2025; Zhang et al., 2025c) for MoE inference, decoupling attention and expert execution onto distinct resources. Although these methods improve MoE inference through disaggregation and routing-aware optimization, sparse and skewed expert activation can still cause imbalanced computation and exposed communication costs during decoding, particularly at small active batch sizes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Qwen3-30B-A3B (b) Mixtral-8x7B-Instruct

Figure 1: End-to-end inference breakdown of a single request batch across four workloads using SGLang on four NVIDIA B200 GPUs(4TP+4EP). `MoE comm. (DeepEP)` and `MoE comp. (DeepGEMM)` denote the token communication and expert computation in the decode phase, respectively. `Others` includes the prefill phase and the remaining decode operations.

Conventional EP in LLM inference follows a dispatch–compute–combine pipeline, where tokens are sent through `all-to-all` collectives, processed by the destination experts, and returned to their source ranks. Unlike tensor parallelism (TP), which follows a regular partitioning scheme, EP communication depends heavily on the input tokens, with data-dependent expert workloads. Both the destinations and the per-rank message sizes vary across layers, requests, and generation steps, coupling communication overhead with routing-induced load imbalance (Li et al., 2023; Jiang et al., 2024c; Zhang et al., 2025b).

This bottleneck is particularly pronounced in decoding. Unlike prefill, which processes many prompt tokens concurrently to amortize communication over large expert batches, decoding offers few token–expert assignments per iteration at small active batch sizes. Figure 1 shows the end-to-end inference breakdown on four B200 GPUs, in which decode-stage MoE computation and communication account for 50%–60% of the total execution time. Meanwhile, Top-K routing exhibits nonuniform expert activation frequencies, as shown in Figures 7, 9, and 11. This phenomenon can concentrate expert work on a subset of ranks, limiting multi-GPU utilization. Although speculative decoding techniques (Leviathan et al., 2023; Chen et al., 2023) can improve computational throughput by generating multiple candidate tokens per iteration, candidate tokens are not always accepted by the target model. Their gains depend on acceptance rates and do not eliminate the need for efficient expert execution.

To summarize, current MoE LLM decoding suffers from three key bottlenecks: *1) Inefficient collective communication.* Dispatch-based EP implementations use DeepEP-style `all-to-all` communication (Zhao et al., 2025b) for token dispatch and combine. At the small active batch sizes typical of decoding, sparse token–expert assignments provide limited work to amortize communication and synchronization costs or overlap communication with computation. *2) Imbalanced Top-K routing and expert computation.* As shown in Appendix A.6, Top-K routing exhibits strong expert preference, which can concentrate token assignments on a subset of EP ranks and lead to substantial inter-rank workload imbalance. *3) Limited scalability beyond Top-K.* In conventional EP, where each expert is stored entirely on a single rank, a token activating K experts can involve at most K EP ranks. Consequently, adding more than K GPUs does not directly increase the parallelism of per-token expert computation, potentially leaving additional GPUs underutilized at small active batch sizes.

We propose `RouteEP`, a routing-aware expert parallelism framework that reduces inter-GPU communication and improves the balance of expert execution for MoE LLM decoding. ❶ We propose a lead-anchored expert grouping algorithm that distributes frequently activated experts across ranks. It selects K lead experts, one for each Top-K position, constructs R expert groups for R EP ranks, and distributes the lead experts and the remaining experts among the R groups according to their group affinity, so that different groups capture distinct position-wise Top-K routing patterns. ❷ We design a routing-aware expert parallelism scheme that distributes Top-K expert computation across EP ranks. Each rank performs local expert computations according to the global routing results

and its assigned expert group, and the partial outputs are globally reduced to obtain the final MoE output, avoiding the conventional token dispatch and combine operations. ❸ We further extend this routing-aware EP framework to larger GPU systems with balanced expert sharding, allowing multiple GPUs to jointly process each selected expert and aggregate the shard contributions through `all-reduce`.

2 BACKGROUND

MoE Inference and Expert Parallelism. Mixture-of-Experts (MoE) models replace selected dense feed-forward blocks with multiple experts, activating a Top-K subset for each token through a learned router (Shazeer et al., 2017). This design increases model capacity while keeping active computation sparse, as demonstrated by GShard (Lepikhin et al., 2020), Switch Transformer (Fedus et al., 2022), Mixtral (Jiang et al., 2024a), and DeepSeekMoE (Dai et al., 2024). Expert parallelism (EP) distributes experts across ranks, commonly following a dispatch–compute–combine pipeline (Rajbhandari et al., 2022; Hwang et al., 2023). During autoregressive decoding, each iteration generates one token per active request. Combining TP with EP is a common configuration, where TP divides the attention and other non-expert computations across GPUs, while EP distributes the expert parameters across GPUs. Together, they reduce per-GPU model memory requirements, enabling the deployment of larger MoE models that exceed the memory capacity of a single GPU.

Optimizations for MoE Serving. Model serving frameworks such as vLLM and SGLang effectively optimize memory management and request execution (Kwon et al., 2023; Zheng et al., 2024). Prefill–decode (PD) disaggregation separates inference phases for independent resource provisioning (Patel et al., 2024; Zhong et al., 2024), while attention–expert (AE) disaggregation decouples attention and expert modules for independent scaling (Zhu et al., 2025; Zhang et al., 2025c). Within MoE layers, Lina, Lancet, and COMET optimize communication scheduling and computation–communication overlap (Li et al., 2023; Jiang et al., 2024c; Zhang et al., 2025b). Routing-aware methods use activation patterns to prefetch offloaded experts (Eliseev & Mazur, 2023; He et al., 2024; Song et al., 2024) or improve expert placement and token locality (Yao et al., 2024; Li et al., 2026b; Yu et al., 2026). These optimizations target different serving bottlenecks. In contrast, `RouteEP` jointly addresses communication overhead, routing-induced load imbalance, and limited intra-token parallelism through offline expert grouping and a routing-aware EP framework with integrated expert sharding.

3 METHODOLOGY

Overview of `RouteEP`. Figure 2 illustrates the workflow of `RouteEP`. Before serving, `RouteEP` collects prefill routing traces from the calibration prompts described in Section 4.1. Based on the collected routing traces, we construct the expert frequency matrix $\mathbf{C}^{\ell} \in \mathbb{N}_0^{K \times E}$ for each layer ℓ, where each entry records the routing frequency of an expert at a specific Top-K position. Then, lead-anchored expert grouping is applied to select K high-frequency lead experts corresponding to the K positions in Top-K routing and evenly distribute them across R expert groups in a round-robin manner, where R denotes the total number of EP ranks. A capacity-constrained assignment places the remaining experts by maximizing a position-wise routing score while keeping the leads fixed.

Guided by the R expert groups, we design a routing-aware EP scheme that eliminates the `all-to-all` dispatch–combine operations in DeepEP (Zhao et al., 2025b). In this parallelism, each EP rank performs partial expert computations for its assigned expert group, and the rank-level partial results are then aggregated through a global `all-reduce` to obtain the final MoE output. We further introduce a fine-grained intra-expert sharding scheme in routing-aware EP that shards individual experts and distributes their computation across GPUs, enabling the decode phase to efficiently scale to systems equipped with more than K GPUs.

3.1 LEAD-ANCHORED EXPERT GROUPING

In this section, we construct a static expert map through position-wise routing modeling, lead selection, and capacity-constrained assignment; Appendix A.1 provides the corresponding algorithms.

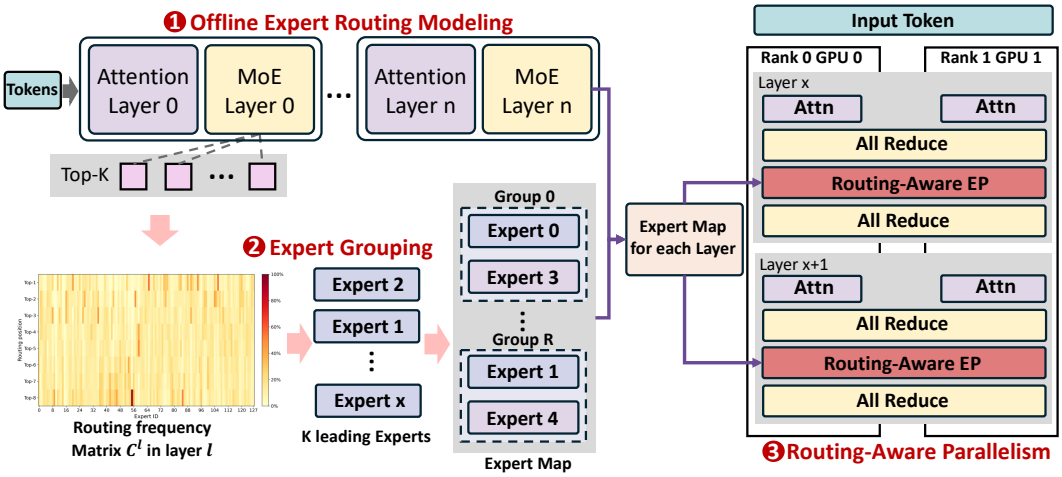


Figure 2: Overview of `RouteEP`. **(1)** Offline profiling captures position-wise Top-K routing frequencies. **(2)** Lead-anchored, capacity-constrained grouping produces a layer-wise expert map. **(3)** Routing-aware decoding executes expert operations locally according to the global Top-K routing results and the local expert groups, and then accumulates the results via `all-reduce`.

Consider an MoE layer ℓ with an expert ID set $\mathcal{E}^\ell = \{0, \dots, E-1\}$ and Top-K routing on R EP ranks, where $1 \le K \le E$, $1 \le R \le E$, and R divides E.

Position-wise routing modeling. Given a calibration trace with N tokens, the gating network in each MoE layer produces the Top-K routes $\mathbf{Y}^\ell \in \{0, \dots, E-1\}^{N \times K}$, where $Y^\ell_{t,s}$ denotes the routed expert ID at position s of Top-K for token t. We define the routing frequency matrix $\mathbf{C}^\ell \in \mathbb{N}_0^{K \times E}$ as

$$\mathbf{C}^\ell_{s,e} = \sum_{t=0}^{N-1} \mathbf{1}\{Y^\ell_{t,s} = e\}, \quad 0 \le s < K,\ 0 \le e < E, \tag{1}$$

which counts how often tokens are routed to expert e at the s-th Top-K position. Algorithm 1 in Appendix A.1 details this counting procedure, which preserves the routing-position order recorded in the trace.

Lead-expert selection and rank binding. Based on $\mathbf{C}^\ell$, we select K distinct lead experts with high routing frequency. Starting from $\Lambda^\ell = \varnothing$, the m-th selection updates Λ^ℓ as

$$\begin{aligned} \lambda^\ell_m &= \arg\max_{e \in \mathcal{E}^\ell \setminus \Lambda^\ell} C^\ell_{m,e}, \\ \Lambda^\ell &= \Lambda^\ell \cup \{\lambda^\ell_m\}, \qquad m = 0, \dots, K-1. \end{aligned} \tag{2}$$

After this construction, we obtain the final Λ^ℓ with K lead experts. We create a collection of R groups, $\mathbf{G}^\ell = (\mathbf{G}^\ell_0, \dots, \mathbf{G}^\ell_{R-1})$, initialize each group to $\varnothing$, and set its capacity to $B = E/R$. The lead experts are then evenly distributed among the groups in $\mathbf{G}^\ell$ in a round-robin manner:

$$\mathbf{G}^\ell_{m \bmod R} = \mathbf{G}^\ell_{m \bmod R} \cup \{\lambda^\ell_m\}, \qquad m = 0, \dots, K-1. \tag{3}$$

This lead-expert selection imposes a uniqueness constraint. When the highest-frequency expert for a given Top-K position has already been chosen, that position selects the highest-frequency expert among the remaining unselected candidates. Algorithm 2 in Appendix A.1 specifies lead selection and rank binding.

Capacity-constrained expert assignment. After initializing the expert groups in Equation 3, we define $\mathcal{L}^\ell_r = \{\lambda^\ell_m \mid m \bmod R = r\}$ as the fixed lead set for rank r and $\mathcal{U}^\ell = \mathcal{E}^\ell \setminus \Lambda^\ell$ as the remaining expert list in ascending order. The residual rank capacity is represented by a slot set $\mathcal{S}^\ell$, defined as

$$\mathcal{S}^\ell = \{(r, c) \mid 0 \le r < R,\ 0 \le c < b^\ell_r\}, \tag{4}$$

where $b_r^\ell = B - |\mathcal{L}_r^\ell|$ denotes the residual capacity of $\mathbf{G}_r^\ell$. The round-robin assignment in Equation 3 yields $|\mathcal{L}_r^\ell| = \lfloor K/R \rfloor + \mathbf{1}[r < K \bmod R]$. Therefore, $|\mathcal{S}^\ell| = |\mathcal{U}^\ell| = M = E - K$, and we formulate the assignment of the remaining experts in $\mathcal{U}^\ell$ as a linear assignment problem. We enumerate the remaining experts as $(u_0, \ldots, u_{M-1})$ and the slots in $\mathcal{S}^\ell$ as $(a_0, \ldots, a_{M-1})$. For $a_i = (r, c)$, we define the cost matrix $\mathbf{D}^\ell \in \mathbb{Z}^{M \times M}$ as

$$D_{i,j}^\ell = - \sum_{\substack{0 \le s < K \\ s \bmod R = r}} C_{s,u_j}^\ell. \tag{5}$$

This matrix accumulates all the routing frequencies satisfying $s \bmod R = r$. Let Φ^ℓ denote the set of permutations of $\{0, \ldots, M-1\}$. The minimum-cost assignment is

$$\pi_\ell^\star = \arg\min_{\pi \in \Phi^\ell} \sum_{i=0}^{M-1} D_{i,\pi(i)}^\ell. \tag{6}$$

Representing the residual capacity of each rank as individual slots converts the equal-capacity constraint into a standard bipartite matching problem, which we solve with the linear-sum assignment solver in SciPy (Virtanen et al., 2020). Algorithm 3 in Appendix A.1 details the cost construction and the recovery of the expert groups. Finally, the expert group assigned to rank r is

$$\mathbf{G}_r^\ell = \mathcal{L}_r^\ell \cup \{u_{\pi_\ell^\star(i)} \mid a_i = (r, c) \in \mathcal{S}^\ell\}. \tag{7}$$

Each remaining expert fills exactly one slot, so the groups form a disjoint partition of $\mathcal{E}^\ell$ with $|\mathbf{G}_r^\ell| = B$. This matching maximizes the position-wise score subject to fixed lead placements and equal group sizes. Note that equal counts alone do not guarantee equal per-token execution workloads.

Offline expert loading. The algorithm outputs the group collection $\mathbf{G}^\ell$, which defines the expert ownership

$$\mathrm{owner}_\ell(e) = r, \qquad e \in \mathbf{G}_r^\ell. \tag{8}$$

Before serving, each rank r loads only the expert parameters in its local group $\mathbf{G}_r^\ell$.

The proposed method partitions experts into equal-sized groups based on position-wise routing frequencies and assigns one group to each EP rank. During decoding, rank r preferentially hosts the experts frequently selected at the Top-K positions m satisfying $m \bmod R = r$. Because the grouping relies on aggregate routing frequencies, individual Top-K routing decisions may still select experts outside the expected groups.

3.2 Routing-Aware Expert Parallelism Framework for Decoding

Disaggregated expert mapping. `RouteEP` retains the original expert mapping and DeepEP execution path for prefill, while decoding uses the offline mapping constructed in Section 3.1. This phase-specific design preserves the throughput-oriented prefill path and specializes expert execution for sparse decode workloads. In a disaggregated deployment, each worker pool can load its corresponding expert layout independently.

Parallelism redesign. DeepEP provides high-performance `all-to-all` dispatch and combine operations for expert parallelism (Zhao et al., 2025b). In prefill, abundant prompt tokens enable batched expert computation and amortize communication costs. Figure 3(a) illustrates this DeepEP-based TP+EP workflow. For each token batch, TP first computes partial attention outputs on each rank using its local attention heads, followed by `reduce-scatter` to distribute the aggregated results. DeepEP then performs token dispatch, expert computation, and combine to produce the MoE outputs, followed by `all-gather` to reconstruct the complete outputs on all ranks. Decoding at small active batch sizes exposes far fewer token–expert assignments than prefill, limiting opportunities to utilize communication bandwidth and distribute expert computation. Figure 3(b) shows the DeepEP-based decoding for a single request, where both selected experts reside on one GPU, illustrating routing-induced imbalance and exposed communication overhead.

`RouteEP` replaces this dispatch-based path with rank-local expert execution, as shown in Figure 3(c). After TP, an `all-reduce` first aggregates the attention TP outputs, making the complete token activations available on every participating rank. Each rank r obtains the original global

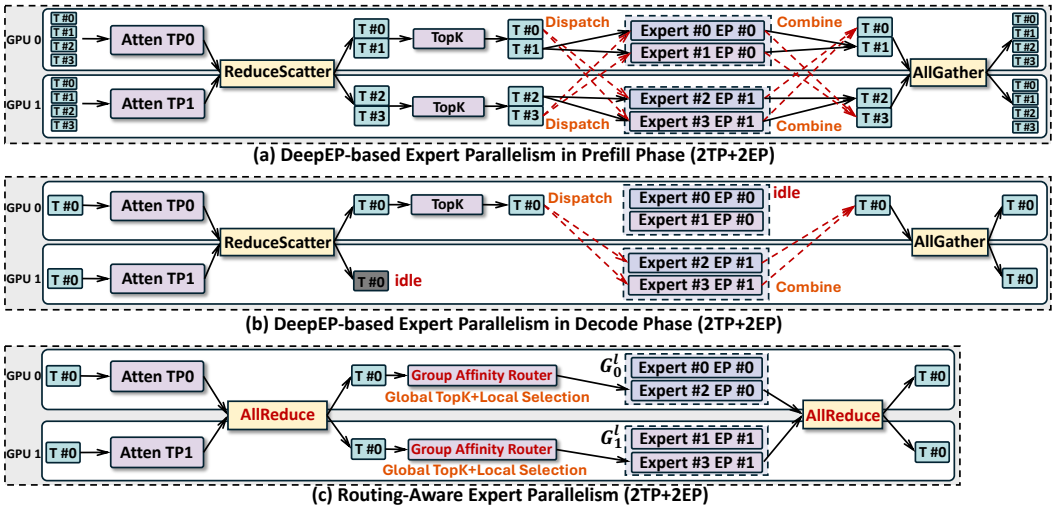


Figure 3: Routing-aware EP with Top-2 routing on two GPUs (2TP+2EP). (a) DeepEP-based prefill scheme. (b) DeepEP-based decoding for a single request. (c) Routing-aware EP in `RouteEP` for a single request. In `RouteEP`, an `all-reduce` replicates the attention output, each EP rank computes its locally owned experts according to the global Top-K routing, and a second `all-reduce` aggregates their weighted outputs.

Top-K routing expert ID list $\mathbf{y}_t^\ell = (Y_{t,0}^\ell, \ldots, Y_{t,K-1}^\ell)$ and performs expert computation for locally owned experts in $\mathbf{G}_r^\ell \cap \mathbf{y}_t^\ell$ to generate the routing-weighted partial outputs, where $\mathbf{G}_r^\ell$ denotes the expert set stored on rank r. Next, a second `all-reduce` sums these partial outputs to reconstruct the MoE output on every rank, replacing the expert dispatch–combine operations and the subsequent `all-gather` operation. The offline grouping algorithm in Section 3.1 influences rank-level expert placement but does not change the global Top-K decisions or routing weights. Each selected expert is evaluated by its owning rank regardless of its routing position. If multiple selected experts reside on the same rank, that rank evaluates all of them. Thus, the execution preserves the original routing decisions, while grouping aims to distribute frequently activated experts without guaranteeing equal per-step workloads.

Expert sharding within routing-aware EP. Building on the routing-aware EP framework, we further integrate efficient intra-expert sharding within local expert groups to increase intra-token parallelism. With unsharded, non-replicated expert placement, each token can engage at most $\min(K, R)$ ranks in parallel, which can leave GPUs underutilized in sparse decode workloads.

Figure 4 illustrates the expert sharding design. For an input $\mathbf{x} \in \mathbb{R}^H$, the fused gate and up projection $\mathbf{W}_{13} \in \mathbb{R}^{H \times 2I}$ produces $\mathbf{g}, \mathbf{u} \in \mathbb{R}^I$, where I is the intermediate dimension. The gated activation is

$$\mathbf{a} = \mathrm{SiLU}(\mathbf{g}) \odot \mathbf{u}, \tag{9}$$

followed by the down projection $\mathbf{W}_2 \in \mathbb{R}^{I \times H}$ to produce the expert output. We partition the intermediate dimension so that each rank stores corresponding slices of both the gate and up projections, together with the matching rows of the down projection.

For Top-K decoding on R EP ranks ($R > K$), the framework organizes the R ranks into K shard groups of $D = R/K$ ranks each. Assuming I is divisible by D, ranks in the same group share the same expert ownership but load distinct intermediate-dimension shards $\mathbf{W}_{13}^{(d)} \in \mathbb{R}^{H \times \frac{2I}{D}}$ and $\mathbf{W}_2^{(d)} \in \mathbb{R}^{\frac{I}{D} \times H}$. In this case, the offline grouping constructs K expert groups, each spanning D shard ranks. Each rank uses the input and global routing results replicated by the first `all-reduce` to evaluate its resident shards of the selected experts.

The final MoE output is aggregated from routing-weighted shard and expert contributions via `all-reduce`, without a separate shard-group reduction. As shown in Figure 4(b), in the Top-1 example with two GPUs, both ranks contribute to the same activated expert. This increases the

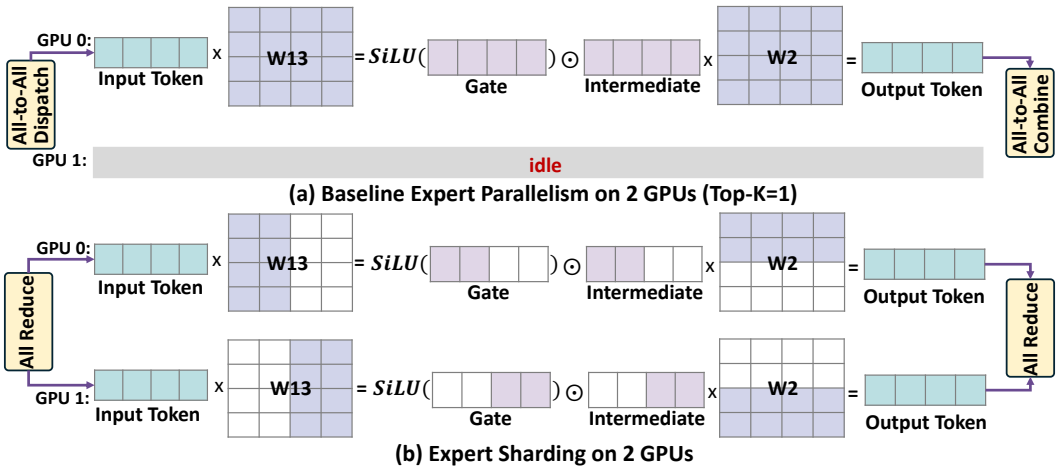


Figure 4: Expert sharding scheme for a Top-1 MoE layer with two EP ranks. **(a)** Original EP implementation using DeepGEMM (Zhao et al., 2025a), where only one GPU performs the expert computation while the other remains idle. **(b)** Sharding during decoding, where both GPUs process the same token with different expert shards to produce partial outputs, followed by `all-reduce` to obtain the final result.

available intra-expert parallelism, although the realized speedup depends on kernel efficiency and communication overhead.

4 Experiments

4.1 Experimental Setup

Experimental environment. We evaluate `RouteEP` on a server equipped with four NVIDIA B200 GPUs, each with 173.2 GiB of HBM3e memory and a peak memory bandwidth of 7.9 TB/s. The GPUs are interconnected via NVLink with a bidirectional bandwidth of 600 GB/s.

Baselines. We compare `RouteEP` against two representative open-source LLM serving frameworks, SGLang (Zheng et al., 2024) and LMDeploy (Contributors, 2023; Zhang et al., 2025a). Both frameworks provide highly optimized execution engines for high-performance LLM inference. SGLang supports efficient batching, PagedAttention, FlashAttention, RadixAttention, and optimized MoE execution through DeepEP (Zhao et al., 2025b) and DeepGEMM (Zhao et al., 2025a). LMDeploy provides continuous batching, blocked KV-cache management, PagedAttention, high-performance CUDA kernels, and optimized MoE execution with support for DeepEP and DeepGEMM where applicable. For each baseline, we enable the framework's optimized communication and expert-computation backends. The configuration details are listed in Appendix A.3

Models and datasets. We evaluate `RouteEP` on three representative open-source MoE LLMs: Qwen3-30B-A3B (Yang et al., 2025), DeepSeek-V2-Lite (DeepSeek-AI et al., 2024), and Mixtral-8x7B-Instruct-v0.1 (Jiang et al., 2024b). For all models and serving systems, we use the same model checkpoints and four GPUs with the default 4-way tensor parallelism (TP) and 4-way expert parallelism (EP) configuration. The experimental workloads cover four representative tasks: instruction following with Alpaca (Taori et al., 2023), Romanian-to-English translation with WMT16 (Bojar et al., 2016), single-document summarization with XSum (Narayan et al., 2018), and mathematical reasoning with AIME 2024 (Jia, 2024). We convert each example into a two-turn conversation and use the same request format across SGLang and LMDeploy. For each dataset, we split the conversations into calibration and evaluation sets at a 4:1 ratio using a fixed random seed and stratification by prompt length. The calibration set contains 2,779 requests with a total of 154,387 prompt tokens. We collect prefill-stage expert-routing traces from the calibration set and use them to construct the expert mapping for each MoE layer. We keep the evaluation set disjoint from the calibration set and use it exclusively for online serving evaluation. Appendix A.2 justifies transferring these rout-

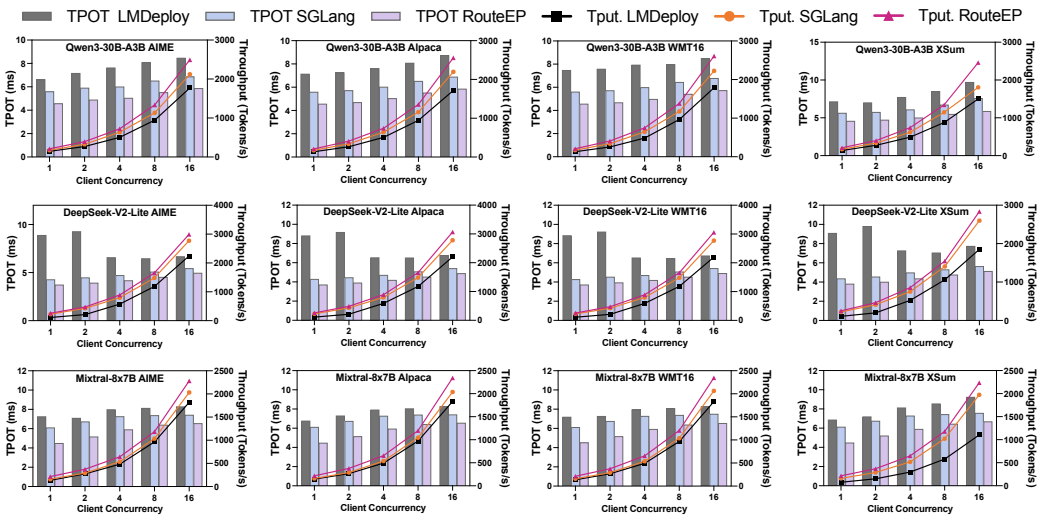


Figure 5: Serving performance across three MoE models and four datasets at client concurrency limits from 1 to 16. Bars show mean TPOT, and lines show output-token throughput. The horizontal axis denotes the client concurrency limit.

ing statistics to decoding: small cross-phase logit perturbations relative to routing margins imply similar position-wise expert frequencies.

LLM serving configuration. For a fair comparison, we evaluate all datasets on the same hardware using identical model checkpoints, input prompts, and generation settings. For the concurrency sweep in Figure 5, we use 64 requests and vary the client concurrency limit over $\{1, 2, 4, 8, 16\}$. All requests generate exactly 512 output tokens with EOS termination disabled, using greedy decoding and the same prompt order across systems. Requests are issued without an arrival-rate limit, subject to the concurrency cap; this cap does not imply a fixed active decode batch size. We disable prefix caching and profiling, warm up each configuration, and perform one measured run per configuration.

Evaluation metrics. To evaluate decoding performance, we primarily use two metrics: **throughput**, and **mean time per output token (TPOT)**. Throughput measures the number of output tokens generated by the serving system per second. TPOT measures the average latency between consecutive output tokens after the first token is generated, reflecting per-token decoding efficiency.

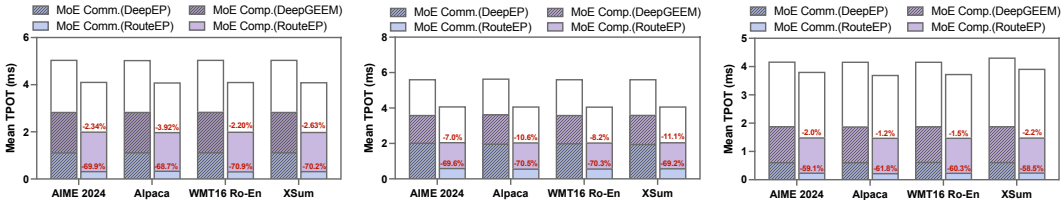
4.2 End-to-End Serving Evaluations

Figure 5 compares the TPOT and output-token throughput of `RouteEP`, SGLang, and LMDeploy as request concurrency increases.

Decode efficiency. For Qwen3-30B-A3B, `RouteEP` achieves average TPOT speedups of $1.20\times$ over SGLang and $1.53\times$ over LMDeploy. For DeepSeek-V2-Lite, `RouteEP` achieves average TPOT speedups of $1.13\times$ over SGLang and $1.84\times$ over LMDeploy. For Mixtral-8x7B, `RouteEP` achieves average TPOT speedups of $1.24\times$ over SGLang and $1.38\times$ over LMDeploy. These results demonstrate that `RouteEP` consistently improves per-token decoding performance across different MoE architectures and serving frameworks, validating the effectiveness of the proposed routing-aware decoding design.

Serving throughput evaluations. We further evaluate the end-to-end throughput of the three models on the four datasets. As shown in Figure 5, `RouteEP` achieves throughput improvements of up to $1.19\times$, $1.12\times$, and $1.23\times$ over SGLang for Qwen, DeepSeek, and Mixtral, respectively. The corresponding improvements over LMDeploy are $1.50\times$, $1.80\times$, and $1.37\times$. These results indicate that the efficient decoding design of `RouteEP` translates into end-to-end serving gains.

Effect of concurrency. Modern LLM serving frameworks dynamically batch multiple concurrent requests during decoding. Therefore, increasing request concurrency generally increases the active decoding batch size, allowing the system to improve GPU utilization and amortize kernel-launch and



(a) Qwen3-30B-A3B (b) Mixtral-8x7B (c) DeepSeek-V2-Lite

Figure 6: Performance breakdown of the single-request decoding step for SGLang and `RouteEP` across three MoE models. Each bar reports mean component time per decode step over 10 requests.

communication overheads across more tokens. At the same time, a larger active batch increases the amount of computation performed at each decoding step, which can lead to higher TPOT. As shown in Figure 5, `RouteEP` consistently outperforms both SGLang and LMDeploy in TPOT and output-token throughput across all evaluated concurrency levels from 1 to 16. These results indicate that `RouteEP` provides the best inference performance among the evaluated state-of-the-art frameworks at all concurrency settings.

4.3 Detailed Breakdown Evaluations

We conduct a component breakdown evaluation for the three models. Specifically, we select 10 requests, perform 64 decoding steps, and calculate the mean breakdown of MoE computation and communication per decoding step. Figure 6 reports this per-step breakdown for Qwen, Mixtral, and DeepSeek in panels (a), (b), and (c), respectively.

According to the results, `RouteEP` reduces communication overhead by 58.5%–70.9% relative to SGLang across the three models. This substantial reduction demonstrates that the proposed routing-aware parallelism effectively lowers communication costs by replacing the conventional `all-to-all` operations. Meanwhile, `RouteEP` reduces MoE computation time by 1.2%–11.1%. These results indicate that the expert grouping scheme improves the routing affinity between experts and EP ranks, leading to a more balanced distribution of expert computation. The reduction in expert computation time is most pronounced for Mixtral-8x7B, which suggests that the proposed expert sharding scheme can better exploit the available GPU parallelism for Top-2 routing on a four-GPU system. In contrast, DeepSeek-V2-Lite exhibits lower reduction in expert computation overhead. As shown by the routing-frequency results in Figure 9 and Figure 10 in Appendix A.6, DeepSeek-V2-Lite inherently exhibits a more balanced expert-routing distribution than the other models, leaving less room for further workload balancing through expert grouping. To evaluate the balance of expert computation, we conduct a rank-level routing analysis in Appendix A.7. The results confirm that `RouteEP` yields a more balanced rank-level workload for all three models.

We further compare the inference performance of the proposed expert shard scheme in Appendix A.4. As shown in Table 1, `RouteEP` exhibits superior inference performance among all datasets, which denotes that the expert sharding in `RouteEP` can significantly improve the resource utilization of multi-GPUs

4.4 Inference Accuracy Evaluations

We use 1,314 GSM8K questions to compare the inference accuracy of `RouteEP` and SGLang. As shown in Table 2, `RouteEP` maintains comparable accuracy to the baseline across the three models.

5 Conclusion

We present `RouteEP`, a routing-aware EP framework for MoE LLM decoding. `RouteEP` combines lead-anchored expert grouping, a routing-aware EP scheme that replaces `all-to-all` dispatch–combine with rank-local expert computation and `all-reduce`, and intra-expert sharding. `RouteEP` substantially reduces communication overhead and better exploits the computing capacity of multi-GPU systems.

REFERENCES

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pp. 131–198. Association for Computational Linguistics, 2016. doi: 10.18653/v1/W16-2301. URL https://aclanthology.org/W16-2301/.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

LMDeploy Contributors. Lmdeploy: A toolkit for compressing, deploying, and serving llm. https://github.com/InternLM/lmdeploy, 2023.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.70/.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Lingxiao Diao, Xinyue Xu, Wanxuan Sun, Cheng Yang, and Zhuosheng Zhang. Guidebench: Benchmarking domain-oriented guideline following for llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11361–11399, 2025.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

Epoch AI. Data on AI models. https://epoch.ai/data/ai-models, 2026. Accessed: September 22, 2026.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Yingqiang Ge, Wenyue Hua, Kai Mei, Juntao Tan, Shuyuan Xu, Zelong Li, Yongfeng Zhang, et al. Openagi: When llm meets domain experts. *Advances in neural information processing systems*, 36:5539–5568, 2023.

Xin He, Shunkang Zhang, Kaijie Tang, Shaohuai Shi, Yuxin Wang, Zihao Zeng, Zhenheng Tang, Xiaowen Chu, Haiyan Yin, Ivor W Tsang, et al. Expertflow: Efficient mixture-of-experts inference via predictive expert caching and token scheduling. *arXiv preprint arXiv:2410.17954*, 2024.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 269–287, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html.

Minghui Jia. AIME 2024. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/Maxwell-Jia/AIME_2024. Accessed: 2026-09-06.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024a. URL https://arxiv.org/abs/2401.04088.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024b. URL https://arxiv.org/abs/2401.04088.

Chenyu Jiang, Ye Tian, Zhen Jia, Shuai Zheng, Chuan Wu, and Yida Wang. Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping. In *Proceedings of Machine Learning and Systems*, volume 6, 2024c. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/339caf45a6fa281cae8adc6465343464-Abstract-Conference.html.

Woosuk Kwon. *vLLM: an efficient inference engine for large language models*. University of California, Berkeley, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020. URL https://arxiv.org/abs/2006.16668.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*,

pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pp. 945–959. USENIX Association, 2023. URL https://www.usenix.org/conference/atc23/presentation/li-jiamin.

Wengang Li, Lingqi Zhang, Toshio Endo, and Mohamed Wahib. Understanding cross-layer contributions to mixture-of-experts routing in llms. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Yan Li, Zhenyu Zhang, Zhengang Wang, Pengfei Chen, and Pengfei Zheng. Semantic parallelism: Redefining efficient MoE inference via model-data co-scheduling. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=MSHPrMpIHZ.

Yinquan Lu, Wenhao Zhu, Lei Li, Yu Qiao, and Fei Yuan. Llamax: Scaling linguistic horizons of llm by enhancing translation capabilities beyond 100 languages. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10748–10772, 2024.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, ChenXue Wang, Shichao Liu, and Qing Wang. Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification. *Proceedings of the ACM on Software Engineering*, 1(FSE):2332–2354, 2024.

Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th international conference on software engineering*, pp. 1–13, 2024.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1206. URL https://aclanthology.org/D18-1206/.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, pp. 118–132, 2024. doi: 10.1109/ISCA59077.2024.00019.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *ACM Transactions on Storage*, 22(4):1–38, 2026.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 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, 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. Promoe: Fast moe-based llm serving using proactive caching. *arXiv preprint arXiv:2410.22134*, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models, 2023. URL https://crfm.stanford.edu/2023/03/13/alpaca.html.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. Contrastive preference optimization: Pushing the boundaries of llm performance in machine translation. *arXiv preprint arXiv:2401.08417*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jinghan Yao, Quentin Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. Exploiting inter-layer expert affinity for accelerating mixture-of-experts model inference. *arXiv preprint arXiv:2401.08383*, 2024. URL https://arxiv.org/abs/2401.08383.

Zhongkai Yu, Yue Guan, Zihao Yu, Chenyang Zhou, Zhengding Hu, Shuyi Pei, Yangwook Kang, Yufei Ding, and Po-An Tsai. Patterns behind chaos: Forecasting data movement for efficient large-scale MoE LLM inference. *arXiv preprint arXiv:2510.05497*, 2026. URL https://arxiv.org/abs/2510.05497. Accepted to the 53rd Annual International Symposium on Computer Architecture.

Li Zhang, Youhe Jiang, Guoliang He, Xin Chen, Han Lv, Qian Yao, Ningsheng Ma, Fangcheng Fu, and Kai Chen. Lmdeploy accelerates mixed-precision llm inference with turbomind. *arXiv preprint arXiv:2508.15601*, 2025a.

Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, Quan Chen, and Xin Liu. COMET: Fine-grained computation-communication overlapping for mixture-of-experts. In *Proceedings of Machine Learning and Systems*, volume 7, 2025b. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/e27ea0cd50b798ff8942caf9203f0992-Abstract-Conference.html.

Zhexiang Zhang, Ye Wang, Yumiao Zhao, Jiayu Xiao, Qianjing Yang, Xiangyu Wang, Jingzhe Jiang, Qizhen Weng, Ruichuan Chen, Shaohuai Shi, et al. Janus: Disaggregating attention and experts for scalable moe inference. *arXiv preprint arXiv:2512.13525*, 2025c.

Chenggang Zhao, Zhean Xu, Liang Zhao, Jiashi Li, Chenhao Xu, Anyi Xu, Shengyu Liu, Kexing Zhou, and Kuai Yu. Deepgemm: clean and efficient blas kernel library on gpu. https://github.com/deepseek-ai/DeepGEMM, 2025a.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. Deepep: an efficient expert-parallel communication library. https://github.com/deepseek-ai/DeepEP, 2025b.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 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. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, et al. Megascale-infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 592–608, 2025.

A APPENDIX

A.1 ALGORITHMIC DETAILS OF OFFLINE EXPERT GROUPING

The offline grouping procedure consists of three functions, applied independently to each MoE layer ℓ. Algorithm 1 converts calibration routes into a position-wise frequency matrix; Algorithm 2 selects distinct lead experts and binds them to rank groups; Algorithm 3 assigns the remaining experts without changing these bindings. Together, they produce the static expert mapping before LLM inference.

Position-wise frequency construction. Algorithm 1 takes the captured routing matrix $\mathbf{Y}^\ell \in \{0, \dots, E-1\}^{N \times K}$, whose entry $\mathbf{Y}^\ell_{t,s}$ is the expert ID at routing position s for token t. It counts occurrences of each expert at each position, yielding $\mathbf{C}^\ell \in \mathbb{N}_0^{K \times E}$ as defined in Equation 1. The procedure preserves the position ordering recorded in the calibration trace.

Algorithm 1: Position-wise routing frequency construction.

Input: $\mathbf{Y}^\ell \in \{0, \dots, E-1\}^{N \times K}$: calibration routes; E: number of experts.
Output: $\mathbf{C}^\ell \in \mathbb{N}_0^{K \times E}$: position-wise routing counts.

Function `ConstructFrequencyMatrix(`$\mathbf{Y}^\ell$, E`)`**:**
  $(N, K) \leftarrow \text{shape}(\mathbf{Y}^\ell)$;
  $\mathbf{C}^\ell \leftarrow \mathbf{0}_{K \times E}$;
  for $s \leftarrow 0$ **to** $K-1$ **do**
    for $e \leftarrow 0$ **to** $E-1$ **do**
      for $t \leftarrow 0$ **to** $N-1$ **do**
        if $\mathbf{Y}^\ell_{t,s} = e$ **then**
          $C^\ell_{s,e} \leftarrow C^\ell_{s,e} + 1$;
  return $\mathbf{C}^\ell$;

Lead selection and rank binding. Algorithm 2 traverses the K routing positions and, following Equation 2, selects for each position the highest-frequency expert that has not yet been chosen. Then, each lead λ^ℓ_m is appended to group $m \bmod R$. The resulting lead set and partially filled groups are passed to the capacity-constrained assignment.

Algorithm 2: Distinct lead selection and rank binding.

Input: $\mathbf{C}^\ell \in \mathbb{N}_0^{K \times E}$: routing counts; R: number of ranks.
Output: Λ^ℓ: set of K distinct leads; $\mathbf{G}^\ell$: R groups with fixed lead placements.

Function `LeadExpertSelection(`$\mathbf{C}^\ell$, R`)`**:**
  $(K, E) \leftarrow \text{shape}(\mathbf{C}^\ell)$;
  Require $1 \le K \le E$, $1 \le R \le E$, and $E \bmod R = 0$;
  $\Lambda^\ell \leftarrow \varnothing$;
  for $m \leftarrow 0$ **to** $K-1$ **do**
    $\lambda^\ell_m \leftarrow \arg\max_{e \in \{0,\dots,E-1\} \setminus \Lambda^\ell} C^\ell_{m,e}$;
    $\Lambda^\ell \leftarrow \Lambda^\ell \cup \{\lambda^\ell_m\}$;
  $\mathbf{G}^\ell \leftarrow (\varnothing, \dots, \varnothing)$; ▷ R empty expert groups
  for $m \leftarrow 0$ **to** $K-1$ **do**
    $r \leftarrow m \bmod R$;
    $\mathbf{G}^\ell_r \leftarrow \mathbf{G}^\ell_r \cup \{\lambda^\ell_m\}$;
  return $\Lambda^\ell, \mathbf{G}^\ell$;

Capacity-constrained assignment. Algorithm 3 completes the expert groups initialized by `LeadExpertSelection` in Algorithm 2. The selected leads remain fixed throughout the as-

signment. For rank r, let $b_r^\ell = B - |\mathbf{G}_r^\ell|$ be the residual capacity of the expert group. Given the K distinct lead experts selected by the preceding procedure, the assignment uses the position-aligned score defined in Section 3.1 and solves Equation 6 to assign the remaining experts to R groups. In this algorithm, `LinearSumAssignment` returns paired row and column indices of a minimum-cost assignment, as provided by `scipy` (Virtanen et al., 2020). Each residual slot receives exactly one remaining expert, ensuring $|\mathbf{G}_r^\ell| = B$ for every rank while preserving all lead placements.

Algorithm 3: Capacity-constrained expert assignment in `RouteEP`.

Input: $\mathbf{C}^\ell \in \mathbb{N}_0^{K\times E}$: routing frequency matrix; Λ^ℓ: K distinct lead experts; $\mathbf{G}^\ell$: R groups initialized by lead selection.
Output: $\mathbf{G}^\ell$: disjoint expert groups, each containing $B = E/R$ experts.

Function `CapacityConstrainedAssignment` ($\mathbf{C}^\ell$, Λ^ℓ, $\mathbf{G}^\ell$, K, E, R) **:**
- **Require** $1 \le K \le E$, $1 \le R \le E$, and $E \bmod R = 0$;
- $B \leftarrow E/R$;
- **Require** $\mathbf{G}^\ell$ contains each lead exactly once, no other experts, and $|\mathbf{G}_r^\ell| \le B$ for every r;
- $\mathcal{U}^\ell \leftarrow \{0, \dots, E-1\} \setminus \Lambda^\ell$;
- $\mathcal{S}^\ell \leftarrow \varnothing$;
- **for** $r \leftarrow 0$ **to** $R-1$ **do**
  - $b_r^\ell \leftarrow B - |\mathbf{G}_r^\ell|$;
  - $\mathcal{S}^\ell \leftarrow \mathcal{S}^\ell \cup \{(r, c) \mid 0 \le c < b_r^\ell\}$;
- **if** $\mathcal{U}^\ell = \varnothing$ **then**
  - **return** $\mathbf{G}^\ell$;
- $M \leftarrow |\mathcal{U}^\ell|$;
- $(a_0, \dots, a_{M-1}) \leftarrow \mathrm{sorted}(\mathcal{S}^\ell)$;
- $(u_0, \dots, u_{M-1}) \leftarrow \mathrm{sorted}(\mathcal{U}^\ell)$;
- **for** $i \leftarrow 0$ **to** $M-1$ **do**
  - $(r, c) \leftarrow a_i$;
  - **for** $j \leftarrow 0$ **to** $M-1$ **do**
    - $D_{i,j}^\ell \leftarrow -\sum_{\substack{0\le s<K \\ s \bmod R = r}} C_{s,u_j}^\ell$; ▷ `Negative position-aligned score`
- $(\mathbf{i}, \mathbf{j}) \leftarrow$ `LinearSumAssignment` ($\mathbf{D}^\ell$) ; ▷ `Minimum-cost perfect matching`
- **for** $p \leftarrow 0$ **to** $M-1$ **do**
  - $(r, c) \leftarrow \mathcal{S}^\ell[\mathbf{i}_p]$;
  - $\mathbf{G}_r^\ell \leftarrow \mathbf{G}_r^\ell \cup \{\mathcal{U}^\ell[\mathbf{j}_p]\}$;
- **return** $\mathbf{G}^\ell$;

A.2 Justification for Prefill-Based Calibration

In this section, we show that prefill routing statistics effectively support expert placement for decoding. The argument separates phase-invariant computation from distribution-dependent calibration: executing the same causal prefix in different phases preserves its mathematical routing computation, but prompt tokens and generated tokens need not have identical routing distributions.

Decomposing routing scores. Following the residual-component perspective of Li et al. (2026a), consider an MoE Transformer with RMSNorm and a linear router. For a token and its causal context x, let $\mathbf{v}(x)$ denote its embedding, and let $\mathbf{a}^j(x)$ and $\mathbf{f}^j(x)$ denote the attention and MoE residual updates in block j, respectively. The residual state immediately before the router normalization in block ℓ is

$$\begin{aligned}\mathbf{h}^\ell(x) &= \mathbf{a}^\ell(x) + \mathbf{f}^{\ell-1}(x) \\ &= \mathbf{v}(x) + \sum_{j=0}^{\ell} \mathbf{a}^j(x) + \sum_{j=0}^{\ell-1} \mathbf{f}^j(x).\end{aligned} \tag{10}$$

Let $\mathbf{W}_g^\ell$ be the router matrix, $\boldsymbol{\gamma}^\ell$ be the normalization gain, and $\mathbf{b}_g^\ell$ be the optional router bias. Define $\mathbf{A}^\ell = \mathbf{W}_g^\ell \operatorname{diag}(\boldsymbol{\gamma}^\ell)$ and $d^\ell(x) = \sqrt{\|\mathbf{h}^\ell(x)\|_2^2/H + \epsilon_{\text{norm}}}$, where H is the hidden dimension. The router logits after applying $\mathrm{RMSNorm}$ can be written as

$$\begin{aligned}\mathbf{z}^\ell(x) &= \mathbf{W}_g^\ell \cdot \mathrm{RMSNorm}(\mathbf{h}^\ell(x)) + \mathbf{b}_g^\ell \\ &= \frac{\mathbf{A}^\ell \cdot \mathbf{h}^\ell(x)}{d^\ell(x)} + \mathbf{b}_g^\ell \\ &= \frac{\mathbf{A}^\ell \cdot \mathbf{v}(x)}{d^\ell(x)} + \sum_{j=0}^{\ell} \frac{\mathbf{A}^\ell \cdot \mathbf{a}^j(x)}{d^\ell(x)} + \sum_{j=0}^{\ell-1} \frac{\mathbf{A}^\ell \cdot \mathbf{f}^j(x)}{d^\ell(x)} + \mathbf{b}_g^\ell.\end{aligned} \tag{11}$$

All components share the normalization denominator of the complete residual state; individual components are not normalized separately. This decomposition identifies the quantities governing routing, rather than assuming that their distributions remain unchanged across phases.

Phase-specific logits and their difference. For the prefill and decoding phases, the router logits for x_q, where $q \in \{\mathrm{pre}, \mathrm{dec}\}$, are given by

$$\begin{aligned}\mathbf{z}_{\mathrm{pre}}^\ell &= \mathbf{b}_g^\ell + \frac{\mathbf{A}^\ell \cdot (\mathbf{v}(x_{\mathrm{pre}}) + \sum_{j=0}^{\ell} \mathbf{a}^j(x_{\mathrm{pre}}) + \sum_{j=0}^{\ell-1} \mathbf{f}^j(x_{\mathrm{pre}}))}{d_{\mathrm{pre}}^\ell}, \\ \mathbf{z}_{\mathrm{dec}}^\ell &= \mathbf{b}_g^\ell + \frac{\mathbf{A}^\ell \cdot (\mathbf{v}(x_{\mathrm{dec}}) + \sum_{j=0}^{\ell} \mathbf{a}^j(x_{\mathrm{dec}}) + \sum_{j=0}^{\ell-1} \mathbf{f}^j(x_{\mathrm{dec}}))}{d_{\mathrm{dec}}^\ell}.\end{aligned} \tag{12}$$

Each phase shares a denominator across its residual components, but the denominators need not agree between the two phases. The logit difference $\Delta\mathbf{z}^\ell = \mathbf{z}_{\mathrm{dec}}^\ell - \mathbf{z}_{\mathrm{pre}}^\ell$ can then be written as

$$\begin{aligned}\Delta\mathbf{z}^\ell = &\frac{\mathbf{A}^\ell \Delta\mathbf{v}}{d_{\mathrm{dec}}^\ell} + \frac{\mathbf{A}^\ell \Delta\mathbf{a}_\Sigma^\ell}{d_{\mathrm{dec}}^\ell} + \frac{\mathbf{A}^\ell \Delta\mathbf{f}_\Sigma^\ell}{d_{\mathrm{dec}}^\ell} \\ &+ \mathbf{A}^\ell \mathbf{h}_{\mathrm{pre}}^\ell \left(\frac{1}{d_{\mathrm{dec}}^\ell} - \frac{1}{d_{\mathrm{pre}}^\ell} \right).\end{aligned} \tag{13}$$

Here, $\Delta\mathbf{v} = \mathbf{v}(x_{\mathrm{dec}}) - \mathbf{v}(x_{\mathrm{pre}})$, $\Delta\mathbf{a}_\Sigma^\ell = \sum_{j=0}^{\ell}(\mathbf{a}^j(x_{\mathrm{dec}}) - \mathbf{a}^j(x_{\mathrm{pre}}))$, and $\Delta\mathbf{f}_\Sigma^\ell = \sum_{j=0}^{\ell-1}(\mathbf{f}^j(x_{\mathrm{dec}}) - \mathbf{f}^j(x_{\mathrm{pre}}))$. This equation provides the bridge from cross-phase representation shifts to routing stability: it expresses the router-logit perturbation entirely in terms of the embedding, attention, MoE-residual, and RMSNorm-scale differences between the coupled prefill and decode contexts. Bounding these terms therefore directly bounds the perturbation of every expert logit. We then show that when this perturbation is smaller than the routing margin, the ordered Top-K assignment is preserved, which in turn bounds the discrepancy between prefill and decode expert-access frequencies.

From logit differences to routing consistency. We next show how the cross-phase logit difference affects expert routing. Let

$$\eta^\ell = \left\| \mathbf{z}_{\mathrm{dec}}^\ell - \mathbf{z}_{\mathrm{pre}}^\ell \right\|_\infty = \max_e \left| z_{\mathrm{dec},e}^\ell - z_{\mathrm{pre},e}^\ell \right| \tag{14}$$

denote the maximum change of any expert logit between a paired prefill and decode context. According to Equation 13, η^ℓ is determined by the cross-phase changes in the embedding, attention residuals, MoE residuals, and normalization scale.

Assume that the router selects the $K < E$ largest logits in descending order. Let $z_{\mathrm{pre},(j)}^\ell$ denote the j-th largest prefill logit, and define the minimum routing margin as

$$m^\ell = \min_{1 \le j \le K} \left(z_{\mathrm{pre},(j)}^\ell - z_{\mathrm{pre},(j+1)}^\ell \right). \tag{15}$$

Since every expert logit changes by at most η^ℓ, the gap between two adjacent logits can decrease by at most $2\eta^\ell$. Therefore,

$$m^\ell > 2\eta^\ell \quad \Longrightarrow \quad \mathbf{Y}_{\mathrm{pre}}^\ell = \mathbf{Y}_{\mathrm{dec}}^\ell, \tag{16}$$

where $\mathbf{Y}_q^\ell$ denotes the ordered Top-K routing result in phase q. Thus, when the cross-phase logit change is smaller than half of the routing margin, the ordered Top-K expert assignment remains unchanged.

We further relate this routing stability to expert-access frequency. Let $Y_{q,s}^\ell$ denote the expert selected at Top-K position s, and define

$$p_{q,s}^\ell(e) = \Pr\left[Y_{q,s}^\ell = e\right] \tag{17}$$

as the access probability of expert e at position s. Let

$$\epsilon^\ell = \Pr\left[m^\ell \le 2\eta^\ell\right] \tag{18}$$

denote the fraction of paired token–context samples for which the routing-stability condition may not hold. Since the selected expert can differ only on these samples, for any expert e we have

$$\left|p_{\mathrm{pre},s}^\ell(e) - p_{\mathrm{dec},s}^\ell(e)\right| \le \epsilon^\ell. \tag{19}$$

Therefore, if the router-logit changes are smaller than the routing margins for most samples, the per-position expert-access frequencies observed during prefill remain close to those during decoding. This directly supports using prefill routing statistics to construct the decode-time expert placement.

A.3 Software Environment and Backend Configuration

The SGLang and LMDeploy environments use Python 3.12.13, PyTorch 2.9.1+cu130, Triton 3.5.1, Transformers 5.5.4, and FlashInfer 0.6.8.post1. The system runs Ubuntu 24.04.5 with NVIDIA driver 580.178.04, and the configured CUDA compiler is nvcc 13.0.88. The communication dependencies are NCCL 2.30.4 and NVSHMEM 3.3.24. We design `RouteEP` mainly based on SGLang v0.5.11 and compare it with LMDeploy 0.17.0. The installed DeepEP and DeepGEMM builds are 1.2.1 and 2.6.1, respectively; both include local modifications for efficient LLM serving. For a fair comparison, all systems use identical checkpoints, BF16 model and KV-cache precision, four-way attention TP, and a KV-cache capacity of 131,072 tokens. The default EP degree is four, and the maximum number of concurrent requests scales from 1 to 16.

A.4 Evaluation for Expert Sharding

In this section, we further evaluate the effectiveness of the expert-sharding design in `RouteEP`. We use Mixtral-8×7B, whose Top-2 routing makes it particularly suitable for evaluating expert sharding across four GPUs. We compare four configurations: the original SGLang baseline (**SGLang**), our routing-aware expert placement without expert sharding (**Route**), SGLang with expert sharding using a configuration with 2-way expert parallelism and 2-way tensor parallelism (**SGLShard**, EP=2, TP=2), and the complete `RouteEP` design that combines routing-aware expert placement with expert sharding. Table 1 reports TPOT and output-token throughput across four workloads and batch sizes from 1 to 16.

Compared with **Route**, introducing expert sharding yields a further $1.04\times$–$1.24\times$ TPOT speedup and improves output-token throughput by $1.03\times$–$1.23\times$ across all evaluated workloads and batch sizes. The improvement is particularly pronounced at small batch sizes, where sparse expert execution provides fewer opportunities to amortize per-rank computation imbalance. As the batch size increases, the performance gap gradually narrows because more tokens are available to improve GPU utilization and balance expert execution.

Compared with **SGLShard**, the complete `RouteEP` achieves an additional $1.03\times$–$1.09\times$ TPOT speedup and $1.02\times$–$1.07\times$ higher throughput. This result shows that expert sharding alone does not fully eliminate expert-computation imbalance. By combining sharding with routing-aware expert placement, `RouteEP` distributes frequently co-activated experts more effectively across ranks and further improves workload balance during decoding.

Overall, these results demonstrate that expert sharding and routing-aware expert placement provide complementary benefits. Expert sharding increases the available parallelism for sparse expert computation, while routing-aware placement improves the distribution of routed tokens across the participating ranks. Their combination enables `RouteEP` to consistently achieve the lowest TPOT and highest throughput across all evaluated batch sizes.

Table 1: Ablation of expert sharding on Mixtral-8x7B. Lower TPOT and higher output-token throughput indicate better performance.

Dataset	Client Concurrency	TPOT (ms)				Throughput (tokens/s)			
		SGLang	Route	SGLShard	RouteEP	SGLang	Route	SGLShard	RouteEP
AIME	1	6.12	5.55	4.83	**4.49**	160.17	176.16	202.97	**216.38**
	2	6.73	6.11	5.50	**5.18**	289.91	318.48	356.07	**373.88**
	4	7.27	6.63	6.17	**5.91**	536.89	587.08	628.56	**642.53**
	8	7.38	6.75	6.61	**6.40**	1033.40	1131.61	1156.99	**1184.42**
	16	7.43	6.82	6.83	**6.55**	2033.25	2210.55	2211.70	**2281.87**
Alpaca	1	6.12	5.55	4.84	**4.48**	159.83	176.04	202.54	**216.86**
	2	6.75	6.12	5.50	**5.16**	290.51	319.99	357.49	**378.09**
	4	7.28	6.64	6.21	**5.94**	538.66	590.59	633.92	**655.87**
	8	7.40	6.75	6.62	**6.42**	1036.34	1143.29	1168.73	**1198.46**
	16	7.43	6.81	6.78	**6.55**	2063.83	2261.66	2289.84	**2342.95**
WMT16	1	6.13	5.61	4.87	**4.54**	159.84	173.43	200.50	**212.87**
	2	6.76	6.12	5.48	**5.17**	288.74	319.94	358.48	**377.03**
	4	7.29	6.62	6.16	**5.92**	538.35	591.87	638.88	**654.37**
	8	7.39	6.75	6.60	**6.39**	1043.80	1140.35	1176.77	**1205.04**
	16	7.50	6.82	6.81	**6.55**	2036.35	2258.01	2279.31	**2346.29**
XSum	1	6.13	5.58	4.90	**4.49**	159.60	174.17	199.31	**216.42**
	2	6.76	6.27	5.50	**5.22**	289.25	308.08	355.36	**369.97**
	4	7.29	6.66	6.12	**5.91**	518.35	571.12	633.27	**649.40**
	8	7.45	6.81	6.64	**6.45**	1022.69	1120.26	1153.76	**1179.13**
	16	7.57	6.96	6.94	**6.69**	1974.59	2154.61	2192.09	**2235.98**

A.5 Inference Accuracy Evaluation

We perform accuracy evaluations to compare `RouteEP` with the SGLang baseline on GSM8K (Cobbe et al., 2021), a benchmark of grade-school mathematical reasoning. Following the SGLang evaluation protocol, we generate responses to 1,314 questions and compute accuracy as the fraction of questions for which the final numerical answer extracted from the model response matches the reference answer. We report the number of correct answers and the corresponding accuracy, together with the absolute change in percentage points (pp) in Table 2.

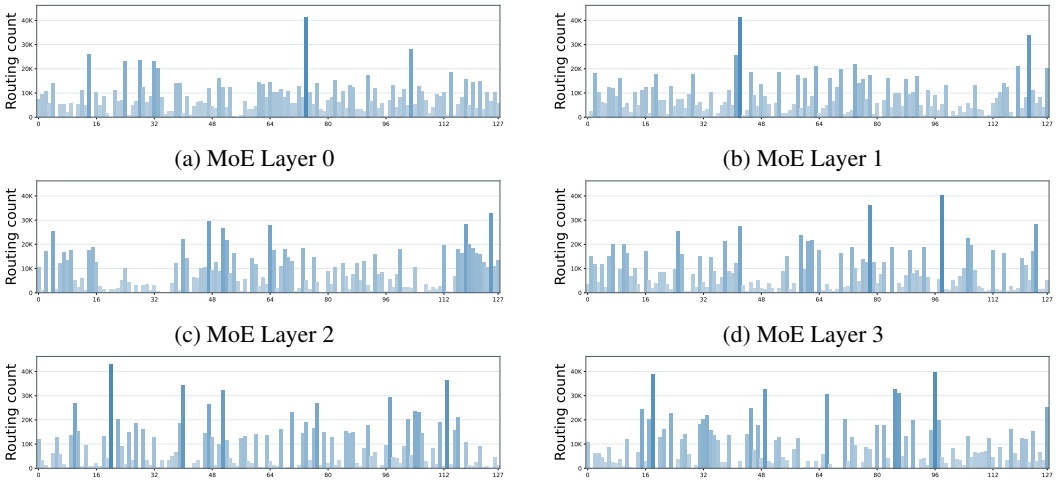
Table 2: Task-level accuracy on 1,314 GSM8K questions. Δ denotes `RouteEP` minus baseline accuracy in percentage points; positive values indicate higher accuracy.

Models	Baseline		RouteEP (ours)		Δ (pp)
	Correct/Total	Acc. (%)	Correct/Total	Acc. (%)	
Qwen3-30B-A3B	1266/1314	96.35	1260/1314	95.89	-0.46
Mixtral-8×7B	843/1314	64.16	844/1314	64.23	$+0.08$
DeepSeek-V2-Lite	891/1314	67.81	909/1314	69.18	$+1.37$

The reported results show that `RouteEP` maintains accuracy comparable to the baseline across the three models. Qwen3-30B-A3B achieves 95.89% versus 96.35%, a decrease of 0.46 pp corresponding to six fewer correct answers. Mixtral-8x7B changes from 64.16% to 64.23% ($+0.08$ pp), while DeepSeek-V2-Lite changes from 67.81% to 69.18% ($+1.37$ pp).

A.6 Routing Frequency Analysis

To characterize the routing patterns of the MoE models before expert grouping, we conduct a detailed token-routing frequency analysis on all three models. Specifically, we first use the calibration set prepared in Section 4 to perform prefill inference for all N tokens and capture the Top-K routing results $\mathbf{Y}^{\ell}$ for each MoE layer ℓ. Based on these routing results, we calculate the token routing frequency for each expert and the matrix $\mathbf{C}^{\ell}$. Figures 7–11 show detailed routing frequency results for selected MoE layers of all three models.

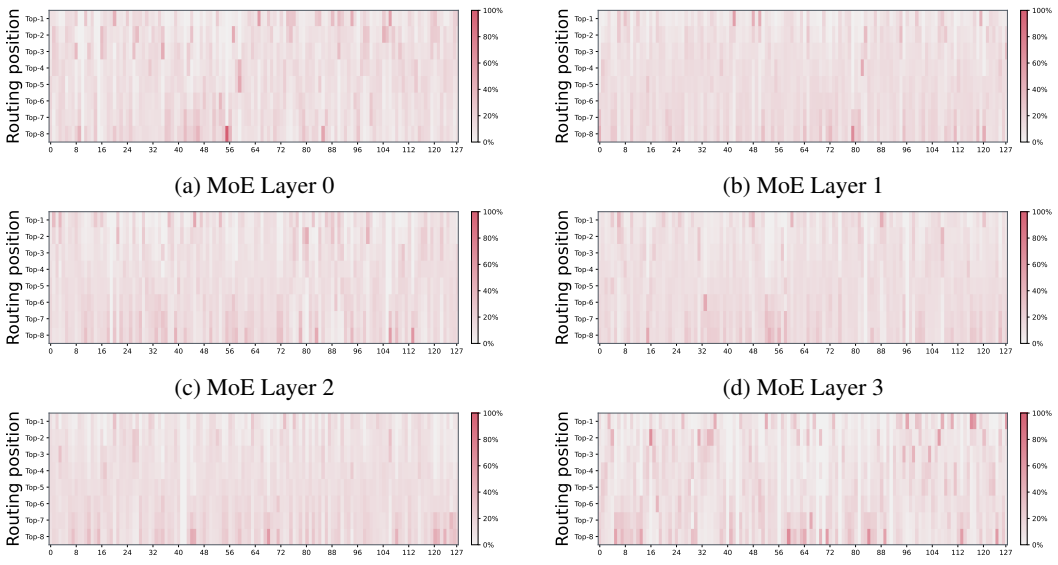


(a) MoE Layer 0 (b) MoE Layer 1

(c) MoE Layer 2 (d) MoE Layer 3

(e) MoE Layer 4 (f) MoE Layer 47

Figure 7: Routing frequencies for each expert ID in representative MoE layers of Qwen3-30B-A3B. Layers 0–4 and the final layer exhibit distinct, layer-dependent routing distributions.



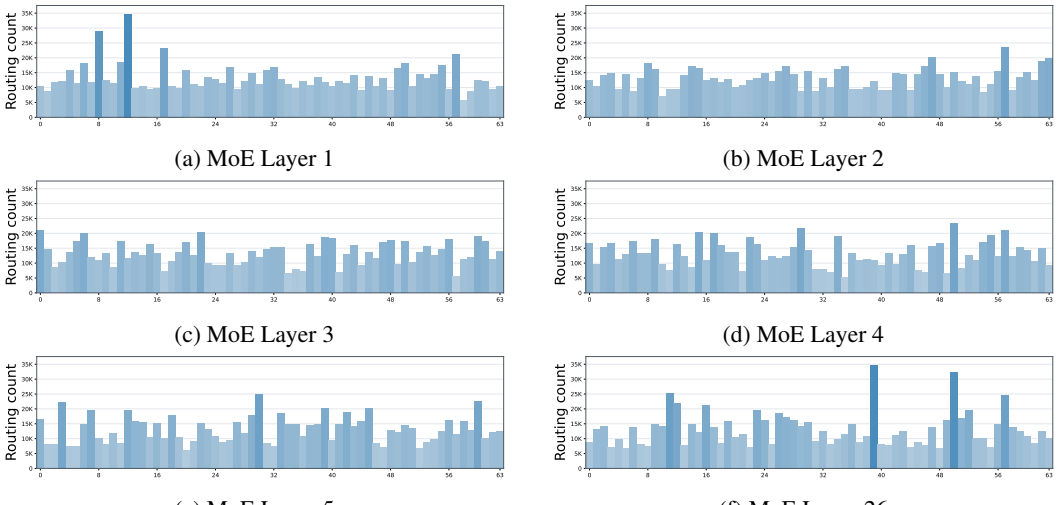
(a) MoE Layer 0 (b) MoE Layer 1

(c) MoE Layer 2 (d) MoE Layer 3

(e) MoE Layer 4 (f) MoE Layer 47

Figure 8: Routing-frequency heatmap across expert IDs for each Top-K position in the Qwen3-30B-A3B model. Layers 0–4 and the final layer show distinct dominant experts for each Top-K position. Token routing for each Top-K position is more balanced than in the other models.

According to the token-routing frequency results in Figure 7, Figure 9, and Figure 11(a), expert workloads are imbalanced across all models, and some experts have high routing frequencies. Although Mixtral-8x7B has only eight experts per layer, its routing behavior is still imbalanced. This observation motivates distributing these high-frequency experts evenly across EP ranks to bal-

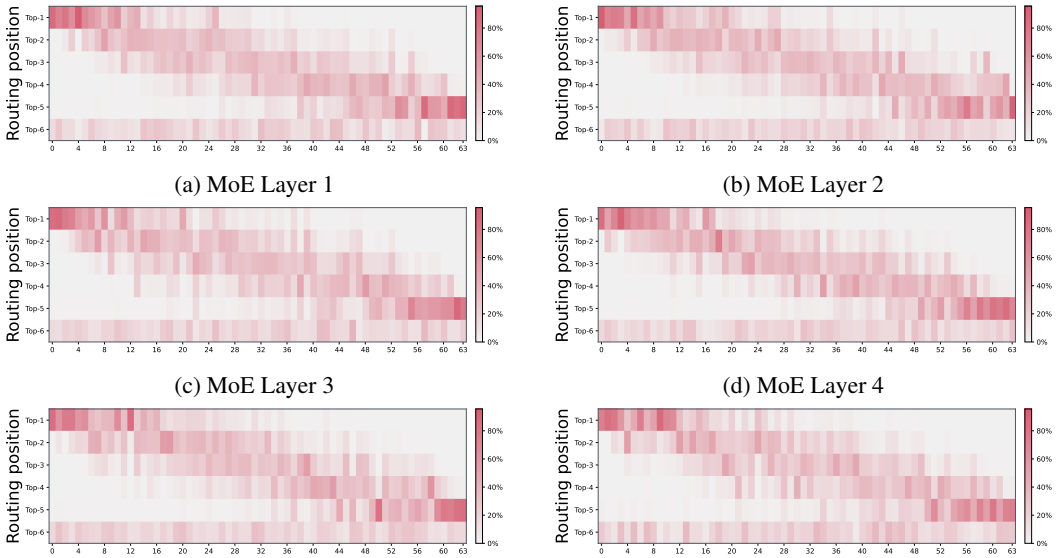


(a) MoE Layer 1 (b) MoE Layer 2

(c) MoE Layer 3 (d) MoE Layer 4

(e) MoE Layer 5 (f) MoE Layer 26

Figure 9: Routing frequencies for each expert ID in representative MoE layers of DeepSeek-V2-Lite. Layers 1–5 and the final layer exhibit distinct, layer-dependent routing distributions.



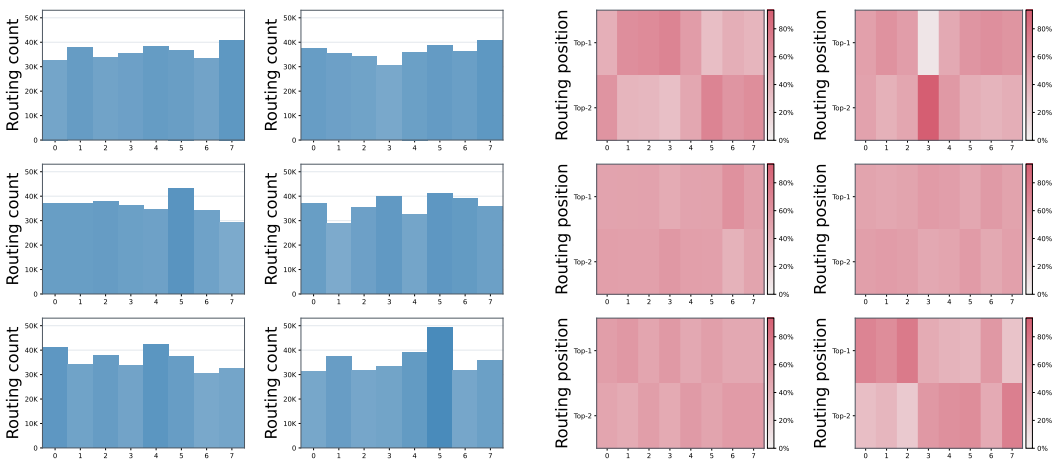
(a) MoE Layer 1 (b) MoE Layer 2

(c) MoE Layer 3 (d) MoE Layer 4

(e) MoE Layer 5 (f) MoE Layer 26

Figure 10: Routing-frequency heatmap across expert IDs for each Top-K position in DeepSeek-V2-Lite. Layers 1–5 and the final layer show distinct dominant experts for each Top-K position.

ance expert computation and reduce synchronization overhead. According to the routing-frequency heatmaps in Figure 8, Figure 10, and Figure 11(b), we observe that different positions in Top-K exhibit distinct expert-routing preferences. In particular, each routing position tends to dispatch tokens to a different set of experts, with different high-frequency experts dominating across Top-K positions. Compared with the other evaluated models, DeepSeek-V2-Lite exhibits a more regular and balanced routing pattern across different Top-K positions. Consequently, its baseline expert mapping already achieves a relatively balanced distribution of expert computation, leaving less room for performance gains through further workload-balancing schemes. As a result, the performance improvement of `RouteEP` on DeepSeek-V2-Lite is smaller than that observed on the other models, as shown in Figure 6(c).

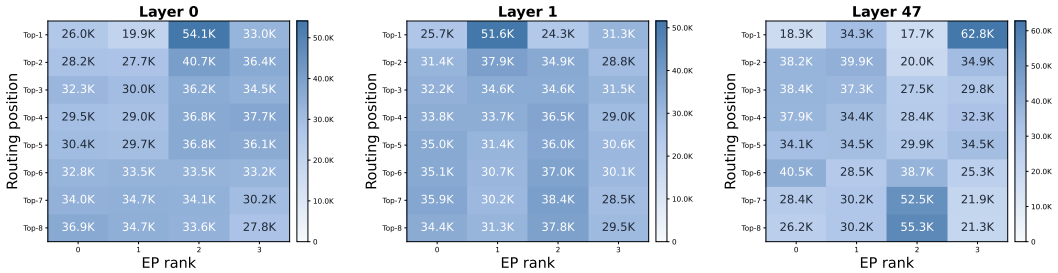


(a) Total routing frequency (b) Routing frequency heatmap

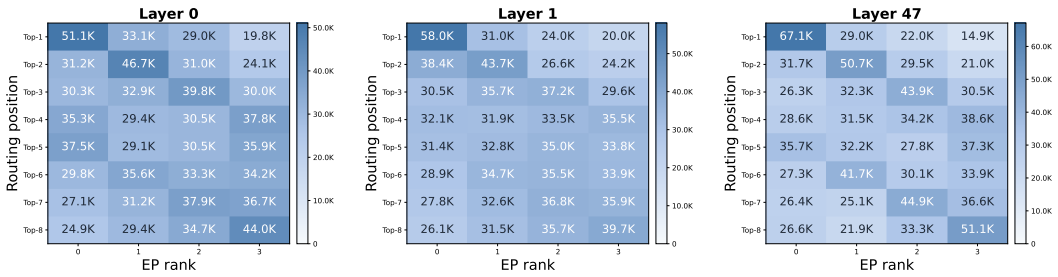
Figure 11: Expert routing frequency evaluations for Layers 0–4 and the final layer of Mixtral-8x7B. (a) Token routing frequencies for each expert ID. (b) Routing-frequency heatmap across expert IDs for each Top-K position.

A.7 Expert Grouping Evaluations

To further evaluate the effectiveness of the expert grouping, we conduct a rank-level routing analysis by comparing the baseline expert mapping with the grouped expert mapping. Figures 12, 13, and 14 present rank-level routing-count heatmaps for each Top-K position, where each cell (i, j) represents the number of tokens routed at the i-th Top-K position to experts located on rank j.



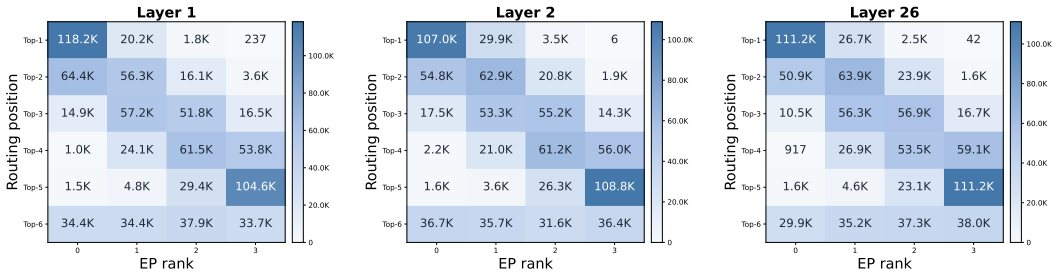
(a) Rank-level routing analysis for the baseline expert distribution.



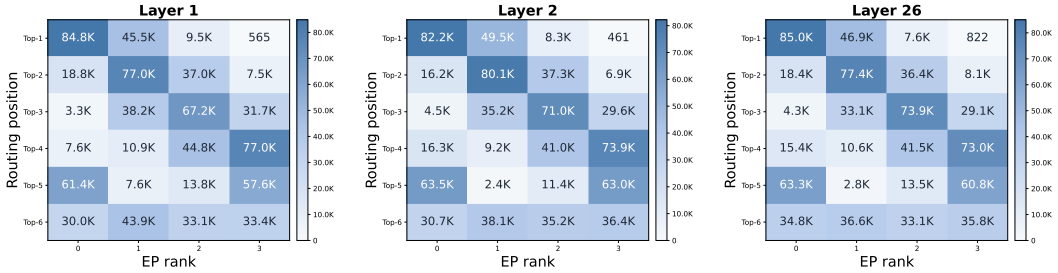
(b) Rank-level routing analysis for the grouped expert distribution.

Figure 12: Rank-level routing-count heatmap across Top-K positions in Qwen3-30B-A3B.

According to Figures 12(a), 13(a), and 14(a), the baseline sequential expert loading results in imbalanced token routing counts across EP ranks, which can lead to workload skew and synchronization stalls during expert computation. After expert grouping, the routing workload becomes substantially more balanced across ranks, as shown in Figures 12(b), 13(b), and 14(b). This improved balance enables more efficient parallel execution and reduces expert computation overhead. These results demonstrate that the proposed expert grouping strategy effectively balances expert workloads across



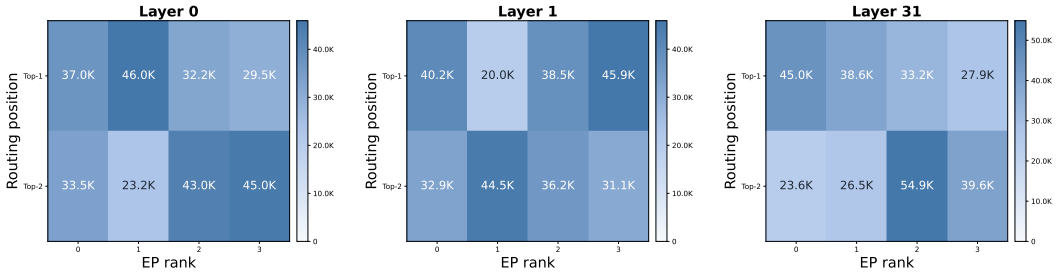
(a) Rank-level routing analysis for the baseline expert distribution.



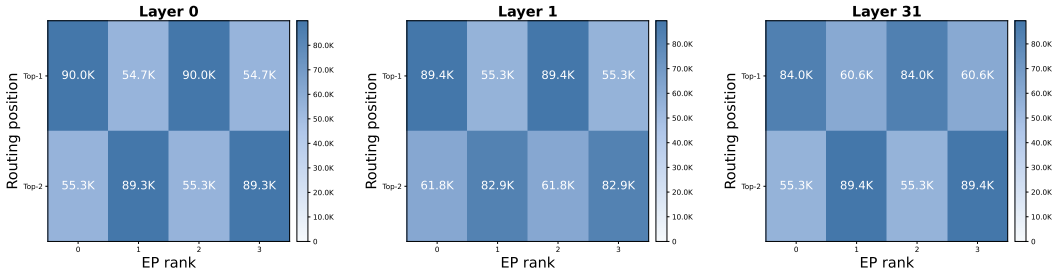
(b) Rank-level routing analysis for the grouped expert distribution.

Figure 13: Rank-level routing-count heatmap across Top-K positions in DeepSeek-V2-Lite.

EP ranks and mitigates the computational overhead caused by routing imbalance, as shown in Figure 6.



(a) Rank-level routing analysis for the baseline expert distribution.



(b) Rank-level routing analysis for the grouped expert distribution.

Figure 14: Rank-level routing-count heatmap across Top-K positions in Mixtral-8x7B.

A.8 AI Use Statement

In this work, we used generative AI tools for organizing experimental data (e.g., collecting metrics from experiment logs into tables), literature searching, drafting parts of the paper, and as a polishing aid for writing and grammar. We have not used generative AI tools for research ideation: the research questions, the experimental design, the analyses, and all claims are the authors' own. All generated work by AI was reviewed by the authors, and its outputs were verified against the results reported

here, references found with AI assistance were checked against the original publications, and all AI-assisted text was checked and revised by the authors. We take full responsibility for the final content of this work.