RANKCAST: PHASE-AWARE RANK BALANCING IN MOE SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern large language model (LLM) inference increasingly adopts disaggregated architectures that separate Prefill (P) and Decode (D) phases. Meanwhile, Expert Parallelism (EP) has been widely adopted in large Mixture-of-Experts (MoE) models serving. However, load imbalance still slows down Mixture-of-Experts (MoE) inference because each layer must wait for the busiest EP rank. The main source of latency differs between prefill and decode. Prefill is compute-bound, and its MoE Layers' latency is determined by the maximum token count of the busiest rank. At the same time, under moderate batch sizes, Decode is memory-bound, so the latency is dominated by the maximum active-expert count across ranks. To solve both problems under a unified framework, we present RankCast, a system that reduces rank-level maximum token counts during prefill and active-expert counts during decode. For every token, RankCast preserves the predicted Top-p experts and chooses the remaining $k-p$ from its Top-L candidates ($L > k$). In prefill, RankCast yields a price per rank that steers flexible assignments to less loaded ranks. In decode, RankCast closes experts on the busiest ranks while preserving the constraints for each token. Both paths are implemented and tested in the state of the art inference frameworks SGLang (Zheng et al., 2024) and DeepEP (Zhao et al., 2025b). We overlap the RankCast's kernel with model execution and reuses the existing DeepEP communication channel for meta data transmission. On Qwen3-235B-A22B with EP$=8$ and Qwen3.5-397B-A17B with EP$=16$, RankCast removes over 75% of the busiest rank's excess prefill load and improves prefill input throughput by up to 19%, while reducing decode time per output token by up to 10%, nearly lossless.

1 INTRODUCTION

Mixture-of-Experts (MoE) architectures are used in model families including Mixtral, LongCat, DeepSeek-V3/V4, Qwen3, and Kimi (Jiang et al., 2024; Meituan LongCat Team, 2025; DeepSeek-AI, 2024; 2026; Yang et al., 2025; Kimi Team, 2026). In MoE model serving, expert parallelism (EP) partitions expert weights across ranks and exchanges token activations through dispatch and combine collectives. These synchronization barriers cause the layer's latency to be bottlenecked by the busiest rank. Although a lot of models deploy per expert imbalance control algotrithms including auxiliary loss and quantile balancing, during serving time, the frozen router still faces batch-level variation and domain drift, and what really matters here is the instantaneous load on each physical EP rank rather than the long-horizon utilization of each expert. DeepSeek-V3 notes that domain shift can cause this imbalance at inference and deploys redundant experts in serving to counter it (DeepSeek-AI, 2024). UltraEP reports pre-balancing inter-rank imbalance ratios of 1.30–4.01 in large-EP training and serving-prefill workloads and observes that expert popularity shifts across microbatches, layers, and data domains (Wei et al., 2026).

PD disaggregation is also well adopted in modern MOE model serving. However, Prefill and decode have different balancing objectives. In prefill, computation and communication grow linearly with the number of tokens a rank receives, and the layer's total latency is determined by the slowest rank due to synchronization barriers. In decode at moderate batch sizes, expert GEMM time is dominated by loading the weights of active experts, and additional tokens matter little until the tokens assigned to a single expert exceed one block's capacity. Our profiling (Appendix D.2) confirms both regimes. Reducing prefill latency therefore requires minimizing the maximum token load across

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.

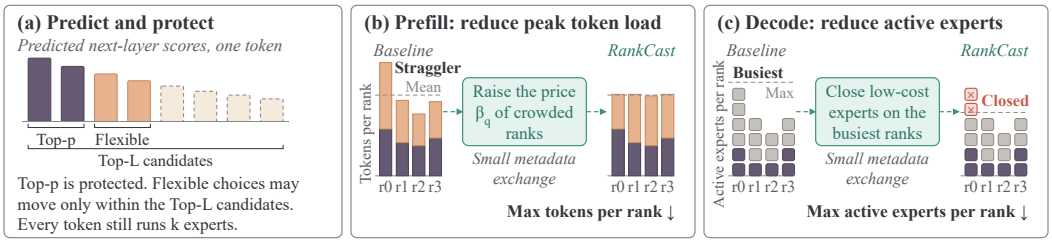


Figure 1: RankCast overview. (a) Predicted next-layer routing defines protected Top-p experts and Top-L candidates. (b) Prefill uses rank prices to redistribute flexible assignments and reduce peak token load. (c) Decode closes low-cost experts to reduce the peak active-expert count while preserving protected experts and at least k candidates per token.

ranks, whereas reducing decode latency requires minimizing the maximum number of active experts across ranks. OEA likewise observes that MoE decode latency scales approximately linearly with the number of active experts in this regime (Oncescu et al., 2026).

Current inference systems typically address this imbalance challenge through expert placement, replication, and predictive prefetching (DeepSeek-AI, 2025; Han et al., 2025; Yang et al., 2026; Zhu et al., 2026). These approaches preserve the router's expert choices but exchange memory or communication bandwidth for load balance. UltraEP shows that placement built from historical load cannot track these fast shifts and can even worsen imbalance (Wei et al., 2026). Replicas occupy HBM that would otherwise hold KV cache; CRAFT reports that conventional replication reduces available KV-cache capacity by 19%–75% and can lower goodput when the concurrency loss exceeds the balancing gain (Zhao et al., 2026). Prefetching avoids permanent residency but must fit expert-weight transfers into a short hiding window and schedule them around token all-to-all (Zhu et al., 2026); across nodes, UltraEP deems moving expert state over the slower scale-out network impractical on the hot path (Wei et al., 2026). Both costs are acute in long-context serving, where KV-cache capacity is scarce and KV-cache transmission adds additional communication pressure.

Rerouting tokens to experts on less loaded ranks in Prefill or to already active experts in Decode provide a new way to avoid both costs. This is the key insight behind RankCast. However, changing a frozen model's expert choices could affect output quality. RANKCAST (Figure 1) therefore limits rerouting through a protected Top-p prefix and a Top-L candidate set, with $p \leq k \leq L$, so balancing changes only lower-ranked choices while keeping k experts per token. For prefill, each rank receives a price that is subtracted from the scores of the candidates it hosts, so crowded ranks receive fewer token assignments.

Decode requires a different objective because token prices do not directly reduce the cost of loading expert weights. RankCast starts from the experts activated by the Top-k routing and closes experts on the busiest ranks (Figure 1c), keeping every Top-p expert open and leaving each token at least k candidates from its Top-L set. The target router then selects its Top-k using its own scores with closed experts suppressed.

To make balancing decisions available before routing, both phases predict the next layer's routing and carry a compact metadata information through DeepEP's existing dispatch communication, so price and mask computation overlap with model execution (Section 4) and introduces no additional synchronization barrier for communication.

Contributions.

- We characterize and formulate EP balancing as a phase-aware, rank-level problem: prefill is bottlenecked by maximum per-rank token load, while decode is dominated by the maximum number of active experts across ranks.
- We develop a prefill algorithm that preserves each token's Top-p prefix and balances the remaining assignments with compact distributed histograms and rank prices.
- We develop a decode algorithm that closes experts to lower the peak active-expert count per rank while preserving each token's protected experts and at least k candidates.

- We implement both paths in SGLang and DeepEP and evaluate them end-to-end. On Qwen3-235B-A22B with EP$= 8$ and Qwen3.5-397B-A17B with EP$= 16$, RankCast removes over 75% of the busiest rank's excess prefill load, improves prefill input throughput by up to 19%, and reduces decode TPOT by up to 10%, at a quality cost below one percentage point.

2 Related Work

Training-time balancing. Switch Transformers, BASE, and Expert Choice balance expert utilization through auxiliary losses, balanced assignment, or expert-side selection (Fedus et al., 2022; Lewis et al., 2021; Zhou et al., 2022). Auxiliary-loss-free balancing instead updates per-expert routing biases from observed loads (Wang et al., 2024), which DeepSeek-V3/V4 combine with a small sequence-wise loss (DeepSeek-AI, 2024; 2026). Kimi K3's Quantile Balancing derives each expert's training-time bias from the dual of a balanced-assignment problem, which makes it a quantile of router-score margins that K3 estimates from all-reduced histograms (Kimi Team, 2026); RankCast's prefill prices play the same dual role for physical ranks at serving time.

Placement, replication, and prefetching. Placement methods keep logical routing but change the expert-to-device mapping: DeepSeek-V3 and EPLB replicate and rearrange hot experts from estimated loads (DeepSeek-AI, 2024; 2025), MoETuner solves a placement ILP over compute and communication skew (Go & Mahajan, 2025), Semantic Parallelism co-schedules placement with rebatching (Li et al., 2026), and Lina allocates more devices to experts predicted to be popular (Li et al., 2023).

Replication and prefetching adapt capacity to hot experts: GRACE-MoE, CRAFT, and Libra allocate replicas by load skew over affinity-based groups, per-layer benefit, or lookahead (Han et al., 2025; Zhao et al., 2026; Yang et al., 2026); PROBE prefetches predicted hot experts within a hiding window (Zhu et al., 2026), and MoonEP prefetches redundant experts planned from the current router outputs (Chen et al., 2026); UltraEP gathers exact post-gating loads per microbatch and jointly plans replication and rerouting (Wei et al., 2026). All of them trade replica storage or expert-state movement for balance.

Inference-time rerouting. Capacity-aware inference (He et al., 2026) limits the number of tokens assigned to each expert and drops or reroutes any tokens that exceed that expert's capacity; LASER reroutes ambiguous tokens to less-loaded experts within a score-derived viable set (Shahout et al., 2025). Both target token-count balance, the prefill objective. RankCast also reroutes at inference time but balances physical-rank load and changes only lower-ranked choices within a limited candidate set.

Decode-time active-expert consolidation. In memory-bound decode, activating another expert can cost more than adding tokens to an active one. Lynx remaps low-affinity assignments onto a smaller active set (Gupta et al., 2024); SERE redirects secondary assignments to similar active experts (Wu et al., 2026); OEA lets additional assignments piggyback on experts the batch already activates (Oncescu et al., 2026). XShare is the closest comparison for the EP objective: it seeds a shared pool with each token's top-ranked experts, grows it by batch-summed gating score up to a preset budget, and, in its EP-aware variant, applies the budget per GPU (Vankov et al., 2026). RankCast instead searches for closures that reduce the maximum active-expert count across ranks subject to each token's predicted candidate-set and protected-prefix constraints. It validates closures jointly because closures on different ranks can affect the same token, and it overlaps the search with model execution through DeepEP (Section 4).

3 Phase-Aware RankCast Algorithms

In the previous sections, we discussed the different optimization objectives for Prefill and Decode. For Prefill, we reduce the maximum token load across ranks and for decode we reduce the maximum active experts of all ranks. In this section, We develop separate algorithms under the same routing constraints: preserving each token's Top-p experts, restricting selection to its Top-L candidates, and selecting exactly k experts. For token i with router scores s_i, we define the candidate set as $\mathcal{C}_i = \mathrm{Top}_L(s_i)$ and the protected set as $\mathcal{F}_i = \mathrm{Top}_p(s_i)$. RankCast compute rank prices in prefill

and an expert mask in decode from predicted next-layer scores. The target layer uses its own scores to select the final experts (Section 4).

3.1 RANKCAST FOR PREFILL

Prefill balances rank load by adjusting each token's $d = k - p$ flexible assignments. Consider N tokens and E experts distributed across P EP ranks. Let $\rho(e)$ be the rank hosting expert e, and let $\mathcal{E}_q$ be the set of experts on rank q. Each token requires d flexible assignments. We therefore retain only its top-d flexible candidates on each rank, as any candidate outside this set cannot appear in the final selection. For token i and rank q, denote their scores by $a_{i,q,1} \geq \cdots \geq a_{i,q,d}$. Missing slots are padded with $-\infty$ and cannot be selected. Let c_q be rank q's target load for flexible assignments, with $\sum_q c_q = Nd$. The target c_q is computed from the balance target after excluding the protected set (Appendix A.2). We choose assignments to maximize their total score while giving each token exactly d choices and matching each rank's target. With $x_{i,q,j} \in \{0, 1\}$ indicating whether a candidate is selected, the problem can be defined as:

$$\max_{\mathbf{x}} \quad \sum_{i,q,j} a_{i,q,j} x_{i,q,j} \text{ s.t.} \qquad \sum_{q,j} x_{i,q,j} = d \ \ \forall i, \qquad \sum_{i,j} x_{i,q,j} = c_q \ \ \forall q. \tag{1}$$

This transportation problem has an integral LP optimum whenever a feasible assignment exists (Appendix A.1). We therefore solve it through its dual. Let α_i be the threshold of token i and β_q be the price of rank q. At the optimum, each token selects its top-d candidates by the price-adjusted score $a_{i,q,j} - \beta_q$. The threshold α_i separates its d-th and $(d+1)$-th adjusted scores. RankCast computes both variables in a single sweep, starting from zero prices. Each token first sets α_i between its d-th and $(d+1)$-th largest flexible scores. Let $w_{i,q,j} = a_{i,q,j} - \alpha_i$ denote the residual score, which measures how far a candidate lies above or below its token's threshold. A candidate on rank q is selected when its residual score exceeds β_q. Each rank therefore sets β_q between its c_q-th and $(c_q + 1)$-th largest residual scores, so that it receives exactly its target load c_q under the current thresholds. With $a^{\downarrow}_{i,(m)}$ denoting the m-th largest score among token i's flexible slots and $w^{\downarrow}_{q,(m)}$ the m-th largest residual score on rank q, the updates can be written as:

$$\alpha_i = \tfrac{1}{2}\big(a^{\downarrow}_{i,(d)} + a^{\downarrow}_{i,(d+1)}\big), \qquad \beta^{\star}_q = \tfrac{1}{2}\big(w^{\downarrow}_{q,(c_q)} + w^{\downarrow}_{q,(c_q+1)}\big). \tag{2}$$

A rank requested by many tokens therefore receives a high price, which moves part of its load to other ranks.

The threshold update needs only the scores of token i. The price update, however, needs the residual scores of all tokens on rank q, which are spread across source ranks. RankCast therefore exchanges more compact histograms instead of individual scores. Let $\tau_0 < \cdots < \tau_B$ delimit B shared bins, and let $H_{q,b}$ be the number of local residual scores $w_{i,q,j}$ on rank q that fall in bin $[\tau_b, \tau_{b+1})$. Each source rank also reports its protected loads $f_q = \sum_i |\mathcal{F}_i \cap \mathcal{E}_q|$, for every rank to compute the same targets c_q. The exchanged data size depends only on P and B (Section 4). After summing the histograms, we scan the bins of rank q from highest to lowest until the count reaches c_q and interpolate within the last bin. This gives an estimate $Q_q(c_q)$ of $w^{\downarrow}_{q,(c_q)}$ with error at most one bin width (Appendix A.3). Since a common shift of all prices does not change any token's ranking, the final price is centered across ranks:

$$\beta_q = Q_q(c_q) - \frac{1}{P} \sum_{q'=1}^{P} Q_{q'}(c_{q'}). \tag{3}$$

Boundary cases, where $c_q = 0$ or all candidates on rank q must be selected, set its price to the upper or lower end of the histogram range (Appendix A.3).

At the target layer, the router keeps the protected experts and selects the remaining d experts by their price-adjusted scores:

$$\mathcal{E}_i^{\text{prefill}} = \mathcal{F}_i \cup \text{Top}_d\{s_{i,e} - \beta_{\rho(e)} \mid e \in \mathcal{C}_i \setminus \mathcal{F}_i\}. \tag{4}$$

3.2 RANKCAST FOR DECODE

For decode, RankCast starts with the experts selected by Top-k routing, and closes experts to reduce the active-expert count on the busiest ranks. A closed expert cannot be selected by any token. Each

token is subject to the same constraints as in prefill. Let $\mathcal{A}_0 = \bigcup_i \mathrm{Top}_k(s_i)$ be the initial expert set and $\mathcal{P} = \bigcup_i \mathcal{F}_i$ the protected set. The objective is

$$\min_{\mathcal{A} \subseteq \mathcal{A}_0} \quad M(\mathcal{A}) = \max_r |\mathcal{A} \cap \mathcal{E}_r| \qquad \text{s.t.} \qquad \mathcal{P} \subseteq \mathcal{A}, \qquad |\mathcal{C}_i \cap \mathcal{A}| \geq k \quad \forall i. \tag{5}$$

RankCast solves this problem greedily. Let $\sigma_i = |\mathcal{C}_i \cap \mathcal{A}| - k$ be the slack of token i, which counts the candidates it can still lose. An active, unprotected expert e is *closable* if $\sigma_i > 0$ for every token i with $e \in \mathcal{C}_i$. Let the quality weight $\omega_{i,e} \in [0, 1]$ denote token i's normalized routing-weight loss estimate for removing expert e. Each rank orders its closable experts by increasing closure cost:

$$\mathrm{cost}(e) = \sum_{\substack{i \\ e \in \mathcal{C}_i}} \frac{\omega_{i,e}}{\sigma_i + 1}. \tag{6}$$

The cost is higher for candidates with larger weights and tokens with fewer alternatives.

Let $A_r = |\mathcal{A} \cap \mathcal{E}_r|$ be the number of active experts on rank r and $M = \max_r A_r$ the count on the busiest rank, which Equation 5 minimizes. One rank cannot lower M alone when another rank has the same count, so each round closes experts on several ranks together. Let $h_r = M - A_r$ be the headroom of rank r below the busiest rank. To lower M by Δ, every rank must keep at most $M - \Delta$ active experts, so rank r closes its $\Delta - h_r$ cheapest closable experts, or none if $\Delta \leq h_r$. Let $e_{r,j}$ be the j-th cheapest closable expert on rank r. Rank r therefore closes $e_{r,j}$ once $\Delta \geq h_r + j$. We call $\mathrm{lev}(e_{r,j}) = h_r + j$ the level of $e_{r,j}$, and set $\mathrm{lev}(e) = +\infty$ for non-closable experts. The maximum Δ can be derived by considering two bounding conditions. First, rank r has only m_r closable experts, therefore $\Delta \leq h_r + m_r$. Second, experts closed together must not exhaust a token's slack. Token i can lose at most σ_i candidates, so Δ must stay below Λ_i, the $(\sigma_i + 1)$-th smallest level among its candidates. The largest feasible reduction is therefore

$$\Delta^\star = \max\Big\{0,\ \min\big(\min_r(h_r + m_r),\ \min_i(\Lambda_i - 1)\big)\Big\}. \tag{7}$$

When $\Delta^\star = 0$, a repair step tries other experts on the busiest ranks (Appendix B.2). The solver repeats these steps until M stops decreasing or the total reduction reaches an optional depth cap D. The closed experts form the router mask of the target layer.

4 System Implementation

We implement RankCast in SGLang and integrate it with DeepEP's normal prefill path and low-latency decode path. The implementation must meet three requirements: the balancing decision must be ready before the next router runs, it must add no collective or synchronization barrier to the per-layer critical path, and every rank must reach the same balancing decision. Both phases therefore use the four-stage pipeline in Figure 2. They predict the next layer's routing scores, exchange compact metadata, compute the balancing decision, and apply it at the next router. Prefill exchanges rank statistics to compute prices, and decode exchanges candidate lists to compute an active-expert mask. Both kinds of metadata ride on the token-count messages that DeepEP already exchanges. Every rank receives the same data and runs the same deterministic computation, so prices and masks need no separate broadcast.

Shared cross-layer router prediction. The balancing decision must be ready before the target layer $\ell + 1$ routes its tokens, so RankCast predicts this routing at the current layer ℓ. As soon as the current hidden states $\mathbf{X}^{(\ell)}$ are available, a side stream applies the next MoE layer's gate weights,

$$\widehat{\mathbf{S}}^{(\ell+1)} = \mathbf{X}^{(\ell)} \mathbf{W}_{\mathrm{gate}}^{(\ell+1)\top} + \mathbf{1}\mathbf{b}^{(\ell+1)\top}, \tag{8}$$

and then the solvers run on these predicted scores while the main stream continues. The prediction determines only the balancing decision. The target layer still computes its own router scores and selects experts from them. On Qwen3-235B, the predicted Top-16 contains 98.7% of the true Top-8 experts and 99.1% of their routing weight (Figure 3(b) in Appendix D.1). RankCast adds two online corrections (Appendix C.2). The bias $\mathbf{b}^{(\ell+1)}$ tracks the mean gap between predicted and measured router scores, and in prefill a per-rank offset raises the price of ranks that stay overloaded. Each is updated every few forwards with one all-reduce, outside CUDA graphs and off the per-layer critical path.

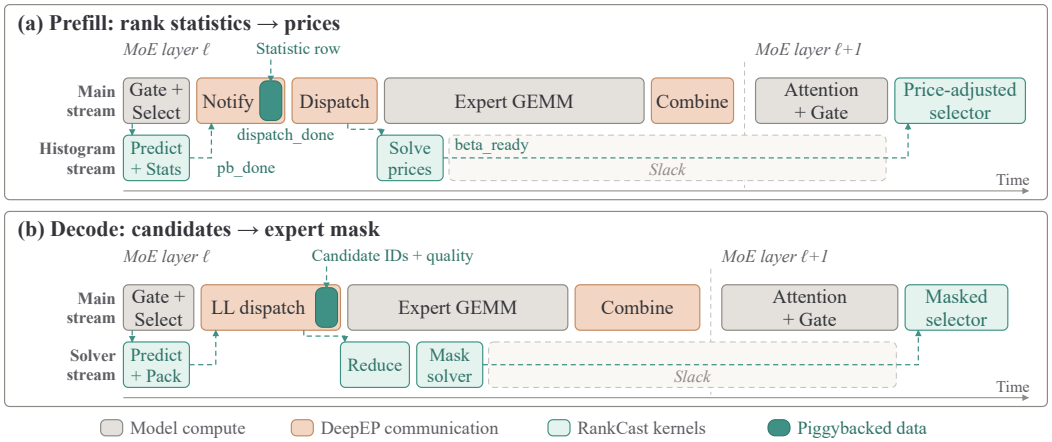


Figure 2: Cross-layer execution of RankCast on one EP rank. Prediction and balancing overlap with model execution, using metadata carried by existing DeepEP communication to compute (a) prefill rank prices or (b) a decode expert mask for the next layer. Dashed dependencies indicate CUDA events; *slack* denotes the interval between solver completion and next-layer selection. LL: low-latency. Timelines are schematic.

Histogram and price pipeline for prefill. Prefill runs the four stages of Figure 2(a) on a side stream. First, a statistics kernel evaluates Equation 8 for the local tokens and builds a statistic row, which includes the histograms $H_{q,b}$ of Section 3.1, the protected loads, and the token count, $P(B+1)+1$ integers in total (about 8 KiB for $P = 8$ and $B = 256$). Second, DeepEP's `notify_dispatch`, which already exchanges each rank's token counts, carries this row to all ranks together with the counts, both within and across nodes. Third, once dispatch completes, a price kernel computes the prices of Equation 3 from the summed rows (Appendix C.1). Finally, the next MoE layer's selector applies Equation 4 with these prices, taking gating weights from the original logits.

Both kernels are small and overlap with model execution (Figure 2(a)). On Qwen3-235B, the statistics and price kernels take 54 and 44 μs per layer. The statistics kernel uses one warp per token and overlaps with the current layer's expert selection and dispatch setup. The price kernel uses a single 128-thread block, and its slack spans the current layer's expert computation and the next layer's attention.

Expert-shedding pipeline for decode. Decode also runs the four stages of Figure 2(b) on a side stream. First, each rank evaluates Equation 8 for its local tokens and writes one candidate row per token, with the token's Top-L experts and their quality weights. Second, the rows ride the token-count messages of DeepEP's low-latency dispatch to every rank, both within and across nodes (Appendix C.3). Third, each rank runs the solver of Section 3.2 in a single CUDA block, which finishes during the current layer's expert computation (Appendix C.4). Finally, the target layer applies the mask by adding a large negative bias to the router score of every closed expert. Its Top-k therefore runs unchanged and never selects a closed expert. The whole path is CUDA-graph compatible (Appendix C.4).

5 Evaluation

5.1 Setup

Models and deployment. We evaluate RankCast on Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025; Qwen Team, 2025) in BF16 (94 MoE layers, 128 experts, $k = 8$) and Qwen3.5-397B-A17B (Qwen Team, 2026) (60 layers, 512 experts, $k = 10$). Both run on two nodes of 8×H800 connected over RoCE, served by SGLang 0.5.12 (Zheng et al., 2024) with DeepEP 1.2.1 (Zhao et al., 2025b) and DeepGEMM (Zhao et al., 2025a). Qwen3-235B runs prefill and decode on sep-

Table 1: Prefill results at three chunked-prefill sizes per model. Throughput gain is relative to SGLang's native Top-k routing (Baseline). Imbalance is the max-to-mean token count across EP ranks. Bold marks the best value of each metric in each row. Online EPLB rebalances every 50 forward passes on ShareGPT and CodeContests; EPLB-R adds 8 redundant experts on Qwen3-235B and 16 on Qwen3.5-397B.

	Tokens	Throughput gain			Imbalance (max/mean)			
Workload	per rank	EPLB	EPLB-R	RankCast	Baseline	EPLB	EPLB-R	RankCast
Qwen3-235B								
ShareGPT	2K	+13.3%	+7.0%	**+16.4%**	1.76	1.15	1.41	**1.09**
	4K	+14.7%	+8.0%	**+18.5%**	1.75	1.13	1.42	**1.08**
	8K	+13.8%	+6.6%	**+19.4%**	1.75	1.11	1.46	**1.08**
CodeContests	2K	+12.7%	+6.4%	**+16.4%**	1.66	1.10	1.35	**1.04**
	4K	+14.8%	+7.0%	**+18.8%**	1.66	1.09	1.36	**1.04**
	8K	+14.2%	+7.5%	**+19.3%**	1.66	1.06	1.39	**1.04**
LongBench	2K	−0.4%	−1.2%	**+7.3%**	1.66	1.60	1.61	**1.16**
	4K	+0.1%	−0.2%	**+7.1%**	1.65	1.58	1.61	**1.15**
	8K	+0.9%	+0.3%	**+4.3%**	1.47	1.39	1.44	**1.10**
Qwen3.5-397B								
ShareGPT	4K	+4.6%	+4.4%	**+10.8%**	1.51	1.14	1.20	**1.09**
	8K	+7.4%	+7.1%	**+11.2%**	1.50	1.11	1.17	**1.07**
	16K	+7.7%	+7.3%	**+10.4%**	1.50	**1.09**	1.13	1.11
CodeContests	4K	+7.1%	+6.5%	**+13.9%**	1.55	1.08	1.19	**1.04**
	8K	+9.9%	+9.1%	**+14.6%**	1.55	1.06	1.17	**1.03**
	16K	+10.4%	+9.7%	**+14.0%**	1.54	1.05	1.15	**1.03**
LongBench	8K	−0.2%	+0.2%	**+4.2%**	1.26	1.15	1.16	**1.05**
	16K	+0.7%	—	**+4.4%**	1.25	1.14	—	**1.05**

arate nodes, each with TP$=8$ and EP$=8$. Qwen3.5-397B runs both phases in one instance across all 16 GPUs, with DP$=16$ attention and EP$=16$. Unless stated otherwise, RankCast uses $L = 16$ candidates ($L = 20$ on Qwen3.5-397B) and protects each token's top $p = 2$ experts in both phases. Prefill uses $B = 256$ histogram bins, and decode uses quality weights without a depth cap. Appendix D.3 reports ablations of p, D, and L.

Workloads and baselines. Prefill experiments measure input throughput on prompts from ShareGPT (ShareGPT, 2023) and CodeContests (Li et al., 2022), tiled or truncated to 4,096 tokens, and on 32,768-token windows of LongBench v2 (Bai et al., 2025). Decode experiments measure the median time per output token (TPOT) on WildChat (Zhao et al., 2024), ShareGPT, and CodeContests prompts at their natural lengths, and on ShareGPT prompts tiled to 1,024 tokens. Quality experiments run eight benchmarks on both models. We compare RankCast with online EPLB (DeepSeek-AI, 2025) in prefill, and with XShare (Vankov et al., 2026) and OEA (Oncescu et al., 2026) in an offline decode comparison. Throughput and quality changes are relative to native Top-k routing on the same deployment, labeled Baseline in the tables. Appendix D.1 lists all settings.

5.2 Throughput

Prefill. Table 1 reports the three workloads on both models. RankCast lowers the imbalance across EP ranks to 1.03–1.16 and removes 76–95% of the busiest rank's excess load over the mean. On ShareGPT and CodeContests, input throughput rises by 16.4–19.4% on Qwen3-235B and 10.4–14.6% on Qwen3.5-397B. On LongBench v2, it rises by 4.2–7.3%, because the 32,768-token inputs make attention a larger share of each step. On ShareGPT and CodeContests, online EPLB (DeepSeek-AI, 2025) removes 73–91% of the excess and raises input throughput by 12.7–14.8% on Qwen3-235B and by 4.6–10.4% on Qwen3.5-397B. In these settings, RankCast gains 2.7–6.8 points more than EPLB. It also leaves a lower imbalance, except on ShareGPT at 16K tokens per rank on Qwen3.5-397B. Redundant experts lower the gain of EPLB on both models, because the extra expert copies use GPU memory and therefore reduce the number of concurrent requests.

Table 2: Median decode TPOT (ms) with 1,024 output tokens per request. Qwen3-235B keeps up to 96 requests in flight, and Qwen3.5-397B up to 128. Prompt gives the median prompt length in tokens. Δ is the relative gain in $1/\mathrm{TPOT}$.

		Qwen3-235B			Qwen3.5-397B		
Workload	Prompt	Baseline	RankCast	Δ	Baseline	RankCast	Δ
WildChat	29	27.68	26.10	$+6.1\%$	41.74	37.67	$+10.8\%$
ShareGPT	91	27.69	26.11	$+6.1\%$	40.76	37.48	$+8.8\%$
ShareGPT, tiled	1,024	27.59	25.58	$+7.9\%$	37.53	34.16	$+9.9\%$
CodeContests	3.5K	27.41	25.33	$+8.2\%$	52.19	46.97	$+11.1\%$

Table 3: Offline comparison of decode shedding with XShare (Vankov et al., 2026) and OEA (Oncescu et al., 2026) on decode batches of Qwen3-235B. M is the mean active-expert count on the busiest rank. Each row compares the methods at the same M. Every method keeps the true top-1 expert of at least 99% of tokens and gives every token k experts. The RankCast column gives the routing probability displaced from each token's Top-k. The XShare and OEA columns give their displaced probability as a multiple of RankCast's.

Workload	M	RankCast	XShare	OEA
ShareGPT	14.56	0.065%	$2.38\times$	$1.94\times$
	13.59	0.180%	$6.02\times$	$4.75\times$
	12.01	0.600%	$3.60\times$	$3.83\times$
	11.59	0.811%	$2.83\times$	$2.84\times$
LiveCodeBench	12.96	0.062%	$4.21\times$	$1.92\times$
	12.05	0.133%	$1.96\times$	$2.25\times$
	10.25	0.428%	$3.00\times$	$4.15\times$
	9.22	0.731%	$2.34\times$	$2.43\times$

On LongBench v2, EPLB and EPLB-R remove only 6–43% of the excess and change throughput by -1.2 to $+0.9\%$.

Decode. Table 2 reports median decode TPOT on four workloads. Decode shedding raises $1/\mathrm{TPOT}$ by 6.1–8.2% on Qwen3-235B and by 8.8–11.1% on Qwen3.5-397B. Natural-length prompts still gain 6.1–11.1%, so the gain does not depend on tiled inputs. On Qwen3-235B, decode shedding lowers the mean number of active experts on the busiest rank from 14.4 to 9.7 (Figure 3(a) in Appendix D.1). Across 11 configurations, median TPOT is linear in this count, with a slope of 0.52 ms per expert ($R^2 = 0.99$). The intercept of 20.1 ms is the part of TPOT that shedding cannot reduce.

Decode baselines. XShare and OEA select experts from the true router scores of the whole batch. Neither has an open-source implementation for expert parallelism. Running them on our cross-layer prediction would therefore require changing their algorithms. We instead compare the three methods offline on router scores captured from Qwen3-235B decode (Table 3). XShare and OEA receive the true scores, whereas RankCast uses its prediction and adds no collective. At the same M, XShare and OEA still displace 1.9–6.0 times as much routing probability as RankCast.

Overhead. Prefill balancing adds about 0.5% to the prefill step time on both models. Decode shedding adds 0.7% to the decode step time on Qwen3-235B and 0.9% on Qwen3.5-397B, mostly through the prediction and masking kernels. Its solver takes 9–17 μs per layer on Qwen3-235B and 58 μs on Qwen3.5-397B. The solver runs on a side stream during the layer's expert computation, so almost none of this time is exposed.

5.3 Quality

Table 4 reports the accuracy change of prefill balancing, decode shedding, and both together. No change exceeds 1 pp. On Qwen3-235B, prefill balancing lowers each of the four ten-repeat benchmarks by 0.3–0.7 pp. Decode shedding changes no benchmark by more than 0.65 pp. With both

Table 4: Quality change in percentage points, relative to native Top-k routing. Prefill enables prefill balancing, Decode enables decode shedding, and Both enables the two together. Baseline gives the accuracy of native routing. Appendix D.1 lists the repeat counts.

	Qwen3-235B				Qwen3.5-397B		
Benchmark	Baseline	Prefill	Decode	Both	Baseline	Prefill	Decode
MMLU-Redux 2.0	0.9105	−0.02	−0.13	−0.17	0.9254	+0.18	−0.01
MATH-500	0.9813	−0.07	−0.20	+0.00	0.9160	−0.07	−0.30
MMMLU	0.8198	−0.16	−0.22	−0.11	0.8673	−0.04	−0.25
RULER	0.8787	−0.40	−0.31	−0.73	0.9419	−0.20	+0.02
GPQA-Diamond	0.7621	−0.67	−0.10	−0.46	0.8576	+0.03	−0.39
AIME 2025	0.6963	−0.32	−0.02	−0.30	0.8707	+0.32	−0.09
IFEval strict	0.8848	−0.31	−0.65	−0.59	0.8854	+0.02	
LiveCodeBench v5	0.7008	−0.43	−0.59	−0.84	0.8467	−0.43	−0.34

enabled, the largest drop is 0.84 pp, on LiveCodeBench. On Qwen3.5-397B, no benchmark drops by more than 0.43 pp. We therefore report a quality cost below 1 pp rather than no degradation.

6 Conclusion and Limitations

We presented RankCast, a unified framework for rank-level load balancing in expert-parallel MoE inference. Prefill and decode share the same routing constraint, which retains each token's predicted Top-p experts. They differ only in the objective, which is the maximum per-rank token load in prefill and the maximum active-expert count in decode. On Qwen3-235B-A22B and Qwen3.5-397B-A17B, RankCast improves prefill input throughput by up to 19% and reduces decode TPOT by up to 10%, at a quality cost below one percentage point.

This work has two limitations. First, our kernels and serving integration are not fully optimized, so a more efficient implementation could further reduce the overhead reported in Section 5.2. Second, limited compute restricted our experiments to at most 16 GPUs with EP$=16$, and we leave larger EP degrees and deployments across more nodes to future work.

AI use statement

We used generative AI tools to implement code: parts of the RankCast solver and its serving integration were AI-assisted. We additionally used them to edit the paper for readability and to write auxiliary scripts. We did not use them for the conceptual framework, hypotheses, experiment design, or interpretation of results; synthetic data, dataset cleaning, qualitative analysis, and translation are not applicable. The authors tested all AI-assisted code against the reference implementation of Appendix C, and take responsibility for the final content of this work.

Ethics statement

This work improves the serving efficiency of existing, publicly released Mixture-of-Experts models and introduces no new model, training procedure, or dataset. All experiments use public model weights and benchmark datasets, involve no human subjects, and collect no personal data. Because RankCast changes routing at inference time, we report the measured quality change of every configuration, including degradations, so that deployers can weigh throughput against accuracy.

Reproducibility statement

Section 5 and Appendix D.1 specify the model, serving stack, hardware, workloads, seeds, warmup and measurement windows, and repetition counts behind every reported number, and Appendix D.3 reports the ablations of the protected prefix p, the depth cap D, the quality weights, and the candidate width L. The prefill solver is specified by Equations 1–4, with its dual derivation, capacity

construction, and histogram error bound in Appendix A; the decode solver by Equations 5–7, with its quality weights, closure order, repair step, and depth cap in Appendix B. Appendix C documents the kernels, the metadata exchange, the online corrections, the candidate-row layout, and the reference implementation against which the GPU solver was validated.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025. doi: 10.18653/v1/2025.acl-long.183. URL https://aclanthology.org/2025.acl-long.183/.

Yutian Chen, Cong Li, Yucheng Wang, and Ming Wei. MoonEP: A perfectly balanced expert parallelism library via dynamic redundant experts. Software repository, 2026. URL https://github.com/MoonshotAI/MoonEP. Accessed 2026-08-09.

DeepSeek-AI. DeepSeek-V3 technical report. arXiv preprint arXiv:2412.19437, 2024. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI. EPLB: Expert parallelism load balancer. Software repository, 2025. URL https://github.com/deepseek-ai/EPLB. Accessed 2026-08-09.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. arXiv preprint arXiv:2606.19348, 2026. URL https://arxiv.org/abs/2606.19348.

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.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile Van Krieken, and Pasquale Minervini. Are we done with MMLU? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5069–5096, 2025. doi: 10.18653/v1/2025.naacl-long.262. URL https://aclanthology.org/2025.naacl-long.262/.

Seokjin Go and Divya Mahajan. MoETuner: Optimized mixture of expert serving with balanced expert placement and token routing. arXiv preprint arXiv:2502.06643, 2025. URL https://arxiv.org/abs/2502.06643.

Vima Gupta, Jae Hyung Ju, Kartik Sinha, Ada Gavrilovska, and Anand Padmanabha Iyer. Lynx: Enabling efficient MoE inference through dynamic batch-aware expert selection. arXiv preprint arXiv:2411.08982, 2024. URL https://arxiv.org/abs/2411.08982.

Yu Han, Lehan Pan, Jie Peng, Ziyang Tao, Hanqi Zhu, Wuyang Zhang, and Yanyong Zhang. GRACE-MoE: Grouping and replication with locality-aware routing for efficient distributed MoE inference. arXiv preprint arXiv:2509.25041, 2025. URL https://arxiv.org/abs/2509.25041.

Shwai He, Weilin Cai, Jiayi Huang, and Ang Li. Capacity-aware inference: Mitigating the straggler effect in mixture of experts. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=LuYFpySWA2.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

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, et al. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024. URL https://arxiv.org/abs/2401.04088.

Kimi Team. Kimi K3: Open frontier intelligence. arXiv preprint arXiv:2607.24653, 2026. URL https://arxiv.org/abs/2607.24653.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE layers: Simplifying training of large, sparse models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6265–6274, 2021. URL https://proceedings.mlr.press/v139/lewis21a.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, 2023. URL https://www.usenix.org/conference/atc23/presentation/li-jiamin.

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*, 2026. URL https://openreview.net/forum?id=MSHPrMpIHZ.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

LiveCodeBench Team. LiveCodeBench release v5. Official benchmark release, 2025. URL https://github.com/LiveCodeBench/LiveCodeBench.

Mathematical Association of America. 2025 American Invitational Mathematics Examination (AIME I and II). Annual mathematics competition, 2025. URL https://maa.org/maa-invitational-competitions/.

Meituan LongCat Team. LongCat-Flash technical report. arXiv preprint arXiv:2509.01322, 2025. URL https://arxiv.org/abs/2509.01322.

Costin-Andrei Oncescu, Qingyang Wu, Wai Tong Chung, Tsai-chuan Wu, Bryan Gopal, Junxiong Wang, Tri Dao, and Ben Athiwaratkun. Opportunistic expert activation: Batch-aware expert routing for faster decode without retraining. In *The Forty-Third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=bSLIDztFwx.

OpenAI. Multilingual Massive Multitask Language Understanding (MMMLU). Dataset, 2024. URL https://huggingface.co/datasets/openai/MMMLU.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Qwen Team. Qwen3-235B-A22B-Instruct-2507. Model card, 2025. URL https://huggingface.co/Qwen/Qwen3-235B-A22B-Instruct-2507.

Qwen Team. Qwen3.5: Towards native multimodal agents. Blog post, February 2026. URL https://qwen.ai/blog?id=qwen3.5. Model card: https://huggingface.co/Qwen/Qwen3.5-397B-A17B.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Rana Shahout, Colin Cai, Yilun Du, Minlan Yu, and Michael Mitzenmacher. From score distributions to balance: Plug-and-play mixture-of-experts routing. arXiv preprint arXiv:2510.03293, 2025. URL https://arxiv.org/abs/2510.03293.

ShareGPT. ShareGPT. Conversation-sharing platform, 2023. URL https://sharegpt.com/.

Daniil Vankov, Nikita Ivkin, Kyle Ulrich, Xiang Song, Ashish Khetan, and George Karypis. XShare: Collaborative in-batch expert sharing for faster MoE inference. arXiv preprint arXiv:2602.07265, 2026. URL https://arxiv.org/abs/2602.07265.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. arXiv preprint arXiv:2408.15664, 2024. URL https://arxiv.org/abs/2408.15664.

Xinming Wei, Chao Jin, Tuo Dai, Yinmin Zhong, Shan Yu, Chengxu Yang, Bingyang Wu, Zili Zhang, Jing Mai, Qianchao Zhu, Zhouyang Li, Yuliang Liu, and Guojie Luo. UltraEP: Unleash MoE training and inference on rack-scale nodes with near-optimal load balancing. arXiv preprint arXiv:2606.04101, 2026. URL https://arxiv.org/abs/2606.04101.

Juntong Wu, Jialiang Cheng, Fuyu Lv, Dan Ou, and Li Yuan. SERE: Similarity-based expert re-routing for efficient batch decoding in MoE models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=98IxaUQtMY.

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. arXiv preprint arXiv:2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Jaehoon Yang, Yushin Kim, Seokwon Moon, Yeonhong Park, and Jae W. Lee. Libra: Effective yet efficient load balancing for large-scale MoE inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=WhxNwgGkAS.

Adrian Zhao, Zhenkun Cai, Zhenyu Song, Lingfan Yu, Haozheng Fan, Jun Wu, Yida Wang, and Nandita Vijaykumar. CRAFT: Fine-grained cost-aware expert replication for efficient mixture-of-experts serving. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 689–700, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/3a7f9e485845dac27423375c934cb4db-Abstract-Conference.html.

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. Software repository, 2025a. URL https://github.com/deepseek-ai/DeepGEMM.

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. Software repository, 2025b. URL https://github.com/deepseek-ai/DeepEP.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1M ChatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*, 2024.

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, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. arXiv preprint arXiv:2311.07911, 2023. URL https://arxiv.org/abs/2311.07911.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*, volume 35, pp. 7103–7114, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/2f00ecd787b432c1d36f3de9800728eb-Abstract-Conference.html.

Qianchao Zhu, Xucheng Ye, Yuliang Liu, Haodong Ouyang, and Chengru Song. PROBE: Co-balancing computation and communication in MoE inference via real-time predictive prefetching. arXiv preprint arXiv:2602.00509, 2026. URL https://arxiv.org/abs/2602.00509.

A Prefill Solver Details

A.1 Dual Derivation of the Rank Prices

Exact LP relaxation. Let slot (i, q, j) be *valid* if $a_{i,q,j} > -\infty$, and let U_q denote the number of valid slots on rank q. Dropping the padded slots and relaxing $x_{i,q,j}$ to $[0, 1]$ gives

$$\max_{\mathbf{x}} \sum_{i,q,j} a_{i,q,j} x_{i,q,j} \quad \text{s.t.} \quad \sum_{q,j} x_{i,q,j} = d \ \ \forall i, \qquad \sum_{i,j} x_{i,q,j} = c_q \ \ \forall q, \qquad 0 \le x_{i,q,j} \le 1. \tag{9}$$

This is a transportation problem. Token i supplies d units, rank q demands c_q units, and each valid slot is an edge of capacity one. Its constraint matrix, including the bounds $x_{i,q,j} \le 1$, is totally unimodular, and d and c_q are integers. Every vertex of the feasible polytope is therefore integral. Whenever a feasible assignment exists, the relaxation has an integral optimum with the same value as Equation 1.

Dual and block updates. Let the threshold α_i and the price β_q be the multipliers of the token and rank constraints, and let $[z]_+ = \max(z, 0)$. The dual function is

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = d \sum_i \alpha_i + \sum_q c_q \beta_q + \sum_{i,q,j} \big[a_{i,q,j} - \alpha_i - \beta_q\big]_+. \tag{10}$$

By complementary slackness, an optimal assignment selects every slot whose adjusted score $a_{i,q,j} - \beta_q$ exceeds α_i and no slot whose adjusted score falls below it. Each token therefore takes its top-d adjusted scores, as in Equation 4.

For fixed $\boldsymbol{\beta}$, let $v^{\downarrow}_{i,(m)}$ denote the m-th largest adjusted score of token i. Any threshold in

$$\alpha_i \in \big[v^{\downarrow}_{i,(d+1)},\, v^{\downarrow}_{i,(d)}\big] \tag{11}$$

minimizes g over α_i. For fixed $\boldsymbol{\alpha}$, let $w^{\downarrow}_{q,(m)}$ denote the m-th largest residual score on rank q, across all source ranks. Any price in

$$\beta_q \in \left[w^{\downarrow}_{q,(c_q+1)},\, w^{\downarrow}_{q,(c_q)}\right] \tag{12}$$

minimizes g over β_q. Starting from $\boldsymbol{\beta} = \mathbf{0}$ and taking the midpoints of both intervals gives Equation 2. If $c_q = 0$ or $c_q = U_q$, the price interval becomes one-sided:

$$\beta_q \geq w^{\downarrow}_{q,(1)} \text{ if } c_q = 0, \qquad \beta_q \leq w^{\downarrow}_{q,(U_q)} \text{ if } c_q = U_q. \tag{13}$$

One sweep and centering. One sweep does not in general reach the dual optimum, so rank q need not receive exactly c_q flexible assignments. Each further sweep would need another histogram exchange, so RankCast runs one sweep. Because $\sum_q c_q = Nd$, raising every β_q and lowering every α_i by the same amount leaves g unchanged. Equation 3 fixes this shift by subtracting the mean price.

A.2 Flexible Target Loads

Let C_q denote the balanced total load of rank q. The Nk assignments are split as evenly as possible across ranks:

$$C_q = \lfloor Nk/P \rfloor + \mathbf{1}[q \leq Nk \bmod P], \qquad \sum_{q=1}^{P} C_q = Nk. \tag{14}$$

Every rank obtains N from the token counts of the statistic rows, so all ranks compute the same C_q. Let f_q denote the protected load of rank q, summed over all source ranks. Every token has p protected experts, so the targets $C_q - f_q$ sum to Nd. A target can be negative or exceed U_q, and clipping it to $[0, U_q]$ alone can change the sum. RankCast therefore subtracts a shared offset μ before clipping:

$$z^{\star}_q(\mu) = \min\{U_q, \max(0, C_q - f_q - \mu)\}, \qquad \sum_q z^{\star}_q(\mu) = Nd. \tag{15}$$

Every token has at least d valid slots, so $\sum_q U_q \geq Nd$. The clipped sum falls continuously from $\sum_q U_q$ to 0 as μ grows, so bisection finds μ. The resulting $\mathbf{z}^{\star}$ is the point closest to $\mathbf{C} - \mathbf{f}$ that satisfies the bounds and the sum. RankCast rounds each $z^{\star}_q$ down and gives the remaining units, one each, to the unsaturated ranks with the largest fractional parts. The integer targets therefore satisfy $0 \leq c_q \leq U_q$ and $\sum_q c_q = Nd$. These bounds do not guarantee a feasible assignment, because several ranks can compete for the slots of the same tokens. The prices remain defined in that case.

A.3 Accuracy of the Histogram Estimate

The histograms use $B = 256$ equal-width bins on $[-W, W]$ with $W = 4$, so each bin has width $\Delta\tau = 2W/B$. Scores outside this range are counted in the first or the last bin. For rank q, the price kernel accumulates bin counts from the top until the count reaches c_q. Let b denote the bin where it stops, and let $H_{q,>b}$ denote the count above this bin. The estimate interpolates linearly inside bin b:

$$Q_q(c_q) = \tau_{b+1} - \Delta\tau\, \frac{c_q - H_{q,>b}}{H_{q,b}}. \tag{16}$$

Integer counts sum exactly, so bin b contains the exact order statistic $w^{\downarrow}_{q,(c_q)}$ whenever b is not an edge bin. Both values then lie in the same bin:

$$\left|Q_q(c_q) - w^{\downarrow}_{q,(c_q)}\right| \leq \Delta\tau. \tag{17}$$

The upper endpoint $w^{\downarrow}_{q,(c_q)}$ of Equation 12 is an exact block minimizer, like the midpoint of Equation 2. Each price is therefore within $\Delta\tau$ of an exact update. Centering shifts every price equally and does not change routing. A token keeps its exact selection whenever the gap between its d-th and $(d+1)$-th price-adjusted scores exceeds $2\Delta\tau$. If $c_q = 0$, the kernel sets the price just above W. If $c_q = U_q$, it sets the price just below $-W$. These values realize Equation 13 within the histogram range.

B Decode Solver Details

B.1 Quality Weights and Closure Order

Quality weights. Let $\hat{s}_{i,e}$ denote the predicted score of expert e for token i, and let $\hat{s}^{\downarrow}_{i,(m)}$ denote the m-th largest predicted score of token i. Let $u_{i,e} = \exp(\hat{s}_{i,e} - \hat{s}^{\downarrow}_{i,(1)})$, and define $u^{\downarrow}_{i,(m)}$ likewise. Closing an expert e of the predicted Top-k approximately hands its routing weight to the $(k+1)$-th expert. The quality weight is this loss, normalized by the loss for the top expert and rounded up to a multiple of $1/3$:

$$\omega_{i,e} = \frac{1}{3}\left\lceil \frac{3\left(u_{i,e} - u^{\downarrow}_{i,(k+1)}\right)}{1 - u^{\downarrow}_{i,(k+1)}} \right\rceil \quad \text{for } e \in \mathrm{Top}_k(\hat{s}_i), \tag{18}$$

and $\omega_{i,e} = 0$ for the other candidates. Each weight therefore takes one of four values and fits in two bits. Without quality weights, the solver sets every $\omega_{i,e}$ to one.

Closure order. Ties in the closure cost are broken by the number of tokens that list the expert as a candidate, fewest first, because closing such an expert reduces the slack of fewer tokens. Remaining ties are broken by expert identifier, so every rank computes the same order.

Closure set. Let $\mathcal{R}(\Delta) = \{e \mid \mathrm{lev}(e) \le \Delta\}$ denote the experts that Section 3.2 closes to lower M by Δ. For $\Delta \le \Delta^\star$, closing $\mathcal{R}(\Delta)$ lowers M by exactly Δ and keeps every token's coverage. $\Delta^\star$ is the largest reduction when every rank closes only its cheapest closable experts. Closing other experts may allow a larger reduction, so the solver does not guarantee an optimum of Equation 5. The solver works on predicted scores, so these guarantees hold for the predicted sets $\mathcal{F}_i$ and $\mathcal{C}_i$.

B.2 Repair Step

$\Delta^\star = 0$ in two cases. Either a busiest rank has no closable expert, or some token would lose more than its slack if every busiest rank closed its cheapest closable expert. In the first case, no closure can lower M. In the second case, closing other experts on the busiest ranks may still lower M by one. The repair step searches for one such expert per busiest rank:

1. Each busiest rank r starts with index $j_r = 1$.
2. Let $\mathcal{U} = \{e_{r,j_r} \mid A_r = M\}$ denote the proposal. A token i *conflicts* with $\mathcal{U}$ if $|\mathcal{C}_i \cap \mathcal{U}| > \sigma_i$.
3. If no token conflicts with $\mathcal{U}$, the solver closes $\mathcal{U}$, and M decreases by one.
4. Otherwise, every busiest rank whose proposed expert is a candidate of a conflicting token sets $j_r \leftarrow j_r + 1$, and the search returns to step 2. The repair fails after eight attempts or when some j_r exceeds m_r.

These ranks advance together, so each attempt is one parallel step. The joint advance can skip a feasible proposal that advancing one rank at a time would find.

B.3 Solver Loop and Depth Cap

The depth cap D bounds the total reduction of M within one MoE layer of one decode step. The default $D = \infty$ imposes no cap. Let δ denote the reduction reached so far, starting from $\delta = 0$. In each round with $\Delta^\star > 0$, the solver closes $\mathcal{R}(\Delta)$ with

$$\Delta = \min(\Delta^\star,\, D - \delta) \tag{19}$$

and sets $\delta \leftarrow \delta + \Delta$. A round with $\Delta^\star = 0$ runs the repair step instead, and a successful repair sets $\delta \leftarrow \delta + 1$. After each round, the solver recomputes the slack, the active counts, M, and the closure costs. It stops when $\delta = D$ or when the repair fails. Every round except the last lowers M by at least one, so the loop ends after at most $M_0 + 1$ rounds, where M_0 is the initial value of M. Once M stops decreasing, the solver closes no further experts, because such closures would move routing weight without lowering M.

C Implementation Details

Both phases predict the routing of the next MoE layer from the hidden states of the current one. The first MoE layer has no predecessor, so its prefill price is the load-feedback offset alone and its decode routing stays native Top-k. The last MoE layer computes no prediction. Only the two online corrections carry state across forwards.

C.1 Prefill Pipeline

The statistics kernel runs on the histogram stream of Figure 2(a) and reads a copy of $\mathbf{X}^{(\ell)}$. The main stream waits for it just before `notify_dispatch` reads the statistic row. `notify_dispatch` already copies each rank's token-count row to every other rank, and RankCast appends its statistic row to this copy. A row holds $P(B+1)+1$ int32 values, or 8,240 bytes for $P=8$ and $B=256$ after padding to 16 bytes. Across nodes, `notify_dispatch` sends the row over the RDMA queue pair of the counts, forwards it to the local ranks over NVLink, and sums the rows before it returns. A rank without tokens sends an all-zero row, because every rank must join the exchange.

Once dispatch completes, the price kernel computes the target loads and raw prices of Appendices A.2 and A.3 from the summed rows. It then subtracts the mean price and adds the load-feedback offset of Appendix C.2. Integer sums and a fixed order of floating-point operations make the prices bit-identical on every rank. At the target layer, a fused Triton kernel replaces the native Top-k and applies Equation 4 with these prices.

C.2 Online Corrections

Score bias. The bias $\mathbf{b}^{(\ell+1)}$ of Equation 8 tracks the mean gap $\bar{g}_e$ between the target layer's router score of expert e and its prediction without the bias. Every 32 forwards, an all-reduce sums the gaps across ranks, and each rank updates

$$b_e \leftarrow (1-\nu)\, b_e + \nu\, \bar{g}_e, \qquad \nu = 0.25. \tag{20}$$

Load feedback. In prefill, a per-rank offset η_q raises the price of ranks that stay overloaded. Every $T=8$ forwards, an all-reduce gives the number of expert assignments n_q that each rank received in recent forwards. Let $\phi_q = \log\!\big(P\, n_q / \textstyle\sum_{q'} n_{q'}\big)$ denote the log load share of rank q, which is zero under an even split, and let $\bar{\phi}$ denote its mean over ranks. The offset moves toward balance:

$$\eta_q \leftarrow \eta_q + \kappa\,(\phi_q - \bar{\phi}), \qquad \kappa = 0.1. \tag{21}$$

Each step is limited to ± 0.069, each offset to ± 1, and the offsets are kept centered. The price kernel adds η_q to the centered price β_q. Both corrections update outside CUDA graphs at the end of a forward.

C.3 Decode Candidate Rows

Let G denote the maximum number of decode tokens per rank. A row lists a token's Top-L predicted experts in descending order, together with the 2-bit code $3\omega_{i,e}$ of each (Appendix B.1). The first p entries form $\mathcal{F}_i$, and the first k form the initial selection. The rows travel in the token-count messages of DeepEP's low-latency dispatch as two byte arrays:

$$\underbrace{[\text{identifiers}]}_{PGL} \;\|\; \underbrace{[\text{codes}]}_{PG\lceil L/4\rceil \text{ bytes}}. \tag{22}$$

For Qwen3-235B ($P=8$, $G=32$, $L=16$), they take 5 KiB per MoE layer. Each rank writes its rows into its own slice of a zeroed buffer, and each receiver sums the buffers of all ranks. The slices are disjoint, so the sum recovers every rank's rows. Across nodes, a rank writes its rows over the RDMA queue pair of its token count before it sends the count. The queue pair delivers in order, so the rows arrive before the count and need no separate signal.

C.4 Decode Solver Kernel

Each rank runs the solver as one CUDA block. The block keeps the candidate graph in shared memory as bit matrices:

$$\mathrm{cmask}[i] = \{e \mid e \in \mathcal{C}_i\}, \qquad \mathrm{tokmask}[e] = \{i \mid e \in \mathcal{C}_i\}. \tag{23}$$

Two more matrices, $\mathrm{mhi}[e]$ and $\mathrm{mlo}[e]$, mark the tokens whose code for expert e has its high or low bit set. Let $\mathrm{sbuck}[j]$ denote the tokens with slack j, and let popc count set bits. The closure cost of Equation 6 then needs only population counts:

$$\mathrm{cost}(e) = \frac{1}{3}\sum_{j=0}^{L-k} \frac{2\,\mathrm{popc}(\mathrm{mhi}[e] \cap \mathrm{sbuck}[j]) + \mathrm{popc}(\mathrm{mlo}[e] \cap \mathrm{sbuck}[j])}{j+1}. \tag{24}$$

Scaling every cost by $3\,\mathrm{lcm}(1, \ldots, L-k+1)$ makes it an integer, so every rank orders the experts identically. For each level, the solver keeps a mask of the experts at that level. Each token counts its candidates level by level to find Λ_i, and a block-wide minimum gives $\Delta^\star$. After the last round, the solver writes a router bias of -10^4 for every closed expert and 0 otherwise. The kernel supports up to 16 ranks, 512 candidate rows, and 32 experts per rank.

CUDA graphs and validation. The solver uses only preallocated buffers on the caller's stream and never synchronizes with the host, so the decode path can be captured in a CUDA graph. It never stops on invalid input, because a stopped rank would stall the dispatch of every other rank. A row with fewer than k valid candidates instead gets zero slack, which keeps that token's experts open. A scalar NumPy reference reproduces the CUDA masks bit for bit on captured Qwen3-235B router scores and on synthetic cases.

D Evaluation Details

D.1 Settings and Protocols

System settings. On Qwen3-235B, SGLang's mini-lb routes requests to the prefill and decode workers, and Mooncake (Qin et al., 2025) transfers the KV cache between them. Prefill uses DeepEP normal mode without CUDA graphs, and decode uses DeepEP low-latency mode with CUDA graphs. No run uses the radix cache. Online EPLB rebalances every 50 forward passes, and the other configurations keep a static expert placement.

Throughput runs. Each run discards its warmup requests and then measures 300 seconds, or 180–240 seconds for the natural-length decode workloads. Prefill baseline and RankCast runs alternate, with two or three runs per configuration. The LongBench v2 and EPLB results are single runs. Each decode configuration runs in one server launch. On Qwen3-235B, each natural-length decode cell averages two runs, and the second run uses a new seed and the reverse order. All other decode cells are single runs.

Prefill. LongBench v2 inputs come from all six domains, with at most four 32,768-token windows per document. Each request generates one output token, and the client keeps a fixed number of requests in flight. On Qwen3-235B, this number is 64 for the 4,096-token workloads and 24 for LongBench v2, which runs with TP attention. On Qwen3.5-397B, it is 128, 192, and 320 at the three chunk sizes, and 32 for LongBench v2. The max-to-mean token ratio is averaged over all MoE layers and over the forward passes that carry at least 90% of the tokens of the largest one. The small-chunk ShareGPT baseline on Qwen3-235B comes from other sessions, whose baseline throughput varies by 0.4%. Table 5 gives the absolute input throughput.

Decode. Each request generates 1,024 output tokens, with end-of-sequence tokens ignored. In the tiled ShareGPT workload, requests arrive at a fixed rate, and the global batch size caps the number in flight. The natural-length workloads keep the global batch size of requests in flight, which is 96 on Qwen3-235B and 128 on Qwen3.5-397B. We report median TPOT, because output throughput does not follow decode speed in these runs. The tiled workload nearly keeps up with its arrival rate, and on short natural-length prompts the bottleneck lies outside decode. On Qwen3.5-397B, TPOT

Table 5: Prefill input throughput (thousand tokens/s) behind the gains of Table 1. †Run with a larger state cache for the linear-attention layers; the gain in Table 1 is relative to a baseline with the same cache.

Workload	Tokens per rank	Baseline	EPLB	EPLB-R	RankCast
Qwen3-235B					
ShareGPT	2K	44.6	50.3	47.5	51.9
	4K	48.7	55.9	52.6	57.7
	8K	50.0	57.0	53.4	59.7
CodeContests	2K	45.4	51.2	48.3	52.8
	4K	50.1	57.5	53.6	59.5
	8K	51.5	58.8	55.3	61.4
LongBench	2K	30.3	30.2	30.0	32.5
	4K	31.8	31.8	31.7	34.0
	8K	33.1	33.4	33.2	34.5
Qwen3.5-397B					
ShareGPT	4K	120.1	125.8	125.5	133.1
	8K	123.2	132.7	132.4	136.9
	16K	122.6	136.1†	135.6†	135.4
CodeContests	4K	120.5	129.1	128.3	137.2
	8K	124.6	137.0	136.0	142.8
	16K	125.1	140.8†	139.9†	142.6
LongBench	8K	122.3	122.0	122.5	127.4
	16K	121.4	122.3	—	126.7

counts only pure decode steps, which excludes the interleaved prefill chunks. The active-expert count on the busiest rank is averaged over layers and steady-state decode steps. Its run-to-run spread is about 0.2 experts, measured on the prefill-balancing runs, which do not change decode routing. Figure 3(a) and Table 6 come from an earlier set of runs on the tiled workload. In those runs, decode shedding without a depth cap gained 9.1%.

Offline decode comparison. Table 3 replays router scores captured from Qwen3-235B decode. Each workload uses two disjoint batches of 96 sequences, with 64 decode steps over all 94 MoE layers. The ShareGPT capture uses 1,024-token contexts, and the LiveCodeBench capture uses 2,048-token contexts. Native routing gives $M = 15.61$ on ShareGPT and $M = 14.11$ on LiveCodeBench. Every method closes experts with the additive bias of Section 4. The native Top-k then runs on the biased true scores. RankCast uses $p = 2$, $L = 16$, and its cross-layer prediction, and its rows come from different depth caps D. RankCast protects the predicted top experts, so it keeps the true top-1 expert of 99.2–99.9% of tokens. XShare follows Algorithm 6 of its paper, and OEA follows its published algorithm. Both receive the true router scores of the batch. A baseline matches a RankCast row when its M is at most 0.15 above RankCast's.

Quality. The quality suite comprises MMLU-Redux 2.0 (Gema et al., 2025), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), MMMLU (OpenAI, 2024), RULER (Hsieh et al., 2024), GPQA-Diamond (Rein et al., 2024), AIME 2025 (Mathematical Association of America, 2025), IFEval strict (Zhou et al., 2023), and LiveCodeBench v5 (Jain et al., 2025; LiveCodeBench Team, 2025). MMMLU uses a fixed 10% stratified subset of each of its 14 languages. RULER uses 100 instances of each of its 13 tasks at 4K, 16K, and 32K tokens. Quality runs use 2K tokens per rank on Qwen3-235B and 1K on Qwen3.5-397B, with server seed 20260809. Generation is non-thinking and samples with temperature 0.7 and nucleus probability 0.8 from the 20 most likely tokens. RULER uses greedy decoding. Each repeat starts a new server session. On Qwen3-235B, MMLU-Redux 2.0, MATH-500, MMMLU, and RULER use three repeats, and the other four benchmarks use ten. Decode shedding uses 13 repeats on GPQA-Diamond and AIME 2025. On Qwen3.5-397B, prefill balancing uses the same counts, except five repeats on AIME 2025. Decode shedding uses two repeats on the first four benchmarks, ten on AIME 2025, and five on the other three. Qwen3.5-397B runs both phases in one instance, so it enables only one of the two at a

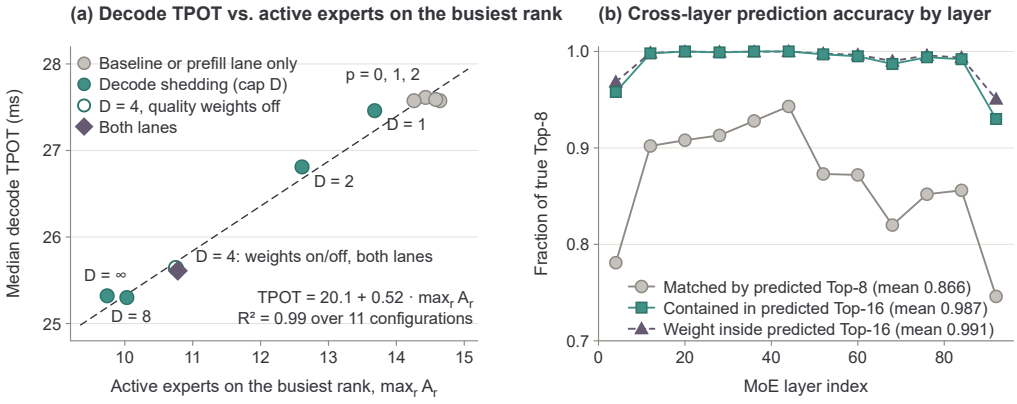


Figure 3: (a) Median decode TPOT against the mean number of active experts on the busiest rank for 11 configurations of Qwen3-235B at ISL 1024 and global batch 96. The dashed line is a least-squares fit. (b) Cross-layer prediction accuracy on 12 sampled MoE layers of Qwen3-235B during prefill. The curves give the share of each token's true Top-8 experts inside the predicted Top-8 and inside the predicted Top-16, and the share of their routing weight inside the predicted Top-16.

time. The change of repeat r is the score of repeat r minus the baseline score of repeat r. On the ten-repeat benchmarks, this change has a standard deviation of 0.7–1.6 pp.

Prediction accuracy. Figure 3(b) uses Qwen3-235B prefill with 2K tokens per rank. It samples the 12 target layers $4, 12, \ldots, 92$ over two forward passes on all eight ranks, 393,216 token–layer pairs in total. For token i, let $\hat{s}_i$ denote its predicted scores and s_i its router scores at the target layer. The Top-L recall is

$$\frac{\left|\mathrm{Top}_8(s_i) \cap \mathrm{Top}_L(\hat{s}_i)\right|}{8}, \tag{25}$$

and the weight recall is the share of the routing weight of $\mathrm{Top}_8(s_i)$ that falls on experts inside $\mathrm{Top}_L(\hat{s}_i)$.

D.2 GEMM Profiling

We profile the grouped expert GEMM of DeepGEMM, which computes both expert projections of one MoE layer on one rank. Figure 4(a–c) measures it offline on one GPU with the expert shapes and experts per rank of both models. Decode and prefill lay out the tokens differently, so we measure each layout.

Figure 4(a) varies the tokens per expert in the decode layout while every expert on the rank is active. Latency changes by less than 1.3% from 1 to 64 tokens per expert, because one 64-token tile holds all tokens of an expert and the time goes to reading its weights. The decode batches of Table 2 place 2.5–6 tokens on an average expert. Decode GEMM time is therefore set by the number of active experts. Figure 4(b) shows that each active expert adds 9.5 μs on Qwen3-235B and 6.2 μs on Qwen3.5-397B. Their ratio of about 1.5 matches the ratio of their expert intermediate sizes, 1,536 and 1,024.

Figure 4(c) repeats the token sweep in the prefill layout. Beyond 512 tokens per expert, latency grows linearly with the token count ($R^2 > 0.999$). The prefill chunks of Table 1 place 1,024–5,120 tokens on an average expert. Prefill GEMM time is therefore set by the token load of the rank. Figure 4(d) confirms this in serving on Qwen3-235B. The GEMM time of each layer and rank grows linearly with the token load of the rank ($R^2 = 0.99$). At the mean load, the part that does not depend on the load is 2.2% of the GEMM time.

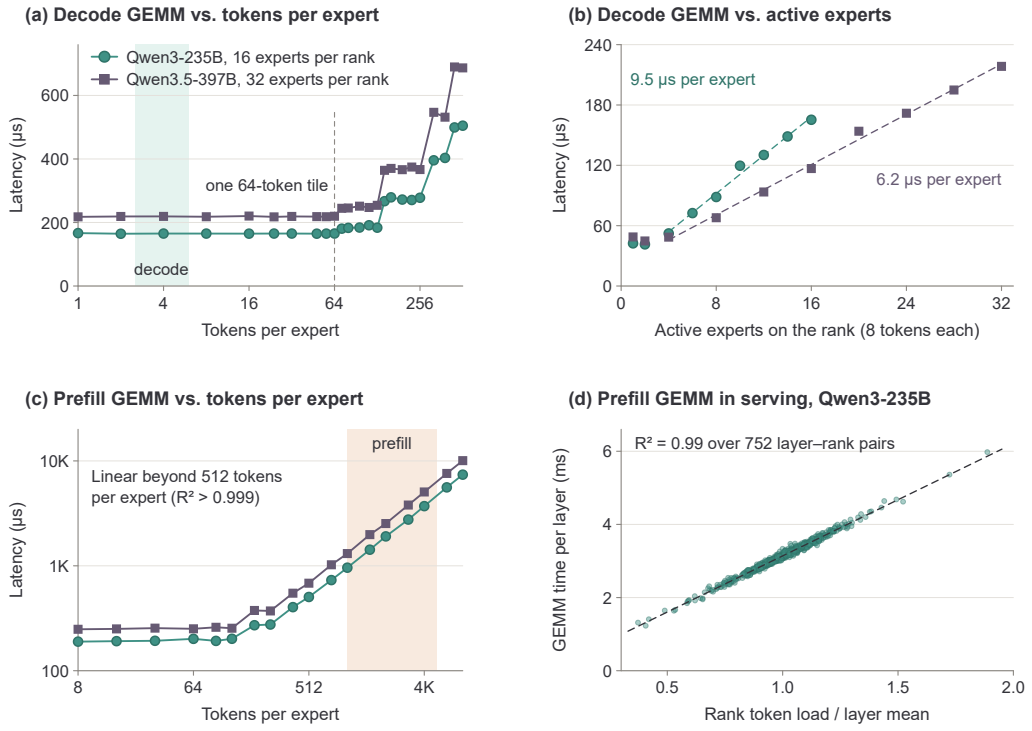


Figure 4: GEMM profiling. (a–c) Offline latency of the grouped expert GEMM for one MoE layer on one GPU, with the expert shapes and experts per rank of Qwen3-235B at EP= 8 and Qwen3.5-397B at EP= 16. (a) Decode layout with every expert on the rank active. (b) Decode layout with 8 tokens per active expert. (c) Prefill layout with every expert on the rank active. The shaded bands mark the tokens per expert at the batch sizes of Table 2 and at the chunk sizes of Table 1. (d) Expert GEMM time of each MoE layer and EP rank during Qwen3-235B prefill at 8K tokens per rank, against the token load of the rank divided by the layer mean. The dashed lines are least-squares fits.

D.3 Ablations

Protected prefix p. The protected prefix costs no prefill throughput. We vary p on Qwen3-235B with ShareGPT 4K inputs at 2K tokens per rank. Input throughput differs by at most 0.3% across $p = 0$, 1, and 2. Imbalance differs by less than 0.01.

Depth cap D and quality weights. Table 6 gives the throughput and quality of decode shedding on Qwen3-235B for each depth cap D with quality weights, and for $D = 4$ without them. Caps of $D \leq 4$ give a smaller decode gain than no cap. A cap of $D = 8$ gives the same gain as no cap. LiveCodeBench accuracy shows no monotone trend in D. A depth cap therefore brings no benefit, and decode runs without one. At $D = 4$, setting every quality weight to one leaves the active-expert count on the busiest rank unchanged. LiveCodeBench then drops by 1.49 pp, against 0.45 pp with the quality weights. The quality weights therefore change which experts close, not how many.

Candidate width L. Table 7 recomputes the prediction accuracy of Figure 3(b) offline for each L. From $L = 12$ to $L = 32$, every four added candidates roughly halve the missed routing weight. The decode candidate rows grow linearly with L. The prefill statistic row does not depend on L. Raising L from 16 to 24 enlarges the candidate rows by 50% and removes 73% of the missed weight. Raising L to 32 doubles the rows and removes 90%. $L = 16$ already keeps 99.1% of the routing weight, so Qwen3-235B uses $L = 16$.

Table 6: Decode shedding on Qwen3-235B for each depth cap D with quality weights, and for $D = 4$ without them, at ISL 1,024 and global batch 96. $\max_r A_r$ is the mean number of active experts on the busiest rank. This count and the gain in $1/\mathrm{TPOT}$ come from the throughput runs. Quality changes are in percentage points. LiveCodeBench and IFEval use ten repeats, or 13 for $D = 4$ with quality weights. MMLU-Redux uses three.

D	Weights	$\max_r A_r$	$\Delta\, 1/\mathrm{TPOT}$	LiveCodeBench v5	IFEval strict	MMLU-Redux 2.0
1	on	13.68	$+0.6\%$	-1.00	-0.35	-0.23
2	on	12.61	$+3.0\%$	-0.31	$+0.04$	-0.20
4	on	10.76	$+7.7\%$	-0.45	-0.11	-0.21
8	on	10.03	$+9.1\%$	-1.27	-0.13	-0.10
∞	on	9.74	$+9.1\%$	-0.59	-0.65	-0.13
4	off	10.75	$+7.7\%$	-1.49	-0.50	-0.26

Table 7: Cross-layer prediction accuracy on Qwen3-235B against the candidate width L, pooled over the 12 sampled layers of Figure 3(b). Top-L recall is the fraction of true Top-8 experts inside the predicted Top-L. Weight recall is the share of routing weight on these experts. Missed weight is the remainder. $L = 16$ is the default on Qwen3-235B.

L	8	12	16	20	24	32
Top-L recall	0.866	0.966	0.987	0.994	0.997	0.999
Weight recall	0.903	0.976	0.991	0.996	0.998	0.999
Missed weight (%)	9.7	2.4	0.92	0.45	0.25	0.095