GLoRE: Global-Local Router-Mass Expert Skipping for Batched MoE Serving

Anonymous authors
Paper under double-blind review

Abstract

Mixture-of-Experts (MoE) scales model capacity without proportionally increasing per-token compute and now underpins many large language models. This sparsity is less effective during serving: decode is dominated by weight movement, so each step's cost depends on the batch's union of selected experts rather than any one token's top-k experts. We introduce **GLoRE**, a runtime expert-masking method that reduces this union through *global expert masking for efficiency* and *local protection for quality*. At each decoding step, GLoRE masks experts with low aggregate router gate mass under a batch-level budget while protecting a token's top expert when its routing weight exceeds a threshold. This adapts expert reduction to each batch while preserving important token-level assignments, and remains robust as batch sizes vary. Implemented as a plug-and-play CUDA kernel for vLLM, GLoRE achieves up to $1.39\times$ decode throughput at batch size 64 with only 2% accuracy loss on Qwen3-30B-A3B, and remains on the accuracy–throughput Pareto frontier across four MoE models. On long chain-of-thought test-time scaling, GLoRE maintains vanilla-level maj@8 accuracy on AIME and GPQA-Diamond while using up to $1.20\times$ fewer GPU-seconds.

1 Introduction

Mixture-of-Experts (MoE) architectures have become a prevailing approach for scaling large language models, with models such as Mixtral (Jiang et al., 2024), DeepSeek-V3(DeepSeek-AI, 2024), Qwen3 (Yang et al., 2025), and Kimi K2 (Kimi Team, 2025) adopting sparse expert architectures. MoE replaces the dense feed-forward (FFN) sublayer with multiple smaller sub-networks, called *experts*, and dynamically activates only a subset for each input token, allowing models to substantially increase parameter capacity without proportionally increasing per-token computation (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022).

During autoregressive decoding, however, the practical sparsity of MoE can be much weaker than its per-token sparsity suggests. Each decoding step produces only one new token per sequence, resulting in low arithmetic intensity and making model-weight movement from high-bandwidth memory (HBM) to on-chip memory a large part of execution cost (Pope et al., 2023; Fu et al., 2024). This issue is amplified in batched serving: although each token activates only a few experts, different tokens can select different experts, requiring the system to access the union of experts selected across the batch. As batch size grows, this union expands with routing diversity, reducing the effective sparsity of expert-weight accesses (Gupta et al., 2024; Wu et al., 2026a; Oncescu et al., 2025).

Prior work reduces MoE inference cost through static compression or runtime expert reduction, using either static information obtained before deployment or runtime routing information. Static approaches fix the expert set or reduction policy before serving(Xie et al., 2024; Lasby et al., 2026) and therefore cannot adapt to the routing distribution of the current batch. Runtime approaches can adapt their decisions online, but they operate at different granularities: some make token-level decisions(Xu et al., 2026; Liu et al., 2026), while others constrain the batch-level expert set(Gupta et al., 2024; Oncescu et al., 2025; Vankov et al., 2026; Wu et al., 2026a;b).

This exposes a inherent *batch–token granularity tension*. Expert-weight movement is a batch-level cost, determined by the distinct experts accessed collectively by a batch, whereas the quality impact of removing an expert is determined at token level. A globally efficient decision can therefore remove an expert important to an individual token, while protecting a fixed number of experts per

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.

token can preserve too many distinct experts as batch size grows. Thus, although existing runtime methods adapt to the serving batch, the tradeoff between batch-level reduction and token-level protection can remain sensitive to batch size and composition. This motivates a design that explicitly coordinates the two granularities while remaining robust as batch sizes vary.

We address this tension with **GLoRE**, which balances batch-level expert reduction with token-level protection using *router gate mass*, a signal already produced by the router. GLoRE dynamically masks low-mass experts at batch scope while selectively protecting important token-level assignments, adapting both decisions to the current routing distribution. It requires no extensive offline calibration or fine-grained profiling and is implemented as a plug-in CUDA kernel for vLLM.

Across four MoE models and nine benchmarks, GLoRE achieves 1.21–$1.39\times$ decode throughput while retaining 93.6–98.2% of vanilla accuracy, and lies on the accuracy–throughput Pareto frontier across all four models. Its retained accuracy varies by at most 2.5 percentage points as the serving batch changes from $B = 8$ to 64, compared with 19.2–79.3 points for online baselines. For long chain-of-thought generation on AIME and GPQA-Diamond, GLoRE matches vanilla maj@8 accuracy within statistical noise while using up to $1.20\times$ fewer GPU-seconds per problem.

Our contributions are summarized as follows:

- We identify and characterize an inherent **batch–token granularity tension** in online MoE serving, where token-level protection can limit batch-level expert reduction as batch size grows, while aggressive batch-level reduction can remove experts important to individual tokens.
- We introduce GLoRE, an online expert-masking method that **masks globally and protects locally**. GLoRE dynamically reduces the batch-level expert union while selectively preserving important token-level routing decisions, providing a robust accuracy–efficiency tradeoff across varying batch sizes.
- We implement GLoRE with custom CUDA kernels in **vLLM**, including graph-safe fused routing and masked reduction, and evaluate its behavior across diverse MoE models and serving workloads. We further demonstrate an application to **test-time scaling** by applying GLoRE to long chain-of-thought generation.

2 Understanding the Batch–Token Tension

We now quantify the tension between *batch-level efficiency* and *token-level quality*, and examine why neither purely global nor fixed-count local masking is sufficient.

Batch-level expert union grows with batch size. Consider a batch of B tokens processed by an MoE layer with N experts and top-k routing (Shazeer et al., 2017). Although each token activates only k experts, the serving system must access the expert union

$$\mathcal{U} = \bigcup_{b=1}^{B} \mathcal{E}_b, \tag{1}$$

where $\mathcal{E}_b$ is the expert set selected by token b. Thus, expert-weight movement is determined by $|\mathcal{U}|$ rather than the per-token sparsity k. Modern MoE models pair a large expert pool with a small k (Dai et al., 2024), so as B increases different tokens activate increasingly diverse experts, causing the union to approach the full expert pool.

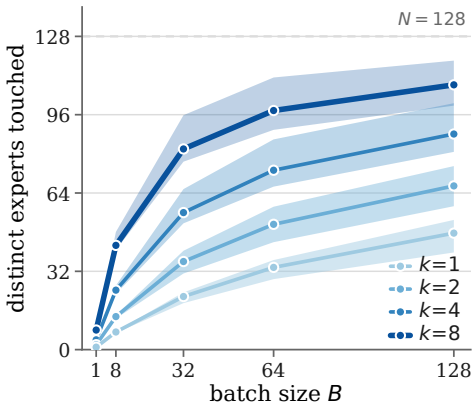

Figure 1: **Batch-level expert union.** Distinct experts accessed per decoding step in Qwen3-30B-A3B under different top-k configurations. Lines: mean over 48 MoE layers; bands: min–max.

For example, in Qwen3-30B-A3B with 128 experts and top-8 routing, a decoding step at $B = 128$ touches 108 experts on average across the 48 MoE layers, with the most diverse layer touching

118 experts (Figure 1). Even with $k = 1$, the union reaches 48 experts at $B = 128$. This gap between token-level sparsity and batch-level weight access is the first manifestation of the batch–token tension.

Global importance does not imply local importance. Reducing the expert union requires removal decisions at batch scope, but aggregate importance can hide token-level variation. An expert that is unimportant in aggregate may still be important to an individual token.

We quantify this disparity using the *token exposure* induced by a global mask. Let s_b denote the fraction of selected-expert importance for token b removed by the global mask, and define

$$R_B = \frac{\mathrm{p99}_b(s_b)}{\mathrm{mean}_b(s_b)}, \tag{2}$$

where p99 denotes the 99th percentile across tokens. Thus, R_B compares the exposure of the most heavily affected tokens with the batch-average exposure.

Under a fixed global masking budget, R_B rises from 1.0 at $B = 1$ to 2.0, 3.3, 3.8, and 4.0 at $B = 8, 32, 64$, and 128, respectively (Figure 12, Appendix D.2). Thus, global masking increasingly produces uneven token exposure as the batch grows, showing that batch-level importance alone cannot provide token-level protection.

Fixed-count protection creates a quality–efficiency tradeoff. A natural response is to retain the r highest-ranked experts for every token among its k routed experts, where $r \leq k$. Let

$$\mathcal{P} = \bigcup_{b=1}^{B} \mathcal{P}_b \tag{3}$$

be the resulting protected union. Because each token protects a fixed number of experts, $|\mathcal{P}|$ generally grows with both r and B. In Qwen3-30B-A3B at $B = 128$, retaining only the highest-ranked expert ($r = 1$) already produces a protected union of 48 experts, while $r = 2$ increases it to 67 experts.

The resulting tradeoff is straightforward: a small r preserves more masking headroom but may provide insufficient protection, while a larger r preserves more of each token's routing choices at the cost of a larger protected union. More importantly, fixed-count protection treats all tokens equally despite differences in their routing concentration. Effective masking should instead allocate protection according to the importance of each token's routing decisions.

3 GLoRE: Mask Globally, Protect Locally

GLoRE instantiates the requirements identified above using router gate mass, which is available before expert execution. At each MoE layer and decoding step, GLoRE (i) aggregates router gate mass per expert over the batch, (ii) masks the lowest-mass experts until their cumulative mass reaches a budget β, and (iii) keeps a masked assignment only when it is the token's top-1 expert with gate share at least μ/k, dropping the rest (Figure 2).

3.1 Design Principles

Router gate mass as a runtime importance signal. For an input x, an MoE layer computes

$$y = \sum_{e \in \mathcal{E}(x)} g_e(x) E_e(x), \tag{4}$$

where $g_e(x)$ is the router gate weight and $E_e(x)$ is the expert output. Gate mass thus provides an execution-free signal of how strongly the router assigns an input to each expert. Appendix D.1 shows that aggregating gate mass over a batch ranks experts by realized contribution with a median Spearman correlation of 0.950 across the 48 MoE layers of Qwen3-30B-A3B (0.886–0.970), while masking by this ranking gives up only $1.08\times$ the contribution of an oracle at the same prune count. Aggregation matters: selection frequency alone ranks experts less accurately (0.890), because batch importance depends on both how often and how strongly an expert is selected. Since gate mass is already produced by top-k routing, GLoRE requires no additional expert evaluation, task-specific calibration data, or offline profiling.



Figure 2: **GLoRE overview.** At each MoE layer and decoding step: **(1)** accumulate router-gate mass per expert over every routed slot of the step; experts never routed this step carry zero mass and are never masked. **(2)** Mask the cheapest prefix while its cumulative share stays within the budget β — one mask per layer, shared by the whole batch. **(3)** A masked slot survives only as the token's top-1 carrying at least μ times a uniform share of its gate mass ($g_1 / \sum g \geq \mu/k$); the rest take the sentinel id at weight 0, never rerouted.

Adaptive global masking and local protection. Routing mass is concentrated among a subset of experts, leaving a low-mass tail that provides opportunities for reducing expert accesses (Chen et al., 2026). A batch-level ranking is nonetheless insufficient: at $\beta = 0.20$, a purely global mask drops about 10% of slots carrying a high share of their token's contribution (Appendix D.1). GLoRE therefore aggregates gate mass across the batch to identify removable experts, while using the same signal at token scope to protect important local assignments. The mass budget β controls the fraction of batch routing mass targeted for removal, while the local threshold μ determines when a masked top-1 assignment is protected. This formulation adapts both decisions to the current routing distribution rather than fixing the number of experts.

Dropping rather than substituting masked experts. GLoRE drops masked expert contributions rather than substituting them with surviving experts. This avoids assuming that an already-accessed expert provides a suitable approximation of the removed contribution. Appendix D.3 tests that assumption on Qwen3-30B-A3B: in most layers even the best-matching surviving expert is nearly orthogonal to the removed output, so substituting it, with or without an optimal rescaling, recovers little more than dropping. Local protection instead preserves masked assignments that carry substantial token-level mass. Masked low-mass contributions are therefore simply skipped.

3.2 Global Mass-Based Masking

At each MoE layer and decoding step, GLoRE aggregates the routing mass of each expert across the batch. Let $g_{b,e}$ denote the gate weight for expert e and token b, so that expert e's aggregate gate mass is $G_e = \sum_{b=1}^{B} g_{b,e}$ and $G_{\text{tot}} = \sum_e G_e$. Sorting experts by ascending G_e, GLoRE masks the longest prefix whose cumulative mass is at most βG_{tot}:

$$m = \max\{\, j : \textstyle\sum_{i \leq j} G_{e_i} \leq \beta G_{\text{tot}} \},\qquad \mathcal{M} = \{e_1, \ldots, e_m\}. \tag{5}$$

Thus, β controls the fraction of aggregate routing mass targeted for removal rather than a fixed number of experts. The mask is rebuilt independently for every MoE layer and decoding step, adapting to the current batch routing distribution.

3.3 Local Mass-Based Protection

Global masking can still remove an expert important to an individual token. GLoRE protects a masked expert when it is the token's top-1 choice and carries sufficient local routing mass:

$$e_b^* \in \mathcal{M} \quad \text{and} \quad \frac{g_{b,e_b^*}}{\sum_{e \in \mathcal{E}_b} g_{b,e}} \geq \frac{\mu}{k}, \tag{6}$$

where e_b^* is token b's highest-weight expert and μ controls the protection threshold. All other masked assignments are dropped rather than rerouted. The resulting batch accesses

$$\widetilde{\mathcal{U}} = \bigcup_{b=1}^{B} \widetilde{\mathcal{E}}_b, \tag{7}$$

where $\widetilde{\mathcal{E}}_b$ contains the unmasked assignments and any protected top-1 assignment.

This two-level construction directly implements the requirement identified in Section 2: global masking controls the batch-level expert union, while local protection limits the exposure of individual tokens.

3.4 Implementation

GLoRE is implemented as a plug-in to vLLM (Kwon et al., 2023) and operates entirely on the routing path. It reuses the router's top-k ids and weights; unprotected masked experts are neither loaded nor executed. The mask is constructed with GPU tensor operations and stored in a persistent fixed-shape buffer, making GLoRE compatible with CUDA-graph replay without host synchronization or re-capture. A fused kernel performs mass aggregation, mask construction, and local protection in one control launch per MoE layer. We also fuse the downstream masked reduction, so a masked layer retains the same two kernel launches as the unmodified path. Masked slots are rewritten to a sentinel id with weight 0 and passed through standard MoE dispatch. Kernel pseudocode (Algorithms 1 and 2) and implementation details are provided in Appendix F.

4 Experiments

We evaluate GLoRE across four MoE models, nine benchmarks, and diverse serving workloads. We compare against static expert compression and runtime expert-reduction approaches, and study the accuracy–efficiency tradeoff, robustness across batch sizes, throughput scaling, local protection, and long chain-of-thought generation. We use Qwen3-30B-A3B as the primary model for detailed analysis and the remaining three models to assess generality.

4.1 Experimental Setup

Models and baselines. We evaluate four MoE models: Qwen3-30B-A3B (Yang et al., 2025), Qwen3.6-35B-A3B (Qwen Team, 2026), Gemma-4-26B-A4B-it (Gemma Team, 2026), and GLM-4.5-Air (GLM-4.5 Team, 2025). We use Qwen3-30B-A3B as the primary model for detailed analysis and the other three models to assess generality. We compare GLoRE with the vanilla model, REAP (Lasby et al., 2026), and SERE (Wu et al., 2026a). REAP represents static expert compression with a fixed pruned model, while SERE represents runtime expert reduction that adapts to the serving batch.

Evaluation. We evaluate nine benchmarks and seven serving workloads spanning diverse task types, concurrency levels, and input/output lengths. We report benchmark accuracy, serving throughput and latency, and long-CoT scaling metrics. Full hardware, software, benchmark, workload, and evaluation configurations are provided in Appendix A.

4.2 Accuracy–Efficiency Tradeoff

We first evaluate GLoRE's accuracy–efficiency tradeoff at $B = 64$.

Table 1 reports per-task accuracy on Qwen3-30B-A3B. GLoRE retains 98.0% of vanilla accuracy while achieving $1.39\times$ vanilla throughput. Its accuracy is comparable to the next-highest-accuracy non-vanilla baseline, SERE $k = 2$ ($57.93 \pm 1.08\%$ vs. $57.45 \pm 0.94\%$), while providing higher throughput ($1.39\times$ vs. $1.16\times$). More aggressive online reduction with SERE $k = 1$ reaches $1.51\times$ throughput but retains only 68.0% of vanilla accuracy. REAP provides a less favorable accuracy–efficiency tradeoff: 25% pruning retains only 54.8% of vanilla accuracy for just $1.05\times$ throughput, with accuracy collapsing to 0.0% on AIME and 5.1% on LiveCodeBench. Increasing pruning to 50% raises throughput to $1.26\times$ but further reduces retained accuracy to 49.3%. Per-task results for all four models, at every batch size, are given in Appendix B.1.

We next evaluate all four MoE models to examine whether this tradeoff generalizes across architectures.

Figure 3 shows that GLoRE lies on the Pareto frontier for all four models, retaining 93.6–98.2% of vanilla accuracy at 1.21–$1.39\times$ throughput. REAP-50% exhibits substantially different accuracy

Method	Accuracy retained (%)	Throughput gain (×)	AIME	GSM8K	M500	MHard	GPQA	MMLU-P	HE+	LCB	LBench
Vanilla	100.0	1.00×	28.33	91.96	75.40	62.08	48.99	70.75	63.41	30.86	60.06
REAP 25%	54.8	1.05×	0.00	84.69	38.40	17.30	29.80	47.00	26.22	5.14	42.90
REAP 50%	49.3	1.26×	0.00	79.76	44.20	21.15	20.20	32.50	22.56	5.14	36.55
SERE k=1	68.0	**1.51**×	3.67	83.69	60.56	17.86	35.15	54.65	50.85	20.91	34.30
SERE k=2	97.2	1.16×	24.00	**91.49**	76.92	63.28	48.48	69.20	60.00	**26.97**	**56.71**
GLoRE	**98.0**	1.39×	**26.33**	90.98	**77.44**	**64.73**	**49.70**	**70.05**	**60.73**	26.86	54.58

Table 1: Accuracy and decode throughput on Qwen3-30B-A3B at $B = 64$. *Accuracy retained* is the nine-task macro average relative to vanilla; *throughput gain* is decode throughput at concurrency 64 relative to vanilla. Per-task accuracy is shown after the rule. Among non-vanilla rows, **bold** and underline mark the best and second-best cells, respectively.

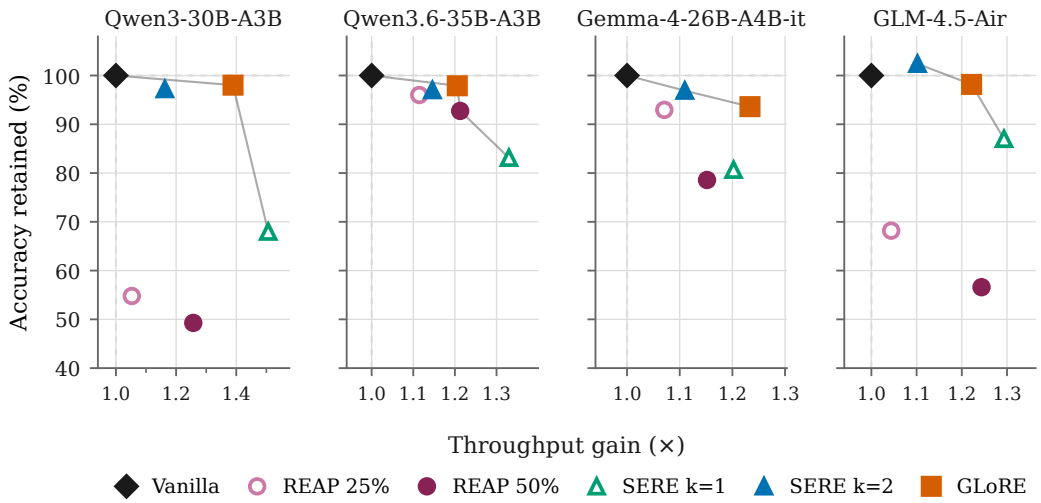

Figure 3: Accuracy–throughput tradeoff at $B = 64$, with one panel per model. Accuracy is the nine-task macro average normalized to the vanilla baseline; throughput is output tokens/s in the concurrency-64 decode regime, likewise normalized. The grey line denotes the Pareto frontier.

across models, retaining 49.3–92.7% despite using the same pruning ratio, indicating sensitivity to the underlying model. GLoRE also achieves higher accuracy and throughput than SERE $k = 2$ on Qwen3-30B-A3B and Qwen3.6-35B-A3B, while the two methods exhibit different accuracy–throughput tradeoffs on Gemma-4 and GLM-4.5-Air.

Overall, these results show that GLoRE maintains a consistent accuracy–efficiency tradeoff across diverse MoE architectures without extensive per-model calibration or offline preprocessing.

4.3 Robustness Across Batch Sizes

We evaluate GLoRE and SERE across batch sizes $B = 8$–64 to assess sensitivity to changing batch composition, which continuous batching (Yu et al., 2022) can vary from iteration to iteration as requests arrive and complete. We focus on these two runtime methods because their routing decisions adapt to the serving batch, unlike static compression methods.

Figure 4 shows a clear difference in batch-size sensitivity. GLoRE's retained accuracy varies by at most 2.5 points across the four models (1.0 on Qwen3-30B-A3B). SERE, in contrast, degrades sharply at small batches. From $B = 8$ to $B = 64$, SERE $k = 2$ improves by 19–38 percentage points, while SERE $k = 1$ improves by 62–79 points. In particular, at $B = 8$, SERE $k = 1$ retains only 1.4–19.0% of vanilla accuracy across models, indicating that its accuracy can degrade substantially when the batch does not provide a sufficiently large expert pool.

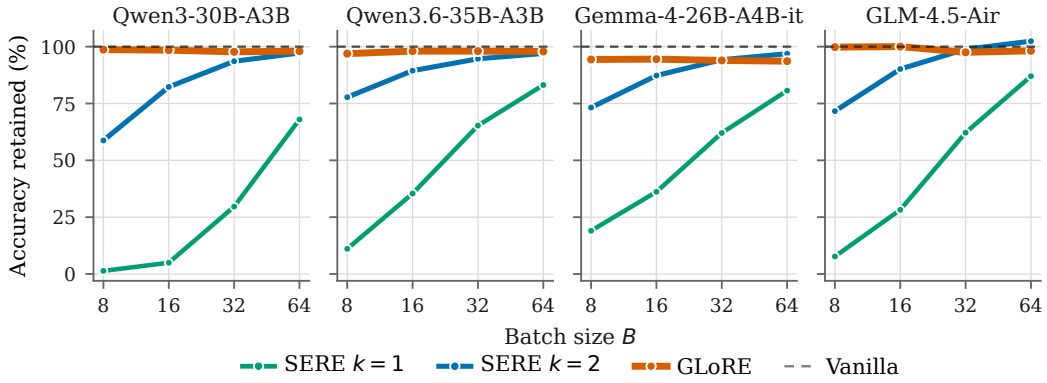

Figure 4: **Accuracy retained across batch sizes**, normalized to each model's own vanilla baseline. GLoRE remains stable across the sweep, while both SERE settings degrade as the batch shrinks.

This sensitivity arises from SERE's fixed per-token selection count: smaller batches produce fewer distinct remaining experts, limiting the substitution pool for rerouting. GLoRE instead uses a fractional mass budget, adapting the number of masked experts to the current routing distribution and batch size, while local protection preserves important assignments. Thus, GLoRE maintains more consistent accuracy across batch sizes, whereas SERE requires a sufficiently large batch to maintain an adequate substitution pool.

The corresponding throughput results are discussed separately in Section 4.4. Detailed evaluation settings are provided in Appendix A, and per-model numbers for the sweep in Appendix B.2.

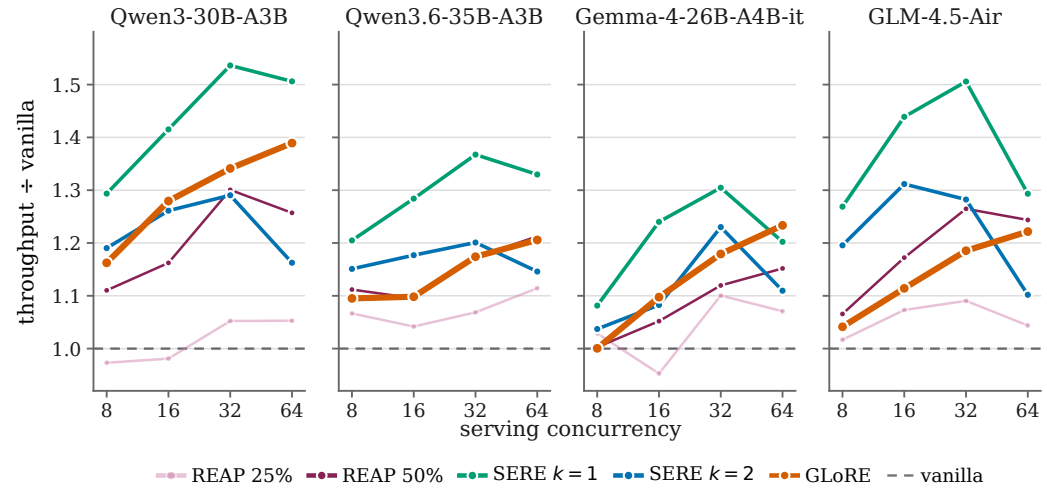

Figure 5: **Fixed-concurrency decode throughput**, normalized to each model's vanilla baseline at the same concurrency.

4.4 Throughput Scaling

Figure 5 shows that GLoRE's throughput continues to increase with concurrency under the tested serving workloads. On Qwen3-30B-A3B, GLoRE improves from $1.16\times$ at concurrency 8 to $1.39\times$ at 64. In contrast, SERE peaks at 32 and declines at 64: from $1.54\times$ to $1.51\times$ for $k{=}1$ and from $1.29\times$ to $1.16\times$ for $k{=}2$. REAP-50% similarly saturates at $1.26\times$ at 64. On the other three models, GLoRE's gain increases or holds from 32 to 64, while both SERE variants decline.

At concurrency 64, GLoRE achieves higher average throughput than SERE $k{=}2$ and REAP across the four models. Combined with its stable accuracy across batch sizes, the increasing throughput gain with concurrency makes GLoRE's accuracy–efficiency tradeoff more favorable at higher con-

currency. Results for the other workloads, including per-cell throughput and TPOT measurements, are provided in Appendix B.3.

4.5 Local Protection Enables Aggressive Masking

GLoRE has two control parameters: the global mass budget β and the local protection threshold μ. Table 2 analyzes how accuracy responds to β with and without protection on Qwen3-30B-A3B at $B = 64$ on a five-task subset (Appendix C), so its averages are not comparable to the nine-task averages above. The sweep characterizes the method rather than selecting β, which is fixed at 0.20 throughout.

Without local protection, increasing the masking budget causes a sharp accuracy collapse: the five-task macro average falls from 69.91 at $\beta = 0.05$ to 57.56 at $\beta = 0.20$ and 25.20 at $\beta = 0.30$. With local protection, accuracy remains near its peak through the default $\beta = 0.20$ and degrades more gradually at larger budgets. The benefit of protection therefore grows with the masking budget, from $+0.3$ points at $\beta = 0.05$ to $+40.4$ points at $\beta = 0.30$.

β	0.05	0.10	0.15	**0.20**	0.25	0.30
no protection ($\mu = 0$)	69.91	67.03	65.92	**57.56**	34.54	25.20
with protection ($\mu = 1.5$)	70.23	70.27	70.00	**70.90**	68.95	65.57
gain from protection	$+0.3$	$+3.2$	$+4.1$	$+13.3$	$+34.4$	$+40.4$

Table 2: **Local protection enables larger global masking budgets.** Qwen3-30B-A3B, $B = 64$, five-task macro average (GPQA-D, GSM8K, HumanEval+, MATH-500, MMLU-Pro); the default budget is in bold. Per-task results and the μ sweep are in Appendix C.

The protection threshold is less sensitive: at $\beta = 0.20$, sweeping $\mu \in [1.2, 2.0]$ changes the macro average by only 2.1 points. We select $\mu = 1.5$ on Qwen3-30B-A3B and use it for all four models (Appendix A.2). Full per-μ results and additional ablations are provided in Appendix C.

4.6 Long-CoT Test-Time Scaling

Long chain-of-thought (CoT) generation (Wei et al., 2022) amplifies the cost of each decoding step over thousands of tokens, making it a natural setting to evaluate whether GLoRE's per-step savings translate into lower test-time compute (Snell et al., 2025). We evaluate Qwen3-30B-A3B-Thinking on AIME 2024+2025 (4K–32K thinking budgets) and GPQA-Diamond (2K–16K), with majority voting over $k = 8$ samples and batch size 16.

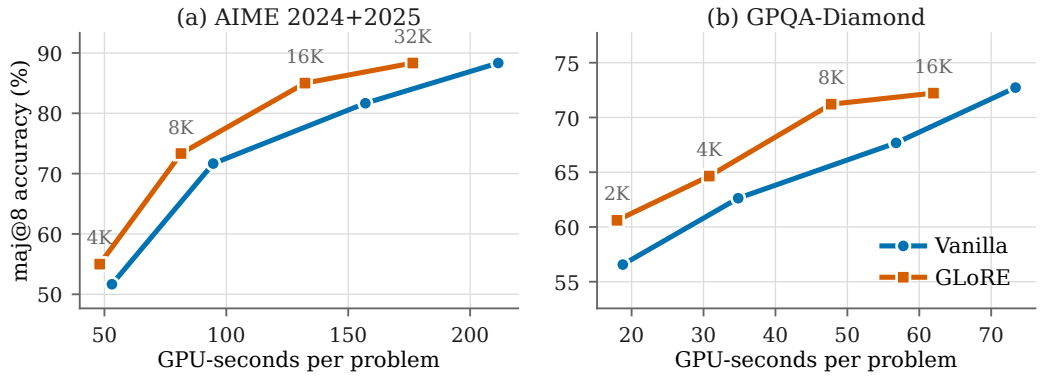

Figure 6: Long-CoT test-time scaling on (a) AIME 2024+2025 and (b) GPQA-Diamond. Accuracy is majority vote over $k = 8$ samples; labels give the thinking budget. Paired 95% CIs are in Table 18.

Figure 6 shows that GLoRE shifts long-CoT generation toward lower compute while maintaining comparable majority-vote accuracy on both tasks. On AIME, GLoRE uses 1.10–$1.20\times$ fewer GPU-seconds per problem from 4K to 32K, with paired maj@8 differences of $+3.3$, $+1.7$, $+3.3$, and 0.0 points; on GPQA-Diamond, it uses 1.05–$1.19\times$ fewer from 2K to 16K, with differences of $+4.0$, $+2.0$, $+3.5$, and -0.5 points. All confidence intervals span zero, so within the tested budgets

GLoRE reduces the cost of long-CoT inference without a statistically significant change in maj@8 accuracy.

These results suggest that GLoRE's per-step savings can translate into lower compute for long-CoT test-time scaling.

Detailed per-budget results and evaluation statistics are provided in Appendix E.

5 Related Work

Existing MoE inference optimization methods differ primarily in the granularity at which expert execution is reduced and whether runtime decisions rely on offline information.

Static expert compression. MoE-Pruner and REAP (Xie et al., 2024; Lasby et al., 2026) permanently remove or merge experts before deployment, producing a fixed compressed model. While this avoids runtime overhead, its accuracy can depend strongly on the workload and domain: the same pruning ratio can produce substantially different quality across models and tasks, as we observe for REAP in our evaluation. GLoRE instead makes expert-reduction decisions from the current serving batch, allowing the reduction to adapt to changing routing distributions.

Token-level runtime reduction. ACE (Xu et al., 2026) performs token-adaptive expert skipping using runtime routing information together with precomputed expert statistics, while Alloc-MoE (Liu et al., 2026) combines offline sensitivity profiling with runtime allocation of expert activation budgets across tokens and layers. By making reduction decisions independently at the token level, these methods do not directly constrain the *batch-level expert union*: different tokens can retain different experts, causing the number of distinct accessed experts to grow with batch size and limiting the resulting weight-movement reduction.

Batch-level runtime reduction. LYNX (Gupta et al., 2024), OEA (Oncescu et al., 2025), XShare (Vankov et al., 2026), and SERE (Wu et al., 2026a) reduce expert access at batch level while protecting some token-level assignments. The latest LYNX revision (v3, May 2026) uses confidence-weighted router affinity to construct an adaptively sized expert set, remapping low-affinity assignments while preserving high-confidence top experts. XShare selects a pool by aggregate gate score under an expert-count budget. OEA keeps each token's top-k_0 experts and fills remaining slots from their union using router scores, while SERE keeps each token's top-k experts and reroutes the rest using an offline similarity matrix. GLoRE instead defines global reduction by a *fraction of aggregate gate mass*, conditions local protection on the top-1 expert's share of the token's routing mass, and drops rather than reroutes remaining masked assignments.

Hybrid offline–runtime methods. NAEE and DiEP (Lu et al., 2024; Bai et al., 2025) use offline analysis to guide expert removal or selection, while ExFold (Wu et al., 2026b) calibrates expert projectors offline and performs phase-specific runtime reduction, using token-level selection for prefill and batch-level expert-pool selection for decode. GLoRE instead derives its decisions directly from the current routing distribution without offline profiling or task-specific calibration.

6 Conclusion

We presented GLoRE, a runtime approach to reducing expert execution in batched MoE serving. By combining batch-level reduction with selective preservation of token routing, GLoRE makes its decisions directly from the routing state at each decoding step, without extensive offline calibration or fine-grained profiling. Across diverse MoE models and benchmarks, GLoRE exhibits consistent accuracy over a range of serving batch sizes while maintaining a favorable accuracy–efficiency tradeoff. The resulting per-step savings also translate to lower generation cost for long chain-of-thought workloads.

AI USE STATEMENT

We used large language models as writing assistants to improve the clarity, grammar, and presentation of the manuscript, and as a search aid to help identify related work. All suggested text was reviewed and revised by the authors, and every retrieved paper was read and verified by the authors before being cited. The method and experimental design are the authors' own, and all experiments were run by the authors. The authors take full responsibility for the content of this paper.

ETHICS STATEMENT

This work studies inference efficiency for existing, publicly released MoE models. All benchmarks and prompt sets used in our experiments are publicly available datasets, used in accordance with their licenses. The work involves no human subjects, no new data collection, and no personally identifiable information.

REFERENCES

Sikai Bai, Haoxi Li, Jie Zhang, Zicong Hong, and Song Guo. DiEP: Adaptive mixture-of-experts compression through differentiable expert pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yuanteng Chen, Peisong Wang, Nanxin Zeng, Yuantian Shao, Shuang Qiu, Gang Li, Jing Liu, and Jian Cheng. Certain head, uncertain tail: Expert-sample for test-time scaling in fine-grained MoE. In *International Conference on Machine Learning (ICML)*, 2026.

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.

Damai Dai, Chengqi Deng, Chenggang Zhao, Runxin 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 (ACL)*, pp. 1280–1297, 2024.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, 2024.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

GLM-4.5 Team. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Vima Gupta, Jae Hyung Ju, Kartik Sinha, Ada Gavrilovska, and Anand Iyer. Lynx: Enabling efficient MoE inference through dynamic batch-aware expert selection. *arXiv preprint arXiv:2411.08982*, 2024. Version 3, revised May 2026.

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 *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2021.

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 *International Conference on Learning Representations (ICLR)*, 2025.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 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 *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2023.

Mike Lasby, Ivan Lazarevich, Nish Sinnadurai, Sean Lie, Yani Ioannou, and Vithursan Thangarasa. REAP the experts: Why pruning prevails for one-shot MoE compression. In *International Conference on Learning Representations (ICLR)*, 2026.

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. In *International Conference on Learning Representations (ICLR)*, 2021.

Baihui Liu, Kaiyuan Tian, Wei Wang, Zhaoning Zhang, Linbo Qiao, and Dongsheng Li. AllocMoE: Budget-aware expert activation allocation for efficient mixture-of-experts inference. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9653–9667, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for MoE large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6159–6172, 2024.

Mathematical Association of America. American invitational mathematics examination (AIME). `https://maa.org/maa-invitational-competitions/`, 2025.

Costin-Andrei Oncescu, Qingyang Wu, Wai Tong Chung, Robert Wu, Bryan Gopal, Junxiong Wang, Tri Dao, and Ben Athiwaratkun. Opportunistic expert activation: Batch-aware expert routing for faster decode without retraining. *arXiv preprint arXiv:2511.02237*, 2025.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

Qwen Team. Qwen3.6-35B-A3B. Model release, `https://huggingface.co/Qwen/Qwen3.6-35B-A3B`, 2026.

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 *Conference on Language Modeling (COLM)*, 2024.

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 (ICLR)*, 2017.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

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.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh Agrawal, Sandeep Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations (ICLR)*, 2025.

Juntong Wu, Jialiang Cheng, Fuyu Lv, Ou Dan, and Li Yuan. SERE: Similarity-based expert re-routing for efficient batch decoding in MoE models. In *International Conference on Learning Representations (ICLR)*, 2026a.

Juntong Wu, Yifei Liu, Junyi Chen, Siqi Fan, Chaoran Feng, Minghao Li, Liujie Zhang, Weihang Cheng, and Li Yuan. ExFold: Unified expert folding for training-free MoE prefill-decode acceleration. *arXiv preprint arXiv:2608.24938*, 2026b.

Yanyue Xie, Zhi Zhang, Ding Zhou, Cong Xie, Ziang Song, Xin Liu, Yanzhi Wang, Xue Lin, and An Xu. MoE-Pruner: Pruning mixture-of-experts large language model using the hints from its router. *arXiv preprint arXiv:2410.12013*, 2024.

Zukang Xu, Zhixiong Zhao, Xing Hu, Jiangyong Yu, Houji Wen, Jun Li, Zhe Jiang, and Dawei Yang. ACE: Adaptive calibration-free expert skipping for MoE-based LLMs. *arXiv preprint arXiv:2609.05228*, 2026.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 521–538, 2022.

A EXPERIMENTAL SETUP

This appendix gives the full configuration behind Section 4.1.

A.1 HARDWARE AND SOFTWARE

All experiments run on a single node with 8× NVIDIA H100 GPUs. Serving and evaluation use vLLM `0.18.1` (Kwon et al., 2023) on its V1 engine, in `bfloat16`, with CUDA graphs enabled. GLoRE is a vLLM plug-in whose masking and protection logic runs in a fused CUDA kernel invoked from the MoE layer and is compatible with CUDA-graph replay (Appendix F). Throughput and latency are measured with `vllm bench serve`, one server started and torn down per cell.

A.2 MODELS, BASELINES, AND GLORE SETTINGS

Model	HF id	total	active	MoE lyrs	experts	top-k
Qwen3-30B-A3B	`Qwen/Qwen3-30B-A3B`	30B	3B	48	128	8
Qwen3.6-35B-A3B	`Qwen/Qwen3.6-35B-A3B`	35B	3B	40	256	8
Gemma-4-26B-A4B-it	`google/gemma-4-26B-A4B-it`	26B	4B	30	128	8
GLM-4.5-Air	`zai-org/GLM-4.5-Air`	106B	12B	45	128	8

Table 3: **The four evaluated MoE models.** Expert counts and top-k are read from each model's `config.json`. Total and active parameters are the vendors' own figures as encoded in the model names. GLM-4.5-Air has a dense first layer (`first_k_dense_replace= 1`) and one shared expert, so 45 of its 46 layers are maskable; for the other three every layer is an MoE layer.

Table 3 summarises the four models. Tensor parallelism is 2 for the first three and 8 for GLM-4.5-Air.

REAP. REAP (Lasby et al., 2026) is a one-shot, offline expert-pruning method. On a calibration corpus, it scores each expert j by router-weighted activation saliency,

$$S_j = \frac{1}{|\mathcal{X}_j|} \sum_{x \in \mathcal{X}_j} g_j(x) \, \|E_j(x)\|_2,$$

and removes the lowest-scoring experts per layer. The original router and top-k are retained, so per-token compute is unchanged and no retraining is performed. We prune 25% and 50% of experts. We evaluate three calibration corpora—code (Evol-CodeAlpaca-v1), math (Tulu-3-SFT-Personas-Math), and web prose (FineWeb-Edu)—for Qwen3-30B-A3B, Qwen3.6-35B-A3B, and Gemma-4-26B-A4B-it, and code only for GLM-4.5-Air. We use the code-calibrated checkpoints for all reported REAP results because they provide the strongest overall accuracy among the calibration choices tested, while also avoiding favoring a calibration domain that differs from our mixed benchmark suite.

SERE. SERE (Wu et al., 2026a) reduces the batch expert set by retaining each token's top-S experts and their batch union as the *primary* set; SERE k=1 and k=2 use S=1 and S=2, respectively. Other selected experts are *secondary* and are matched to their most similar primary expert using a precomputed expert-similarity matrix. Following the original paper, we compute this matrix using Frobenius similarity over expert activations on FineWeb-Edu (400 sequences of 128 tokens). Secondary assignments are re-routed when similarity exceeds threshold ρ; otherwise the expert is preserved as *critical*. We use ρ=0, which re-routes all secondary assignments and disables critical-expert preservation. We do not tune a positive ρ because the layer-normalized similarity scale varies substantially across models and layers; for example, at ρ=0.5, the median layer admits 39% of off-diagonal pairs on Qwen3-30B-A3B but under 0.% on Qwen3.6-35B-A3B and GLM-4.5-Air. Thus, ρ=0 provides a consistent threshold across models and represents SERE's most aggressive setting.

GLoRE. GLoRE uses the same global budget $\beta = 0.20$ on every model. The local threshold is $\mu = 1.5$ on every model, selected on Qwen3-30B-A3B (Appendix C) and applied unchanged to the other three models. GLoRE requires no task-specific calibration data and no offline artifact.

Benchmark	n	shots	max new tokens
AIME 2024+2025 (Mathematical Association of America, 2025)	60	4	8192
GSM8K (Cobbe et al., 2021)	1319	5	2048
MATH500 (Hendrycks et al., 2021)	500	4	4096
MATH-Hard (Hendrycks et al., 2021)	1324	4	4096
GPQA-Diamond (Rein et al., 2024)	198	0	4096
MMLU-Pro (400) (Wang et al., 2024)	400	0	4096
HumanEval+ (Liu et al., 2023)	164	0	2048
LiveCodeBench (Jain et al., 2025)	175	0	4096
LiveBench-mixed (White et al., 2025)	581	0	12288

Table 4: **The nine accuracy benchmarks.** n is the number of items actually run and the shot count is shared by every model and arm; all are scored with greedy decoding. Dataset revisions and content hashes are pinned per task and recorded with each cell.

Workload	Input len.	Output len.	Max concurrency
C8	128	256	8
C16	128	256	16
C32	128	256	32
C64	128	256	64
LongGen	128	1024	64
LongPrompt	1024	256	64
PrefillHeavy	8192	64	64

Table 5: **The seven serving workloads.** Each is a `vllm bench serve` run with random prompts of the stated input and output length. The prompt count is at least $10\times$ the concurrency so the achieved batch tracks the cap; realised batch sizes are in Appendix B.3.

A.3 Benchmarks and Serving Workloads

Table 4 lists the nine accuracy benchmarks.

Serving throughput is measured over the seven synthetic workloads of Table 5.

Every speed number is a ratio to that model's own vanilla cell in the same workload. PrefillHeavy is a prefill-dominated control, where little benefit is expected (Appendix B.3).

A.4 Metrics and Evaluation Protocol

Accuracy is measured offline with greedy decoding (temperature 0, top-p 1; the long-CoT runs instead sample at temperature 0.6, Appendix E) at batch sizes $\{8, 16, 32, 64\}$. Batch sizes are pinned at engine construction so the mask operates over the stated batch. Per-task stop sequences and answer extractors are fixed across arms. SERE and GLoRE use five seeds per cell; vanilla and REAP use single runs. We report per-task accuracy and the nine-task macro average, with sample standard deviation across seeds. This variation is smaller than the scorer's binomial standard error for every task; for AIME (n=60), the latter is ≈6 percentage points.

All arms use the same task evaluators. AIME, MATH-500, and MATH-Hard use boxed-answer equality (`math_boxed_equal`), with last-number extraction as a secondary metric. GSM8K uses strict exact match on `####`. GPQA-Diamond and MMLU-Pro follow the `openai/simple-evals` protocol with `Random(0)` option shuffling and `n_repeats`$= 1$. HumanEval+ uses the EvalPlus test suite, LiveCodeBench uses its official harness at pass@1, and LiveBench-mixed uses its vendored per-subtask scorers.

Serving metrics are output-token throughput and mean time per output token from `vllm bench serve`. Accuracy and serving measurements use separate runs, so each Pareto point pairs independent measurements from the same configuration. For long-CoT generation, we report avg@k, pass@k, maj@k (Wang et al., 2023), and GPU-seconds (Appendix E).

B Full Experimental Results

B.1 Per-Task Accuracy at $B = 64$

Method	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
Vanilla	43.33	96.74	86.60	79.08	63.13	81.50	67.07	45.14	76.69	71.08
REAP 25%	41.67	**96.36**	**86.40**	79.68	46.97	75.00	68.90	43.43	**75.76**	68.24
REAP 50%	**50.00**	94.69	85.20	**79.98**	36.87	57.25	**73.78**	**49.14**	66.34	65.92
SERE k=1	30.33	92.77	77.24	69.47	50.61	72.75	58.42	26.63	53.59	59.09 $\pm$0.71
SERE k=2	39.00	95.94	84.60	79.28	**62.73**	**80.70**	64.39	44.12	69.79	68.95 $\pm$0.39
GLoRE	42.33	96.03	85.28	78.32	62.42	80.60	63.41	44.57	73.27	**69.58** $\pm$0.43

Table 6: **Per-task accuracy on Qwen3.6-35B-A3B at $B = 64$. Overall** is their macro average with the sample spread across runs. SERE and GLoRE carry five seeds; vanilla and REAP are single-run. Among non-vanilla rows, **bold** and underline mark the best and second-best cells, respectively; the shaded row is GLoRE.

Method	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
Vanilla	63.33	95.91	87.40	82.10	69.70	84.50	92.68	62.29	69.55	78.61
REAP 25%	**56.67**	94.92	**87.20**	**81.87**	52.53	71.25	**93.29**	**59.43**	60.46	73.07
REAP 50%	31.67	92.27	79.60	67.60	37.88	51.75	90.85	53.14	51.14	61.77
SERE k=1	25.67	91.11	77.72	67.99	60.00	72.50	84.64	35.89	55.29	63.42 $\pm$0.68
SERE k=2	**56.67**	**95.39**	86.12	80.02	68.18	**81.35**	91.46	59.31	**67.04**	**76.17** $\pm$0.62
GLoRE	55.00	94.16	82.40	75.94	**68.44**	79.88	92.07	53.14	61.27	73.59 $\pm$0.14

Table 7: **Per-task accuracy on Gemma-4-26B-A4B-it at $B = 64$. Overall** is their macro average with the sample spread across runs. SERE and GLoRE carry five seeds; vanilla and REAP are single-run. Among non-vanilla rows, **bold** and underline mark the best and second-best cells, respectively; the shaded row is GLoRE.

Method	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
Vanilla	21.66	91.73	58.30	39.73	58.09	75.38	84.15	36.57	51.51	57.46
REAP 25%	5.00	82.49	46.40	30.29	32.83	57.00	54.88	8.57	35.18	39.18
REAP 50%	3.33	73.46	45.80	28.47	19.19	39.00	53.66	7.43	22.36	32.52
SERE k=1	10.00	89.45	59.16	41.22	50.50	67.65	72.32	18.06	41.54	49.99 $\pm$0.62
SERE k=2	**27.67**	92.23	**61.92**	**48.27**	57.17	**75.10**	82.81	**32.91**	**51.48**	**58.84** $\pm$1.15
GLoRE	21.33	**92.92**	56.88	39.71	**57.27**	74.00	**83.91**	30.74	50.94	56.41 $\pm$0.58

Table 8: **Per-task accuracy on GLM-4.5-Air at $B = 64$. Overall** is their macro average with the sample spread across runs. SERE and GLoRE carry five seeds; vanilla and REAP are single-run. Among non-vanilla rows, **bold** and underline mark the best and second-best cells, respectively; the shaded row is GLoRE.

Tables 6–8 give per-task accuracy for every method at $B = 64$, the setting of the main accuracy results (Section 4.2), on the three models not already tabulated there; Qwen3-30B-A3B is Table 1 in the main text. For vanilla and offline-compressed REAP, each token's routing decision is independent of the serving batch; consequently, neither requires batch-size-specific accuracy evaluation. Both GLoRE and SERE settings were additionally run at $B = 8$, 16 and 32; those results are reported in Appendix B.2.

Across the three models, GLoRE remains relatively close to vanilla on most tasks, although the magnitude and location of per-task differences vary by model. On Qwen3.6-35B-A3B, GLoRE achieves 69.58% overall accuracy, the highest among the non-vanilla methods (next: SERE $k = 2$ at 68.95%), with its largest drops on HumanEval+ and LiveBench. On Gemma-4-26B-A4B-it, GLoRE retains 93.6% of vanilla accuracy, with larger differences on AIME, MATH-Hard, LiveCodeBench, and LiveBench. On GLM-4.5-Air, GLoRE retains 98.2% of vanilla accuracy, with less than 1.5 points of difference on eight of nine tasks; LiveCodeBench is the main exception.

REAP-50% exhibits substantial task-specific collapse, particularly on GPQA-Diamond and MMLU-Pro for Qwen3.6 and Gemma-4, and additionally on AIME, LiveCodeBench, and LiveBench for GLM-4.5-Air. The two SERE settings illustrate a clear quality–reduction tradeoff: increasing the retained experts from $k = 1$ to $k = 2$ substantially improves accuracy across all three models, raising the overall accuracy from 59.09% to 68.95% on Qwen3.6, from 63.42% to 76.17% on Gemma-4, and from 49.99% to 58.84% on GLM-4.5-Air. The improvement is especially pronounced on AIME, MATH, MATH-Hard, and LiveCodeBench, indicating that the more aggressive $k = 1$ setting can incur substantial task-level quality losses.

B.2 Accuracy Across Batch Sizes

Method	B	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
SERE k=1	8	0.00	0.02	0.04	0.05	1.11	0.75	0.12	0.00	5.19	0.81 ±0.12
	16	0.00	2.18	2.36	0.26	2.93	7.60	2.93	0.80	7.30	2.93 ±0.34
	32	1.00	55.89	17.36	3.64	8.29	14.80	25.12	11.54	20.06	17.52 ±0.76
	64	3.67	83.69	60.56	17.86	35.15	54.65	50.85	20.91	34.30	40.18 ±0.94
SERE k=2	8	4.00	79.15	44.40	23.79	29.60	50.80	29.27	13.49	37.95	34.72 ±0.55
	16	15.33	88.55	66.40	48.63	42.63	62.75	50.37	18.86	44.18	48.63 ±0.87
	32	19.67	90.71	75.32	60.00	47.17	67.00	60.13	25.94	51.79	55.30 ±0.67
	64	24.00	**91.49**	76.92	63.28	48.48	69.20	60.00	**26.97**	**56.71**	57.45 ±0.94
GLoRE	8	**27.67**	**91.40**	**78.16**	**63.56**	**49.09**	**68.95**	**61.34**	**28.00**	**56.76**	**58.33** ±0.71
	16	**29.67**	**91.10**	**76.80**	**63.96**	**49.60**	**69.45**	**61.22**	**26.51**	**55.44**	**58.19** ±0.77
	32	**26.67**	**91.27**	**78.04**	**64.40**	**49.29**	**68.15**	**62.32**	**26.06**	**53.61**	**57.76** ±0.48
	64	**26.33**	90.98	**77.44**	**64.73**	**49.70**	**70.05**	**60.73**	26.86	54.58	**57.93** ±1.08

Table 9: **Per-task accuracy across batch sizes on Qwen3-30B-A3B.** At each batch size, **bold** and underline mark the best and second-best of the three methods; shaded rows are GLoRE. **Overall** is the mean over five seeds with its spread. Vanilla and REAP have no sweep and are omitted.

Method	B	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
SERE k=1	8	0.00	23.56	7.24	2.01	3.64	10.00	3.42	0.00	21.05	7.88 ±0.48
	16	2.00	59.08	42.88	26.00	10.81	23.05	30.37	1.94	30.35	25.16 ±0.92
	32	16.00	84.58	68.44	56.48	34.44	52.40	53.66	9.60	41.97	46.40 ±0.74
	64	30.33	92.77	77.24	69.47	50.61	72.75	58.42	26.63	53.59	59.09 ±0.71
SERE k=2	8	25.67	85.11	73.08	62.60	46.26	72.25	54.51	20.00	58.03	55.28 ±0.67
	16	35.00	92.71	80.40	72.66	53.64	77.50	63.29	33.37	63.57	63.57 ±0.49
	32	41.00	95.07	84.00	77.07	58.08	78.85	62.93	41.03	67.58	67.29 ±0.72
	64	39.00	95.94	84.60	**79.28**	**62.73**	**80.70**	**64.39**	44.12	69.79	68.95 ±0.39
GLoRE	8	**37.67**	**96.16**	**85.84**	**78.31**	**62.22**	**80.85**	**62.93**	**43.43**	**72.71**	**68.90** ±0.41
	16	**38.33**	**96.13**	**84.92**	**78.54**	**63.64**	**81.20**	**64.39**	**46.17**	**73.82**	**69.68** ±0.22
	32	**44.33**	**96.03**	**84.56**	**77.61**	**62.63**	**81.45**	**63.90**	**43.43**	**73.16**	**69.68** ±0.59
	64	**42.33**	**96.03**	**85.28**	78.32	62.42	80.60	63.41	**44.57**	**73.27**	**69.58** ±0.43

Table 10: **Per-task accuracy across batch sizes on Qwen3.6-35B-A3B.** At each batch size, **bold** and underline mark the best and second-best of the three methods; shaded rows are GLoRE. **Overall** is the mean over five seeds with its spread. Vanilla and REAP have no sweep and are omitted.

Tables 9–12 give the per-task numbers behind Figure 4, for the three arms that have a batch sweep.

GLoRE shows limited batch-size sensitivity at both the aggregate and per-task levels. Its overall accuracy varies by less than 1.5 points across $B = 8$–64 on every model, and the per-task values generally fluctuate rather than follow a systematic trend with batch size. In contrast, SERE's accuracy depends strongly on the serving batch size. For k=1, overall accuracy varies by 39.4–51.2 points between $B = 8$ and 64, falling to near zero on several reasoning and coding tasks at small batches. SERE k=2 is less sensitive than k=1, but its overall accuracy still varies by 13.7–22.7 points over the same range, far more than GLoRE's.

The resulting gap is concentrated at small batches. At $B = 8$, GLoRE exceeds SERE k=2 by 13.6–23.6 points across the four models, whereas the two methods are much closer at $B = 64$. This behavior is consistent with SERE's batch-level substitute pool: at small batches, fewer primary

Method	B	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
SERE k=1	8	0.00	21.09	4.40	1.57	28.28	40.50	7.44	0.34	30.86	14.94 ±0.41
	16	0.67	50.08	24.32	12.11	33.33	47.10	46.58	3.32	38.23	28.42 ±0.75
	32	10.67	78.20	59.40	45.44	47.98	61.40	76.83	12.80	46.24	48.77 ±0.80
	64	25.67	91.11	77.72	67.99	60.00	72.50	84.64	35.89	55.29	63.42 ±0.68
SERE k=2	8	20.00	82.05	70.72	55.15	52.33	70.20	85.13	26.06	56.35	57.55 ±0.63
	16	39.67	91.98	81.32	70.83	61.62	75.75	90.12	45.60	60.66	68.62 ±0.63
	32	54.00	**94.51**	**84.48**	**77.13**	66.26	79.90	**91.83**	**54.74**	**64.08**	**74.10** ±0.34
	64	**56.67**	**95.39**	**86.12**	**80.02**	68.18	**81.35**	91.46	**59.31**	**67.04**	**76.17** ±0.62
GLoRE	8	**54.17**	**94.39**	**83.40**	**75.53**	**66.42**	**80.50**	**93.90**	**57.43**	**61.91**	**74.18** ±0.91
	16	**56.66**	**94.01**	**83.70**	**75.53**	**68.94**	**80.75**	**92.07**	**55.71**	**61.20**	**74.29** ±0.64
	32	**57.22**	94.12	82.90	75.80	**66.67**	**80.12**	91.46	54.28	60.97	73.85 ±1.05
	64	55.00	94.16	82.40	75.94	**68.44**	79.88	**92.07**	53.14	61.27	73.59 ±0.14

Table 11: **Per-task accuracy across batch sizes on Gemma-4-26B-A4B-it.** At each batch size, **bold** and underline mark the best and second-best of the three methods; shaded rows are GLoRE. **Overall** is the mean over five seeds with its spread. Vanilla and REAP have no sweep and are omitted.

Method	B	AIME	GSM8K	MATH	M-Hard	GPQA	MMLU	HE+	LCB	LiveB	**Overall**
SERE k=1	8	0.00	9.33	1.00	0.00	6.06	13.75	1.22	0.00	8.47	4.43
	16	0.00	39.08	14.20	4.30	20.20	38.62	11.29	2.29	16.00	16.22 ±0.62
	32	1.67	78.70	39.45	16.60	34.09	56.50	57.47	8.57	28.51	35.73 ±0.71
	64	10.00	89.45	59.16	41.22	50.50	67.65	72.32	18.06	41.54	49.99 ±0.62
SERE k=2	8	5.83	69.59	45.20	26.09	45.96	60.12	66.77	15.14	35.68	41.16 ±0.78
	16	15.84	85.56	**59.30**	**40.64**	53.79	70.38	78.05	20.00	42.66	51.80 ±0.60
	32	**24.16**	89.99	**64.40**	**49.09**	55.94	72.12	80.34	27.14	48.20	**56.82** ±0.81
	64	**27.67**	92.23	**61.92**	**48.27**	57.17	**75.10**	82.81	**32.91**	**51.48**	**58.84** ±1.15
GLoRE	8	**26.11**	**92.80**	**57.47**	**40.38**	**59.60**	**74.08**	**82.93**	**30.86**	**51.84**	**57.34** ±0.91
	16	**28.34**	**92.70**	56.33	39.85	**58.92**	**74.58**	**83.74**	**31.62**	**51.19**	**57.48** ±0.62
	32	20.00	**93.05**	54.73	40.46	**56.06**	**73.50**	**84.76**	**31.62**	**50.07**	56.03 ±0.34
	64	21.33	**92.92**	56.88	39.71	**57.27**	74.00	**83.91**	30.74	50.94	56.41 ±0.58

Table 12: **Per-task accuracy across batch sizes on GLM-4.5-Air.** At each batch size, **bold** and underline mark the best and second-best of the three methods; shaded rows are GLoRE. **Overall** is the mean over five seeds with its spread. Vanilla and REAP have no sweep and are omitted.

experts are available as alternatives for removed assignments, while a larger batch provides a richer substitute pool. GLoRE instead recomputes its global mask from the current routing distribution and conditionally protects important local assignments, avoiding reliance on a sufficiently large shared substitute pool.

B.3 Serving Performance

Serving performance is measured with `vllm bench serve` over the seven workloads defined in Appendix A.3. The table below reports the achieved batch range across the four models; the engine sustains slightly below the configured maximum concurrency, with the gap remaining small for all decode workloads.

Workload	achieved batch	role
C8	7.0–7.7	low-concurrency decode
C16	14.8–15.4	
C32	29.8–30.9	
C64	60.5–61.5	serving-batch decode
LongGen	62.7–63.4	queue-bound
LongPrompt	57.2–59.5	mixed input/output lengths
PrefillHeavy	52.4–53.8	prefill-dominated (control)

We provide additional serving results beyond the C8–C64 concurrency sweep of Section 4.4. Figures 7–10 report the three remaining workload configurations for each model: LongGen, Long-

Prompt, and PrefillHeavy. Each figure reports output-token throughput and mean time per output token (TPOT), both relative to that model's own vanilla baseline under the same workload.

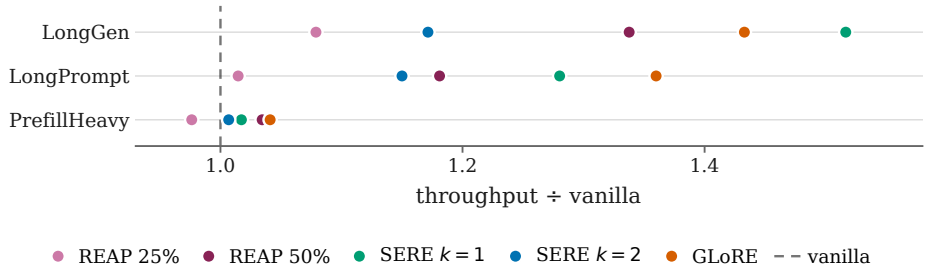

Figure 7: **Qwen3-30B-A3B** on the three non-concurrency workloads, throughput relative to this model's own vanilla in the same workload.

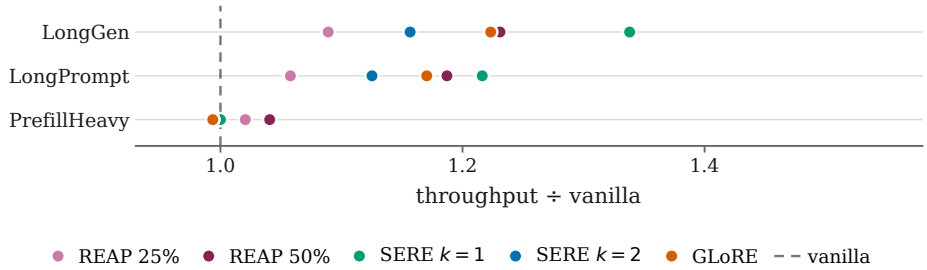

Figure 8: **Qwen3.6-35B-A3B** on the three non-concurrency workloads, throughput relative to this model's own vanilla in the same workload.

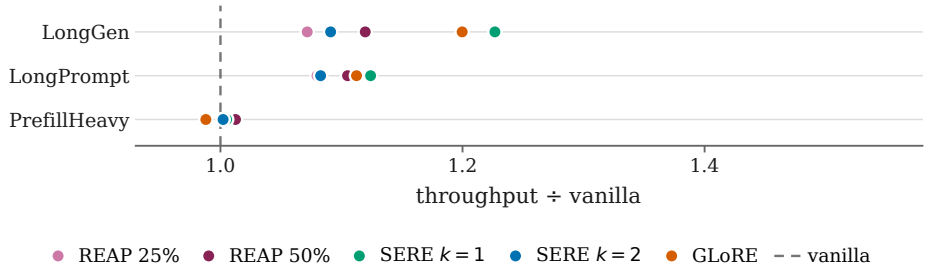

Figure 9: **Gemma-4-26B-A4B-it** on the three non-concurrency workloads, throughput relative to this model's own vanilla in the same workload.

Across the decode-oriented workload variants, GLoRE maintains substantial serving gains. On Qwen3-30B-A3B, for example, GLoRE reaches 7,117 tokens/s on LongGen and 5,627 tokens/s on LongPrompt, compared with 4,967 and 4,138 tokens/s for vanilla. The corresponding TPOT decreases from 12.72 to 8.80 ms on LongGen and from 14.19 to 9.99 ms on LongPrompt. Similar improvements are observed across the other three models, although their magnitude varies with model architecture and workload.

LongGen particularly highlights the benefit of decode-time expert reduction: with four times as many generated tokens as the standard decode workload, per-step savings accumulate over a substantially longer generation. GLoRE achieves 1.18–1.43× the vanilla throughput across the four models on LongGen, while reducing TPOT by approximately 15–31%. LongPrompt also shows consistent improvements, with throughput reaching 1.11–1.36× vanilla and TPOT reductions of approximately 13–30%.

In contrast, PrefillHeavy is a prefill-dominated control, where little benefit is expected. GLoRE's throughput remains close to vanilla across all four models, ranging from 0.99× to 1.04× of the corresponding vanilla throughput, while TPOT remains similarly close. This contrasts with the larger gains on decode-oriented workloads and is consistent with GLoRE's savings coming from decode-time weight movement, which this workload barely exercises.

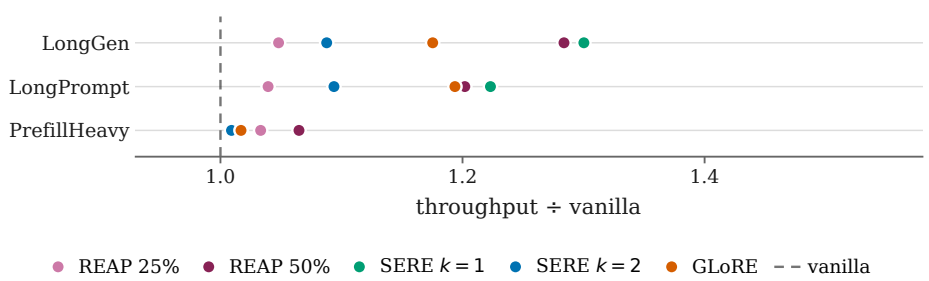

Figure 10: **GLM-4.5-Air** on the three non-concurrency workloads, throughput relative to this model's own vanilla in the same workload.

Model	Method	C8	C16	C32	C64	LongGen	LongPrompt	PrefillHeavy
Qwen3-30B-A3B	Vanilla	1062	1782	2892	4830	4967	4138	514
	REAP 25%	1034	1748	3043	5085	5359	4198	502
	REAP 50%	1180	2071	3761	6072	6644	4887	532
	SERE k=1	**1374**	**2521**	**4442**	**7274**	**7533**	5298	523
	SERE k=2	1265	2247	3731	5615	5818	4759	517
	GLoRE	1235	2279	3878	6710	7117	**5627**	**535**
Qwen3.6-35B-A3B	Vanilla	980	1679	2644	4062	4294	3670	536
	REAP 25%	1046	1749	2826	4526	4677	3882	547
	REAP 50%	1090	1839	3100	4926	5286	4357	**558**
	SERE k=1	**1181**	**2156**	**3616**	**5401**	**5746**	**4464**	536
	SERE k=2	1128	1976	3175	4654	4967	4130	533
	GLoRE	1073	1844	3104	4897	5253	4296	533
Gemma-4-26B-A4B-it	Vanilla	1335	2304	3772	6086	6006	4406	547
	REAP 25%	1373	2195	4151	6517	6437	4757	551
	REAP 50%	1338	2424	4223	7010	6724	4868	**554**
	SERE k=1	**1444**	**2857**	**4921**	7317	**7368**	**4952**	550
	SERE k=2	1384	2494	4640	6753	6553	4770	548
	GLoRE	1336	2528	4448	**7507**	7205	4901	540
GLM-4.5-Air	Vanilla	907	1411	2344	4049	4131	3280	394
	REAP 25%	922	1514	2556	4226	4330	3409	408
	REAP 50%	966	1654	2964	5035	5304	3942	**420**
	SERE k=1	**1151**	**2030**	**3529**	**5236**	**5372**	**4012**	401
	SERE k=2	1084	1850	3006	4461	4494	3588	398
	GLoRE	944	1572	2778	4946	4856	3915	401

Table 13: **Serving throughput across all workloads** (output tokens/s, `vllm bench serve`). Compare within a model and workload. Among each model's non-vanilla rows, **bold** and underline mark the highest and second-highest values; the shaded row is GLoRE.

PrefillHeavy spends ≈0.8% of its time in decode, so the MoE layer is compute-bound and a mask that removes weight transfers has almost nothing to remove. Every method returns between 0.98× and 1.07× there. We report it as a control rather than as a workload, and it is the reason the seven-regime geometric mean (1.163× for GLoRE) sits below the decode-only figure of 1.19×. The two mixed workloads, by contrast, sit at ≈60 co-resident requests and reproduce the concurrency-64 result: 1.25× on LongGen and 1.21× on LongPrompt for GLoRE.

Tables 13 and 14 report the complete absolute throughput and TPOT measurements for every model–method–workload combination, including the vanilla baseline. Absolute values are not directly comparable across models, which use different parameter counts and tensor-parallel degrees, or across workloads, which differ in input/output length mix. Every ratio reported in this paper is computed relative to the vanilla measurement for the same model and workload and can therefore be recomputed from these tables.

Model	Method	C8	C16	C32	C64	LongGen	LongPrompt	PrefillHeavy
Qwen3-30B-A3B	Vanilla	7.22	8.63	10.68	12.65	12.72	14.19	103.69
	REAP 25%	7.40	8.75	10.13	11.93	11.80	13.95	105.89
	REAP 50%	6.44	7.34	8.09	10.00	9.48	11.60	**99.98**
	SERE k=1	**5.41**	**5.88**	**6.73**	**8.25**	**8.35**	10.81	102.15
	SERE k=2	5.92	6.68	8.09	10.71	10.85	12.16	103.22
	GLoRE	6.11	6.60	7.72	8.76	8.80	**9.99**	100.27
Qwen3.6-35B-A3B	Vanilla	7.12	8.84	11.43	14.90	14.67	15.85	97.75
	REAP 25%	6.95	8.50	10.67	13.45	13.49	15.17	96.24
	REAP 50%	6.62	8.03	9.61	12.13	11.87	13.23	**94.94**
	SERE k=1	**6.01**	**6.67**	**8.14**	**11.03**	**10.93**	**12.85**	98.29
	SERE k=2	6.43	7.38	9.34	12.72	12.68	13.95	98.82
	GLoRE	6.77	7.97	9.56	12.16	11.94	13.35	99.64
Gemma-4-26B-A4B-it	Vanilla	5.57	6.64	7.89	10.03	10.43	12.97	97.14
	REAP 25%	5.48	6.49	7.25	9.29	9.77	12.10	96.78
	REAP 50%	5.70	6.28	7.27	8.77	9.40	11.79	**96.38**
	SERE k=1	**5.21**	**5.23**	**6.07**	8.17	**8.51**	11.52	96.46
	SERE k=2	5.43	6.06	6.49	8.94	9.62	12.04	97.04
	GLoRE	5.61	6.01	6.77	**8.01**	8.64	**11.30**	97.94
GLM-4.5-Air	Vanilla	8.33	10.84	13.11	15.11	15.34	18.04	134.96
	REAP 25%	8.14	10.01	11.96	14.49	14.58	17.16	131.01
	REAP 50%	7.74	9.12	10.23	11.81	11.89	14.50	**127.46**
	SERE k=1	**6.40**	**7.30**	**8.46**	**11.32**	**11.69**	**14.06**	131.97
	SERE k=2	6.84	8.08	10.05	13.47	14.05	15.98	134.18
	GLoRE	7.98	9.68	11.00	12.24	12.98	14.74	134.61

Table 14: **Serving latency across all workloads** (mean TPOT, ms, `vllm bench serve`). Compare within a model and workload. Among each model's non-vanilla rows, **bold** and underline mark the lowest and second-lowest values; the shaded row is GLoRE.

C Ablations: Global Budget, Local Protection, and Parameter Sensitivity

We use five representative tasks—GPQA-Diamond, GSM8K, HumanEval+, MATH-500, and MMLU-Pro—to reduce evaluation cost while covering diverse task types. On Qwen3-30B-A3B at $B = 64$, Table 15 varies the global masking budget β with and without local protection to measure how masking strength affects accuracy and how much local protection contributes. This sweep is an effect analysis, not a hyperparameter search: all other experiments use $\beta = 0.20$. Table 16 varies the protection threshold μ at the default budget.

β	protection	GPQA-D	GSM8K	HE+	MATH	MMLU	**macro**
0.05	off	47.47	90.67	62.80	76.60	72.00	69.91
	μ=1.5	47.47	90.67	65.85	76.40	70.75	70.23
0.10	off	50.00	90.45	53.05	75.40	66.25	67.03
	μ=1.5	51.01	90.75	64.63	77.20	67.75	70.27
0.15	off	46.46	90.22	54.27	73.40	65.25	65.92
	μ=1.5	47.98	90.83	63.41	76.80	71.00	70.00
0.20	off	31.31	81.65	46.34	66.00	62.50	57.56
	μ=1.5	51.01	91.81	64.02	77.40	70.25	70.90
0.25	off	25.76	0.76	31.71	56.20	58.25	34.54
	μ=1.5	50.51	90.37	62.20	75.40	66.25	68.95
0.30	off	20.20	0.00	28.05	31.00	46.75	25.20
	μ=1.5	42.93	89.76	58.54	71.60	65.00	65.57

Table 15: **Budget sweep with and without local protection**, Qwen3-30B-A3B, $B = 64$, five-task accuracy. The shaded row is the default setting. Five-task macro, **not** comparable to the nine-task averages used elsewhere.

protection	GPQA-D	GSM8K	HE+	MATH	MMLU	**macro**
off	31.31	81.65	46.34	66.00	62.50	57.56
μ=1.2	50.00	91.28	62.20	78.20	66.50	69.64
μ=1.3	46.46	91.74	65.24	76.80	68.50	69.75
μ=1.5	51.01	91.81	64.02	77.40	70.25	70.90
μ=1.8	53.54	90.75	62.80	75.40	68.50	70.20
μ=2.0	46.46	91.21	61.59	75.80	68.75	68.76

Table 16: **Protection threshold sweep** at the default budget $\beta = 0.20$, Qwen3-30B-A3B, $B = 64$, five-task accuracy. The shaded row is the default setting.

percentile	Qwen3-30B-A3B	Qwen3.6-35B-A3B	Gemma-4	GLM-4.5-Air
p5	1.31	1.30	1.25	1.28
p10	1.38	1.39	1.32	1.39
p20	1.48	1.51	1.42	1.52
p25	1.52	1.56	1.47	1.58
p50	1.73	1.82	1.74	1.86
p75	2.03	2.18	2.15	2.23
p90	2.40	2.66	2.79	2.73
p95	2.69	3.06	3.39	3.16

Table 17: **Which μ sits at each percentile** of a model's top-1 share $(g_1/\sum_j g_j)\,k$. Setting μ to a model's p20 value protects the 80% of tokens whose top-1 is at least that decisive. All four models use $k = 8$.

Without local protection, increasing β causes a pronounced accuracy collapse: the five-task macro average decreases from 69.91 at $\beta = 0.05$ to 57.56 at $\beta = 0.20$ and 25.20 at $\beta = 0.30$. With $\mu = 1.5$, accuracy remains substantially more stable, ranging from 70.23 to 65.57 over the same budget range. Thus, local protection becomes increasingly important as the masking budget grows.

At the default $\beta = 0.20$, varying μ from 1.2 to 2.0 changes the Qwen3 macro average by only 2.14 points (Table 16), from 68.76 to 70.90. A repeated $\mu = 1.5$ run differs by 2.25 points, suggesting

that differences of this scale should not be over-interpreted. We therefore select $\mu = 1.5$ on Qwen3-30B-A3B and apply it unchanged to the other three models.

Table 17 interprets μ in terms of the distribution of top-1 routing shares. A token's top-1 share, expressed in units of a uniform slot, is $(g_1/\sum_j g_j)k$; thus, μ acts as a quantile knob rather than an absolute weight. For example, setting μ to a model's p20 value admits the 80% of tokens whose top-1 assignment is at least that decisive. The four routers agree closely around this decision region: p5 spans 1.25–1.31 and p20 spans 1.42–1.52, while p95 spans 2.69–3.39. Hence, a common μ transfers because the relevant decision boundary is similar across models, despite differences in the distribution tails. The shared setting $\mu = 1.5$ lies near p20 on Qwen3-30B-A3B, Qwen3.6-35B-A3B, and GLM-4.5-Air and just above p25 on Gemma-4.

D ANALYSIS

D.1 WHY ROUTER GATE MASS

An MoE layer writes $y = \sum_{e \in \mathcal{E}(x)} g_e(x) E_e(x)$ into the residual stream (Eq. 4). Masking expert e for token t removes

$$c_e(t) = \|g_e(t) E_e(h_t)\| = g_e(t) \|E_e(h_t)\|, \tag{8}$$

where the second equality follows from the non-negative gate weight. Thus, the batch contribution magnitude $C_e = \sum_t c_e(t)$ depends jointly on routing strength and expert-output magnitude.

However, C_e is unavailable before expert execution because $\|E_e(h_t)\|$ is unknown. GLoRE therefore uses aggregate gate mass G_e (Section 3.2) as an execution-free proxy. We evaluate whether it supports GLoRE's two decisions: global expert ranking and local protection.

Setup. We instrument Qwen3-30B-A3B and record the applied gate weight $g_e(t)$ and pre-gate expert output norm $\|E_e(h_t)\|$ for every routed slot. Statistics are computed within each layer and reported as the median with the layer-wise range, avoiding confounding from the approximately $14\times$ depth-wise growth in output norms.

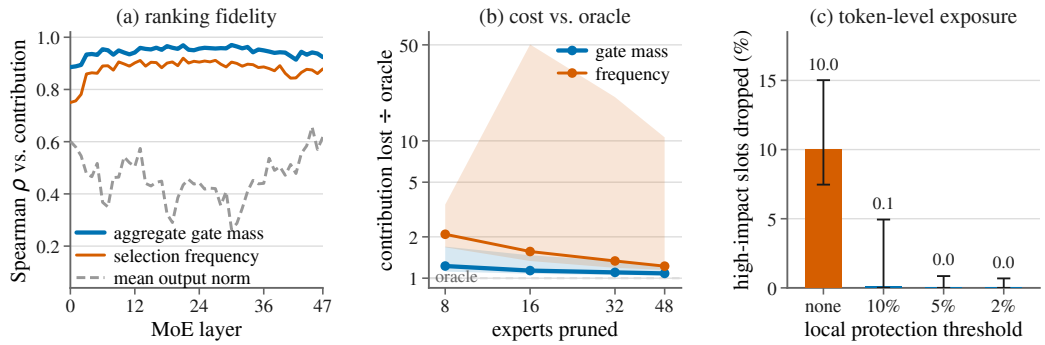

Figure 11: **Router gate mass as a runtime importance signal**, Qwen3-30B-A3B, 48 MoE layers. **(a)** Spearman correlation of each signal with realized contribution C_e (Eq. 8). **(b)** Contribution lost by removing a fixed number of experts, relative to an oracle that removes the smallest-C_e experts. **(c)** High-impact slots dropped at $\beta = 0.20$, without and with local protection. Lines are medians; shading and whiskers give the layer-wise range.

Gate mass ranks experts by realized contribution. We compare aggregate gate mass with realized contribution C_e, selection frequency, and mean output norm. Gate mass achieves a median Spearman correlation of 0.950 with C_e, compared with 0.890 for selection frequency and 0.454 for mean output norm (Figure 11a).

For a fixed number of removed experts, we compare each ranking with an oracle that removes the smallest-C_e experts. At $\beta = 0.20$, gate-mass masking removes 18.1% of contribution mass (12.5–19.4% across layers) at $1.08\times$ the oracle cost, compared with $1.22\times$ for frequency at the median and $10.6\times$ on the worst layer. At smaller prune counts, gate mass remains within $1.23\times$ of the oracle, whereas frequency reaches $50\times$ (Figure 11b).

A global ranking alone leaves individual tokens exposed. We call a routed slot *high-impact* when it carries at least 20% of its token's realized contribution. At $\beta = 0.20$, a purely global mask drops 10.0% of high-impact slots, while the most exposed tokens lose 68% of their contribution mass (62–84% across layers). This reproduces the batch–token tension of Section 2 using realized contributions.

The router signal provides only limited separation: firings on the most exposed tokens have a median gate weight $1.24\times$ that of other firings (1.09–$1.64\times$), with a lift above 1 for 83% of rare experts. This motivates token-level protection rather than relying solely on the global threshold.

Gate weight provides an effective local protection signal. GLoRE protects masked top-1 assignments using their gate weights. Against realized contribution share, the protection score achieves a median Spearman correlation of 0.927 (0.784–0.947), compared with 0.898 (0.758–0.936) for gate weight alone. Protection reduces the fraction of high-impact slots dropped from 10.0% to 0.0% at the median and at most 0.9% across layers (Figure 11c).

Together, these results support aggregate gate mass for global expert ranking and token-level gate weights for protecting important local assignments.

D.2 Token Exposure Under a Global Mask

Section 2 summarizes the batch–token disparity using the exposure ratio R_B (Eq. 2). Figure 12 provides a layer-averaged view for Qwen3-30B-A3B. We apply a cheapest-first global mask with $\beta = 0.20$ and no local protection to the routing capture. The line reports the mean R_B across the 48 MoE layers, while the shaded region shows the layer-wise minimum–maximum range.

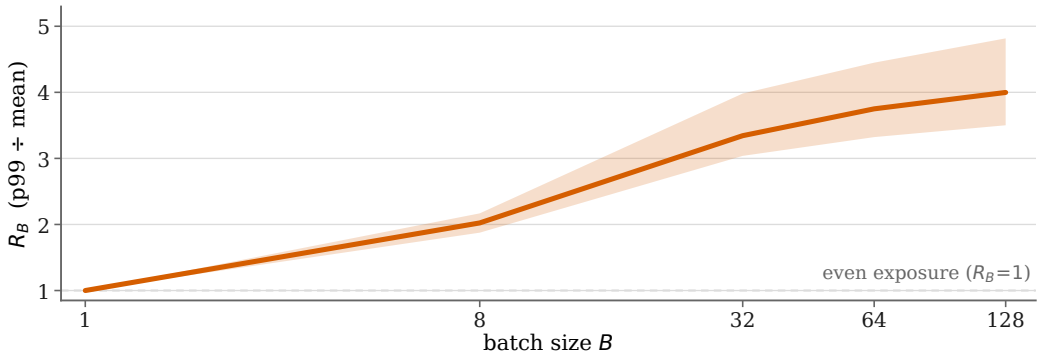

Figure 12: **Token exposure under global masking.** Layer-averaged exposure ratio R_B (Eq. 2) for Qwen3-30B-A3B under a cheapest-first global mask with $\beta = 0.20$ and no local protection. The line shows the mean across 48 MoE layers; the shaded region shows the layer-wise minimum–maximum range.

The layer-averaged trend shows that the disparity grows rapidly with batch size: R_B increases from 1.0 at $B = 1$ to 2.0, 3.3, 3.8, and 4.0 at $B = 8, 32, 64$, and 128, respectively. Most of the increase occurs by $B = 32$. The layer-wise range indicates that this behavior is observed across the model rather than being driven by a single layer. Thus, as routing diversity grows with batch size, a fixed global masking budget can expose some tokens to substantially more removal than others, motivating explicit token-level protection in addition to batch-level masking.

D.3 Dropping Versus Substituting a Masked Expert

Section 3.3 drops masked assignments rather than rerouting them to surviving experts. We test this choice by comparing the reconstruction error of dropping expert A with substituting expert B. For a masked slot with gate weight g, dropping incurs error $g\|E_A(h_t)\|$, whereas substitution incurs $g\|E_A(h_t) - E_B(h_t)\|$. Thus, substitution helps only when

$$\cos\theta > \frac{\|E_B(h_t)\|}{2\|E_A(h_t)\|}, \tag{9}$$

which requires $\cos\theta > 0.5$ at comparable norms. We measure whether real substitutes satisfy this condition by force-forwarding every expert on sampled tokens and selecting the best surviving expert by cosine similarity.

Figure 13 reports results on nine MoE layers of Qwen3-30B-A3B, with 1,794–2,107 masked slots per layer and a median substitute-to-removed norm ratio n_B/n_A of 0.61–0.96. The best-survivor cosine is an oracle: identifying the substitute requires expert outputs that masking seeks to avoid, so deployable rules based on precomputed similarity (Wu et al., 2026a) or calibrated reconstruction (Wu et al., 2026b) can only perform worse on this objective. Even under this upper bound, the median best-survivor cosine is at most 0.013 from layers 6–42, with the interquartile range straddling

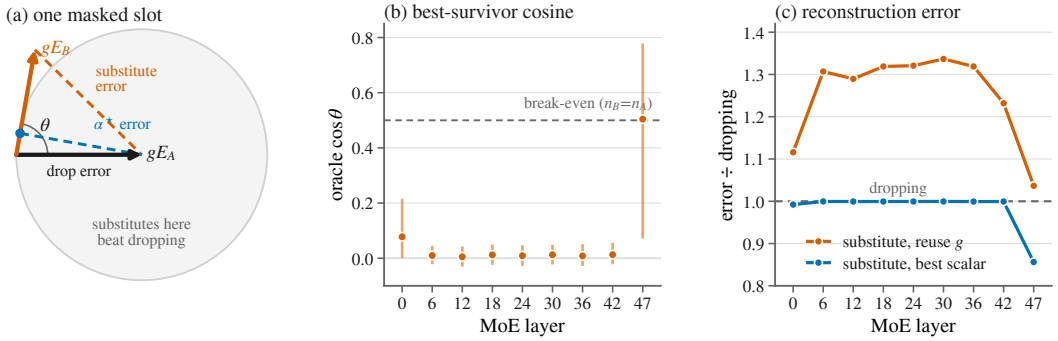

Figure 13: **Dropping versus substituting a masked expert**, Qwen3-30B-A3B. (a) Schematic of one masked slot (not data): a substitute beats dropping only inside the shaded disc (Eq. 9); drawn at $\theta = 80°$ and the measured norm ratio 0.85. (b) Median cosine between the removed output and its best surviving substitute (an oracle), with the p10–p90 range; dashed: break-even at equal norms. (c) Median error relative to dropping, reusing g and with the optimal scalar $\alpha^\star$; values above 1 are worse than dropping.

zero. Reusing the masked slot's gate weight is therefore 1.23–1.34× worse than dropping and improves on dropping for none of the masked slots in these layers.

Allowing an optimal scalar rescaling provides little additional benefit. For substitute norm n_B, removed-expert norm n_A, and cosine $\cos\theta$, the per-slot optimal coefficient is

$$\alpha^\star = g\,\frac{n_A \cos\theta}{n_B},$$

leaving error $g\,n_A\sqrt{1-\cos^2\theta}$. This oracle rescaling still leaves error at 0.999× the drop error from layers 6–42, because near-orthogonal substitutes have an optimal coefficient near zero. The first and last MoE layers differ: at layer 47, the oracle cosine reaches 0.50 and substitution improves on dropping for half of the masked slots.

These results should be interpreted as a per-slot reconstruction analysis rather than an end-task evaluation. It considers one-to-one substitution and one model under prefill; methods that coalesce multiple excluded routes onto a shared expert may obtain different residuals (Wu et al., 2026b). Thus, the measurement does not rule out substitution in general.

It instead shows that, under batch-level masking on Qwen3-30B-A3B, rerouting a masked assignment to a single surviving expert provides little recovery of the removed contribution, even with oracle target selection and scalar rescaling. Dropping therefore avoids the additional calibration corpus or similarity matrix required by such substitution schemes.

E LONG-COT TEST-TIME SCALING

E.1 EVALUATION PROTOCOL

We evaluate Qwen3-30B-A3B-Thinking on AIME 2024+2025 (60 problems) with thinking budgets of 4K, 8K, 16K, and 32K tokens at batch size 16. For each budget, we sample $k = 8$ chains per problem at temperature 0.6 (top-p 0.95, top-k 20) and use majority voting for evaluation. The thinking budget is the only swept scaling variable; k is fixed to reduce the variance of the accuracy estimate. Cost is measured as GPU-seconds per problem: each arm's problems are split across two concurrently running shards, and we sum each shard's wall-clock time multiplied by its tensor-parallel degree and divide by the number of problems. GPQA-Diamond (198 questions) follows the same protocol at budgets of 2K, 4K, 8K, and 16K. Its answers are extracted leniently: the simple-evals `Answer: X` pattern when present, and otherwise the first answer line after the reasoning in common forms such as `$C$`, `\boxed{C}`, or `**C**`; vanilla scores are unchanged by this rule. Confidence intervals are paired bootstrap 95% intervals over problems.

E.2 PER-BUDGET RESULTS

		GPU-s per problem			maj@8 (%)		
Task	Budget	Vanilla	GLoRE	speedup	Vanilla	GLoRE	Δ [95% CI]
AIME	4K	53.1	48.1	1.10×	51.67	55.00	+3.3 [−5.0, +11.7]
	8K	94.6	81.4	1.16×	71.67	73.33	+1.7 [−3.3, +8.3]
	16K	157.0	132.3	1.19×	81.67	85.00	+3.3 [−3.3, +10.0]
	32K	211.5	176.5	1.20×	88.33	88.33	0.0 [−5.0, +5.0]
GPQA	2K	18.8	18.0	1.05×	56.57	60.61	+4.0 [−1.0, +9.1]
	4K	34.8	30.8	1.13×	62.63	64.65	+2.0 [−2.5, +6.6]
	8K	56.8	47.7	1.19×	67.68	71.21	+3.5 [−1.0, +8.6]
	16K	73.4	62.0	1.18×	72.73	72.22	−0.5 [−4.5, +3.5]

Table 18: **Long-CoT test-time scaling results** with $k = 8$ majority voting. GPU-s is GPU-seconds per problem (Appendix E.1); Δ is GLoRE minus vanilla maj@8 with its paired 95% CI.

		avg@8 (%)			pass@8 (%)		
Task	Budget	Vanilla	GLoRE	Δ [95% CI]	Vanilla	GLoRE	Δ [95% CI]
AIME	4K	44.79	33.54	−11.3 [−18.1, −5.0]	65.00	58.33	−6.7 [−16.7, +1.7]
	8K	59.17	53.54	−5.6 [−11.3, −0.4]	78.33	78.33	0.0 [−6.7, +6.7]
	16K	71.88	68.54	−3.3 [−8.3, +1.7]	91.67	88.33	−3.3 [−8.3, 0.0]
	32K	85.42	82.08	−3.3 [−6.5, −0.2]	90.00	90.00	0.0 [0.0, 0.0]
GPQA	2K	50.63	49.12	−1.5 [−5.4, +2.3]	67.17	70.20	+3.0 [−1.5, +7.6]
	4K	55.74	57.51	+1.8 [−1.3, +4.9]	75.25	76.26	+1.0 [−3.0, +5.1]
	8K	65.28	65.59	+0.3 [−2.2, +2.7]	82.32	82.32	0.0 [−4.0, +4.0]
	16K	70.45	70.14	−0.3 [−2.5, +1.9]	83.84	86.36	+2.5 [−1.5, +6.6]

Table 19: **Per-sample (avg@8) and coverage (pass@8) accuracy** for the runs of Table 18. avg@8 is the mean accuracy of the eight samples; pass@8 counts a problem as solved if any sample is correct. Δ is GLoRE minus vanilla with its paired 95% CI.

Table 18 shows that GLoRE uses 1.10–$1.20\times$ fewer GPU-seconds per problem on AIME and 1.05–$1.19\times$ fewer on GPQA-Diamond while maintaining statistically comparable maj@8 accuracy: every paired 95% confidence interval spans zero. On GPQA-Diamond, repeated GLoRE runs at 16K varied by 2.5 points in maj@8, the same scale as the differences above, and at 2K about 85% of samples in both arms reach the budget, so that rung mostly reflects budget forcing and gives the smallest saving ($1.05\times$).

Table 19 shows that the comparable maj@8 accuracy on AIME comes from voting over samples rather than from unchanged per-sample accuracy: GLoRE's avg@8 is lower at every budget, significantly so at 4K, 8K, and 32K (-11.3, -5.6, and -3.3 points), while pass@8 is unchanged within its confidence intervals. On GPQA-Diamond, avg@8 and pass@8 stay within ± 3 points of vanilla at every budget, with no significant difference.

On AIME, interpolating the vanilla accuracy–compute curve suggests equal-cost gains of $+4.1$–$+6.9$ percentage points, but a slope-based analysis does not yield a statistically significant gain at any budget. We therefore report the more conservative result: GLoRE reduces the compute required to obtain comparable majority-vote accuracy.

The measured reduction is an end-to-end cost result based on sampling-pool wall-clock time, so it reflects both execution efficiency and generated sequence length. Finally, due to resource constraints, we evaluate long-CoT scaling at batch size 16, where GLoRE removes fewer experts than at larger serving batches. Thus, the observed 1.10–$1.20\times$ GPU-second reduction is likely conservative relative to our larger-batch serving results.

F FUSED CUDA KERNEL

GLoRE uses one fused CUDA kernel per masked MoE layer, `routing_mass_step_fused`, together with `masked_moe_sum`, which replaces vLLM's stale-slot clear and top-k reduction. The pseudocode below illustrates the kernel's logic; identifier names follow our implementation. GLoRE exposes two hyperparameters, β and μ, set to 0.20 and 1.5 in our experiments. The source also supports a prune-count cap, mass floor, and batch guard, but all are disabled in our experiments and omitted from the pseudocode (`min_active_frac`$=0$, `mass_floor`$=0$, `guard_top1`=FALSE).

F.1 LAUNCH AND PRECONDITIONS

The kernel is launched on a *single block* with `threads` $= 1024$ if $Bk \geq 8192$ and 512 otherwise, raised to 1024 when it would fall below N; dynamic shared memory is $(N{+}1)$ floats for the mass array plus $(N{+}1)$ bytes for the guard flags. The host asserts contiguous $[B, k]$ `topk_ids` (`int32` or `int64`) and `topk_weights` (`float32`, written in place), a contiguous `bool skip_mask` of $N{+}1$ elements, $0 < N \leq 1024$ — the single-block selection assumes one thread per expert. If $B = 0$ or $k = 0$ the mask is zeroed and the kernel returns, so a stale mask cannot leak into the step.

Phase A — build the mask (`rms_build_mask_block`, whole block)

---

per-expert gate mass for this step, in shared memory

1: $\texttt{s_mass}[e] \leftarrow 0$ for $e = 0..N$; **sync**

2: **for** each slot $i = 0..Bk-1$ strided by `blockDim`:
$e \leftarrow \text{clamp}(\texttt{ids}[i], 0, N)$; $\text{atomicAdd}(\texttt{s_mass}[e],\ \texttt{w}[i])$; **sync**

3: $\texttt{total} \leftarrow \max\big(\sum_{e<N} \texttt{s_mass}[e],\ 10^{-9}\big)$ (warp-shuffle reduction; the sentinel row N is excluded)

one thread per expert: rank by mass, then apply the budget

4: **for** each expert $e < N$ (one thread per expert):

$\texttt{take} \leftarrow (\texttt{s_mass}[e] > 0)$ (experts untouched this step are never masked)

if `take`:

$S \leftarrow \big\{ f < N : \texttt{s_mass}[f] > 0,$
$\quad (\texttt{s_mass}[f] < \texttt{s_mass}[e]) \vee (\texttt{s_mass}[f] = \texttt{s_mass}[e] \wedge f < e) \big\}$

$\texttt{cum} \leftarrow \sum_{f \in S} \texttt{s_mass}[f]$

$\texttt{take} \leftarrow (\texttt{cum} + \texttt{s_mass}[e])/\texttt{total} \leq \beta$

$\texttt{skip_mask}[e] \leftarrow \texttt{take}$

5: $\texttt{skip_mask}[N] \leftarrow$ FALSE (sentinel stays unmasked so dropped slots never re-trigger); **sync**

Algorithm 1: Mask construction. Step 4 ranks each expert by counting cheaper experts rather than sorting: N comparisons per thread across N threads, which keeps the build single-block and makes the build→apply barrier a `__syncthreads()`. The tie-break $f < e$ makes the prefix deterministic under equal masses.

Phase B — apply to one token row (`rms_apply_row`, one thread per token, grid-strided)

1: `base` $\leftarrow bk$; in one pre-write pass: $j^\star \leftarrow \arg\max_j$ `w`$[\texttt{base}+j]$ (first max wins), `row_sum` $\leftarrow \sum_j$ `w`$[\texttt{base}+j]$
scale-free top-1 protection, written as a product so a zero row sum cannot divide
2: `decisive` $\leftarrow (\mu > 0) \wedge (\texttt{w}[\texttt{base}+j^\star] > 0) \wedge \big(\texttt{w}[\texttt{base}+j^\star] \cdot k \;\geq\; \mu \cdot \texttt{row_sum}\big)$
optional mass floor (inert when `mass_floor` = 0): protect heaviest-first
3: **if** `mass_floor` $> 0 \wedge k \leq 64$: seed `rescued` $\leftarrow \{j^\star\}$ if `decisive`;
while `kept` $<$ `floor_eff` $\cdot$ `total`: protect the heaviest still-masked slot, or **break** if none remains
write pass: drop to the sentinel, never reroute
4: **for** $j = 0..k-1$: $e \leftarrow \text{clamp}(\texttt{ids}[\texttt{base}+j], 0, N)$;
`rescue` $\leftarrow$ (`decisive` $\wedge\, j = j^\star$) $\vee$ ($j \in$ `rescued`)
if `skip_mask`$[e] \wedge \neg$ `rescue`: `risk_dropped` += `w`$[\texttt{base}+j]$; `ids`$[\texttt{base}+j] \leftarrow \varnothing$; `w`$[\texttt{base}+j] \leftarrow 0$
5: reduce (`risk_dropped`, `row_sum`) by warp shuffle, then across warps, then atomicAdd into `risk_accum`$[2]$

Algorithm 2: Per-token application. One thread owns a whole row, not a slot: the share test in step 2 reads the row's maximum and total gate mass, which the same kernel may zero in step 4, so a thread-per-slot mapping would race one slot's write against another's read of the same row. Step 2 is the implemented form of the local rule — a *ratio*, $\texttt{w}_1/\texttt{row_sum} \geq \mu/k$, which needs no assumption that the row sums to one. That matters across models: DeepSeek-V2-Lite routes with `norm_topk_prob`=FALSE and its rows sum to 0.391, where an absolute threshold would fire on a wholly different share of tokens than on Qwen3.

F.2 Downstream Reduction

Dropped slots carry the sentinel id at weight 0, so the expert-output cache holds stale rows at those positions. Stock vLLM clears them and then sums over k; `masked_moe_sum` fuses the pair, summing only live slots:

$$\texttt{output}[t,:] \;=\; \sum_k \big[\texttt{topk_ids}[t,k] \in [0,N)\big] \cdot \texttt{cache}[t,k,:], \tag{10}$$

with one block per token, threads striding the hidden dimension, the per-slot live flags computed once into shared memory, and accumulation in `float32` regardless of the cache dtype. Because the cache already carries $g_k E_k(x)$, this matches the clear-then-sum result for every kept slot up to `float32` accumulation order, and it removes the separate clear launch and its writes.

F.3 Load-Bearing Implementation Details

First, the shared-memory pointers in Phase A carry no `__restrict__` qualifier. With it, `nvcc` kept stale values across the `__syncthreads()` barriers and silently produced an all-FALSE mask. Second, build and apply are fused into a dedicated kernel rather than chaining the two standalone ones, which removes one kernel dispatch per masked layer.

G LIMITATIONS AND FUTURE WORK

Where GLoRE helps. GLoRE is most beneficial when decode is memory-bound and the batch is large enough for the expert union to approach the full pool. Its throughput gain therefore increases with serving concurrency (Section 4.4) and is small at low concurrency. It provides little benefit when prefill dominates: prefill is more compute-bound and already accesses many experts, with GLoRE achieving 0.99–$1.04\times$ vanilla throughput on the prefill-dominated workload (Appendix B.3).

Batch adaptation and composition. Although the number of experts masked by GLoRE adapts to the routing distribution and batch size, the global mass budget β itself is fixed. The same β may therefore be overly conservative for some batches and overly aggressive for others. Moreover, because the mask is computed from the whole batch, a token's masking decisions depend on the routing distribution of the other requests, so the same request can encounter different expert masks under different batch compositions, even with greedy decoding. Local protection limits the effect on the token's most decisive assignment but does not eliminate this dependence. Adaptive budgets or batch-invariant masking could improve robustness across workloads and serving conditions.

Local protection. GLoRE's local protection rule does not guarantee that every important token-level assignment is preserved. It protects only masked top-1 assignments whose gate weight exceeds μ/k, so important non-top-1 assignments or assignments with weaker gate weights may still be dropped. More fundamentally, GLoRE uses router gate weight as an execution-free proxy and therefore does not directly know which masked expert contributes most to the token's final output. A more accurate importance estimator could provide stronger protection, at the cost of additional computation or profiling.

Scope of the evaluation. Our evaluation uses a single node of $8\times$ H100 GPUs with tensor parallelism, four MoE models, and batch sizes up to 64. Expert-parallel and multi-node deployments and much larger batches are not evaluated. The long-CoT study is limited to a small set of reasoning evaluations and uses batch size 16, while its GPU-second measure reflects both execution time and changes in generation length.

Future directions. Several directions follow from these limitations. First, the global masking budget could adapt across layers or decoding steps based on batch size, routing concentration, or observed quality signals, rather than using a fixed β. Second, stronger execution-free importance signals could improve local protection by identifying important assignments beyond top-1 gate weight. Third, the drop-versus-substitute decision could be made layer-aware: substitution provides little benefit in most interior layers but can help near the network boundary (Appendix D.3). Finally, combining GLoRE with expert-parallel load balancing or offline expert compression, and evaluating it on larger batches, multi-node deployments, and larger-batch test-time scaling, would clarify its applicability beyond the current setting.