BLAZE: BIAS-DRIVEN LOAD-AWARE ZERO-OVERHEAD EXPERT ROUTING FOR MOE LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Expert-parallel Mixture-of-Experts (MoE) inference is usually bottlenecked by routing-induced load imbalance: overloaded experts and EP ranks create stragglers in All-to-All communication and grouped-GEMM execution. Existing placement-level approaches mitigate persistent hotspots by replicating expert weights, but this consumes GPU memory and does not directly correct transient router imbalance at inference time. We present BLAZE, a token-drop-free and replica-free router-side controller for expert-parallel MoE serving. BLAZE redirects only confidence-ambiguous expert choices away from overloaded experts or EP ranks, while locking high-confidence semantic decisions and computing mixture weights from the original router scores. To make redirection deployable in end-to-end serving, BLAZE further uses an overhead-aware activation gate that bypasses the controller on small effective MoE batches where redirection overhead is unlikely to pay off. Across GPT-OSS-20B and Qwen3-30B-A3B, BLAZE stays within 1.6 percentage points of vanilla accuracy, while random or worst-ranked replacement collapses accuracy by 44–49 percentage points; on GPT-OSS-120B, three complete MLPerf accuracy runs of BLAZE all pass the official threshold with no loss relative to vanilla. On DeepSeek-R1, the overhead-aware gate avoids a 16% TPOT regression from always-on redirection. Under MLPerf-style serving with TP = EP = 8 at SLO-feasible load, BLAZE turns a vanilla TTFT violation on GPT-OSS-20B into an SLO pass (26.7% lower TTFT P99) and, in a matched QPS sweep, still meets the latency SLO at 1.75 QPS, whereas capacity-aware routing fails from 1.25 QPS; on Qwen3-30B-A3B, all methods meet the SLO and BLAZE delivers the highest completed-token throughput. BLAZE also improves offline throughput by 8.9% on GPT-OSS-120B.

1 INTRODUCTION

Mixture-of-Experts (MoE) language models (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022) scale model capacity by routing each input token to only a small number of "expert" subnetworks. This sparse activation makes large language models (LLMs) practical to serve, but it also makes the learned router a systems-critical component. Under expert parallelism (EP), experts are partitioned across graphics processing units (GPUs), and each MoE layer must dispatch tokens to the GPU ranks hosting the selected experts and combine the outputs through collective communication. Because these collectives synchronize across ranks, their latency is governed by the slowest participant. Overloaded experts therefore create stragglers that inflate end-to-end tail latency.

Existing systems often address persistent routing skew through placement-level mechanisms such as Expert Parallelism Load Balancing (EPLB), which uses observed routing statistics to redistribute experts and may replicate hot experts across devices (DeepSeek-AI, 2024; vLLM Contributors, 2026). Redirecting work to an exact replica is functionally safe because the replica has identical weights, leaving the expert computation unchanged. However, placement-level balancing changes model residency rather than directly modifying the router's decisions for the current batch. Replication consumes additional GPU memory, while redistributing or replicating experts requires the expert placement to be updated. These mechanisms are therefore best suited to persistent hotspots and do not directly correct transient, batch-specific imbalance among the original experts.

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.

This paper asks whether expert replication is necessary for inference-time load balancing. Modern MoE training recipes often use auxiliary load-balancing losses, router-bias updates, or related mechanisms to encourage broader expert utilization during pretraining. This reduces the likelihood that non-top experts are merely under-trained backups: many experts may have received sufficient training signal to serve as semantically meaningful alternatives for some tokens. This observation motivates a hypothesis: when the router assigns comparable confidence to multiple experts, can a non-top-ranked expert replace an overloaded top choice without degrading model quality? If this hypothesis holds, the serving system can alleviate expert imbalance without replicating expert weights. Instead of copying hot experts to other GPUs, the router can redirect ambiguous tokens from overloaded experts to less-loaded router-ranked alternatives.

This opportunity raises two challenges. The first challenge is semantic safety. Unlike replica-based EPLB, redirecting a token to a different expert changes the model function. A redirection mechanism must therefore be selective: high-confidence semantic choices should remain fixed, while only low-confidence choices or fallback slots are exposed to load feedback. In addition, mixture weights should still be computed from the original semantic router scores, ensuring that load feedback changes which experts are selected without changing how selected experts are blended. A second challenge is control-path amortization. BLAZE is an active router-side mechanism: it executes extra score adjustment, routing metadata generation, load-count accumulation, and feedback updates. These costs are small, but they are not free. In small effective-batch regimes, there may be too little imbalance-driven work to save, so executing the redirection path can cost more than it recovers. Thus, a practical router-side controller must determine, at runtime, when the current MoE invocation is large enough for BLAZE's redirection path to pay off and when the controller should bypass itself and return to vanilla routing.

We present BLAZE, a bias-driven, low-overhead, replica-free inference-time router for expert-parallel MoE serving. BLAZE treats routing as a constrained closed-loop control problem: it observes expert and rank load pressure and applies lightweight proportional feedback to router scores. The controller is confidence-aware, locking high-margin top choices while redirecting ambiguous choices toward less-loaded next-ranked experts. It is also overhead-aware as an end-to-end serving policy: it enables redirection only when the current MoE invocation has a sufficiently large effective batch to amortize BLAZE's router-side control path and the communication effects it can introduce. This naturally favors prefill and chunked prefill, where many prompt-token representations enter each MoE layer together; for small decode batches, BLAZE falls back to original routing to avoid TPOT regressions. The individual signals that BLAZE uses, router confidence and load, are natural in hindsight; the nontrivial question is *which* assignments can be changed without destroying model quality, and how to make that rule cheap enough for a graph-captured continuous-serving engine. We show that the answer is a sharp boundary rather than a smooth trade-off: arbitrary replacement loses 44–49 percentage points of accuracy, and even preserving the top-1 expert while randomizing the fallback slots loses 24–34 points. BLAZE stays inside this boundary only because three choices are coupled: high-margin top-1 decisions are locked, only ambiguous assignments are exposed to load feedback, and mixture weights are computed from the original semantic scores rather than the penalized selection scores. This paper makes the following contributions:

- **Replica-free expert load balancing.** We show that inference-time MoE load balancing need not rely on expert replication. BLAZE redirects confidence-ambiguous tokens to less-loaded experts, requiring no retraining, token dropping, expert migration, or duplicated expert weights.
- **Confidence-aware semantic redirection.** We characterize the accuracy boundary of expert substitution and use it to design BLAZE's confidence-aware routing policy, which stays within 1.6 percentage points of vanilla accuracy on GPT-OSS-20B and Qwen3-30B-A3B and shows no observed accuracy loss on GPT-OSS-120B under the full MLPerf accuracy protocol.
- **Overhead-aware activation for end-to-end serving.** We derive an activation gate tailored to BLAZE's router-side overhead: it enables redirection only when the current MoE invocation has a sufficiently large effective batch to amortize the extra control path and communication effects, a condition commonly met in prefill and chunked prefill. For small decode batches, the gate falls back to vanilla routing, avoiding a 16% TPOT regression from always-on redirection on DeepSeek-R1. In MLPerf-style serving at SLO-feasible load, BLAZE converts a GPT-OSS-20B TTFT violation into an SLO pass, sustains the latency SLO at higher load than capacity-aware routing, and matches or exceeds native EPLB throughput on Qwen3-30B-A3B without replicating any expert weights.

2 METHOD

The design is driven by two observations. First, expert identity is semantic: arbitrary expert replacement can severely degrade accuracy. Thus, BLAZE must preserve high-confidence routing decisions and redirect only choices that are semantically ambiguous. Second, the BLAZE router path must pay for itself. At small effective batches, the extra redirection logic, metadata movement, load accounting, and communication effects can outweigh the straggler reduction it creates. Thus, BLAZE must also decide when to bypass itself. Figure 1 summarizes the router. For each MoE layer, BLAZE first checks an overhead-aware activation gate. If the current effective token count is below the activation threshold, BLAZE bypasses all redirection and calls the original router. Otherwise, it forms a load-adjusted routing score from three terms: the original semantic score, an expert-level load penalty, and an EP-rank-level load penalty. A confidence-aware rule locks high-margin top-1 choices, while low-margin top-1 choices and fallback slots are selected using the load-adjusted score. Mixture weights are always computed from the original semantic scores, not the load-adjusted scores.

2.1 ACCURACY-PRESERVING EXPERT REDIRECTION

Let $s_{\ell,e}(x)$ denote the original semantic router score of token x for expert e at MoE layer ℓ. This score includes the model's native routing bias when such a bias exists. Vanilla routing selects the Top-K experts under $s_{\ell,e}(x)$ and computes mixture weights from the same scores. BLAZE changes only the expert-selection rule; it does not change the expert outputs or the mixture-weight computation.

The key accuracy guardrail is a margin test on the original semantic scores. Let $e_1(x)$ and $e_2(x)$ be the top-1 and top-2 experts under $s_{\ell,e}(x)$, and define the semantic margin

$$\Delta_\ell(x) = s_{\ell,e_1}(x) - s_{\ell,e_2}(x). \quad (1)$$

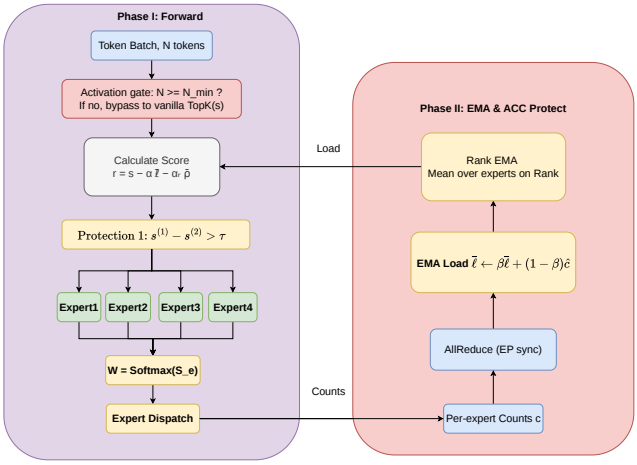

Figure 1: Illustration of BLAZE router-side mechanism.

If $\Delta_\ell(x) > \tau$, BLAZE treats the top-1 decision as high-confidence and locks $e_1(x)$ into the selected expert set. If $\Delta_\ell(x) \leq \tau$, the top-1 position is considered ambiguous and may also be redirected by the load-aware router. In both cases, the remaining fallback slots are chosen by the feedback-controlled score described in §2.2. This rule reflects the empirical feasibility boundary established in §3.2: preserving only the top-1 expert is not sufficient when fallback experts are random, but preserving high-confidence top-1 choices while selecting fallback experts load-aware keeps routing inside an accuracy-safe redirection band.

Offline threshold calibration. Because $\Delta_\ell(x)$ is a router-logit margin, its absolute scale can vary across model families. BLAZE therefore treats τ as a calibrated serving parameter rather than as a hand-tuned constant. Given a small calibration trace and a target fraction p_{redir} of low-margin tokens to make eligible for top-1 redirection, we set $\tau_{\text{offline}} = Q_{p_{\text{redir}}}(\{\Delta_\ell(x)\}_{\text{calib}})$. Since the lock condition is $\Delta_\ell(x) > \tau$, this fixes the redirection budget before deployment and then runs with a scalar threshold in the CUDA routing kernel. What transfers across architectures is this calibration procedure, not a single threshold value: the same target quantile maps to $\tau \approx 0.78$ on GPT-OSS-20B but $\tau \approx 5.09$ on Qwen3-30B-A3B (Appendix E). We therefore recommend the online quantile tracker with $q = 0.90$ as the cross-model default, since it is the only strategy that improves accuracy on both evaluated models, and keep a fixed threshold as a throughput-oriented option that avoids online histogram updates. Unless otherwise noted, the serving rows in this paper use this fixed-threshold option; Table 1 quantifies its accuracy cost.

Selection–weight decoupling. Let $\mathcal{S}_\ell(x)$ be the selected expert set after redirection. BLAZE computes mixture weights from the original semantic scores:

$$w_{\ell,e}(x) = \frac{\exp(s_{\ell,e}(x))}{\sum_{j\in\mathcal{S}_\ell(x)} \exp(s_{\ell,j}(x))}, \qquad e \in \mathcal{S}_\ell(x). \tag{2}$$

For sigmoid-routed MoEs, BLAZE uses the model's native sigmoid-and-renormalize form instead of softmax. In all cases, load feedback affects which experts are selected, but not how selected experts are blended. Given the selected expert set, the output computation is algebraically the same as vanilla routing on that set.

2.2 LOAD- AND RANK-AWARE FEEDBACK

BLAZE formulates expert balancing as closed-loop negative feedback. At each step, it observes how many token assignments each expert receives, updates a moving average of recent expert load, and subtracts a proportional penalty from future routing scores.

For layer ℓ and expert e, let $c_{\ell,e}^{(t)}$ be the number of token assignments routed to expert e at step t, and normalize it by the layer mean, $\hat{c}_{\ell,e}^{(t)} = c_{\ell,e}^{(t)} / \big(\frac{1}{E}\sum_j c_{\ell,j}^{(t)} + \epsilon\big)$, so that it has mean approximately one independent of batch size. BLAZE maintains an exponential moving average of this load and subtracts it from the semantic score:

$$\bar{\ell}_{\ell,e}^{(t)} = \beta \bar{\ell}_{\ell,e}^{(t-1)} + (1-\beta)\hat{c}_{\ell,e}^{(t)}, \qquad r_{\ell,e}^{\text{expert}}(x) = s_{\ell,e}(x) - \alpha \bar{\ell}_{\ell,e}, \tag{3}$$

where an expert with $\bar{\ell}_{\ell,e} > 1$ has recently received more than its fair share of tokens and α is the proportional gain. This penalty steers future ambiguous tokens away from recently overloaded experts while leaving high-confidence semantic decisions protected by the margin lock.

Rank-aware feedback under expert parallelism. Expert-level balance alone is not sufficient under expert parallelism. The serving system synchronizes at the EP-rank level: a hot rank can delay All-to-All dispatch, All-to-All combine, and the subsequent grouped-GEMM MoE backend even when no single expert appears extreme in isolation. BLAZE therefore adds a second feedback term at EP-rank granularity.

Let $r(e)$ be the EP rank that owns expert e, and let $\mathcal{E}_r$ be the experts placed on rank r. BLAZE computes the rank load as the mean EMA load of the experts on that rank, $\bar{\rho}_{\ell,r} = |\mathcal{E}_r|^{-1} \sum_{e\in\mathcal{E}_r} \bar{\ell}_{\ell,e}$, and the final adjusted score used for redirection is

$$r_{\ell,e}(x) = s_{\ell,e}(x) - \alpha \bar{\ell}_{\ell,e} - \alpha_{\text{rank}} \bar{\rho}_{\ell,r(e)}. \tag{4}$$

The expert-level term discourages routing to hot experts; the rank-level term discourages routing to hot EP ranks. This hierarchical feedback targets the actual EP bottleneck: both local expert hotspots and rank-level stragglers.

The rank penalty is conservative by design. We use $\alpha_{\text{rank}} < \alpha$ so that rank-level feedback nudges ambiguous choices away from hot ranks without overwhelming the semantic router signal. For grouped routers such as DeepSeek-R1, BLAZE preserves the pretrained group-level routing prior: group selection remains semantic-only, and expert- and rank-aware feedback are applied only inside the selected groups during expert Top-K selection.

Global load aggregation. Each EP rank observes only a partial token stream. To keep the controller state consistent, BLAZE accumulates per-layer expert counts on device, packs the $L \times E$ count buffer, and performs one `AllReduce(SUM)` across the EP group before updating the EMA. This buffer contains counts, not activations or weights, so it is small and amortized across all MoE layers.

2.3 OVERHEAD-AWARE ACTIVATION AND END-TO-END ROUTER

The feedback controller should be active only when redirection can amortize the overhead of the BLAZE router path itself. At small effective batches, redirecting tokens may improve expert-count balance but also spread work across more experts and EP ranks, increasing All-to-All fanout, routing metadata movement, scheduling overhead, and grouped-GEMM fragmentation. In this regime, the control-path and communication overhead can dominate the straggler reduction, so executing the load-aware router is not necessarily a latency win.

BLAZE therefore uses an overhead-aware activation gate whose runtime proxy is the current MoE token count. Let N denote the effective token count entering a MoE layer. If $N < N_{\min}$, BLAZE returns the vanilla router output and bypasses the redirection path; otherwise, it enables the feedback-controlled router. This threshold is not a capacity limit and never drops tokens. It only decides whether the load-adjusted routing score is worth executing for the current MoE invocation.

We estimate $N_{\min}$ from the point at which the BLAZE router path can amortize its overhead. A MoE invocation with N tokens creates about NK expert assignments, so each of the E/EP experts on a rank expects $\bar{t} = NK/(E/\mathrm{EP})$ assignments. With B_M the backend grouped-GEMM M-tile size, we take one tile per local expert ($\bar{t} \geq B_M$) as the conservative amortization point. Because the practical crossover also depends on router skew, collectives, scheduling, and CUDA-Graph bucketization, we validate a small activation range, $N_{\min} = \mathrm{bucket_up}(O_{\mathrm{route}}\,(E/\mathrm{EP})/K)$ with $O_{\mathrm{route}} \in [0.25B_M, B_M]$, where bucket_up rounds upward to a system-friendly CUDA-Graph or scheduler bucket. The lower end corresponds to an aggressive quarter-tile activation point and the upper end corresponds to the conservative one-tile default. We use $B_M = 128$ as the grouped-GEMM tile proxy on the evaluated H200 backend, so the activation range is $O_{\mathrm{route}} \in [32, 128]$ assignments per local expert. The threshold is therefore not a semantic hyperparameter and not a capacity constraint; it is a serving-system knob that prevents BLAZE's own router overhead from being paid on small MoE calls where vanilla routing is faster.

Algorithm 1 BLAZE inference-time expert-redirection router

Require: Semantic router scores $s_{\ell,e}(x)$, token count N_ℓ, routing fanout K, expert-load EMA $\bar{\ell}_{\ell,e}$, rank-load estimate $\bar{\rho}_{\ell,r}$, gains $\alpha, \alpha_{\mathrm{rank}}$, confidence threshold τ, activation threshold $N_{\min}$

Ensure: Selected experts $\mathcal{S}_\ell(x)$ and mixture weights $w_\ell(x)$ for the unchanged MoE backend

if $N_\ell < N_{\min}$ **then**
  {True bypass: no redirection and no controller side effects}
  for each token x **do**
    $\mathcal{S}_\ell(x) \leftarrow \mathrm{TopK}_e\, s_{\ell,e}(x)$
    $w_\ell(x) \leftarrow \mathrm{Normalize}\big(s_{\ell,\mathcal{S}_\ell(x)}(x)\big)$
  end for
  return $\mathcal{S}_\ell(x), w_\ell(x)$
end if
Reset expert count buffer $C_{\ell,e} \leftarrow 0$
for each token x **do**
  for each candidate expert e **do**
    $r_{\ell,e}(x) \leftarrow s_{\ell,e}(x) - \alpha\bar{\ell}_{\ell,e} - \alpha_{\mathrm{rank}}\bar{\rho}_{\ell,r(e)}$
  end for
  $e_1 \leftarrow \arg\max_e s_{\ell,e}(x)$
  $e_2 \leftarrow$ second-largest expert under $s_{\ell,e}(x)$
  $\mathcal{S}_\ell(x) \leftarrow \emptyset$
  if $s_{\ell,e_1}(x) - s_{\ell,e_2}(x) > \tau$ **then**
    {Accuracy guardrail: preserve high-confidence top-1}
    $\mathcal{S}_\ell(x) \leftarrow \mathcal{S}_\ell(x) \cup \{e_1\}$
    Add the top $K-1$ experts under $r_{\ell,e}(x)$ excluding e_1 to $\mathcal{S}_\ell(x)$
  else
    {Low-confidence top-1 is also redirectable}
    $\mathcal{S}_\ell(x) \leftarrow \mathrm{TopK}_e\, r_{\ell,e}(x)$
  end if
  $w_\ell(x) \leftarrow \mathrm{Normalize}\big(s_{\ell,\mathcal{S}_\ell(x)}(x)\big)$
  {Mixture weights use semantic scores, not adjusted scores}
  for each selected expert $e \in \mathcal{S}_\ell(x)$ **do**
    $C_{\ell,e} \leftarrow C_{\ell,e} + 1$
  end for
end for
return selected experts $\mathcal{S}_\ell(x)$ and semantic mixture weights $w_\ell(x)$

End-to-end router. Algorithm 1 gives the full runtime path. It is executed for each MoE layer. The only persistent state is the per-layer expert-load EMA, the per-layer rank-load estimate, and the per-layer count buffer. Expert weights, expert placement, token capacity, and the MoE backend are unchanged. At the end of each step, BLAZE all-reduces the accumulated counts across the EP group, normalizes them, and applies Eq. (3) and the rank-load average; Appendix C describes how $N_{\min}$ is resolved at initialization. The serving backend sees the same interface as vanilla routing: selected expert IDs, mixture weights, and token assignments.

3 EXPERIMENTS

We evaluate BLAZE along three research questions. First, can expert redirection preserve task accuracy? Second, when does router-side load balancing become a net latency win rather than an extra scheduling and communication cost? Third, does the resulting controller improve end-to-end Server serving against practical baselines?

3.1 EXPERIMENTAL SETUP

Table 5 summarizes the main evaluation setup. Unless otherwise stated, experiments run on a single 8×H200-SXM node with tensor parallelism and expert parallelism both equal to 8. We use the TensorRT-LLM endpoint stack and NVIDIA's MLPerf Inference harness to generate MLPerf-format latency summaries. These are internal A/B measurements using the MLPerf Server load generator and latency accounting; they are not official MLPerf submissions because GPT-OSS-20B and Qwen3-30B-A3B are not MLPerf-registered benchmarks. We report completed samples/s, output tokens/s, time-to-first-token (TTFT), and time-per-output token (TPOT). The Server scenario uses the MLPerf token-latency service targets of TTFT P99 $\leq$ 3000 ms and TPOT P99 $\leq$ 80 ms. Server comparisons in the main text are run at SLO-feasible and boundary target QPS (bounded 300-request LoadGen cells with Poisson arrivals); saturated QPS = 3–4 runs are reported only as diagnostics in Appendix I. Appendix C.2 gives the request, dataset, and metric protocol.

3.2 E1: ACCURACY-SAFE EXPERT REDIRECTION

The central risk in inference-time expert balancing is that the router is not merely a load dispatcher: its Top-K decisions encode semantic information. We therefore evaluate whether routing decisions can be safely redirected on the full AIME25 + GPQA-Diamond accuracy set; Appendix H.1 defines the accuracy protocol and each routing condition used in this experiment. Table 17 and Figure 2 summarize the combined result. The appendix breakdown in Table 13 shows the same pattern within both task families. Random Top-k and Worst-k are especially destructive on AIME25, while the load-aware policies preserve most of the vanilla signal on both math and science questions. These results support the design choice in §2: BLAZE should not treat experts as interchangeable capacity slots. Instead, it only redirects through a load-aware, confidence-gated rule.

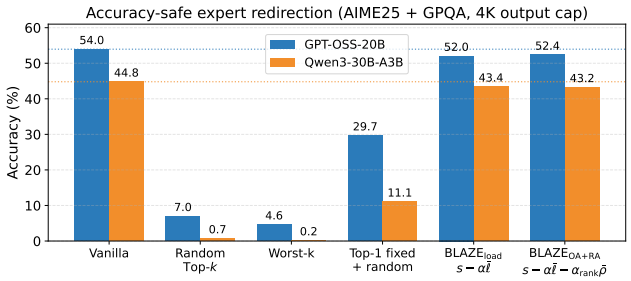

Figure 2: Accuracy-safe expert redirection on the full AIME25+GPQA-Diamond set with a 4096-token output cap. Random and worst-k replacement collapse accuracy, and preserving only the top-1 expert is insufficient. The full BLAZE remains within 1.6 pp of vanilla on both architectures. Dotted lines mark each model's vanilla accuracy.

How much routing changes, and what it costs. A validated device-side observer counts a token as *redirected* when at least one of its semantic Top-K experts changes. Table 1 varies only the lock threshold τ at fixed gains (α=0.15, α_{rank}=0.02, β=0.85) on all 1,230 AIME25+GPQA-Diamond prompts. Redirect volume is nearly constant across τ while accuracy is not: on Qwen3-30B-A3B, τ=5.0 loses 2.2 pp relative to τ=0.5 with an essentially unchanged redirected fraction. Because the bounded load penalty caps how many decisions can flip, raising τ draws the same number of redirections from higher-confidence decisions, which is where semantic cost enters. The aggregate redirect fraction is therefore not a reliable proxy for accuracy; what matters is which margins are unlocked. This is why τ should be set by quantile calibration, which pins the flip pool to each model's own low-confidence percentile, rather than by a single global constant (§2.1).

Accuracy at scale under the full MLPerf protocol. The redirect-feasibility study above is greedy and single-run. To separate run-to-run variation from systematic degradation, we also evaluate GPT-OSS-120B under the complete MLPerf v6.0 accuracy protocol (AIME25, GPQA-Diamond, and LiveCodeBench-V6; 4,395 scored generations; temperature 1.0; 32,768-token cap; unmodified official evaluator) with one vanilla and three complete BLAZE executions on 8×H200 with TP = EP = 8. Table 2 shows that every BLAZE execution passes the 82.30% MLPerf threshold, and the three-run mean is 0.47 pp above vanilla. The base-item-clustered 95% intervals of the paired differences all include zero, so we claim no accuracy improvement, only the absence of an observed loss. Appendix H.3 gives per-run results and additional request-order, Qwen3-235B-A22B, and H100 checks.

Table 1: Routing-change statistics and paired accuracy on all 1,230 AIME25+GPQA-Diamond prompts (greedy, 4,096-token cap). Only τ varies. Paired vanilla accuracy in this run is 54.23% (GPT-OSS-20B) and 43.90% (Qwen3-30B-A3B). "Redir." is the share of all tokens with at least one changed expert; "Slots" is the share of changed expert slots.

Model	τ	Accuracy (%)	Δ vs. vanilla (pp)	Redir. (%)	Slots (%)
GPT-OSS-20B	0.5	53.58	−0.65	14.91	4.19
GPT-OSS-20B	1.5	52.20	−2.03	15.39	4.32
GPT-OSS-20B	5.0	53.58	−0.65	15.19	4.26
Qwen3-30B-A3B	0.5	44.47	+0.57	6.03	1.92
Qwen3-30B-A3B	1.5	44.07	+0.16	6.04	1.93
Qwen3-30B-A3B	5.0	42.28	−1.63	6.02	1.95

Table 2: GPT-OSS-120B accuracy under the MLPerf v6.0 accuracy protocol (one vanilla execution and three complete BLAZE executions; mean ± SD over BLAZE runs).

Condition	AIME25	GPQA-D.	LCB-V6	MLPerf score
Vanilla	81.67	75.96	84.90	83.44
BLAZE (3 runs)	85.69 ± 1.05	75.83 ± 0.36	85.38 ± 0.37	83.91 ± 0.26
Δ (pp)	+4.03	−0.14	+0.48	+0.47

3.3 E2: WHY OVERHEAD-AWARE ROUTER ACTIVATION IS NECESSARY

The activation gate is motivated by the overhead of BLAZE's router path. Load redirection reduces expert stragglers, but executing the redirection path also adds router work, metadata handling, and All-to-All scheduling pressure. At small effective token counts, the MoE grouped-GEMM backend is already operating near its tile floor, so improving load uniformity may not remove enough compute work to pay for this extra control-path and communication cost. At larger batches, however, overloaded experts create real stragglers and the same controller becomes beneficial. Figure 3 visualizes this effect on DeepSeek-R1. The ablation uses a 600-prompt endpoint microbenchmark with client concurrency 256 and a 4000-token generation cap; Appendix G reports the numeric table. Always-on redirection ($N_{\min} = 0$) raises mean TPOT from 68.24–68.28 ms to 79.16 ms, a 16.0% regression on decode-sized effective batches. Setting the overhead-aware gate to $N_{\min} = 256$ restores TPOT to 68.18 ms by bypassing those small MoE calls. At the same concurrency-256 setting, BLAZE_{BA} improves output throughput by 4.2%, and $\text{BLAZE}_{\text{BA+RA}}$ improves it by 7.4%, while TPOT stays essentially flat. A GPT-OSS-20B `BLAZE_MIN_BATCH` sweep shows the same non-monotonic behavior, with the moderate threshold $N_{\min} = 128$ improving samples/s by 8.5% over always-on routing (Figure 7); Appendix G.1 adds a prefill-heavy DeepSeek-V4-Pro diagnostic. The gate is thus an overhead-amortization safeguard, not an accuracy mechanism.

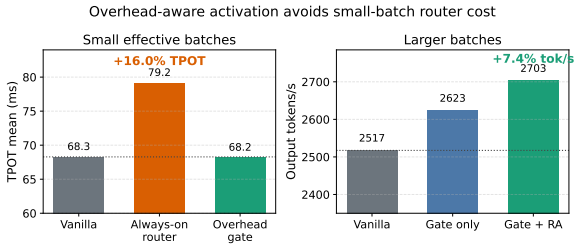

Figure 3: DeepSeek-R1 overhead-aware activation ablation. Left: always-on redirection hurts decode-sized effective batches, increasing mean TPOT by 16.0%; the $N_{\min} = 256$ gate restores baseline TPOT by taking the vanilla-router bypass. Right: at the concurrency-256 serving point, the same gate keeps redirection active where it is useful; rank-aware feedback adds another throughput gain.

3.4 E3: END-TO-END COMPARISON AND OVERHEAD

We next evaluate complete Server-scenario behavior. Table 3 compares methods at SLO-feasible and boundary operating points rather than under saturation. All TRT-LLM rows use the same engine, one 8×H200 node, TP = EP = 8, CUDA Graphs enabled, and the same bounded LoadGen request protocol within each model; the SLO is TTFT P99 $\leq$ 3 s and TPOT P99 $\leq$ 80 ms.

On GPT-OSS-20B at target QPS = 1, vanilla violates the TTFT target while BLAZE passes, with 26.7% lower TTFT P99; vanilla retains the lower TPOT P99. At QPS = 2 all methods violate TTFT,

Table 3: MLPerf-style Server comparison at SLO-feasible (QPS = 1) and boundary (QPS = 2) load with TP = EP = 8 on 8×H200, TensorRT-LLM with CUDA Graphs enabled. SLO: TTFT P99 $\leq$ 3 s and TPOT P99 $\leq$ 80 ms. GPT-OSS-120B Offline rows are included for throughput context. Saturated QPS = 4 results, including the Libra/SGLang rows, are in Appendix I.

Model	Backend	Load	Method	Req/s	Completed tok/s	TTFT P99	TPOT P99	SLO
GPT-OSS-20B	TRITON	QPS = 1	Vanilla	0.91	2570	3526 ms	**12.86 ms**	fail
GPT-OSS-20B	TRITON	QPS = 1	BLAZE	**0.92**	2151	**2584 ms**	18.04 ms	pass
GPT-OSS-20B	TRITON	QPS = 2	Vanilla	1.84	4414	65505 ms	106.34 ms	fail
GPT-OSS-20B	TRITON	QPS = 2	Capacity-Aware	1.92	4306	11931 ms	192.17 ms	fail
GPT-OSS-20B	TRITON	QPS = 2	BLAZE	**1.95**	**4643**	**3323 ms**	**18.84 ms**	fail
Qwen3-30B-A3B	WIDEEP	QPS = 1	Vanilla	0.73	2790	2773 ms	19.17 ms	pass
Qwen3-30B-A3B	WIDEEP	QPS = 1	EPLB	**0.94**	3458	2827 ms	**17.03 ms**	pass
Qwen3-30B-A3B	WIDEEP	QPS = 1	BLAZE‡	0.93	**3561**	**2771 ms**	18.59 ms	pass
GPT-OSS-120B	TRITON	Offline	Vanilla	23.87	12219	n/a	n/a	–
GPT-OSS-120B	TRITON	Offline	BLAZE	**26.00**	**13608**	n/a	n/a	–

Notes. Server cells are bounded 300-request LoadGen runs, not official MLPerf results; within each model all cells share one request protocol, which differs from that of Appendix J. ‡ Fixed τ=5.0 (accuracy cost in Table 1). EPLB uses 144 slots (16 replicas). Missing baselines: Appendix F.

but BLAZE degrades far more gracefully (3.3 s versus 65.5 s for vanilla and 11.9 s for Capacity-Aware). For a matched comparison with Capacity-Aware at SLO-feasible load, we use a separate GPT-OSS-20B QPS sweep in which all cells share one shorter-output request protocol (Appendix J). In that sweep, Capacity-Aware passes at QPS = 1.0 but fails from QPS = 1.25 onward, whereas BLAZE still passes at QPS = 1.75; from QPS = 0.55 to 0.70, BLAZE improves completed tokens/s by 1.7–4.4% over vanilla and lowers TTFT P99 at every point. We treat these as observed rather than deterministic boundaries, since repeat-level P99 variation remains. On Qwen3-30B-A3B at QPS = 1, all three methods pass; BLAZE delivers the highest completed tokens/s and the lowest TTFT P99, while native EPLB retains the lowest TPOT, so we do not claim universal dominance over either baseline. BLAZE reaches this without the 0.84 GiB/GPU of replicated expert weights that EPLB uses here (Table 18).

Missing baselines reflect structural limits rather than omitted comparisons: TRT-LLM EPLB has a GPT-OSS model-path integration limitation, and Capacity-Aware's dynamic-shape `nonzero()` cannot be captured by WIDEEP CUDA Graphs. The released Libra (Yang et al., 2026) artifact's fast path never activated under our continuous-batching trace, which we report as an integration boundary rather than a comparison in Libra's intended setting (Appendix F).

Controller overhead. To separate the routing controller's intrinsic cost from end-to-end scheduling effects, we run a router-only no-op microbenchmark in which the BLAZE kernels execute as in production but the final expert assignment is forced back to vanilla, isolating the per-call routing kernel, the once-per-step EMA update, and the EP `AllReduce`. Table 10 reports sub-millisecond overhead for a full forward across all MoE layers: 267–288 μs on GPT-OSS-20B (24 MoE layers) and 503–592 μs on Qwen3-30B-A3B (48 MoE layers). Per token per layer this is 2.73 ns and 3.01 ns at a 4096-token batch, on the order of a handful of register operations. To test whether this cost is visible end to end, we run a controlled fixed-work experiment on GPT-OSS-20B at QPS = 1 in which every response is fixed to 128 output tokens and prompts, request order, and the Poisson arrival schedule are identical. We compare vanilla, a *Shadow* setting that runs the production controller with zero penalties (bit-identical routing), and the active controller, over two counterbalanced repetitions. Shadow exposes a small bookkeeping cost (+2.3 ms mean TTFT), which active routing recovers; the paired-bootstrap 95% intervals of the active-minus-vanilla differences in mean and P99 TTFT, TPOT, and end-to-end latency all include zero (Appendix K). BLAZE therefore adds no detectable net latency at fixed work, while its benefits appear as serving pressure increases (Table 3); we do not claim a statistically significant fixed-work speedup.

Sensitivity and input-dependent skew. The default gains are not a narrow optimum: at the SLO-feasible GPT-OSS-20B point, varying $\alpha \in \{0.05, 0.15, 0.30\}$ or $\alpha_{\text{rank}} \in \{0, 0.02, 0.05\}$ keeps every setting SLO-compliant with a 1.8% throughput spread, and rank feedback lowers TTFT P99 by 15% on a rank-skewed GPT-OSS-20B stress workload (Appendix L). In a length-matched study that stratifies prompts by pre-redirection expert and rank skew, the throughput gain is larger in the

higher-skew stratum (+41.8% versus +16.1% output tokens/s) with no observed accuracy loss in either stratum (Appendix M). Both are small-sample diagnostics.

Memory footprint. BLAZE does not replicate experts and does not allocate runtime token-routing buffers proportional to the request stream. Its entire persistent device state is the per-layer expert-load EMA, the counts buffer, and a handful of scalar parameters — ~ 37 KB total at GPT-OSS-120B's 36-layer, 128-expert configuration (Appendix C, "Device state layout"). Expert-replication load balancers instead pay for every redundant expert slot (Appendix H.6; Table 18). EPLB consumes 0.28–1.11 GiB per GPU on GPT-OSS-20B (depending on whether the replicated expert weights are stored in the model's native MXFP4 or in bf16) and 0.84 GiB per GPU on Qwen3-30B-A3B. BLAZE's contribution to that column is below the table's resolution. This is not a side observation: KV-cache pool size is set as a fraction of free GPU memory after weights are loaded, so every gibibyte freed by not replicating experts is a gibibyte that the engine can return to KV cache and continuous-batching capacity.

4 Related Work

Training-time routing and balance. Switch Transformers (Fedus et al., 2022) uses an auxiliary balancing loss; ST-MoE (Zoph et al., 2022) adds a separate router-stability z-loss. BASE Layers (Lewis et al., 2021) solves balanced token assignment, while Expert Choice (Zhou et al., 2022) lets experts select tokens. Soft MoE (Puigcerver et al., 2024) uses differentiable mixtures, and DeepSeek-MoE (Dai et al., 2024) combines fine-grained and shared experts. Loss-Free Balancing (Wang et al., 2024), adopted in DeepSeek-V3 (DeepSeek-AI, 2024), updates biases from training loads, using biased scores for selection and original scores for mixture weights. BLAZE applies feedback during inference with fixed model weights, confidence protection, and an overhead-aware activation gate. **MoE execution systems.** DeepSpeed-MoE (Rajbhandari et al., 2022) optimizes training and inference; Tutel (Hwang et al., 2023) adapts parallelism and pipelining. FasterMoE (He et al., 2022) and FlexMoE (Nie et al., 2023) address dynamic workloads and placement in distributed training. MegaBlocks (Gale et al., 2023) uses block-sparse computation for variable expert token counts. BLAZE changes eligible assignments before backend execution. **Expert placement and replication.** EPLB redistributes and optionally replicates experts while preserving their identities; vLLM supports online rebalancing (DeepSeek-AI, 2024; vLLM Contributors, 2026). LLEP (Nguyen et al., 2026) redistributes excess tokens and associated expert parameters. Libra (Yang et al., 2026) predicts expert activations, combining replication and token sharding while overlapping balancing overhead with computation. BLAZE uses no extra expert slots ($E_{\text{slots}} = E$), changing eligible expert choices while protecting high-confidence top-1 assignments. **Inference-time expert selection.** Pre-gated MoE (Hwang et al., 2024) anticipates expert use to overlap loading and computation. Lynx (Gupta et al., 2026) remaps low-affinity assignments to activate fewer experts and reduce decode-time bandwidth pressure. BLAZE instead spreads eligible assignments across less-loaded experts and EP ranks to reduce stragglers. Its contribution combines inference-time feedback, confidence protection, and activation gating; biased selection and inference-time remapping themselves have precedents. **Latency-aware LLM serving.** Orca (Yu et al., 2022) uses iteration-level scheduling; vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) optimize KV-cache management and reuse. Sarathi-Serve (Agrawal et al., 2024) schedules chunked prefills, while DistServe (Zhong et al., 2024) separates prefill and decode to meet TTFT and TPOT targets. BLAZE's MoE-layer activation gate responds to the scheduler's effective token batches.

5 Conclusion

BLAZE improves expert-parallel MoE serving by treating routing as a controlled redirection problem rather than a capacity-only load-balancing problem. Its confidence guardrail preserves high-margin semantic choices, its load and rank feedback redirect only ambiguous assignments, and its overhead-aware gate bypasses the controller when the MoE call is too small to amortize router-side cost. Across GPT-OSS-20B and Qwen3-30B-A3B, BLAZE preserves near-vanilla accuracy while improving tail latency and extending the SLO-feasible load range in the evaluated engines and workloads, without dropping tokens, replicating experts, retraining, or changing the MoE backend interface. Its accuracy is also preserved on GPT-OSS-120B under the full MLPerf accuracy protocol.

AI USE STATEMENT

Generative AI tools were used only to polish the writing of this paper; the authors are responsible for all content.

ETHICS STATEMENT

This work studies inference-time routing for Mixture-of-Experts language models and does not involve human subjects, personal data, or new datasets. All models and benchmarks used are publicly released.

REPRODUCIBILITY STATEMENT

Appendix C describes the TensorRT-LLM and vLLM integration and the routing kernel, Appendix C.2 specifies the serving protocol, and Appendix D lists all hyperparameters. Appendices J–M give the protocol of every additional serving and latency analysis, including request counts, seeds, and statistical procedures. All evaluation benchmarks and base models are publicly available, and code will be made available upon acceptance.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 1280–1297. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.70. URL https://doi.org/10.18653/v1/2024.acl-long.70.

DeepSeek-AI. Deepseek-v3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. In *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. mlsys.org, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5a54f79333768effe7e8927bcccffe40-Abstract-mlsys2023.html.

Vima Gupta, Jae Hyung Ju, Kartik Sinha, Ada Gavrilovska, and Anand Padmanabha Iyer. Lynx: Enabling efficient MoE inference through dynamic batch-aware expert selection. *arXiv preprint arXiv:2411.08982v3*, 2026. URL https://arxiv.org/abs/2411.08982v3. Originally released in 2024.

Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In Jaejin Lee, Kunal Agrawal, and Michael F. Spear (eds.), *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, pp. 120–134. ACM, 2022. doi: 10.1145/3503221.3508418. URL https://doi.org/10.1145/3503221.3508418.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. mlsys.org, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, 2024. URL https://arxiv.org/abs/2308.12066.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

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

Xuan-Phi Nguyen, Shrey Pandit, Austin Xu, Caiming Xiong, and Shafiq Joty. Least-loaded expert parallelism: Load balancing an imbalanced mixture-of-experts. *CoRR*, abs/2601.17111, 2026. doi: 10.48550/ARXIV.2601.17111. URL https://doi.org/10.48550/arXiv.2601.17111.

Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement. *Proc. ACM Manag. Data*, 1(1):110:1–110:19, 2023. doi: 10.1145/3588964. URL https://doi.org/10.1145/3588964.

Joan Puigcerver, Carlos Riquelme Ruiz, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=jxpsAj7ltE.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Proceedings of Machine Learning Research, pp. 18332–18346. PMLR, 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Virtual Event / Valencia, Spain, May 30 - June 3, 2020*, pp. 446–459. IEEE, 2020.

doi: 10.1109/ISCA45697.2020.00045. URL https://doi.org/10.1109/ISCA45697.2020.00045.

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 *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

vLLM Contributors. Expert parallel deployment: Expert parallel load balancer (EPLB). vLLM v0.20.1 documentation, 2026. URL https://docs.vllm.ai/en/v0.20.1/serving/expert_parallel_deployment/. Accessed September 26, 2026.

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

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

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 2022, Carlsbad, CA, USA, July 11-13, 2022*, pp. 521–538. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y. Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/2f00ecd787b432c1d36f3de9800728eb-Abstract-Conference.html.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022. URL https://arxiv.org/abs/2202.08906.

APPENDIX

This appendix is organized as follows. Appendix A gives the theoretical motivation for BLAZE's bias-driven score and its connection to the optimal expert assignment. Appendix B states the limitations of BLAZE. Appendix C describes the TensorRT-LLM and vLLM integration, the CUDA routing kernel, and the persistent controller state layout. Appendix C.2 specifies the MLPerf-style Server and Offline scenario protocol used throughout the evaluation. Appendix D lists all hyperparameters and notational conventions. Appendix E studies the accuracy–throughput trade-off as the confidence threshold τ varies, including the online quantile alternative. Appendix F explains baseline availability and contrasts BLAZE with Libra. Appendix G gives additional activation-gate sweep results, including the GPT-OSS-20B `BLAZE_MIN_BATCH` sweep and the DeepSeek-V4-Pro prefill-heavy diagnostic. Appendix H reports the full redirect-feasibility accuracy tables for AIME25 and GPQA-Diamond, the per-run GPT-OSS-120B MLPerf accuracy evaluation, and request-order, Qwen3-235B-A22B, and H100/H200 accuracy checks. The memory-overhead derivation compares EPLB's replica footprint with BLAZE's negligible persistent state. The remaining appendices report the saturated QPS = 4 Server diagnostics (Appendix I), the GPT-OSS-20B QPS sweep (Appendix J), phase-level, fixed-work, and prefill-dominant latency analyses (Appendix K), gain sensitivity and rank-skew stress tests (Appendix L), and behavior under input-induced routing skew (Appendix M).

A THEORETICAL STABILITY OF BLAZE

This appendix formalizes the three guarantees used by the main text. The first is an inference-stability statement: BLAZE changes only low-regret routing slots, so the output and task loss stay close to vanilla routing whenever the pretrained router margin is a reliable local confidence signal. The second is a closed-loop load-balancing statement: the EMA feedback controller converges to a steady state near uniform expert load; with the rank-aware term, the same argument also controls EP-rank load. The third is an efficiency statement: when straggler time is an increasing function of expert and rank imbalance, the convergence bounds imply an expected serving latency reduction whenever the straggler reduction exceeds the router overhead.

A.1 NOTATION

Please refer to Table 4 used in the paper.

A.2 ASSUMPTIONS

A1. Bounded local MoE sensitivity. For every layer, changing one selected expert slot whose semantic-score regret is at most η changes the layer output by at most $G_{\mathrm{moe}}\eta$. Changing m slots changes the output by at most $mG_{\mathrm{moe}}\eta$. The downstream network and evaluation loss are locally Lipschitz with constant G_{net} with respect to the collection of MoE-layer perturbations. This assumption is the formal version of the margin-calibration study in §3.2: semantic score gaps are treated as a local proxy for how costly a redirection is.

A2. Bounded feedback range. During active routing, the normalized EMA states remain in $[1 - D_E, 1 + D_E]$, and the rank states remain in $[1 - D_R, 1 + D_R]$. Thus the maximum penalty difference between two experts is

$$\Gamma_{\mathrm{BA}} = 2\alpha D_E, \qquad \Gamma_{\mathrm{BA+RA}} = 2\alpha D_E + 2\alpha_{\mathrm{rank}} D_R. \tag{5}$$

The clipping and finite-batch normalization used by the implementation make this a mild bounded-state assumption.

A3. Monotone smooth routing response. Fix a layer and condition on the batch-aware gate being active. Let $p_e(u, \rho)$ be the expected normalized assignment rate to expert e when the expert-load state is u and the rank-load state is ρ; for $\mathrm{BLAZE}_{\mathrm{load}}$ and $\mathrm{BLAZE}_{\mathrm{BA}}$, the rank term is disabled and ρ is ignored. The response map is locally strongly balancing in the expert-load state: for the uniform vector $\mathbf{1}$,

$$\|p(u, \rho) - \mathbf{1}\|_\infty \le (1 - \kappa)\|u - \mathbf{1}\|_\infty + \xi_N, \qquad 0 < \kappa \le 1. \tag{6}$$

Table 4: Notation used in the BLAZE theory.

Symbol	Meaning
$\ell \in \{1, \dots, L\}$	MoE layer index
$e \in \{1, \dots, E\}$	Expert index in a layer
$r \in \{1, \dots, R\}$	EP-rank index, with $R = \mathrm{EP}$
$\mathcal{E}_r$	Experts placed on rank r
$r(e)$	EP rank that owns expert e
x	Token representation entering an MoE layer
N	Effective token count in the current MoE call
K	Number of experts selected per token
$s_{\ell,e}(x)$	Original semantic router score
$e_1(x), e_2(x)$	Top-1 and top-2 experts under $s_{\ell,e}(x)$
$\Delta_\ell(x)$	Semantic margin $s_{\ell,e_1}(x) - s_{\ell,e_2}(x)$
τ	Top-1 locking threshold
$\mathcal{S}^0_\ell(x)$	Vanilla Top-K expert set
$\mathcal{S}^B_\ell(x)$	BLAZE expert set
$w_{\ell,e}(x)$	Mixture weight computed from original scores
$c^{(t)}_{\ell,e}$	Assignments to expert e at step t
$\hat{c}^{(t)}_{\ell,e}$	Count normalized by the layer mean count
$\bar{\ell}^{(t)}_{\ell,e}$	Expert-load EMA state
$\bar{\rho}^{(t)}_{\ell,r}$	Rank-load state, mean of $\bar{\ell}^{(t)}_{\ell,e}$ on rank r
α	Expert-load feedback gain
α_{rank}	Rank-load feedback gain
β	EMA smoothing factor
$N_{\min}$	Batch-aware activation threshold
B_M, σ	Backend tile size and activation scale used to set $N_{\min}$
$\delta_E(t)$	Expert imbalance $\max_e \lvert\bar{\ell}^{(t)}_{\ell,e} - 1\rvert$
$\delta_R(t)$	Rank imbalance $\max_r \lvert\bar{\rho}^{(t)}_{\ell,r} - 1\rvert$
$T^{(t)}$	Per-layer serving time model at step t
Ω	Per-layer overhead of the active BLAZE controller
λ_E, λ_R	Latency sensitivity to expert and rank imbalance

Here $\xi_N = O(N^{-1/2})$ is the finite-batch sampling error. Intuitively, overloaded experts receive larger penalties, and the ambiguous token mass is large enough that routing probability moves away from them.

A4. Rank-level monotonicity. For $\mathrm{BLAZE}_{\mathrm{BA+RA}}$, the rank-aggregated response obeys the same condition:

$$\|P_R(u, \rho) - \mathbf{1}\|_\infty \le (1 - \kappa_R)\|\rho - \mathbf{1}\|_\infty + \xi_{N,R}, \qquad 0 < \kappa_R \le 1, \tag{7}$$

where P_R is the expected normalized assignment rate per EP rank and ρ is the rank-load state. This is exactly the additional structure introduced by Eq. 4.

A5. Straggler-dominated serving-time model. For active batches, the expected per-layer MoE serving time admits the first-order decomposition

$$\mathbb{E}[T^{(t)}] = T_{\mathrm{avg}} + \lambda_E \delta_E(t) + \lambda_R \delta_R(t) + \Omega, \tag{8}$$

where T_{avg} is the balanced grouped-GEMM and communication time, $\lambda_E, \lambda_R \ge 0$ measure sensitivity to expert and EP-rank stragglers, and Ω is the active-router overhead. For $\mathrm{BLAZE}_{\mathrm{load}}$ and $\mathrm{BLAZE}_{\mathrm{BA}}$, set $\lambda_R = 0$ and $\delta_R(t) = 0$ in this model. For inactive batches, $\Omega = 0$ because the batch-aware gate takes the true bypass path.

A.3 Main Theorem

The statement below covers the ablations used in the paper. Setting $N_{\min} = 0$ gives the always-on load-feedback router $\mathrm{BLAZE}_{\mathrm{load}}$; setting $\alpha_{\mathrm{rank}} = 0$ with the batch gate gives $\mathrm{BLAZE}_{\mathrm{BA}}$; enabling both the gate and rank penalty gives full $\mathrm{BLAZE}_{\mathrm{BA+RA}}$.

Theorem 1 (accuracy-stable, load-stable, and efficiency-conditioned BLAZE). Consider one deployed MoE model whose router is replaced by Algorithm 1. Suppose Assumptions A1–A3 hold for active batches, and let $m_\ell(x)$ be the number of BLAZE-selected slots that differ from vanilla Top-K at layer ℓ for token x. Let δ_E^0 and δ_R^0 denote the corresponding steady-state imbalance of vanilla routing under the same traffic distribution.

If $N < N_{\min}$, the batch-aware gate returns the vanilla routing result exactly, so the inference perturbation and load-controller update are both zero. If $N \geq N_{\min}$, then for $\text{BLAZE}_{\text{load}}$ or BLAZE_{BA} ($\alpha_{\text{rank}} = 0$),

$$|\mathcal{L}_B(x) - \mathcal{L}_0(x)| \leq G_{\text{net}} G_{\text{moe}} \sum_{\ell=1}^{L} m_\ell(x)\, \Gamma_{\text{BA}}, \tag{9}$$

where $\mathcal{L}_0$ and $\mathcal{L}_B$ are the vanilla and BLAZE losses. Moreover, the expert-load EMA converges linearly up to finite-batch noise:

$$\delta_E(t) \leq (1 - (1-\beta)\kappa)^t \delta_E(0) + \frac{\xi_N}{\kappa}. \tag{10}$$

If Assumption A5 holds, then after t active steps the expected per-layer serving-time improvement over vanilla is at least

$$\Delta T_{\text{BA}}(t) \geq \lambda_E \left[\delta_E^0 - (1 - (1-\beta)\kappa)^t \delta_E(0) - \frac{\xi_N}{\kappa} \right] - \Omega. \tag{11}$$

In particular, the steady-state improvement is positive whenever $\lambda_E(\delta_E^0 - \xi_N/\kappa) > \Omega$.

If Assumption A4 also holds, then full $\text{BLAZE}_{\text{BA+RA}}$ satisfies

$$|\mathcal{L}_B(x) - \mathcal{L}_0(x)| \leq G_{\text{net}} G_{\text{moe}} \sum_{\ell=1}^{L} m_\ell(x)\, \Gamma_{\text{BA+RA}}, \tag{12}$$

and the rank imbalance obeys

$$\delta_R(t) \leq (1 - (1-\beta)\kappa_R)^t \delta_R(0) + \frac{\xi_{N,R}}{\kappa_R}. \tag{13}$$

Under Assumption A5, its expected per-layer serving-time improvement is at least

$$\begin{aligned} \Delta T_{\text{BA+RA}}(t) \geq \lambda_E &\left[\delta_E^0 - (1 - (1-\beta)\kappa)^t \delta_E(0) - \frac{\xi_N}{\kappa} \right] \\ &+ \lambda_R \left[\delta_R^0 - (1 - (1-\beta)\kappa_R)^t \delta_R(0) - \frac{\xi_{N,R}}{\kappa_R} \right] - \Omega. \end{aligned} \tag{14}$$

The steady-state improvement is positive whenever the expert- and rank-level straggler reduction exceeds the router overhead:

$$\lambda_E \left(\delta_E^0 - \frac{\xi_N}{\kappa} \right) + \lambda_R \left(\delta_R^0 - \frac{\xi_{N,R}}{\kappa_R} \right) > \Omega. \tag{15}$$

Thus the batch-aware version preserves exact vanilla behavior when balancing is not amortized, while the active controller reaches a steady state whose expert and rank imbalance are $O(\xi_N/\kappa)$ and $O(\xi_{N,R}/\kappa_R)$, respectively. In the large-batch regime selected by §2.3, this balance translates into expected serving efficiency gains exactly when the saved straggler time dominates the controller overhead.

Proof. *Step 1: exact bypass.* When $N < N_{\min}$, Algorithm 1 returns $\mathcal{S}_\ell^0(x) = \text{TopK}_e\, s_{\ell,e}(x)$ and computes the mixture weights from the original semantic scores. This is exactly the vanilla MoE computation. The implementation also skips count writes, AllReduce, and EMA updates on this path. Therefore the loss perturbation, controller-state change, and active-router overhead are all zero.

Step 2: per-slot semantic regret under expert feedback. Assume $N \geq N_{\min}$ and $\alpha_{\text{rank}} = 0$. Define the semantic regret of selecting expert e instead of a vanilla expert e^0 by $s_{\ell,e^0}(x) - s_{\ell,e}(x)$. The BA

adjusted score is $r_{\ell,e}(x) = s_{\ell,e}(x) - \alpha\bar{\ell}_{\ell,e}$. Pair each selected BLAZE expert that is not in the vanilla Top-K set with one vanilla Top-K expert that was excluded by BLAZE. This pairing exists because both routes select K experts. Since the selected BLAZE experts are the Top-K experts under the adjusted score among the eligible candidates, each such pair satisfies

$$s_{\ell,e^0}(x) - \alpha\bar{\ell}_{\ell,e^0} \le s_{\ell,e}(x) - \alpha\bar{\ell}_{\ell,e}. \tag{16}$$

Rearranging gives

$$s_{\ell,e^0}(x) - s_{\ell,e}(x) \le \alpha(\bar{\ell}_{\ell,e^0} - \bar{\ell}_{\ell,e}) \le 2\alpha D_E = \Gamma_{\mathrm{BA}}, \tag{17}$$

where the last inequality uses Assumption A2. This proves that every load-redirected slot whose vanilla counterpart was eligible for adjusted-score selection has regret at most Γ_{BA}.

Step 3: top-1 guardrail closure. We show every changed slot has semantic regret at most Γ_{BA} by exhaustive case analysis.

- $\Delta_\ell(x) > \tau$ *(top-1 locked).* The guardrail forces $e_1(x)$ into $\mathcal{S}^B_\ell(x)$, so it is never in the symmetric difference. Only fallback slots can differ from vanilla, and Step 2 bounds each by Γ_{BA}.
- $\Delta_\ell(x) \le \tau$ *(top-1 unlocked),* e_1 *wins the adjusted-score competition.* Then $e_1 \in \mathcal{S}^B_\ell(x)$; no regret is incurred by this slot. Other differing slots are handled by Step 2.
- $\Delta_\ell(x) \le \tau$ *(top-1 unlocked),* e_1 *loses.* Then $e_1 \notin \mathcal{S}^B_\ell(x)$ and it is paired in Step 2 with the Blaze expert e that displaced it. The adjusted-score comparison gives $s_{\ell,e_1}(x) - s_{\ell,e}(x) \le \Gamma_{\mathrm{BA}}$.

In every case the per-slot semantic regret is at most Γ_{BA}, as claimed.

Step 4: convert slot regret to loss perturbation. By Assumption A1, changing $m_\ell(x)$ slots in layer ℓ with per-slot regret at most Γ_{BA} changes that MoE-layer output by at most $m_\ell(x)G_{\mathrm{moe}}\Gamma_{\mathrm{BA}}$. Summing over layers and applying the downstream Lipschitz constant G_{net} proves Eq. 9.

Step 5: rank-aware regret. For BA+RA, the adjusted score is $r_{\ell,e}(x) = s_{\ell,e}(x) - \alpha\bar{\ell}_{\ell,e} - \alpha_{\mathrm{rank}}\bar{\rho}_{\ell,r(e)}$. The same adjusted-score comparison gives

$$\begin{aligned} s_{\ell,e^0}(x) - s_{\ell,e}(x) &\le \alpha(\bar{\ell}_{\ell,e^0} - \bar{\ell}_{\ell,e}) + \alpha_{\mathrm{rank}}(\bar{\rho}_{\ell,r(e^0)} - \bar{\rho}_{\ell,r(e)}) && (18)\\ &\le 2\alpha D_E + 2\alpha_{\mathrm{rank}} D_R = \Gamma_{\mathrm{BA+RA}}. && (19)\end{aligned}$$

Repeating Steps 3–4 proves Eq. 12.

Step 6: expert-load contraction. For one layer, let $\hat{p}^{(t)}$ be the vector of realized normalized assignment rates at step t. The actual EMA update is

$$\bar{\ell}^{(t+1)} = \beta\bar{\ell}^{(t)} + (1-\beta)\hat{p}^{(t)}. \tag{20}$$

Because A3's term $\xi_N = O(N^{-1/2})$ is explicitly defined to absorb finite-batch sampling error, the bound in Eq. 6 holds for the realized rate $\hat{p}^{(t)}$, not just its expectation: $\|\hat{p}^{(t)} - \mathbf{1}\|_\infty \le (1-\kappa)\|\bar{\ell}^{(t)} - \mathbf{1}\|_\infty + \xi_N$. Subtracting $\mathbf{1}$ and applying A3 to $\hat{p}^{(t)}$,

$$\begin{aligned} \|\bar{\ell}^{(t+1)} - \mathbf{1}\|_\infty &\le \beta\|\bar{\ell}^{(t)} - \mathbf{1}\|_\infty + (1-\beta)\|\hat{p}^{(t)} - \mathbf{1}\|_\infty && (21)\\ &\le \big[\beta + (1-\beta)(1-\kappa)\big]\|\bar{\ell}^{(t)} - \mathbf{1}\|_\infty + (1-\beta)\xi_N && (22)\\ &= (1-(1-\beta)\kappa)\delta_E(t) + (1-\beta)\xi_N. && (23)\end{aligned}$$

Unrolling this scalar recursion gives

$$\delta_E(t) \le (1-(1-\beta)\kappa)^t\delta_E(0) + (1-\beta)\xi_N\sum_{i=0}^{t-1}(1-(1-\beta)\kappa)^i, \tag{24}$$

and the geometric sum is at most $1/((1-\beta)\kappa)$, proving Eq. 10.

Step 7: rank-load contraction. For BA+RA, average the realized assignment rate $\hat{p}^{(t)}$ over each rank $\mathcal{E}_r$ to obtain the realized rank-assignment rate $\hat{P}^{(t)}_R$. By the same argument as Step 6, A4's $\xi_{N,R}$ absorbs finite-batch sampling error, so A4 applies to $\hat{P}^{(t)}_R$ directly and gives the one-step inequality

$$\delta_R(t+1) \le (1-(1-\beta)\kappa_R)\delta_R(t) + (1-\beta)\xi_{N,R}. \tag{25}$$

Unrolling the same geometric series proves Eq. 13.

Step 8: expected efficiency gain. Let T_0 be the vanilla per-layer serving time under the same traffic. For BLAZE_{BA} ($\lambda_R = 0$ by A5), Assumption A5 gives $\mathbb{E}[T_0] = T_{\text{avg}} + \lambda_E \delta_E^0$ and $\mathbb{E}[T_{\text{BA}}^{(t)}] = T_{\text{avg}} + \lambda_E \delta_E(t) + \Omega$. Therefore

$$\Delta T_{\text{BA}}(t) = \mathbb{E}[T_0 - T_{\text{BA}}^{(t)}] = \lambda_E(\delta_E^0 - \delta_E(t)) - \Omega. \tag{26}$$

Substituting Eq. 10 proves Eq. 11. For BA+RA, A5 gives $\mathbb{E}[T_0] = T_{\text{avg}} + \lambda_E \delta_E^0 + \lambda_R \delta_R^0$ and $\mathbb{E}[T_{\text{BA+RA}}^{(t)}] = T_{\text{avg}} + \lambda_E \delta_E(t) + \lambda_R \delta_R(t) + \Omega$, so the comparison includes both expert and rank straggler terms:

$$\Delta T_{\text{BA+RA}}(t) = \lambda_E(\delta_E^0 - \delta_E(t)) + \lambda_R(\delta_R^0 - \delta_R(t)) - \Omega. \tag{27}$$

Substituting the upper bounds from Eqs. 10 and 13 proves Eq. 14. Taking $t \to \infty$ yields the positive-gain condition in Eq. 15. This completes the proof.

B LIMITATIONS

BLAZE's throughput gains are most pronounced in prefill and chunked-prefill phases where large effective token batches produce genuine expert imbalance; in low-concurrency decode-only serving, the overhead-aware gate correctly bypasses redirection and falls back to vanilla routing, so gains in that regime are limited. The confidence threshold τ and activation threshold $N_{\min}$ should be calibrated per model family and deployment configuration, as distribution shift between calibration traffic and production traffic may alter the accuracy–throughput trade-off. All experiments use a single 8×H100/H200 node with intra-node NVLink; multi-node EP deployments with slower inter-node interconnects may exhibit different overhead profiles and may require re-validation before deployment. Serving-performance claims are scoped to the evaluated engines (TensorRT-LLM, and vLLM for one diagnostic), workloads, and SLO-feasible operating points. Qwen3-30B-A3B has a single SLO-feasible Server operating point, and the fine-grained QPS sweep covers only GPT-OSS-20B. For GPT-OSS-120B and Qwen3-235B-A22B we provide accuracy evidence but no Server-scenario throughput comparison against other balancers. Libra is evaluated only through its released SGLang artifact; we did not port it to TensorRT-LLM/WIDEEP, since a faithful port would require rebuilding its activation predictor, replica placement, token sharding, and buffer lifecycle inside the graph-captured engine. The τ sweep varies the lock threshold at fixed gains; a joint gain-and-threshold dose study is left to future work. None of the serving results are official MLPerf submissions.

C IMPLEMENTATION DETAILS

BLAZE is implemented as a drop-in router-side module for TensorRT-LLM, consisting of a CUDA routing kernel and a lightweight Python orchestration layer. The persistent controller state adds approximately 37 KB of GPU memory in the GPT-OSS-120B 36-layer, 128-expert configuration.

Code integration path. The TensorRT-LLM integration changes only the MoE routing method. The model files keep the original gate projection unchanged; when `ENABLE_DYNAMIC_BIAS_ROUTER=1`, the fused-MoE setup selects `DynamicBiasMoeRoutingMethod` in `_torch/modules/fused_moe/routing.py`. This method receives the ordinary router logits and lazily creates a per-device manager in `dynamic_bias/router_manager.py`. The manager owns the persistent count buffers, EMA buffers, runtime scalar parameters, and the CUDA/Triton router backend. GPT-OSS and Qwen-style flat routers use `DynamicBiasMoeRoutingMethod`; DeepSeek-R1 uses the corresponding `DeepSeekV3BlazeRoutingMethod` wrapper around the WIDEEP noaux routing path.

After the transformer forward, the model entry point calls `dynamic_bias_step`. This is the only place where the per-layer counts accumulated during active BLAZE routing are folded into `load_ema`. The call forwards the MoE EP process group when available, so attention-DP plus EP deployments can aggregate counts with `AllReduce(SUM)` before the EMA update. When no process group is registered, the step uses the rank-local count view; in the TP = EP setting used by the main measurements, each rank sees the same router stream.

Overhead-aware true bypass. The production code implements overhead-aware activation before entering the BLAZE manager. The threshold is controlled by `BLAZE_MIN_BATCH` (or the `DEEPSEEK_BLAZE_MIN_BATCH` alias). If this variable is an integer, it is used directly as $N_{\min}$. If it is set to `auto` or `adaptive`, the helper `_resolve_blaze_min_batch` uses the one-tile default

$$N_{\min}^{\text{default}} = \text{clip}_{[\text{floor,cap}]}\left(\text{bucket_up}\left(B_M \cdot \frac{E/\text{EP}}{K}\right)\right). \tag{28}$$

The helper also reports the corresponding overhead-aware validation band by replacing B_M with $O_{\text{route}} \in [0.25B_M, B_M]$. With $B_M = 128$, this is 32–128 assignments per local expert. The factor $(E/\text{EP})/K$ converts this assignment budget into the number of input tokens needed to reach that budget, because each token contributes K expert assignments. The main experiments use explicit serving thresholds from this small system-friendly band, as listed in Table 6.

When the effective MoE token count satisfies $N < N_{\min}$, the routing method returns the vanilla router result directly: `RenormalizeMoeRoutingMethod` for flat GPT-OSS/Qwen-style routers and the original DeepSeek noaux-TC router for DeepSeek-R1. This path does not instantiate the BLAZE manager if it has not already been created, does not launch the BLAZE CUDA router, does not write count buffers, and does not trigger an EMA update or EP AllReduce. In the TRITON fused-MoE path, the same small-batch condition skips the BLAZE intercept and falls through to the default `TritonEPRouter`, so the downstream bitmatrix layout remains exactly the backend's vanilla layout.

Threshold resolution. Algorithm 2 resolves $N_{\min}$ once at router initialization. It supports both manual overrides and automatic backend-aware thresholding: a manual integer value is useful for ablations ($N_{\min} = 0$ corresponds to always-on redirection), while automatic mode computes the threshold from the backend tile size, local expert count, and routing fanout. The threshold is not a capacity limit and never drops tokens; when the gate is inactive, the model behavior is exactly vanilla for that MoE call.

Routing kernel. For active batches, the CUDA backend contains a fixed-τ fast path and a general diagnostic path. In the fast path, one 128-thread block routes four tokens: each warp handles one token and each lane owns four contiguous experts. In the general path, one 128-thread block routes one token with one thread per expert. Both paths execute the same logical steps:

1. **Parallel score computation.** The router computes the semantic baseline $\texttt{sem}[e] = \texttt{logits}[e] + \texttt{bias_base}[e]$ and the adjusted score $\texttt{score}[e] = \texttt{sem}[e] - \alpha \cdot \texttt{load_ema}[e] - \alpha_{\text{rank}} \cdot \texttt{rank_load}[r(e)]$ when the rank-aware term is enabled. Runtime parameters are read from device memory, making the kernel CUDA-Graph safe: the host can update them with `cudaMemcpyAsync` between graph replays without re-capturing the graph.
2. **Guarded Top-K selection.** The top-1/top-2 margin is computed on the semantic baseline. If the gap exceeds τ, the semantic top-1 expert is locked into the selected set; the remaining slots are chosen from the load-penalized scores. Otherwise, all slots are chosen from the load-penalized scores. The selected expert weights are always computed as $\text{softmax}(\texttt{sem}[\text{selected}])$, not from the penalized scores.
3. **Count and metadata update.** The selected experts issue `atomicAdd` updates to the per-layer expert-count array. When the fused MoE backend requests it, the router also writes the selected-expert bitmatrix consumed by the downstream packing path.

This design keeps the dense score computation parallel while preserving the original semantic scores for probability weights and safety statistics. The kernel is invoked once per MoE layer.

EMA update and skipped batches. After the transformer forward, `router_manager.step` checks whether any layer called the active BLAZE route path. If all layers were bypassed by the overhead-aware gate, the step returns early: existing EMA values are kept, no stale token count is used, and no spurious EMA decay is applied. When active routing did occur, the update kernel normalizes each expert count by the layer's mean assignment count, $N_{\text{tokens}}K/E$, applies $\bar{\ell}_{\ell,e} \leftarrow \beta\bar{\ell}_{\ell,e} + (1-\beta)\hat{c}_{\ell,e}$, and resets the count buffer. The implementation also supports

BLAZE_STEP_INTERVAL>1; in that case counts and token totals are accumulated across skipped update steps so the normalization remains consistent.

Expert parallelism and co-batching. When an EP process group is available, the manager copies the full $L \times E$ count matrix to a pre-allocated device buffer and issues `AllReduce(SUM)` before the EMA update. For the $L = 36, E = 128$ configuration, this matrix is only $36 \times 128 \times 4$ bytes, or 18 KB per update. Continuous batching does not mix request contents: attention state and KV cache remain per request. Other users' tokens can affect only the anonymous aggregate load statistics that shape later low-confidence routing decisions; the controller never reads prompt text or hidden states from another request.

Device state layout. All persistent state resides in device memory:

Buffer	Type	Size
`counts[36][128]`	int32	18 KB
`load_ema[36][128]`	float32	18 KB
`d_params[3]`	float32	12 B
`gap_hist[64]`	int32	256 B
`flip_cnt[36]`	int32	144 B
`lock_cnt[36]`	int32	144 B
`drop_cnt[36]`	int32	144 B
Total		$\sim$37 KB

Runtime scalar parameters, including α and τ, are stored in a device-side array `d_params` with a host-side mirror. Updates are performed via `cudaMemcpyAsync`, allowing the Python-side safety monitor and optional online quantile controller to adjust parameters between CUDA Graph replays without triggering re-capture.

Safety monitor overhead. The accuracy-statistics counters (`flip_cnt`, `lock_cnt`, `drop_cnt`) are updated via `atomicAdd` inside the routing kernel, guarded by a host-controlled sampling flag `enable_accuracy_stats`. This flag is active only during $t \bmod 50 \in \{0, 1\}$ and disabled for the remaining 48 steps, restricting counter-related atomic contention to 4% of decode steps.

Total overhead. We report controller overhead with the router-only no-op microbenchmark in Table 10. That measurement executes the active BLAZE routing kernels, the once-per-step EMA update, and the EP `AllReduce`, but forces the final expert assignment back to vanilla so that backend scheduling effects are excluded. The measured full-forward controller cost is 267–288 μs on GPT-OSS-20B and 503–592 μs on Qwen3-30B-A3B; these are the overhead numbers used throughout the paper.

C.1 Experimental scope and naming

The main redirect-feasibility accuracy study uses only AIME25 and GPQA-Diamond. LiveCodeBench V6 appears only in the separate GPT-OSS-120B MLPerf accuracy summary in Appendix H; it is not part of the GPT-OSS-20B/Qwen3-30B-A3B router-intervention comparison. For routing conditions, $\text{BLAZE}_{\text{load}}$ denotes expert-load feedback with the confidence guardrail, BLAZE_{OA} adds the overhead-aware activation gate, and $\text{BLAZE}_{\text{OA+RA}}$ adds both the overhead-aware gate and rank-aware feedback.

C.2 MLPerf-Style Scenario Protocol

We use NVIDIA's MLPerf Inference LoadGen interface (Reddi et al., 2020) as a repeatable traffic generator and latency accountant, but the runs are internal A/B measurements rather than official MLPerf submissions. In MLPerf terminology, the Query Sample Library (QSL) is a fixed table of tokenized prompts, and the System Under Test (SUT) is the TensorRT-LLM or SGLang serving endpoint. LoadGen sends query sample IDs to the SUT; the SUT retrieves the corresponding tokenized

Table 5: Primary experimental configuration. GPT-OSS-20B and Qwen3-30B-A3B are used for accuracy and Server experiments; DeepSeek-R1 is used for the overhead-aware activation ablation because its WIDEEP backend exposes the small-batch TPOT regression most clearly.

Component	Configuration
Hardware	Main GPT-OSS-20B and Qwen3-30B-A3B Server, accuracy, and activation-threshold runs use one 8× NVIDIA H200-SXM 141GB node. The DeepSeek-V4-Pro vLLM diagnostic in Appendix G.1 also uses one 8× H200-SXM 141GB node. GPT-OSS-120B Offline throughput and guardrail-throughput rows use one 8× NVIDIA H100-SXM 80GB node
Parallelism	Main Server rows use TP = 8, PP = 1, EP = 8; GPT-OSS-120B Offline throughput rows use TP = 4, PP = 1, EP = 4
Runtime	TensorRT-LLM 1.3.0rc8 endpoint, PyTorch backend
Redirect-feasibility accuracy data	AIME25 and GPQA-Diamond
Redirect-feasibility decoding	Greedy, max output tokens = 4096
Serving protocol	MLPerf-style Server at SLO-feasible and boundary target QPS (main text), saturated QPS = 4 (Appendix I), and MLPerf-style Offline throughput runs; see Appendix C.2
MoE backend	GPT-OSS-20B: TRITON; Qwen3/DeepSeek-R1: WIDEEP
BLAZE controller	GPT-OSS-20B: expert-load penalty + rank-aware penalty; Qwen3-30B-A3B/DeepSeek-R1: expert-load penalty + overhead-aware gate + rank-aware penalty
Overhead-aware $N_{\min}$	Server rows: GPT-OSS-20B does not enable the overhead-aware gate; Qwen3-30B-A3B uses $N_{\min} = 64$ in the main TP = EP = 8 experiments. Accuracy ablations use the condition-specific routers defined in Appendix H.1

prompt, performs generation with the configured output cap, and reports completion events back to LoadGen.

Dataset splits. Throughput and latency rows for GPT-OSS-20B, Qwen3-30B-A3B, and GPT-OSS-120B use the same fixed 6,396-sample PerformanceOnly query table. This table is the GPT-OSS v4 performance split: synthetic PubMed-summarization prompts, with maximum input sequence length 15,330 and maximum output sequence length 10,240. The prompt set is tokenized or encoded for the target model/runtime before serving, and the query table is held fixed across methods within each model comparison. The main redirect-feasibility accuracy experiments are separate from these serving runs: they use AIME25 and GPQA-Diamond as described in Table 5 and Appendix H. The GPT-OSS MLPerf accuracy split contains 4,395 samples from AIME, GPQA, and LiveCodeBench, but it is not the source of the Server throughput numbers.

Server scenario. The Server scenario emulates an online service. LoadGen issues individual requests according to its Server arrival schedule at the target QPS (QPS = 1 and 2 in the main table; each such cell is a bounded 300-request run that retains LoadGen's INVALID early-stopping label, while the saturated QPS = 4 runs in Appendix I use the full 6,396-sample QSL). Concretely, LoadGen samples inter-arrival times from the Server scenario's Poisson process, so the target QPS controls the mean arrival rate but individual requests still arrive with online-service burstiness. The serving endpoint performs continuous batching over this live request stream. For streaming generation, the SUT reports a first-token event when the first output token is produced and a completion event when the request finishes; LoadGen then computes completed samples/s, output tokens/s, TTFT P99, and TPOT P99. We use the MLPerf token-latency targets TTFT P99 ≤ 3000 ms and TPOT P99 ≤ 80 ms to interpret whether a run is latency-compliant. Within each model comparison, all methods use the same QSL, target-QPS setting, output cap, and LoadGen accounting; only the serving engine or routing policy changes.

Offline scenario. The Offline scenario measures maximum batch-throughput rather than online latency. LoadGen makes the query samples available to the SUT at once, so the endpoint can keep the device saturated using large batches and continuous batching without obeying a target

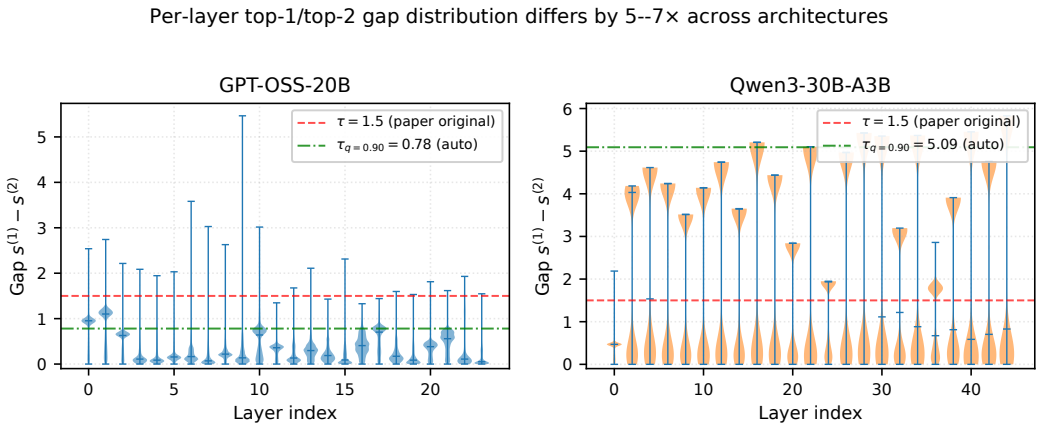

Figure 4: Per-layer top-1/top-2 gap distributions differ substantially across architectures. Online quantile calibration maps the same target q to model-specific thresholds, which is why a single fixed threshold does not transfer across architectures.

arrival rate. In our GPT-OSS-120B PerformanceOnly runs, LoadGen repeats/permutates the 6,396-sample performance QSL to issue 51,168 sample instances in one Offline query. We therefore report samples/s and output tokens/s for Offline rows. TTFT and TPOT are marked n/a in Table 3 because Offline latency percentiles describe a full saturated batch drain rather than per-request online service behavior.

D Hyperparameters and Notation

Table 6 summarizes the controller hyperparameters, ablation names, serving settings, and mathematical symbols used throughout the paper. Unless otherwise stated, BLAZE denotes the full $\text{BLAZE}_{\text{OA+RA}}$ configuration.

E Guardrail Calibration Tradeoffs

BLAZE's confidence guardrail uses the semantic top-1/top-2 gap

$$\Delta_\ell(x) = s_{\ell,e_1}(x) - s_{\ell,e_2}(x) \tag{29}$$

to decide whether the original top-1 expert is too confident to redirect. The throughput-oriented serving path uses a fixed global threshold, but that threshold is selected offline from a target redirection budget rather than chosen as an arbitrary logit-scale constant. We collect a short calibration trace, compute the margin distribution, and choose $\tau = Q_{p_{\text{redir}}}(\Delta)$ so that approximately a fraction p_{redir} of low-margin tokens is exposed to top-1 redirection. The resulting empirical value $\tau = 1.5$ is the fixed threshold used by most of our serving runs. A single fixed value does not transfer across architectures: the same $q = 0.90$ target maps to $\tau \approx 0.78$ on GPT-OSS-20B and $\tau \approx 5.09$ on Qwen3-30B-A3B, so $\tau = 1.5$ lies outside the intended lock band on both. What transfers is the calibration procedure, either the offline quantile fit or the online tracker below.

For comparison, we also implement an online quantile rule. The operator specifies a target quantile q, and the controller sets

$$\tau^{(t)} = Q_q(\{\Delta_\ell(x)\}_{\text{recent}}), \tag{30}$$

where the quantile is computed from recent observed gaps. Since a token is locked when $\Delta_\ell(x) > \tau$, this also exposes a direct policy knob: $q = 0.90$ targets a 10% locked-token fraction, $q = 0.80$ targets 20%, and so on. Figure 4 shows that equal lock-fraction targets can map to very different absolute thresholds across architectures, while Figure 5 illustrates the target-lock-fraction interpretation of the quantile rule.

Table 6: Hyperparameters, ablation names, and notation used in this paper.

Symbol / parameter	Value	Description
BLAZE *Configurations*		
$\text{BLAZE}_{\text{load}}$	load feedback	Expert-load feedback with the confidence guardrail; no overhead-aware gate or rank-aware penalty.
BLAZE_{OA}	load + OA	Adds overhead-aware activation; sets $\alpha_{\text{rank}} = 0$.
$\text{BLAZE}_{\text{OA+RA}}$ / BLAZE	load + OA + RA	Full router used by main BLAZE rows unless otherwise stated.
Feedback Controller		
α	0.15 default	Expert-load EMA penalty gain in Eq. 3. It multiplies a mean-normalized load signal, so the same value is used without per-model retuning on GPT-OSS-20B, GPT-OSS-120B, and Qwen3-30B-A3B.
β	0.85 default	EMA smoothing factor for expert load in Eq. 3.
α_{rank}	0.02 / 0.0025	Rank-aware feedback gain in Eq. 4. 0.02 for the flat GPT-OSS and Qwen3 routers; 0.0025 for grouped DeepSeek-R1 routing; 0 in BLAZE_{OA} ablations. The rank term is kept smaller than α because it only breaks ties among already-ambiguous choices.
$c_{\ell,e}^{(t)}$	measured	Token assignments to expert e in layer ℓ at step t.
$\hat{c}_{\ell,e}^{(t)}$	normalized count	Count normalized by the layer mean assignment count.
$\bar{\ell}_{\ell,e}$	device EMA	Per-layer, per-expert load EMA.
$\bar{\rho}_{\ell,r}$	rank-load EMA	Mean load EMA over experts placed on EP rank r.
Confidence Guardrail		
$s_{\ell,e}(x)$	model router score	Original semantic router score for token x, layer ℓ, expert e.
$\Delta_\ell(x)$	$s_{\ell,e_1}(x) - s_{\ell,e_2}(x)$	Top-1/top-2 semantic gap used by the confidence guardrail.
τ	online q=0.90 recommended	Margin threshold. Online quantile calibration with $q = 0.90$ is the recommended cross-model default. Serving rows use a fixed threshold as the throughput-oriented option ($\tau = 1.5$ unless noted; $\tau = 5.0$ for the Qwen3-30B-A3B row of Table 3).
p_{redir}	calibration target	Target fraction of low-margin tokens made eligible for top-1 redirection in the offline calibration rule $\tau = Q_{p_{\text{redir}}}(\Delta)$.
q	0.90 / 0.80	Quantile target for the online calibration ablation; $q = 0.90$ for the accuracy comparison and $q = 0.80$ in the GPT-OSS-120B throughput sweep.
`gap_hist`	64 bins over $[0, 8]$	Device histogram for recent gap quantiles; halved every 50 update steps.
Overhead-Aware Activation		
N	measured	Effective token count entering the MoE layer.
B_M	128	Backend grouped-GEMM tile-size proxy used by the activation rule.
O_{route}	$[0.25B_M, B_M]$	Overhead-aware activation range in §2.3; the upper end is the one-tile default.
$N_{\min}$	§2.3	Activation threshold. In the reported TP = EP = 8 Server rows, GPT-OSS-20B does not enable the overhead-aware gate; Qwen3-30B-A3B uses 64 from its overhead-aware range; GPT-OSS-120B uses 128 in the throughput setting.
`bucket_up`	$\{32, 64, 128, 256, 512, 1024\}$	CUDA-Graph batch buckets used to round the raw activation threshold upward.
Manual $N_{\min} = 0$	always-on	Ablation that forces redirection for every effective batch.
Model and Serving Notation		
L	model-dependent	Number of MoE layers; GPT-OSS-20B uses 24, GPT-OSS-120B uses 36, and Qwen3-30B-A3B uses 48.
E	model-dependent	Number of routed experts: GPT-OSS-20B uses 32; Qwen3-30B-A3B and GPT-OSS-120B use 128; DeepSeek-R1 uses 256.
K	model-dependent	Routing fanout: GPT-OSS models use $K = 4$; Qwen3-30B-A3B and DeepSeek-R1 use $K = 8$.
EP, TP, PP	8, 8, 1 main runs	Main runs use EP = TP = 8 and PP = 1; adaptive-q sweep uses TP = EP = 4.
$\mathcal{S}_\ell(x)$	selected set	Expert set selected for token x at layer ℓ after any redirection.
$w_{\ell,e}(x)$	semantic weights	Mixture weights computed from original router scores, not adjusted scores.
Server QPS	1, 2 (main); 0.55–1.75 sweep	Target QPS for MLPerf-style Server experiments; saturated QPS = 4 runs are in Appendix I.
Output cap	4096	AIME25 and GPQA-Diamond cap for the redirect-feasibility accuracy study.

Global vs. per-layer thresholding. We use an offline-calibrated fixed global threshold as the throughput-oriented default, and evaluate three related variants. *Offline re-fit global* estimates a new scalar threshold from a calibration set. *Offline per-layer* keeps a separate τ_ℓ for each MoE layer, which can track layer-wise heterogeneity but introduces a vector of thresholds that must be stored, validated, and re-estimated when the deployment distribution changes. *Online global* keeps a single running histogram over all sampled layers and recomputes τ during serving. It preserves

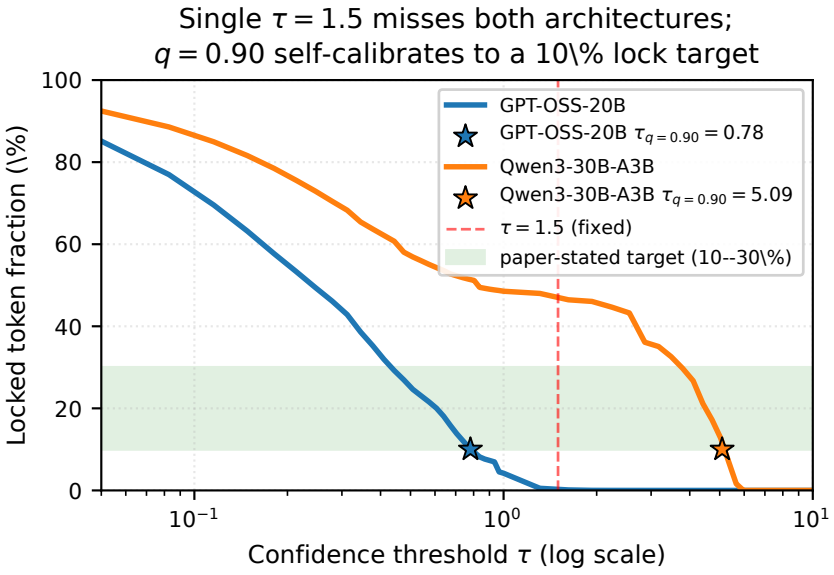

Figure 5: Locked-token fraction as a function of the absolute guardrail threshold. Quantile calibration uses q to target a lock fraction directly; for example, $q = 0.90$ targets roughly 10% locked tokens on both models.

Table 7: Accuracy comparison for confidence-threshold calibration strategies. Values are the accuracies plotted in Figure 6; Δ is relative to fixed $\tau = 1.5$ for the same model.

Strategy	GPT-OSS-20B	Δ	Qwen3-30B-A3B	Δ
Vanilla	35.77	-1.06	35.37	+1.22
Offline fixed global $\tau = 1.5$	36.83	+0.00	34.15	+0.00
Offline re-fit global τ	34.55	-2.28	35.61	+1.46
Offline per-layer τ_ℓ	35.37	-1.46	35.53	+1.38
Online global quantile ($q = 0.90$)	**37.56**	**+0.73**	34.88	**+0.73**

one user-facing knob q, adapts to live traffic, and avoids maintaining per-layer calibration artifacts, but it also adds a small online-control path and does not maximize throughput in our serving sweep.

Table 7 reports the accuracy comparison summarized in Figure 6. Offline per-layer and online global quantile calibration are useful diagnostics for the accuracy/portability tradeoff. Online $q = 0.90$ is the only strategy that improves accuracy on both models (+0.73 pp each), whereas fixed $\tau = 1.5$ costs 1.22 pp relative to vanilla on Qwen3-30B-A3B. We therefore recommend online $q = 0.90$ as the cross-model default. The fixed threshold remains a throughput-oriented option because it avoids online updates and has the best throughput in Table 8.

Online implementation. The CUDA path keeps a single global 64-bin histogram of recent $\Delta_\ell(x)$ values over the range $[0, 8]$. The routing kernel can subsample tokens and layers before issuing the histogram `atomicAdd`; the controller update step launches a one-block quantile kernel that writes the current τ into `d_params`. The histogram is halved every 50 update steps by default, giving the estimate a sliding-window behavior. Because the routing kernel reads τ from device memory, changing τ does not require CUDA Graph re-capture.

Automated q search and throughput. We then swept q and the update interval on GPT-OSS-120B Offline to quantify the throughput cost of online calibration. Table 8 reports the completed valid harness runs. The adaptive configurations preserve most of the fixed-threshold throughput gain while avoiding the need to reuse the same empirical τ across deployments. However, the

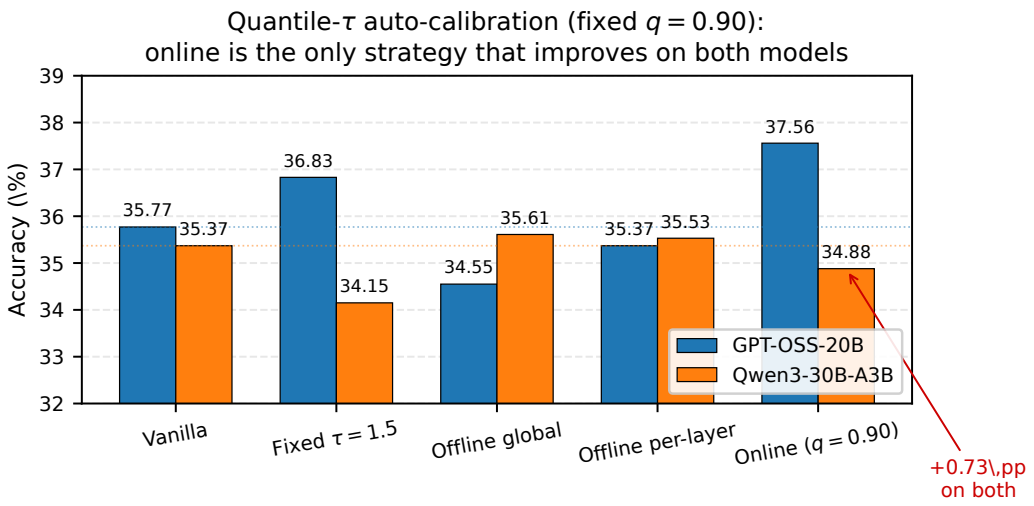

Figure 6: Accuracy comparison of fixed τ, offline global calibration, offline per-layer calibration, and online quantile calibration. Online quantile calibration improves accuracy over fixed $\tau = 1.5$ on both models in this comparison and is the recommended default, while Table 8 shows the throughput advantage of the fixed-threshold option.

Table 8: Guardrail-throughput sweep on GPT-OSS-120B Offline. All rows are valid PerformanceOnly harness summaries on 8×H100 with TP = EP = 4 and the TRITON MoE backend. The update interval is measured in controller update steps.

Guardrail	q	Update interval	Samples/s	Tokens/s	Δ vs. no router
No router	–	–	13.7373	17678.2	+0.0%
Offline fixed $\tau = 1.5$	–	–	14.0897	18525.6	+4.8%
Online quantile	0.70	10	13.7700	18068.6	+2.2%
Online quantile	0.75	10	13.8061	18155.8	+2.7%
Online quantile	0.80	10	13.8771	18215.7	+3.0%
Online quantile	0.80	20	13.8986	18223.2	+3.1%
Online quantile	0.80	50	13.9821	18342.2	+3.8%
Online quantile	0.85	10	13.8594	18229.5	+3.1%

fixed global threshold has the highest throughput in this sweep: fixed $\tau = 1.5$ improves tokens/s by 4.8% over the no-router baseline, while the best online setting ($q = 0.80$ with update interval 50) improves tokens/s by 3.8%. The fixed threshold is therefore the throughput option, and online quantile calibration is the accuracy-oriented, portable default.

F Baseline Method Notes

TRT-LLM EPLB on GPT-OSS-20B. EPLB was attempted with both TRT-LLM 1.3.0rc8 and rc13, but neither produced a valid Server run due to an integration bug in the GPT-OSS MoE path/model wiring. The row is therefore marked **n/a**.

Capacity-Aware on Qwen3-30B-A3B. Capacity-Aware's Python routing path issues a dynamic-shape `nonzero()` on overload masks, which is incompatible with CUDA Graph capture on WIDEEP. Disabling CUDA Graph and the autotuner produces a working off-graph run, but that is not a fair comparator against a CUDA-Graph-on BLAZE configuration. This is a structural property of Python-side capacity routing rather than a TRT-LLM version bug, and it motivates BLAZE's graph-safe CUDA-kernel design (Appendix C).

Libra and SGLang rows. Libra (Yang et al., 2026) predicts next-layer expert activations and combines expert replication with adaptive token sharding to balance prefill. Table 9 contrasts its mechanism with BLAZE. The Libra rows in Table 19 run the authors' released supplementary SGLang patch, not our reimplementation. The released Libra patch targets Qwen/GLM-style SGLang MoE paths; a GPT-OSS-20B Libra baseline would require a separate SGLang model-path port, so Libra is reported only for Qwen3-30B-A3B. SGLang rows disable CUDA Graphs because Libra's published scripts pass `-disable-cuda-graph` and its `forward_mid` path uses static symmetric-memory `topk_ids` buffers whose shapes do not match the variable effective shapes of our continuous-batching trace. Under our workload the fast path therefore never activated, and Libra-on matched SGLang vanilla (0.51 req/s). This is an integration boundary of the released artifact under our workload, not evidence that BLAZE outperforms Libra in its intended setting. Porting Libra to TensorRT-LLM/WIDEEP with CUDA Graphs is possible in principle, but it would require rebuilding the activation predictor, replica placement, token sharding, computation overlap, and the dynamic buffer and graph lifecycle, so the result would measure our implementation choices rather than published Libra.

Table 9: Mechanism-level contrast between Libra and BLAZE.

Axis	Libra (Yang et al., 2026)	BLAZE
Balancing lever	Predicted execution plan, replication, token sharding	Online load penalty in router selection
Expert weights	Additional exact replicas	No replicas ($E_{\text{slots}} = E$)
Semantic effect	Original expert identity preserved via a replica	Only low-margin choices may redirect; mixture weights use original scores
Runtime target	Published SGLang prefill path	Graph-captured TRT-LLM/WIDEEP continuous batching

G ADDITIONAL ACTIVATION-GATE RESULTS

This appendix gives the concrete protocols behind the overhead-aware activation experiment in §3.3. The DeepSeek-R1 ablation uses a 600-prompt endpoint microbenchmark with client concurrency 256, a 4000-token generation cap, and seed 7. We compare always-on redirection ($N_{\min} = 0$) against the overhead-aware gate used in the ablation ($N_{\min} = 256$). The "small-batch" rows refer to decode-sized effective MoE calls observed during the benchmark; they are not a separate low-concurrency client setting. The concurrency-256 rows report the same serving regime when larger prefill/chunked-prefill MoE calls can amortize the router-side balancing cost.

G.1 DEEPSEEK-V4-PRO PREFILL-HEAVY PROTOCOL

Table 12 is a diagnostic LoadGen Server run on a single 8×H200-SXM node using vLLM with data parallel size 8 and expert-parallel routing enabled. The model is `deepseek-ai/DeepSeek-V4-Pro`. The endpoint uses `max_model_len=4096`, `max_num_seqs=512`, `max_num_batched_tokens=512`, an FP8 KV cache, and `enforce_eager`. LoadGen drives the OpenAI-compatible completion endpoint using the same Server-style Poisson arrival process as Appendix C.2, with target QPS 1.2. This is a short matched-trace diagnostic rather than an official MLPerf minimum-query-count run; both rows use the same request trace and LoadGen latency accounting. Prompts are token IDs from the DeepSeek-R1 evaluation QSL `mlperf_deepseek_r1_dataset_4388_fp8_eval.pkl`, truncated to 3140 input tokens, and the requested output length is 2 tokens. This makes the run intentionally prefill-heavy: TTFT and full request latency primarily measure the long-prefill path, while TPOT is reported for completeness.

The baseline row is the same vLLM endpoint with BLAZE disabled. The BLAZE row enables the router with $N_{\min} = 256$, $\alpha = 0$, $\alpha_{\text{rank}} = 0.0002$, $\tau = 0.15$, and $\beta = 0.85$. Full latency is measured from LoadGen issue time to request completion. This diagnostic is an end-to-end mechanism check for the prefill regime, not an official MLPerf submission row.

Table 10: Router-controller overhead. One forward includes all MoE layers plus the BLAZE update step.

Model	Extra time / forward	Extra time / token / layer at 4096 tokens
GPT-OSS-20B	267–288 μs	2.73 ns
Qwen3-30B-A3B	503–592 μs	3.01 ns

Table 11: DeepSeek-R1 overhead-aware activation microbenchmarks. All rows use 600 prompts, client concurrency 256, a 4000-token generation cap, and seed 7; `min_batch` is the BLAZE activation threshold $N_{\min}$.

Condition	min_batch	α_{rank}	Output tok/s	TPOT mean
Vanilla, small-batch run	–	0	623.99	68.24 ms
Always-on router	0	0	546.12	79.16 ms
Overhead-aware gate	256	0	654.36	68.18 ms
Vanilla, concurrency 256	–	0	2517.3	68.80 ms
BLAZE_{OA}	256	0	2623.4	69.00 ms
$\text{BLAZE}_{\text{OA+RA}}$	256	0.0025	2702.9	69.04 ms

H ADDITIONAL ACCURACY RESULTS

This appendix reports the GPT-OSS-120B MLPerf accuracy summary and the underlying numerics for the redirect-feasibility study of §3.2 (AIME25 and GPQA-Diamond). Unless otherwise noted, the redirect-feasibility runs use greedy decoding (`temperature = 0`) and `max_output_tokens = 4096` on the same TP = EP = 8 H200 deployment described in Table 5. LiveCodeBench V6 is included only in the separate GPT-OSS-120B MLPerf accuracy summary in Table 14; we do not use it for the GPT-OSS-20B/Qwen3-30B-A3B redirection-feasibility comparison.

H.1 ACCURACY PROTOCOL AND ROUTING CONDITIONS

The redirect-feasibility accuracy experiment is an offline generation study, not an MLPerf Server throughput run. For each model and routing condition, we run the same AIME25 and GPQA-Diamond prompts through the TensorRT-LLM endpoint, decode greedily, cap generation at 4096 tokens, and then apply the same task-specific answer extraction and scoring scripts. The reported overall score is sample-weighted over AIME25's 240 examples and GPQA-Diamond's 990 examples. The routing intervention is applied only inside MoE expert selection; model weights, expert kernels, prompt formatting, and the answer scorer are unchanged across conditions.

We use the following condition names in Table 13 and Table 16.

- **Vanilla.** The model's native Top-K router is used without any runtime load feedback or expert replacement.
- **Random Top-k.** A destructive stress test that replaces the semantic Top-K expert set with K randomly selected distinct experts for each routed token. This tests whether routed experts can be treated as interchangeable capacity slots.
- **Worst-k.** An adversarial stress test that selects the lowest-scoring K experts under the original router scores. This provides a lower-bound sanity check that the router scores carry semantic signal.
- **Top-1 fixed + random fallback.** The original semantic top-1 expert is preserved, but the remaining $K - 1$ fallback experts are selected randomly from the other experts. This isolates whether protecting only the top-1 expert is sufficient for accuracy.

Table 12: DeepSeek-V4-Pro prefill-heavy vLLM diagnostic on 8×H200. LoadGen uses long-prefill, two-output-token requests at target QPS 1.2. The BLAZE row uses $N_{\min} = 256$.

Metric	Baseline	BLAZE	Change
Samples/s	1.12	1.13	+0.9%
Full latency P99	5.17 s	2.95 s	-42.9%
TTFT P99	4.95 s	2.49 s	-49.7%
TPOT P99	444.02 ms	416.93 ms	-6.1%

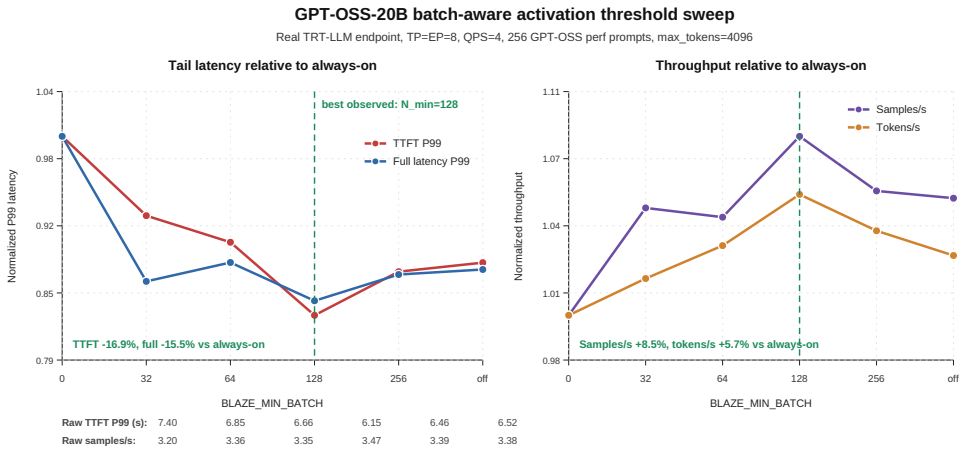

Figure 7: GPT-OSS-20B overhead-aware activation-threshold diagnostic. We sweep `BLAZE_MIN_BATCH` on a real TensorRT-LLM endpoint with TP = EP = 8, QPS = 4, 256 GPT-OSS performance prompts, and a 4096-token output cap. This quick mechanism run is not the reported GPT-OSS-20B Server configuration; it shows that a moderate activation threshold can avoid always-on router overhead while retaining useful load-balancing gains. In this sweep the best observed point is $N_{\min} = 128$, improving samples/s by 8.5% and output tokens/s by 5.7% relative to always-on routing.

- **Capacity-Aware.** A score-thresholded capacity-aware routing baseline with token-drop strength $\gamma = 1.0$. When an expert exceeds its capacity, the lowest-scored assignments to that expert are removed and the remaining weights are renormalized. We include this row for GPT-OSS-20B; the same accuracy-cycle run was not completed for Qwen3-30B-A3B.
- BLAZE$_{\text{load}}$. The confidence-gated expert-load-feedback router. High-margin semantic top-1 choices are locked, low-margin choices and fallback slots can be redirected by the expert-load penalty, and mixture weights are still computed from the original semantic scores. This ablation disables overhead-aware activation and rank-aware feedback.
- BLAZE$_{\text{OA+RA}}$. The full router used as BLAZE in the main paper unless otherwise stated. It adds the overhead-aware activation gate and rank-aware feedback to BLAZE$_{\text{load}}$.

H.2 GPT-OSS-120B MLPERF ACCURACY

Table 14 reports the per-execution MLPerf v6.0 accuracy results summarized in Table 2. All executions use the same checkpoint, full 4,395-sample dataset, QSL setup, Offline AccuracyOnly scenario, decoding configuration (high reasoning, temperature 1.0, top-p 1.0, top-k 0, 32,768-token cap), and 8×H200 TP = EP = 8 runtime; the router is the only intended condition change, and every execution returns all 4,395 scored generations. The composite score follows the MLPerf task-score aggregation. At matched request IDs, BLAZE runs 1/2/3 versus vanilla have 164/170/197 wins, 150/137/173 losses, and 4,081/4,088/4,025 unchanged outcomes. Their QSL-aligned, base-item-

Table 13: Dataset-level accuracy breakdown.

Model	Condition	AIME25	GPQA-Diamond	Overall
GPT-OSS-20B	Vanilla	35.15%	58.50%	53.95%
GPT-OSS-20B	Capacity-Aware	37.92%	57.63%	53.78%
GPT-OSS-20B	Random Top-k	0.83%	8.48%	6.99%
GPT-OSS-20B	Worst-k	0.00%	5.76%	4.63%
GPT-OSS-20B	Top-1 fixed + random fallback	12.18%	33.87%	29.67%
GPT-OSS-20B	BLAZE$_{\text{load}}$	36.40%	55.81%	52.04%
GPT-OSS-20B	BLAZE$_{\text{OA+RA}}$	36.13%	56.28%	52.37%
Qwen3-30B-A3B	Vanilla	17.50%	51.41%	44.80%
Qwen3-30B-A3B	Random Top-k	0.00%	0.91%	0.73%
Qwen3-30B-A3B	Worst-k	0.00%	0.20%	0.16%
Qwen3-30B-A3B	Top-1 fixed + random fallback	0.00%	13.74%	11.06%
Qwen3-30B-A3B	BLAZE$_{\text{load}}$	15.42%	50.20%	43.41%
Qwen3-30B-A3B	BLAZE$_{\text{OA+RA}}$	16.67%	49.70%	43.25%

clustered 95% intervals are $[-0.48, +0.94]$, $[-0.10, +1.60]$, and $[-0.38, +1.27]$ pp. Because the H200 configuration is our local adaptation, these are accuracy controls rather than formal MLPerf submissions.

Table 14: GPT-OSS-120B MLPerf v6.0 accuracy per execution (4,395 samples; AIME25 30×8, GPQA-Diamond 198×5, LiveCodeBench-V6 1,055×3). All executions exceed the 82.30% threshold.

Condition	AIME25	GPQA-Diamond	LiveCodeBench-V6	MLPerf score
Vanilla	196/240 (81.67%)	752/990 (75.96%)	2,687/3,165 (84.90%)	83.44%
BLAZE run 1	203/240 (84.58%)	754/990 (76.16%)	2,692/3,165 (85.06%)	83.67%
BLAZE run 2	206/240 (85.83%)	747/990 (75.46%)	2,715/3,165 (85.78%)	84.19%
BLAZE run 3	208/240 (86.67%)	751/990 (75.86%)	2,700/3,165 (85.31%)	83.88%
BLAZE mean ± SD	85.69 ± 1.05%	75.83 ± 0.36%	85.38 ± 0.37%	83.91 ± 0.26%
Mean − vanilla	+4.03 pp	−0.14 pp	+0.48 pp	+0.47 pp

H.3 RUN-TO-RUN VARIATION AND SCALE CHECKS

Request-order variation. The redirect-feasibility protocol is greedy, so a sampling seed has no effect. BLAZE is nevertheless stateful across concurrent requests because its load EMA evolves with request order, so request order is the relevant source of run-to-run variation. We run three deterministic request-order permutations per condition on a fixed 200-prompt subset (100 AIME25 + 100 GPQA-Diamond; greedy; 4,096-token cap). All 12 cells return 200/200 scored responses. Table 15 shows that both mean gaps are far smaller than the order-induced standard deviation and have opposite signs. This is a subset study with three permutations, so it bounds rather than resolves run-to-run variation.

Table 15: Request-order robustness on a 200-prompt subset (three permutations; mean ± SD).

Model	Vanilla (%)	BLAZE (%)	Paired vanilla−BLAZE (pp)
GPT-OSS-20B	47.17 ± 3.33	48.00 ± 4.09	−0.83 ± 6.83
Qwen3-30B-A3B	36.17 ± 2.75	36.00 ± 0.50	+0.17 ± 3.21

GPT-OSS-120B under the redirect-feasibility protocol. On the full 1,230-prompt greedy protocol with a 4,096-token cap, vanilla scores 61.71% (759/1,230) and BLAZE 61.63% (758/1,230), a gap of −0.08 pp with 110 wins, 111 losses, and 1,009 unchanged outcomes (AIME25 42.92/42.08%, GPQA-Diamond 66.26/66.36%).

Qwen3-235B-A22B scale check. A paired Qwen3-235B-A22B control (94 layers, 128 experts, top-8; 200 prompts; greedy) is exactly tied at 33.0% (66/200 each) with 19 wins and 19 losses. We treat this 200-prompt check as supporting evidence only; we do not report Server-scenario throughput for this model.

Same configuration on H100 and H200. With the same controller settings ($\alpha = 0.15$, $\tau = 1.5$, $\beta = 0.85$) and no hardware-specific retuning, GPT-OSS-120B AIME25 (30×8 generations, MLPerf methodology) scores 81.67% (vanilla) versus $85.69 \pm 1.05\%$ (BLAZE, three runs) on H200, and 83.75% versus 84.58% (one paired run with identical per-request seeds) on H100. We do not interpret these as significant improvements; they show no observed regression when the same configuration is reused across the two evaluated GPU types.

H.4 AIME25 AND GPQA-DIAMOND: PER-CONDITION TABLE

Table 16 reports the per-dataset accuracy of every routing condition used in §3.2, on both architectures. The Capacity-Aware row (with token-drop strength γ=1.0) is included on GPT-OSS-20B as an additional baseline; it was not run on Qwen3-30B-A3B in this evaluation cycle.

Table 16: Accuracy on AIME25 (240 problems) and GPQA-Diamond (990 problems) under each routing condition. "Top-1 + random" is "Top-1 fixed, $k-1$ random fallback"; $\text{BLAZE}_{\text{load}}$ is the load-aware load-penalty routing without the overhead-aware gate; $\text{BLAZE}_{\text{OA+RA}}$ adds the overhead-aware gate and rank-aware penalty. "Capacity-Aware (γ=1)" refers to the score-thresholded capacity-aware drop baseline.

	GPT-OSS-20B		Qwen3-30B-A3B	
Condition	AIME25	GPQA-D.	AIME25	GPQA-D.
Vanilla	35.15	58.50	17.50	51.41
Random Top-k	0.83	8.48	0.00	0.91
Worst-k	0.00	5.76	0.00	0.20
Top-1 + random fallback	12.18	33.87	0.00	13.74
Capacity-Aware (γ=1)	37.92	57.63	—	—
$\text{BLAZE}_{\text{load}}$	36.40	55.81	15.42	50.20
$\text{BLAZE}_{\text{OA+RA}}$	36.13	56.28	16.67	49.70

The accuracy gap between BLAZE and Vanilla is at most 2.71 pp (GPT-OSS-20B GPQA-Diamond) and at most 1.71 pp (Qwen3-30B-A3B GPQA-Diamond), while the random- and worst-k baselines collapse to single-digit accuracy on both benchmarks. The Top-1-fixed-but-random-fallback condition recovers some accuracy but still leaves a 24–34 pp deficit relative to Vanilla, showing that protecting the top-1 expert alone is insufficient.

H.5 LIVECODEBENCH V6 SCOPE

LiveCodeBench V6 (LCB) appears in this manuscript only as part of the GPT-OSS-120B MLPerf accuracy split reported in Table 14. We do not report an LCB redirection-feasibility ablation for GPT-OSS-20B or Qwen3-30B-A3B, so LCB is not included in the main accuracy-preservation claims in §3.2. This avoids mixing the GPT-OSS-120B MLPerf accuracy summary with the separate AIME25+GPQA-Diamond router-intervention study.

H.6 MEMORY-OVERHEAD DERIVATION

For a SwiGLU MoE expert, the expert MLP stores gate, up, and down projection weights. With hidden size d_{hidden}, intermediate size $d_{\text{intermediate}}$, and b bytes per stored weight, one expert in one MoE layer therefore requires $3d_{\text{hidden}}d_{\text{intermediate}}b$ bytes. If a replication-based balancer replaces the original E experts with E_{slots} physical expert slots per layer, the additional weight storage across L MoE layers is

$$3 \cdot d_{\text{hidden}} \cdot d_{\text{intermediate}} \cdot (E_{\text{slots}} - E) \cdot L \cdot b. \tag{31}$$

Under expert parallelism, expert weights are sharded across the EP group, so the per-rank cost is this total divided by the EP world size. Table 18 uses this formula for the EPLB rows. BLAZE has

Algorithm 2 Overhead-aware activation-threshold resolution

Require: Total experts E, EP world size EP, routing fanout K, backend tile size B_M, bucket set $\mathcal{B} = \{32, 64, 128, 256, 512, 1024\}$, floor N_{floor}, cap N_{cap}, user setting u
Ensure: Activation threshold $N_{\min}$ and validation band $[N_{\min}^{\text{low}}, N_{\min}^{\text{default}}]$
if u is an integer **then**
  {Manual override; $u = 0$ means always-on redirection}
  return u
end if
$E_{\text{local}} \leftarrow E/\text{EP}$
{Expected assignments per local expert are $N \cdot K/E_{\text{local}}$}
$O_{\text{low}} \leftarrow 0.25 B_M$
$O_{\text{default}} \leftarrow B_M$
$N_{\min}^{\text{low}} \leftarrow \text{clip}_{[N_{\text{floor}}, N_{\text{cap}}]}(\text{bucket_up}(O_{\text{low}} E_{\text{local}}/K))$
$N_{\min}^{\text{default}} \leftarrow \text{clip}_{[N_{\text{floor}}, N_{\text{cap}}]}(\text{bucket_up}(O_{\text{default}} E_{\text{local}}/K))$
{Default uses one tile; sensitivity sweeps use the band}
return $N_{\min}^{\text{default}}$ and $[N_{\min}^{\text{low}}, N_{\min}^{\text{default}}]$

Table 17: Overall accuracy on AIME25 + GPQA-Diamond. Each non-vanilla condition modifies only expert routing; decoding, data, and endpoint settings are held fixed. BLAZE preserves near-vanilla accuracy while avoiding the catastrophic failures of arbitrary expert replacement.

Routing condition	GPT-OSS-20B	Δ	Qwen3-30B-A3B	Δ
Vanilla	53.95%	+0.00 pp	44.80%	+0.00 pp
Capacity-Aware	53.78%	-0.17 pp	n/a	n/a
Random Top-k	6.99%	-46.96 pp	0.73%	-44.07 pp
Worst-k	4.63%	-49.32 pp	0.16%	-44.63 pp
Top-1 fixed + random fallback	29.67%	-24.29 pp	11.06%	-33.74 pp
$\text{BLAZE}_{\text{load}}$	52.04%	-1.92 pp	43.41%	-1.38 pp
$\text{BLAZE}_{\text{OA+RA}}$	**52.37%**	**-1.59 pp**	**43.25%**	**-1.54 pp**

$E_{\text{slots}} = E$ because it does not replicate experts; its persistent state is the controller buffer footprint reported in Appendix C.

I SATURATED SERVER DIAGNOSTICS (QPS = 4)

Table 19 and Figure 8 report MLPerf-style Server runs at target QPS = 4 on the full 6,396-sample PerformanceOnly QSL. At this load every method violates the TTFT target, so these runs characterize overload behavior rather than an SLO-compliant deployment point; the main-text comparison uses SLO-feasible load instead (Table 3). Under saturation, BLAZE improves completed samples/s by 18.6% on GPT-OSS-20B and 29.0% on Qwen3-30B-A3B and is the only GPT-OSS-20B configuration that meets the TPOT target. Capacity-Aware's apparent $+5.1\%$ tokens/s over vanilla on GPT-OSS-20B is a queueing artifact: its TPOT P99 of 897.39 ms exceeds the 80 ms target by more than $11\times$, so LoadGen counts throughput only over the small fraction of samples that reached the served window. Capacity-Aware drops the lowest-scored tokens at overloaded experts, while BLAZE preserves every token by redirecting only ambiguous choices. The Libra/SGLang rows are discussed in Appendix F.

J GPT-OSS-20B SERVER QPS SWEEP

Table 20 reports GPT-OSS-20B Server cells from SLO-feasible load to the SLO boundary, with TP = EP = 8 and CUDA Graphs enabled. Each cell contains 300 requests with zero transport errors. All cells in this sweep share one request protocol whose outputs are shorter (about 1,100 completed tokens per request) than those of the Table 3 protocol (about 2,300–2,800), so throughput is comparable within this table but not with Table 3. The QPS = 1.0 BLAZE cell is the default setting

Table 18: Per-rank GPU-memory overhead of routing-controller methods evaluated in this paper. "Extra slots" is per MoE layer. GPT-OSS-20B is reported in both MXFP4 (the model's native expert-weight precision) and a bf16 equivalent for comparison with bf16 baselines. BLAZE's 37 KB persistent state is below the table's GiB resolution. The 0.844 GiB figure is the EPLB footprint on Qwen3-30B-A3B; we do not measure Libra's footprint, which depends on its replica plan.

Method (Model)	Extra slots/layer	Extra weight bytes	Per-GPU (EP=8)
BLAZE (any model)	0	0	≈ 0
EPLB (Qwen3-30B-A3B, bf16)	16	6.75 GiB	0.844 GiB
EPLB (GPT-OSS-20B, MXFP4)	8	2.22 GiB	0.278 GiB
EPLB (GPT-OSS-20B, bf16-eq.)	8	8.90 GiB	1.11 GiB
Libra (Qwen3-30B-A3B)	redundant replicas	+ replica weights	+ replica weights
	+ symmetric-memory `topk_ids_buf` sized by SEQ_LENS_SUM×topk		

Table 19: Saturation diagnostic: MLPerf-style Server comparison at QPS=4 with TP = EP = 8. Vanilla, Capacity-Aware, EPLB, and BLAZE use TensorRT-LLM unless noted. For Libra, we report both SGLang vanilla (Libra disabled) and Libra-on rows on Qwen3-30B-A3B.

Model	Method	Engine/backend	Graph	Samples/s	Tokens/s	TTFT P99	TPOT P99
GPT-OSS-20B	Vanilla	TRT-LLM / TRITON	ON	3.34	8232.71	262.83 s	1181.72 ms
GPT-OSS-20B	Capacity-Aware	TRT-LLM / TRITON	ON	3.49	8650.80†	213.43 s	897.39 ms
GPT-OSS-20B	EPLB	TRT-LLM / TRITON	ON	n/a	n/a	n/a	n/a
GPT-OSS-20B	BLAZE	TRT-LLM / TRITON	ON	**3.96**	8581.93	**15.41 s**	**28.29 ms**
Qwen3-30B-A3B	Vanilla	TRT-LLM / WIDEEP	ON	2.24	8409.33	1134.76 s	404.88 ms
Qwen3-30B-A3B	Capacity-Aware	TRT-LLM / WIDEEP	ON	n/a	n/a	n/a	n/a
Qwen3-30B-A3B	EPLB	TRT-LLM / WIDEEP	ON	2.22	8557.00	1152.90 s	402.10 ms
Qwen3-30B-A3B	SGLang vanilla	SGLang	OFF	0.51	752.76	10643.77 s	87.78 ms
Qwen3-30B-A3B	Libra (released artifact)	SGLang	OFF	0.51	751.62	10637.68 s	88.32 ms
Qwen3-30B-A3B	BLAZE	TRT-LLM / WIDEEP	ON	**2.89**	**9277.42**	**639.28 s**	**278.94 ms**

Notes. **n/a**: method did not produce a valid Server result. † Queueing artifact: throughput is counted only over samples that reached the SUT before cutoff. Libra's fast path did not activate under this continuous-batching trace, so the Libra row matches SGLang vanilla; this is an integration diagnostic, not a comparison in Libra's intended setting. SGLang rows disable CUDA Graphs; all TRT-LLM rows keep them enabled.

of the gain sweep in Table 23. Across the matched points from QPS = 0.55 to 0.70, BLAZE improves completed-token throughput by 1.7–4.4% over vanilla and lowers TTFT P99 at every point; at QPS = 0.65, BLAZE passes while vanilla does not. At QPS = 1.25, 1.50, and 1.75, BLAZE passes with TTFT P99 of 2.548, 2.703, and 2.804 s, whereas Capacity-Aware fails with 3.223, 3.767, and 3.478 s. Within this protocol, Capacity-Aware passes at QPS = 1.0, so its observed SLO boundary lies between 1.0 and 1.25 QPS, whereas BLAZE remains compliant at least up to 1.75 QPS; we did not run this protocol beyond 1.75. Because of repeat-level P99 variation and some non-monotonicity (vanilla fails at 0.65 but passes at 0.70), we treat these as observed rather than deterministic boundaries. The sweep covers only GPT-OSS-20B; for Qwen3-30B-A3B we have a single SLO-feasible point (Table 3).

K LATENCY ANALYSIS

Phase breakdown. Table 21 compares engine-side P99 timers for GPT-OSS-20B at QPS = 1 (299 accepted requests per method). Means also fall (server TTFT 164.68→136.29 ms; context forward 69.91→63.98 ms; decode step 5.50→5.25 ms). One field worsens: postprocessing/first-token delivery P99 rises from 18.05 to 211.94 ms (mean 3.91→5.66 ms), yet total TTFT P99 still falls; P99 components are not additive. These timers localize the gain to context/prefill forwards and iteration tails without attributing it to a specific All-to-All or GEMM kernel. The decode-step figure is a correlated engine metric over different generated-step counts, not evidence of rerouting at every decode step. Endpoint TTFT starts at request receipt, whereas LoadGen TTFT starts at scheduled arrival and includes pre-admission waiting, hence the difference in scale from Table 3. This breakdown covers only GPT-OSS-20B and compares BLAZE with vanilla.

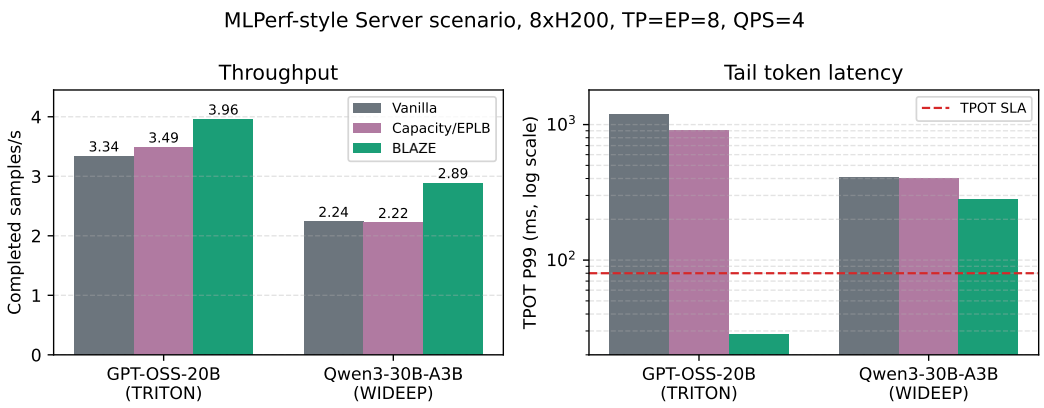

Figure 8: Saturation diagnostic: MLPerf-style Server comparison on 8×H200, TP = EP = 8, QPS=4. Left: completed samples/s. Right: TPOT P99 on a log scale; the dashed red line marks the 80 ms Server service target. At this load all methods violate the TTFT target.

Table 20: GPT-OSS-20B QPS sweep under a single shorter-output request protocol (not comparable in throughput with Table 3). Entries are completed tokens/s / TTFT P99 (ms) / TPOT P99 (ms), followed by SLO status; "–" marks cells not run under this protocol.

Target QPS	Vanilla	Capacity-Aware	BLAZE
0.55	578 / 2494 / 7.78, pass	586 / 2614 / 7.51, pass	593 / 2327 / 7.30, pass
0.60	639 / 2553 / 7.49, pass	638 / 2752 / 7.65, pass	667 / 2465 / 7.59, pass
0.65	691 / 3154 / 8.23, fail	691 / 2651 / 7.66, pass	713 / 2509 / 7.72, pass
0.70	742 / 2668 / 7.70, pass	743 / 2992 / 7.93, pass	755 / 2312 / 7.78, pass
1.00	–	1064 / 2682 / 8.32, pass	1067 / 2476 / 8.46, pass
1.25	–	TTFT 3223, fail	TTFT 2548, pass
1.50	–	TTFT 3767, fail	TTFT 2703, pass
1.75	–	TTFT 3478, fail	TTFT 2804, pass

Controlled fixed-work analysis. On the same stack at QPS = 1.0, we fix every response to exactly 128 output tokens and use identical prompts, request IDs and order, and Poisson arrival schedule (seed 240731). We run two counterbalanced repetitions (Shadow→Active→Vanilla and Vanilla→Active→Shadow); all six cells complete 300/300 requests with zero transport or output-length errors. Vanilla disables BLAZE; Shadow runs the production CUDA controller with zero penalties ($\alpha = \alpha_{\text{rank}} = 0$), and a separate exact-route gate confirms that it preserves the complete semantic Top-K set for 100% of routed tokens; Active uses $\alpha = 0.15$, $\alpha_{\text{rank}} = 0.02$, $\tau = 1.5$, and $\beta = 0.85$. Table 22 reports two-level paired-bootstrap 95% intervals over the repetitions and matched requests. For Active−Vanilla, the TTFT/TPOT/E2E P99 differences are $-1.86/+0.08/-12.09$ ms, and their intervals also include zero.

Prefill-dominant run. Capping the output at one token removes autoregressive decoding while keeping first-token generation. At client concurrency 32 on the same 1,230 AIME25+GPQA-Diamond prompts, GPT-OSS-20B request throughput rises from 114.10 to 164.57 req/s and prompt throughput from 31,960 to 46,097 tok/s (+44.2%). This run covers only GPT-OSS-20B; the end-to-end Server results in Table 3 remain the deployment-relevant evidence. Appendix G.1 gives a complementary prefill-heavy DeepSeek-V4-Pro diagnostic.

L GAIN SENSITIVITY AND RANK FEEDBACK

Selection of α and α_{rank}. The expert gain α multiplies a mean-normalized EMA load signal, so its magnitude is normalized across expert counts; we use $\alpha = 0.15$ and $\beta = 0.85$ without per-

Table 21: GPT-OSS-20B phase-level P99 latency at QPS = 1.

P99 metric (ms)	Vanilla	BLAZE	Change
Queue	8.28	6.89	-16.8%
Preprocessing	333.29	235.00	-29.5%
Prefill-to-first-token	621.29	356.96	-42.5%
Context GPU forward	421.99	234.96	-44.3%
Decode-step GPU forward	45.11	12.05	-73.3%
Server TTFT	900.78	778.21	-13.6%

Table 22: Controlled fixed-work latency contrasts on GPT-OSS-20B (ms; mean differences with 95% paired-bootstrap intervals).

Contrast	TTFT mean	TPOT mean	E2E mean
Shadow − Vanilla	$+2.339$ [0.921, 3.543]	$+0.044$ [-0.029, 0.118]	$+7.870$ [-0.758, 16.549]
Active − Shadow	-1.811 [-3.355, -0.188]	-0.059 [-0.143, 0.022]	-9.249 [-21.278, 2.087]
Active − Vanilla	$+0.528$ [-1.909, 2.717]	-0.015 [-0.036, 0.006]	-1.380 [-5.782, 3.310]

model retuning on GPT-OSS-20B (E=32, K=4), GPT-OSS-120B (E=128, K=4), and Qwen3-30B-A3B (E=128, K=8). The rank term is deliberately smaller because it should only break ties among already-ambiguous choices. Hardware-dependent kernel amortization is handled separately by $N_{\min}$, which is derived from the grouped-GEMM tile size. We do not claim either gain is universally optimal.

One-factor sweep. Table 23 varies one gain at a time at the SLO-feasible GPT-OSS-20B Server point (target QPS = 1), using the shorter-output protocol of Appendix J. All five unique settings achieve 0.96 req/s and pass both SLOs, and the full token-throughput spread is 1.8%, so the default is not a narrow optimum. These are single-run, 300-request cells on one model, so we do not use their ordering to claim that α_{rank} helps on a lightly loaded, weakly rank-skewed trace.

Table 23: Gain sensitivity on GPT-OSS-20B at QPS = 1 (single runs). Entries list the three settings in order.

Sweep	Completed tok/s	TTFT P99 (ms)	TPOT P99 (ms)
$\alpha \in \{0.05, 0.15, 0.30\}$, $\alpha_{\text{rank}} = 0.02$	1072.6 / 1067.3 / 1084.3	2527.7 / 2475.5 / 2433.5	8.08 / 8.46 / 7.88
$\alpha_{\text{rank}} \in \{0, 0.02, 0.05\}$, $\alpha = 0.15$	1087.0 / 1067.3 / 1074.3	2359.4 / 2475.5 / 2480.5	8.11 / 8.46 / 7.79

Rank-skew stress test. To isolate rank feedback, we construct a fixed-work workload on GPT-OSS-20B using only the pre-redirection semantic Top-K: from AIME25 we select 20 prompts with aggregate semantic rank max/mean 1.779 and rank CV 0.407, repeat each 50 times, and issue the 1,000 one-token requests at concurrency 16. We run three interleaved repetitions per setting; separate observer cells collect routing statistics and are excluded from timing. Table 24 shows that prompt throughput is non-decreasing and TTFT P99 is lower in all three paired repetitions, at the cost of a small P50 increase. The 20-prompt full-generation check (four wins, four losses) is only a no-gross-regression diagnostic. Rank feedback is therefore not claimed to help every naturally balanced trace; it becomes useful when expert-level control alone leaves measurable EP-rank skew.

M BEHAVIOR UNDER INPUT-INDUCED ROUTING SKEW

Domain labels are not a reliable proxy for router skew, so we construct matched lower- and higher-skew buckets using a score fixed before any performance run. For each prompt and layer, we count the original semantic Top-K assignments before redirection. Expert skew is the maximum expert count divided by the mean; rank skew is defined analogously after summing experts on each EP rank. We average each ratio across layers and define $S = \frac{1}{2}(\text{expert max/mean} + \text{rank max/mean})$, so $S = 1$ means uniform load. We deduplicate prompts, optimally length-match 28 AIME25/GPQA-Diamond pairs, and place the smaller-S member of each pair in the lower-skew bucket, keeping the 20 pairs with the largest within-pair difference. Each bucket has 10 AIME and 10 GPQA prompts; mean

Table 24: Rank-skew stress test on GPT-OSS-20B (median over three paired repetitions).

Metric	$\alpha_{\text{rank}} = 0$	$\alpha_{\text{rank}} = 0.02$	Change
Prompt tok/s	24,383.4	24,755.7	$+1.53\%$
TTFT P50 (ms)	115.30	118.03	$+2.37\%$
TTFT P99 (ms)	341.40	289.88	-15.09%
Selected-rank max/mean	1.5001	1.4788	-1.42%
Selected-rank CV	0.2784	0.2677	-3.83%
Accuracy diagnostic	6/20	6/20	tie

input length is 206.70/206.35 tokens and mean S is 2.924/3.057. These are relative strata within a matched cohort, not universal thresholds. No post-redirection or performance statistic enters bucket construction; observers are disabled for performance, and every cell starts a fresh graph-enabled server.

Table 25: Routing response to input-induced skew. E/R denotes expert/rank max-to-mean load.

Bucket	Pre-route E/R	BLAZE-selected E/R	Redirected tokens/slots	Semantic top-1 removed
Lower skew ($S = 2.924$)	4.086 / 1.762	3.429 / 1.596	42.38% / 11.70%	0.56%
Higher skew ($S = 3.057$)	4.310 / 1.803	3.575 / 1.627	44.09% / 12.26%	0.61%

Table 26: Accuracy–serving trade-off by skew bucket (vanilla / BLAZE; 20 prompts per bucket). All four cells pass the request SLOs.

Bucket	Req/s	Output tok/s	TTFT P99 (ms)	TPOT P99 (ms)	Accuracy
Lower skew	0.818 / 0.964 (+17.9%)	1302.3 / 1511.3 (+16.1%)	325.0 / 299.9	4.49 / 4.49	25% / 35%
Higher skew	0.776 / 1.030 (+32.8%)	1145.1 / 1623.8 (+41.8%)	315.7 / 318.7	5.47 / 4.47	45% / 50%

Higher input-induced skew produces modestly more corrective freedom, while the original semantic top-1 is removed for only about 0.6% of tokens in either bucket. Paired BLAZE wins/losses are 3/1 (lower skew) and 3/2 (higher skew). With 20 prompts per stratum we do not claim an accuracy improvement; neither stratum shows aggregate degradation, while the throughput benefit is larger in the higher-skew stratum, at the cost of a small TTFT P99 increase there. Batch size and routing skew are orthogonal: the activation gate decides whether a batch is large enough to amortize control, and within an active batch the expert and rank EMA sets the penalty, so hot experts and ranks receive stronger feedback, whereas a uniform load adds the same offset to every expert and leaves the Top-K unchanged.