ALBUS: NETWORK-AWARE EXPERT LOCALIZATION AND DYNAMIC BALANCING WITH UNIFIED SCHEDULING FOR MOE SERVING

Anonymous Authors
Paper under double-blind review

ABSTRACT

Mixture-of-experts (MoE) models expand the capacity of large language models by increasing the number of expert parameters while activating only a small subset for each token. This sparse activation allows model capacity to grow without a proportional increase in per-token computation. The resulting growth in model size increasingly requires serving systems to distribute experts across multiple nodes, where fast intra-node links and slower inter-node connections form a hierarchical network. Compared with single-node serving, this deployment amplifies cross-node communication costs and GPU load imbalance, limiting the inference efficiency of large MoE models. Existing load-balancing methods alleviate imbalance through redundant expert placement but do not sufficiently integrate hierarchical network topology into their load-balancing optimization. We present Albus, a system that jointly optimizes communication and load balancing for MoE inference on hierarchical networks. Albus introduces expert-locality enhancement and communication-aware load balancing to optimize routing and redundant expert placement. It combines these innovations with predictive scheduling and expert-replica prefetching adapted to hierarchical networks, forming a complete execution pipeline for MoE serving. Evaluations on GLM-4.7 and DeepSeek-R1 show that Albus maintains comparable accuracy to native inference across six benchmarks covering knowledge, reasoning, language understanding, and truthfulness. Compared with native SGLang inference, Albus achieves a $1.71\times$ prefill speedup and $1.56\times$ decode throughput.

1 INTRODUCTION

Mixture-of-experts (MoE) is now a widely used architecture for large language models. Early sparse MoE layers used relatively few experts and selected one or two experts per token. Recent models instead employ hundreds of smaller routed experts and often a larger top-k, the number of experts selected for each token (DeepSeek-AI et al., 2025). Serving these larger models requires distributing expert weights across more GPUs through expert parallelism (EP), which shards experts across GPU ranks and increasingly spans multiple nodes. Meanwhile, a larger top-k creates more token–expert assignments: each selected expert requires dispatching the token to its hosting rank and returning its output for combination. Larger EP deployments spread these assignments across more devices and nodes, while larger top-k increases their number. Together, these trends increase communication traffic and make inference more sensitive to uneven work across ranks, with the busiest rank delaying the entire MoE layer.

MoE system research accelerates inference at several levels. Communication libraries optimize token exchange (Zhao et al., 2025b), while grouped expert kernels enable efficient expert computation (Zhao et al., 2025a); routing and request co-scheduling improve activation locality (Li et al., 2026); caching and offloading manage limited accelerator residency (Kamahori et al., 2025; Xue et al., 2025); and expert placement or replication redistributes work across devices (DeepSeek-AI, 2025). These directions are complementary rather than independent: routing changes the demand to be balanced, placement determines where that demand communicates, and a new placement helps only if its replicas become ready before dispatch. This coupling motivates treating demand shaping, rank-level placement, and replica preparation as one runtime problem.

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.

Serving stacks distribute experts across ranks and use optimized All-to-All communication and grouped expert GEMMs to execute this sparse graph (Hwang et al., 2023; Rajbhandari et al., 2022; Zhao et al., 2025a;b; Zheng et al., 2024b). Once EP spans multiple servers, however, the cluster exposes two qualitatively different communication domains: high-bandwidth NVLink/NVSwitch inside a node and substantially slower InfiniBand or RoCE across nodes. Scaling EP simultaneously places fewer experts on each rank, so one hot expert more directly becomes one hot GPU. Figure 1 shows these coupled effects. From EP8 to EP32, dispatch and combine rise from 22.0% to 48.4% of layer latency on GLM-4.7 and from 22.2% to 43.6% on DeepSeek-R1; the critical-rank expert-compute cost rises from $1.61\times$ to $3.26\times$ and from $1.91\times$ to $4.08\times$ the corresponding rank average. These results show that optimizing token communication or expert computation in isolation is insufficient at large EP.

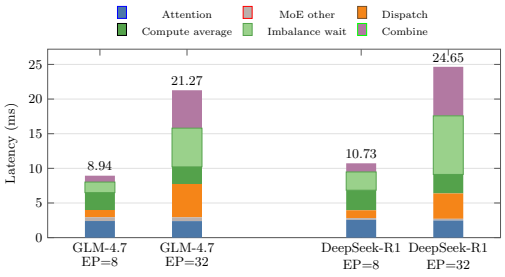


Figure 1: Single-layer inference breakdown for GLM-4.7 and DeepSeek-R1 at EP8 (one node) and EP32 (four nodes), with DP-attention=EP. Compute comprises mean rank time and waiting for the slowest rank.

Existing MoE serving systems use redundant experts to alleviate load imbalance: placing copies of popular experts on additional GPUs allows their token workloads to be shared across ranks. However, existing approaches do not adequately address the combined challenges of large-scale EP inference on hierarchical networks. Static and periodically updated placements rely on historical loads and can lag behind changing demand (DeepSeek-AI, 2025). Predictive systems anticipate next-layer demand and prepare replicas during current-layer execution (Yang et al., 2026), but extending this workflow across nodes requires fitting slower expert transfers into a limited preparation window. Exact-load methods avoid prediction mismatch by using actual gate outputs, yet leave even less time for replica preparation (Chen et al., 2026; Wei et al., 2026). Moreover, balancing work across replicas can send tokens across node boundaries, trading lower rank imbalance for higher communication costs. Thus, redundant experts alone are insufficient: effective load balancing at large EP scales must jointly account for the network hierarchy, token communication, and timely replica preparation. Dynamic predictive systems must also overlap scheduling and replica preparation with a short window of MoE execution, risking contention with the main execution path for GPU streaming multiprocessors (SMs) and NVLink, PCIe, and network interface controller (NIC) bandwidth—a challenge compounded by the multiple transfer paths and slower inter-node links in hierarchical networks.

To address this challenge, Albus provides a cross-node MoE inference framework for hierarchical clusters. It jointly reduces cross-node token communication and rank-level load imbalance by combining locality-aware demand shaping with communication-aware redundant expert placement, while preparing the required expert replicas ahead of use. The framework turns predicted demand into a safe executable layout without replacing the realized router decision, thereby connecting communication reduction, load balancing, and online expert preparation in one serving workflow.

Our contributions are threefold:

1. We design a cross-node MoE inference framework for hierarchical clusters. Its execution pipeline overlaps demand prediction, plan generation, and expert preparation with model execution to avoid exposing additional overhead on the main execution path, while minimizing resource contention to limit interference with foreground computation and communication.
2. We propose a locality-aware dynamic top-k algorithm and a communication-aware load-balancing algorithm to address communication overhead and load imbalance in MoE inference at large EP scales.
3. We evaluate Albus on five MoE models with 30B–671B parameters and show that it can maintain average benchmark quality comparable to the baseline. Compared with native SGLang inference, Albus achieves a $1.71\times$ prefill speedup and $1.56\times$ decode throughput.

2 Related Work

MoE Execution Infrastructure and Data Movement. DeepEP (Zhao et al., 2025b) provides efficient token dispatch and combine for expert parallelism, NCCL supplies collective and point-to-point communication primitives, DeepGEMM (Zhao et al., 2025a) enables efficient expert computation, and SGLang (Zheng et al., 2024b) integrates these capabilities into an inference serving runtime. Albus builds on this infrastructure to optimize routing, redundant expert placement, and replica preparation. Expert prefetching has a different objective from token communication: Albus aims to overlap expert-weight preparation with ongoing computation without interfering with the model's main execution path. This requires keeping the prefetching workload's SM usage low, leaving compute resources available for model execution.

Expert Load Balancing and Placement by Demand Horizon. MoE training uses capacity constraints, auxiliary load-balancing losses, or router-bias updates to balance assignments across experts (DeepSeek-AI et al., 2025; Fedus et al., 2022; Lepikhin et al., 2020), but expert-level balance during training does not ensure balanced physical-rank workloads under inference-time routing distributions. Periodic EPLB (DeepSeek-AI, 2025) collects expert-load statistics over a serving interval and then recalibrates the replica layout; changes in request mix and batch composition can make that layout stale before the next update. Albus instead predicts next-layer expert demand from the preceding layer's hidden states during inference and updates redundant expert placement online. Its decisions therefore reflect the current serving workload rather than training-time or previous-interval statistics, while the lookahead provides time to prepare replicas before use. Libra (Yang et al., 2026) is a dynamic load-balancing system that predicts next-layer expert demand. It combines AllGather-based token transport with local MoE computation to hide scheduling overhead. However, these mechanisms do not readily extend to hierarchical multi-node deployments, and its load-balancing decisions do not explicitly optimize topology-dependent communication costs. Albus targets multi-node MoE inference and jointly optimizes communication and rank-level load balance through redundant expert placement. Exact-load systems remove the prediction gap by making decisions after the true gate. UltraEP (Wei et al., 2026) and MoonEP (Chen et al., 2026) exploit high-bandwidth GPU interconnects to support fine-grained replication and rerouting from this late signal. The placement information is maximally current, but it arrives immediately before dispatch. This timing is viable when all GPUs in the EP group are connected through high-bandwidth links; on conventional multi-node RDMA clusters, cross-node expert-state movement generally cannot be hidden in the same interval. Albus uses one-layer-ahead prediction to provide a longer preparation window for cross-node expert replication over hierarchical networks than is available after gating.

3 Background and Motivation

3.1 MoE Inference under Expert Parallelism

MoE routing. Consider a sparse MoE layer with E routed experts and a nonempty token set $\mathcal{T}$. For token $t \in \mathcal{T}$ and expert $e \in \{1, \dots, E\}$, let $a_{t,e}$ be the router score and k ($1 \le k \le E$) the number of experts selected per token. The operator TopK_e selects the k highest-scoring expert indices, giving the selected set $\mathcal{K}_t$:

$$\mathcal{K}_t = \mathrm{TopK}_e(a_{t,e}, k), \tag{1}$$

Their outputs are combined using the model's native routing weights.

EP load imbalance. Expert parallelism (EP) distributes experts across P GPUs, each represented by a rank. Let $s, r \in \{1, \dots, P\}$ index source and serving ranks, respectively, and $s(t)$ be token t's source rank. The destination table D maps each source–expert pair (s, e) to a serving rank $D[s, e]$ hosting expert e. The demand matrix A counts actual router selections: $A[s, e]$ is the number of tokens from s selecting e. With $\mathbb{I}[\cdot]$ equal to one when its condition holds and zero otherwise,

$$A[s, e] = \sum_{t \in \mathcal{T}} \mathbb{I}[s(t) = s]\, \mathbb{I}[e \in \mathcal{K}_t]. \tag{2}$$

Under D, the rank load $L_r(D; A)$ counts token–expert assignments executed on rank r:

$$L_r(D; A) = \sum_{s,e} A[s, e]\, \mathbb{I}[D[s, e] = r]. \tag{3}$$

Each selected expert contributes one assignment per token, even when several execute on the same rank. The EP imbalance $\mathcal{I}(D; A)$ is the maximum rank load divided by the mean:

$$\mathcal{I}(D; A) = \frac{\max_r L_r(D; A)}{\frac{1}{P}\sum_r L_r(D; A)}. \tag{4}$$

$\mathcal{I}(D; A) = 1$ indicates perfect balance; larger values indicate greater imbalance.

Remote nodes. Let $\mathrm{node}(r)$ denote rank r's node. For token t, define $\mathcal{R}_t(D)$ as the set of nodes serving its selected experts under D, excluding its source node $\mathrm{node}(s(t))$. With $|\cdot|$ denoting set size, the remote-node metric $\mathcal{C}(D)$ is the mean number of remote nodes per token:

$$\mathcal{C}(D) = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} |\mathcal{R}_t(D)|, \tag{5}$$

Each remote node counts once per token; service within the source node contributes zero. A lower $\mathcal{C}(D)$ indicates less cross-node communication in MoE.

3.2 The Opportunity in Tail Experts

For token t, sort the baseline top-k scores as $a_{t,(1)} \geq \cdots \geq a_{t,(k)}$. The leading experts contribute most of the mixture weight, whereas tail experts usually carry much smaller weights and may still require execution on remote nodes. Despite their small weights, directly dropping these experts can remove useful contributions and substantially degrade model quality. As Figure 2 shows, however, a remote tail expert often has a score close to that of the highest-scoring local alternative. Replacing the remote tail expert with this local alternative can therefore avoid inter-node communication while preserving model quality. We call this opportunity tail localization.

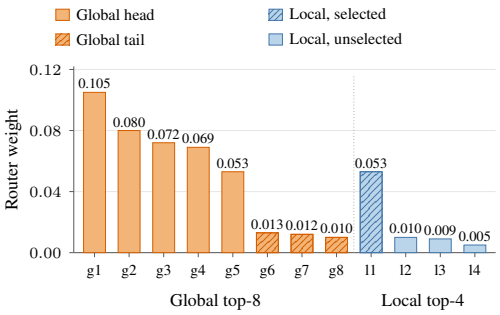


Figure 2: Tail experts and the highest-scoring unselected local experts have similar router weights (Qwen3.6-35B-A3B, EP8, PG-19).

If load balancing focuses solely on equalizing rank loads, it may redirect localized token–expert assignments to remote nodes, offsetting the communication savings from tail localization. Moreover, communication accounts for a larger share of layer latency in hierarchical multi-node deployments than in single-node deployments. We therefore need a load-balancing algorithm that accounts for both expert demand and network topology to jointly optimize communication and load imbalance.

4 Method

Effective load balancing on hierarchical networks requires both a communication-aware assignment and enough time to prepare its expert replicas. Albus addresses the assignment problem through two complementary components: locality-aware expert top-k exploits the tail-localization opportunity to reduce communication through expert selection, while communication-aware load balancing jointly optimizes communication and rank-level load balance by using predicted source–expert demand and network topology to determine redundant expert placement and token assignments. Figure 3 illustrates the Albus framework, including the execution pipeline and the Prediction and Expert transfer modules, which prepare next-layer assignments and expert weights alongside model execution.

Let ℓ denote the MoE layer currently executing. We reuse its MoE input as a proxy for the input to layer $\ell+1$ and apply the next layer's router to predict its expert demand. The predicted source–expert counts are denoted by $\hat{A}[s, e]$. After gathering these counts, only GPU0 within each node executes the planner; the resulting destination table and replica expert transfer tasks are then published. Prediction, planning, and publication overlap layer ℓ's dispatch. The realized router output at layer $\ell+1$ still determines which experts execute.

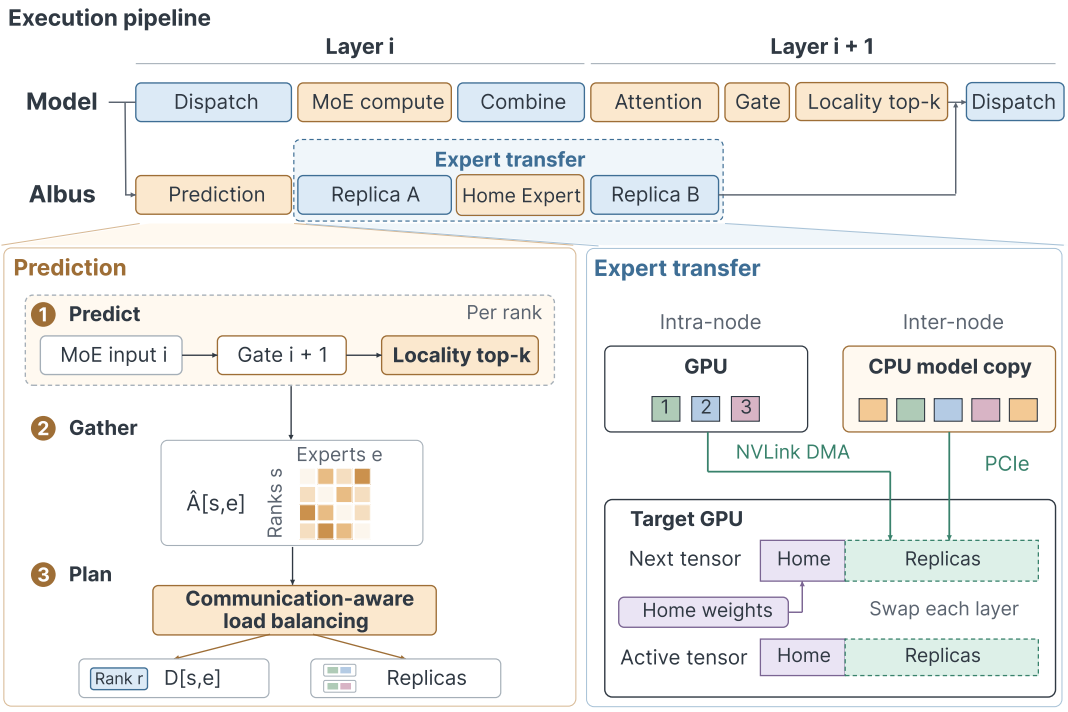


Figure 3: Albus overview. Prediction, planning, and expert transfer overlap model execution. Both prediction and actual top-k routing use the locality enhancement in Section 4.1. Communication-aware load balancing (Section 4.2) uses predicted demand $\hat{A}[s, e]$ to determine the destination table $D[s, e]$ and redundant expert placement.

Albus uses double buffering to overlap expert preparation with inference. The MoE kernel takes an expert tensor containing the weights of the experts executed on a GPU. Each GPU maintains two execution buffers, each assembling this tensor from the weights of its locally owned *home experts* and the prepared replicas. One buffer supplies the current layer's MoE computation while the other is populated with the next layer's experts. The buffers exchange roles across successive layers, allowing storage to be reused without overwriting weights still in use. Home expert transfer copies weights from local GPU storage; replica expert transfer fetches weights from same-node GPUs over NVLink or the resident CPU model copy over PCIe.

We split replica expert transfer into two equal parts, overlapping one with layer ℓ's MoE computation and the other with layer $\ell + 1$'s attention. This overlaps communication with computation and distributes transfer work across two windows, reducing pressure on either stage. Home expert transfer mainly consumes high-bandwidth memory (HBM) bandwidth, so we overlap it with layer ℓ's combine with limited interference. Lightweight prediction likewise runs during dispatch. Together, these choices aim to limit contention for the main execution path's NVLink, NIC, and SM resources.

4.1 Locality-Aware Expert Top-k

For a token t originating on rank $s = s(t)$, let $\mathcal{E}_{\text{local}}(s)$ denote the experts on the source rank in a single-node deployment, or on the source node in a multi-node deployment, excluding redundant experts in both cases. Let $\alpha \geq 1$ be the locality multiplier, $\tilde{a}_{t,e}$ the adjusted ranking score, and $\mathcal{C}_t$ the resulting candidate set. Using the original score $a_{t,e}$ from Equation 1, we compute

$$\tilde{a}_{t,e} = a_{t,e}\alpha^{\mathbb{I}[e \in \mathcal{E}_{\text{local}}(s)]}, \qquad \mathcal{C}_t = \text{TopK}_e(\tilde{a}_{t,e}, k). \tag{6}$$

This changes expert selection while retaining the original scores for output weighting.

We observe that not every token needs to activate all k experts: when routing probability mass concentrates on a few leading experts, skipping low-weight tail experts offers an opportunity to

reduce computation while preserving model quality. For models with normalized original router scores, we introduce a skip parameter $\beta \geq 0$ and threshold $\tau = \beta/E$, using the uniform probability $1/E$ as a reference scale. Filtering the locality-enhanced scores gives the surviving set $\mathcal{S}_t = \{e \in \mathcal{C}_t : \tilde{a}_{t,e} > \tau\}$; the same threshold applies to local and remote candidates. However, routing concentration varies across tokens and layers, and small individual weights do not imply negligible total contribution. We observe that thresholding alone can skip too many experts in some layers, sharply degrading model quality. We therefore require each token at every MoE layer to retain at least p experts, where $1 \leq p \leq k$. If fewer than p candidates survive, we restore skipped candidates in descending order of their original scores. The final selected set $\mathcal{K}_t^{\text{Albus}}$ is

$$\mathcal{K}_t^{\text{Albus}} = \begin{cases} \mathcal{S}_t, & |\mathcal{S}_t| \geq p, \\ \mathcal{S}_t \cup \text{TopK}_{e \in \mathcal{C}_t \setminus \mathcal{S}_t}(a_{t,e}, p - |\mathcal{S}_t|), & \text{otherwise.} \end{cases} \tag{7}$$

We use this final set as $\mathcal{K}_t$ when computing demand in Equation 2. The layer combines selected outputs using renormalized original scores. For sigmoid-gated models, we apply only the locality multiplier, preserve the original k, and retain native output weighting; filtering and minimum-retention controls are disabled. No model parameters are retrained.

4.2 Communication-Aware Load-Balancing Plan

Let $\mathcal{T}_s$ contain the tokens originating on rank s, and let $\hat{\mathcal{K}}_t$ denote the expert set predicted by applying Section 4.1's selection rule to the proxy input. We aggregate predicted demand as

$$\hat{A}[s, e] = \sum_{t \in \mathcal{T}_s} \mathbb{I}[e \in \hat{\mathcal{K}}_t]. \tag{8}$$

Only these PE counts, rather than token identities, are passed to the planner. It constructs $D[s, e]$ using network topology and the predicted rank loads $L_r(D; \hat{A})$ from Equation 3. The corresponding predicted imbalance is $\mathcal{I}(D; \hat{A})$.

Let $H(e)$ denote expert e's home rank. The planner starts from $D[s, e] = H(e)$ and repeatedly considers demand assigned to the most heavily loaded rank. A destination rank is feasible if it already hosts the expert or has a free replica slot. When the source node already hosts the expert, candidates are restricted to that node; otherwise, feasible destinations are considered globally. Within this set, the planner selects the least-loaded rank and admits a move only when its post-move load is below the current hottest rank's load. Among admissible source–expert chunks, it moves the largest first. Iteration stops when rank loads are sufficiently balanced or no admissible move remains.

If the destination rank does not already host the expert, the planner adds a replica there. If a move introduces an expert's first replica in a node, all demand for that expert originating within the node is redirected to the new copy, and affected rank loads are recomputed. During decoding, replica expert transfer is restricted to GPUs within the same node. See Appendix A.2 for algorithmic details.

5 Evaluation

5.1 Setup

Models. We evaluate five MoE models: Qwen3-235B-A22B (Yang et al., 2025) at EP8 on one node; Qwen3.6-35B-A3B (Qwen Team, 2026) and Qwen3-30B-A3B (Yang et al., 2025) at EP4 within one node; and GLM-4.7 (GLM Team et al., 2025) and DeepSeek-R1 (DeepSeek-AI, 2025) at EP32 across four nodes each. All deployments use data-parallel attention with DP-attention=EP: each GPU independently computes attention for its own tokens, while MoE layers dispatch tokens across the EP group for expert computation and combine the results. Following Section 4.1, local experts reside on the source GPU for Qwen and on any GPU in the source node for GLM-4.7 and DeepSeek-R1. The locality multiplier α enhances only local home experts, excluding replicas.

Environment and Settings. We implement all methods in SGLang 0.5.10 on nodes with eight GPUs connected by NVLink/NVSwitch and inter-node RoCE links. NVLink/NVSwitch provides unidirectional bandwidth on the order of hundreds of GB/s, while CPU–GPU PCIe and inter-node RoCE links provide tens of GB/s. The SGLang backend uses DeepEP (Zhao et al., 2025b) for token

dispatch and combine and DeepGEMM (Zhao et al., 2025a) for MoE computation. Each GPU has two redundant expert slots, with at most one copy of any given expert on a GPU. During decoding, each GPU may additionally send out at most two experts. Datasets and evaluation details are provided in Appendix A.1.

Baselines. We compare Albus against native SGLang inference, the EPLB algorithm (DeepSeek-AI, 2025), and Libra++. EPLB is calibrated with data from the corresponding evaluation dataset. To better support multi-node load balancing, we integrate Libra's predictive load-balancing algorithm (Yang et al., 2026) into the same execution pipeline used by Albus and denote this adapted system as Libra++. This enables comparing balancing algorithms under the same runtime organization.

5.2 Model Quality

Table 1 reports quality and speedup on larger models; routing calibration appears in Appendix F, and small-model ablations in Appendix G (Table 7). Accuracy is reported as a percentage; *Avg* is the average over the listed tasks. Speedup is baseline latency divided by method latency under the same model and batch configuration. At $\alpha = 2$, Albus can maintain model quality comparable to the baseline (Table 1). Increasing α can provide greater speedups, but may degrade model quality. We therefore use $\alpha = 2$ in subsequent system performance experiments to assess acceleration with limited impact on model quality.

Table 1: Quality–efficiency trade-off on larger MoE models. For Qwen3-235B-A22B, we set $p = 5$ and sweep the parameter pair (α, β); GLM-4.7 and DeepSeek-R1 use locality-only routing, so β is not applicable. Avg is the average over six datasets. Speedup is the arithmetic mean of baseline-to-method latency ratios on LMSYS-Chat-1M at input lengths of 1K, 2K, 4K, and 8K tokens. All three models use native SGLang inference as the baseline.

Model	Config. (α, β)	C-Eval	MMLU	MMLU-Pro	LIMO	RTE	TruthfulQA	Avg	Speedup
Qwen3-235B-A22B	**Baseline**	**86.6**	**87.5**	**77.3**	94.4	**82.0**	**60.7**	81.4	1.00×
	(2.0, 4.0)	86.5	87.0	76.9	**97.4**	**82.0**	60.4	**81.7**	1.13×
	(2.0, 6.0)	85.4	86.2	75.1	91.4	81.2	57.8	79.5	1.25×
	(3.0, 4.0)	86.3	87.3	76.7	94.7	81.6	60.5	81.2	1.11×
	(3.0, 6.0)	84.7	85.4	74.0	90.3	79.4	58.2	78.7	1.26×
GLM-4.7	**Baseline**	88.0	**89.2**	78.4	93.2	72.9	63.9	80.9	1.00×
	(1.5, –)	**88.3**	89.1	**79.8**	**93.8**	**74.0**	**68.7**	**82.3**	1.08×
	(2.0, –)	87.8	88.7	79.6	93.3	73.3	66.9	81.6	1.16×
	(2.5, –)	87.1	88.7	79.4	89.2	70.8	64.6	80.0	1.22×
	(3.0, –)	85.7	88.0	78.0	81.2	71.5	63.3	78.0	1.26×
DeepSeek-R1	**Baseline**	**90.2**	**90.7**	82.6	91.2	74.0	58.1	81.1	1.00×
	(1.5, –)	**90.2**	90.5	**82.7**	**91.6**	74.0	**58.9**	**81.3**	1.08×
	(2.0, –)	89.8	90.4	82.2	90.8	**74.4**	58.5	81.0	1.17×
	(2.5, –)	89.5	90.2	81.8	82.7	72.2	57.1	78.9	1.25×
	(3.0, –)	88.8	89.7	80.3	71.6	71.5	57.7	76.6	1.30×

5.3 Inference Speedup

We evaluate prefill latency and throughput, and separately compare decode time per output token (TPOT) and token throughput. Redundant expert placement metrics and layer timings help explain the gains. All latency comparisons use matching model and workload configurations.

Prefill latency. Figure 4 compares four per-rank PG-19 prefill loads: 4K, 8K, 12K, and 16K. Relative to native SGLang, Albus achieves 1.48–$1.74\times$ speedup on GLM-4.7 and 1.64–$1.87\times$ on DeepSeek-R1. Across both models and all four loads, Albus achieves average speedups of $1.71\times$, $1.33\times$, and $1.10\times$ over native SGLang, EPLB, and Libra++, respectively.

Load balance and node locality. At $\alpha = 2$, Albus reduces $\mathcal{I}(D; A)$ from native SGLang's 3.37 to 1.27 on GLM-4.7 and from 3.45 to 1.28 on DeepSeek-R1, also improving on EPLB and Libra++. The gap to Albus Oracle reflects errors in the demand predictions used for load balancing. Albus

also reduces communication $\mathcal{C}(D)$ by 22.8% on GLM-4.7 and 23.2% on DeepSeek-R1 relative to Libra++. This relative reduction increases with α over the tested range from 1 to 3. Thus, rank balance improves while preserving node locality.

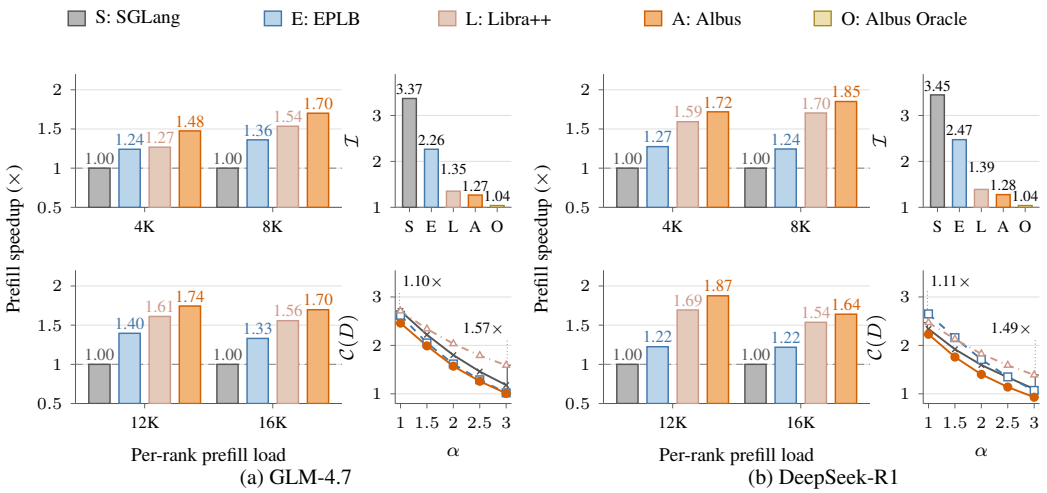


Figure 4: Prefill speedup, load imbalance $\mathcal{I}(D; A)$, and communication $\mathcal{C}(D)$ on PG-19 for GLM-4.7 and DeepSeek-R1. Each rank processes one request, and prefill load denotes its length. The communication sweep applies each α to all methods, including SGLang. Both imbalance and communication metrics are averaged over the four tested request lengths. For prefill and imbalance, the baseline is native SGLang inference; Albus, Libra++, and EPLB use $\alpha = 2$. Albus Oracle reports predicted imbalance $\mathcal{I}(D; \hat{A})$, while other methods report imbalance $\mathcal{I}(D; A)$ from actual routing.

Layer latency breakdown. Table 2 shows that Albus achieves per-layer speedups of 1.59× over native SGLang and 1.20× over Libra++. Albus and Libra++ have similar expert computation times, while Albus spends less time in dispatch and combine. After obtaining the actual top-k routing decisions, Libra++ performs an additional all-gather to reconcile its planned load budgets with realized demand. This operation is recorded under MoE other, which is 1.25 ms higher than native SGLang. The all-gather exchanges demand counts for 160 experts, each represented by a 4-byte `int32` value, totaling 640 B per rank; data transfer takes less than 0.2 ms. Most of the additional elapsed time reflects rank synchronization. Without this all-gather, the synchronization wait would instead be included in dispatch timing.

Table 2: Per-layer latency breakdown (ms) on GLM-4.7, including attention and MoE execution. The baseline is native SGLang inference; Libra++ and Albus use $\alpha = 2$. Total is the full-layer latency.

Method	Dispatch	Compute	Combine	MoE other	Attention	Predict[a]	Plan[a]	Replica expert transfer[b]	Home expert transfer[a]	Total
Baseline	7.13	15.05	7.39	1.32	5.86	0.00	0.00	0.00	0.00	36.75
Libra++	5.85	7.81	5.52	2.57	5.86	0.11	0.41	2.10	0.19	27.62
Albus	3.74	7.66	4.42	1.39	5.90	0.11	0.22	1.96	0.19	**23.11**

[a]Overlapped with communication; [b]overlapped with computation.

SLO-constrained prefill serving. Figure 5(a,b) compares Albus at $\alpha = 2$ with native SGLang under a 5 s service-level objective (SLO) for time to first token (TTFT). Albus increases the highest tested request rate meeting the SLO from 76 to 124 RPS for median (P50) TTFT (1.63×) and from 60 to 108 RPS for the 95th percentile (P95; 1.80×).

Prefill throughput. Figure 5(c) compares native SGLang inference, locality enhancement alone ($\alpha = 2$), Albus ($\alpha = 2$), and Albus with locality enhancement disabled. With CPU scheduling included, their average throughputs are 75.5k, 85.2k, 105.7k, and 91.9k tokens/s, respectively. Albus achieves 1.40× the throughput of native SGLang. Even with locality enhancement disabled (i.e., selecting the same top-k experts as native SGLang), Albus still achieves 1.22× its throughput.

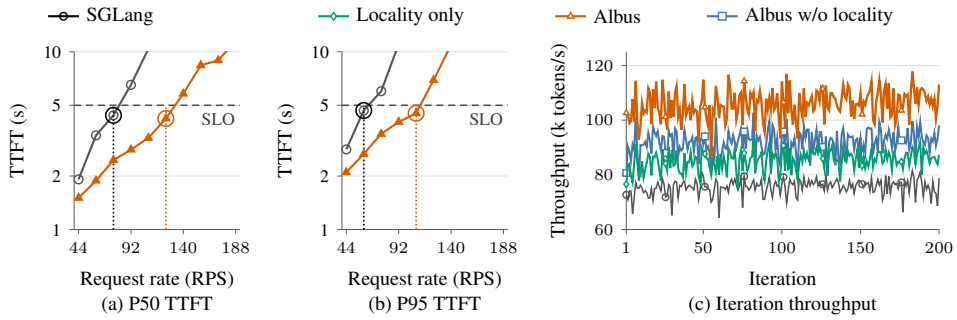


Figure 5: Serving TTFT and prefill throughput of GLM-4.7 on PG-19 with request lengths up to 8K tokens. The SLO is 5 s, and RPS denotes the number of requests arriving per second. Throughput is measured over 200 iterations of prefill serving. Locality only and Albus both use $\alpha = 2$. Albus w/o locality denotes Albus with locality enhancement disabled and load balancing enabled.

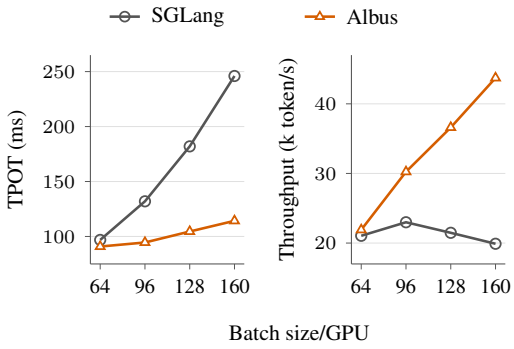


Figure 6: Decode TPOT and throughput of GLM-4.7 on PG-19 with inputs of up to 2K tokens.

Decode performance. Figure 6 shows that Albus achieves an average $1.59\times$ speedup in TPOT and $1.56\times$ the throughput of native SGLang. At small batch sizes, short execution times limit overlap, and performance also depends on how many experts are activated. As batches grow, longer execution windows enable more overlap. Albus keeps prediction and planning lightweight and limits expert transfers into and out of each GPU, allowing preparation to fit within these windows. Native SGLang's TPOT grows roughly linearly as hot ranks bottleneck execution, while Albus distributes work more evenly across GPUs and substantially slows TPOT growth.

Memory overhead. Table 3 reports memory usage on one of the 32 GPUs serving GLM-4.7. A BF16 expert occupies $w = 3 \cdot 5120 \cdot 1536 \cdot 2/2^{30}$ GB for three projections. At EP32, this GPU owns $160/32 = 5$ home experts. Albus reuses two buffers across layers, each holding five home experts and two replicas: $2(5+2)w \approx 0.62$ GB beyond resident weights. EPLB retains two replicas in each of $92 - 3 = 89$ MoE layers: $89 \cdot 2w \approx 7.82$ GB. Load imbalance causes hot ranks to process more tokens, requiring larger intermediate activation tensors during expert computation and increasing peak MoE activation memory. Albus uses 92.1% less extra weight space and has a 46.5% lower activation peak than EPLB. Although its buffers add 0.62 GB, Albus cuts peak MoE activations from the baseline's 12.64 to 5.11 GB. This 7.53 GB reduction outweighs the buffering cost, resulting in lower peak GPU memory usage.

Table 3: Memory usage on one of 32 GPUs serving GLM-4.7 on PG-19 at 16K prefill tokens/GPU. The baseline is native SGLang inference; Albus and EPLB use $\alpha = 2$. GB denotes 2^{30} bytes.

Method	Extra expert weights	Peak MoE activations
SGLang	0.00	12.64
Albus	0.62	5.11
EPLB	7.82	9.56

6 Conclusion

We presented Albus, a MoE serving system that jointly optimizes communication and load balance on hierarchical networks. It integrates locality-aware expert selection and communication-aware redundant expert placement with predictive scheduling and expert prefetching. Evaluations show comparable model quality, lower prefill latency, and higher prefill and decode throughput.

AI USE STATEMENT

We used generative AI tools to polish wording, suggest synonyms, improve sentence fluency, and check grammar. We also used these tools to survey the research area and identify prior work related to our research ideas. In the Evaluation section, AI tools helped draft descriptions of experimental results based on our experimental figures.

REPRODUCIBILITY STATEMENT

We use publicly available model checkpoints that can be downloaded from Hugging Face. The experimental environment and system configurations are specified in Section 5. Albus's execution pipeline is described in Section 4, with implementation details and overlap measurements in Appendices A and B. Hyperparameter settings are reported in Section 5 and Appendices A.3, F, and G.

REFERENCES

Yutian Chen, Cong Li, Yucheng Wang, and Ming Wei. Moonep: A perfectly balanced expert parallelism library via dynamic redundant experts. https://github.com/MoonshotAI/MoonEP, 2026.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

DeepSeek-AI. Expert parallelism load balancer (EPLB). GitHub repository, 2025. URL https://github.com/deepseek-ai/EPLB.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff,

Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, jiayi lei, Yao Fu, Maosong Sun, and Junxian He. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 62991–63010. Curran Associates, Inc., 2023. doi: 10.52202/075280-2749. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/c6ec1844bec96d6d32ae95ae694e23d8-Paper-Datasets_and_Benchmarks.pdf.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 269–287. Curan, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/5616d34cf8ff73942cfd5aa922842556-Paper-mlsys2023.pdf.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 56099–56115, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/8cd1ce03ea58b3d7dfd809e4d42f08ea-Paper-Conference.pdf.

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, 2020. URL https://arxiv.org/abs/2006.16668.

Yan Li, Zhenyu Zhang, Zhengang Wang, Pengfei chen, and Pengfei Zheng. Semantic parallelism: Redefining efficient moe inference via model-data co-scheduling. In

C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 148580–148599, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/f0552f14388d95b19740dee809f5cad1-Paper-Conference.pdf.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL https://aclanthology.org/2022.acl-long.229/.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. https://huggingface.co/datasets/open-r1/codeforces, 2025.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1911.05507.

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, 2022. URL https://arxiv.org/abs/2201.05596.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

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

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024. URL https://arxiv.org/abs/2405.14333.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. Moe-infinity: Efficient moe inference on personal machines with sparsity-aware expert cache, 2025. URL https://arxiv.org/abs/2401.14361.

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

Jaehoon Yang, Yushin Kim, Seokwon Moon, Yeonhong Park, and Jae W. Lee. Libra: Effective yet efficient load balancing for large-scale moe inference. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 98341–98362, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/9ff1ac9a659085fed0735362cafe5e53-Paper-Conference.pdf.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning, 2025. URL https://arxiv.org/abs/2502.03387.

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

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, Joseph E Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 22225–22257, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5f9bfdfe3685e4ccdbc0e7fb29cccf2a-Paper-Conference.pdf.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024b. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

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

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

A Implementation Details

A.1 Datasets and Evaluation

Datasets. We evaluate model quality using lm-evaluation-harness (Gao et al., 2024). The evaluation datasets are C-Eval (Huang et al., 2023), MMLU (Hendrycks et al., 2021), MMLU-Pro (Wang et al., 2024), LIMO (Ye et al., 2025), IFEval (Zhou et al., 2023), SciQ (Welbl et al., 2017), RTE (Wang et al., 2019), and TruthfulQA-MC2 (Lin et al., 2022). Table 4 summarizes the metric used for each benchmark. For inference speedup evaluation, we use PG-19 (Rae et al., 2019), LMSYS-Chat-1M (Zheng et al., 2024a), Codeforces (Penedo et al., 2025), BookCorpusOpen (Zhu et al., 2015), and DeepSeek-Prover (Xin et al., 2024). Routing calibration uses WikiText-103 (Merity et al., 2016).

Table 4: Datasets and metrics used for accuracy evaluation.

Task	Metric
C-Eval	CoT Accuracy
MMLU	CoT Accuracy
MMLU-Pro	CoT Accuracy
LIMO	CoT Accuracy
IFEval	Accuracy
SciQ	Log-likelihood Accuracy
RTE	Log-likelihood Accuracy
TruthfulQA-MC2	Log-likelihood MC2 Accuracy

A.2 Communication-Aware Load Balancing

Algorithm Albus planner

Input: $\hat{A}[s,e]$, predicted demand; $H(e)$, home rank; $\mathrm{node}(r)$, node mapping; P, EP size; N, replica slots per rank; $\mathrm{stop_rate}$, stopping multiplier
Output: $D[s,e]$, serving rank; $X[e,r] \in \{0,1\}$, whether rank r hosts expert e

$D[s,e] \leftarrow H(e)$; $X[e,r] \leftarrow \mathbb{I}[r = H(e)]$
$L_r(D;\hat{A}) \leftarrow \sum_{s,e} \hat{A}[s,e]\,\mathbb{I}[D[s,e]=r]$ for all r
$B \leftarrow \mathrm{stop_rate} \cdot \frac{1}{P}\sum_r L_r(D;\hat{A})$
while $\max_r L_r(D;\hat{A}) > B$ **do**
    $h \leftarrow$ most-loaded rank; initialize candidate moves $\mathcal{M} \leftarrow \emptyset$
    for all (s,e) with $D[s,e]=h$ and $\hat{A}[s,e]>0$ **do**
        $\mathcal{F} \leftarrow \{r \neq h : r \text{ hosts } e \vee \mathrm{CanReplicate}(e,r)\}$ ▷ Replica can be moved to rank r
        Restrict $\mathcal{F}$ to the source node if it already hosts e
        if $\mathcal{F} = \emptyset$ **then continue**
        $d \leftarrow$ least-loaded rank in $\mathcal{F}$, breaking ties by rank ID
        if $L_d(D;\hat{A}) + \hat{A}[s,e] < L_h(D;\hat{A})$ **then** add (s,e,d) to $\mathcal{M}$
    end for
    if $\mathcal{M} = \emptyset$ **then break**
    $(s,e,d) \leftarrow$ move in $\mathcal{M}$ with the largest predicted demand $\hat{A}[s,e]$
    if the node containing d does not yet host e **then**
        Redirect that node's demand for e to rank d in D
    end if
    $D[s,e] \leftarrow d$
    if rank d does not host e **then** reserve a slot and set $X[e,d] \leftarrow 1$
    Update the affected rank loads
end while
return D, X

A.3 Kernel Implementation

We implement separate CUDA kernels for prefill and decode planning based on the algorithm in Appendix A.2. The framework sets the stopping multiplier $\mathrm{stop_rate} = 1.01$. Only local rank 0 on each node launches these kernels, using a single cooperative thread array (CTA) per invocation to limit contention with foreground computation. All intermediate solver data are stored in shared

memory. Expert placement is encoded with rank bitmasks, enabling fast queries of whether a rank or node already hosts an expert. Each rank has two replica slots. Decode additionally limits each rank to at most two outgoing expert copies and restricts replication to the node containing the expert's home rank, accommodating the shorter overlap window during decoding. With GLM-4.7 and 32 EP ranks, mean solver latency is approximately 0.30 ms for prefill and 0.19 ms for decode.

Prefill. The prefill kernel uses 256 threads organized into eight warps, which evaluate experts hosted by the most-loaded rank in parallel. Each warp first selects its best feasible candidate. Warp 0 then reduces across these warp-level candidates to select the move with the largest predicted demand. Finally, a single thread commits the move and updates the affected rank loads and replica state.

Decode. The restriction to replication within the home node partitions planning into independent subproblems by destination node. In the four-node configuration, the decode kernel uses one CTA with 128 threads, with one warp assigned to each node. Each warp considers demand from all source ranks for experts whose home ranks reside in its assigned node. Each node can independently commit one move per iteration. A single CTA synchronization at the end of each iteration checks whether any node can continue; the solver exits when no node has a feasible move. The decode solver runs entirely on the GPU and supports CUDA Graph capture and replay, without CPU intervention between iterations.

A.4 Overlap Stream Details

Figure 3 already illustrates the overlap schedule. Replica expert transfer is divided into two parts, Replica 0 and Replica 1, which overlap layer ℓ's MoE computation and layer $\ell + 1$'s attention, respectively. Replica 0 starts once dispatch and planning are complete; Replica 1 starts after combine. Execution waits for replica transfers scheduled in the MoE computation window before entering combine, and for all required expert transfers before the next layer's dispatch. These waits provide a safe fallback when transfers extend beyond their intended overlap windows. During decoding, if the MoE computation window is insufficient, the corresponding replica-transfer work is deferred to the attention window and must still complete before the next dispatch.

The implementation uses separate CUDA streams for these operations. The plan stream performs prediction, demand-count collection, and plan generation. The home copy stream handles home expert transfer, while the replica copy stream handles replica expert transfer. Both types of expert transfer use CUDA asynchronous DMA copies on their respective streams. During decoding, the plan stream and home copy stream are captured in the CUDA Graph together with the main execution stream; replica expert transfer runs outside the graph.

A.5 Runtime Details

Albus runtime. Albus uses one background submission thread per rank, with pre- and post-execution hooks triggering next-layer planning and prefetching. This thread manages dependencies and asynchronous submission while the main thread continues submitting forward operations. Prediction, load statistics, planning, and model computation execute on the GPU; asynchronous DMA moves weights. A double-buffered expert bank overlaps current-layer computation with next-layer preparation, with events and readiness signals ensuring that weights are ready before use.

During decoding, CUDA Graph captures prediction, demand-count exchange, planning, fixed-address home expert transfer, and replica-transfer requests and waits. Each replay generates a new plan on the GPU. The background thread submits dynamic replica copies to a separate DMA stream outside the graph and signals completion to release dependent graph operations.

EPLB runtime. EPLB keeps each layer's experts and replicas resident on GPUs. It collects actual loads during forward passes and periodically rebalances (DeepSeek-AI, 2025): affected weights are transferred, routing mappings are updated, and subsequent forward passes use the new layout.

Libra++ runtime. Libra uses AllGather dispatch, separate local and remote MoE passes, and CPU-based token sharding (Yang et al., 2026). Across nodes, AllGather broadcasts tokens to all GPUs, generating more traffic than destination-specific All-to-All. Large EP leaves little GPU-local work per rank; splitting MoE computation into two passes can reduce grouped-GEMM efficiency and incur extra kernel launches. CPU-based sharding also introduces CPU–GPU synchronization, complicating

CUDA Graph capture during decoding. We therefore evaluate Libra's predictive load-balancing algorithm using the Albus runtime, denoting this multi-node adaptation as Libra++.

B Overlap Measurements

Foreground interference. We isolate the infrastructure cost during prefill by removing the algorithmic benefits of Albus and enabling its overlapped execution pipeline. Table 5 reports attention and MoE other at L4, and dispatch, compute, and combine at L3. The aggregate overhead of these five components is below 1%, indicating limited interference with the main execution path in this test.

Table 5: Prefill foreground times (ms), independently rounded.

Component	Baseline	Albus
Attention	5.76	5.84
MoE other	1.04	1.07
Dispatch	3.66	3.59
Compute	8.97	9.01
Combine	3.86	3.89
Total	23.30	23.39

Figure 7 shows the number of expert copies requested from each source rank in a redundant expert placement plan during actual Albus prefill inference. Requests are concentrated on a few source ranks, revealing hotspots in expert-transfer demand.

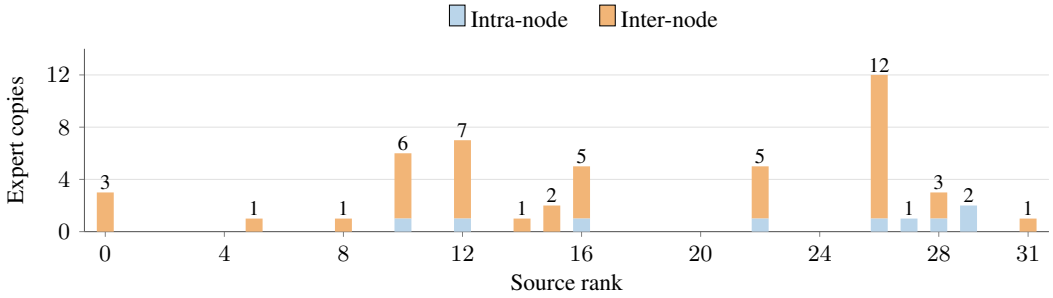


Figure 7: Expert copies requested from each source rank in one prefill redundant expert placement plan: 50 in total, including 9 intra-node and 41 inter-node requests. Intra-node denotes requests from GPUs within the source node; inter-node denotes requests from GPUs on other nodes.

Compute–transfer overlap. Table 6 evaluates prefill compute–transfer overlap with 8K tokens/rank: 64K token–expert assignments for DeepGEMM under top-8 routing, and 8K tokens for attention. Naive uses direct RDMA. For replica expert transfer, Albus uses NVLink DMA for experts homed within the node and retrieves other experts from a resident CPU model copy over PCIe. Albus divides replica expert transfer into two parts, one per compute window. The table reports each part separately; the total transfer time is $1.22 + 1.17 = 2.39$ ms. Within each window, transfer overlaps computation.

Table 6: Compute and replica expert transfer times (ms) during prefill with 8K tokens/rank.

Execution mode	DeepGEMM (64K)		Attention (8K)	
	Compute	Transfer	Compute	Transfer
Standalone	4.49	—	5.25	—
Naive	4.57	11.30	5.25	11.61
NCCL	7.71	6.89	5.60	8.00
Albus	4.48	1.22	5.19	1.17

NCCL's GPU communication kernels share SM resources with foreground computation; its measured compute time increases by 71.87% for DeepGEMM and 6.63% for attention. Naive and Albus keep compute times close to standalone execution. However, Naive remains vulnerable to concentrated requests at source ranks (Figure 7), and its transfers outlast both compute windows. Albus's two-stage transfers finish within each window while preserving near-baseline computation times.

During decoding, Albus restricts replica expert transfer to GPUs within the same node, using high-bandwidth NVLink. It also limits the numbers of incoming and outgoing experts per rank to avoid transfer hotspots. Under these constraints, 2 experts can be transferred within 0.33 ms in our measurement.

C Next-Layer Prediction Quality

We evaluate next-layer prediction for GLM-4.7 and DeepSeek-R1 on Codeforces, BookCorpusOpen, PG-19, and DeepSeek-Prover, using Albus at $\alpha = 2$ with 8K prefill tokens per source rank.

Prediction metrics. Let $\widehat{\mathcal{K}}_t$ denote token t's predicted expert set and $\mathcal{K}_t$ its actual set, with $k = 8$. Top-k accuracy is

$$\mathrm{Acc}_{\text{top-}k} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{|\widehat{\mathcal{K}}_t \cap \mathcal{K}_t|}{k}. \tag{9}$$

The rank-load normalized mean absolute error (NMAE) is

$$\mathrm{NMAE} = \frac{\sum_{r=1}^{P} \left| L_r(D; \hat{A}) - L_r(D; A) \right|}{\sum_{r=1}^{P} L_r(D; A)}. \tag{10}$$

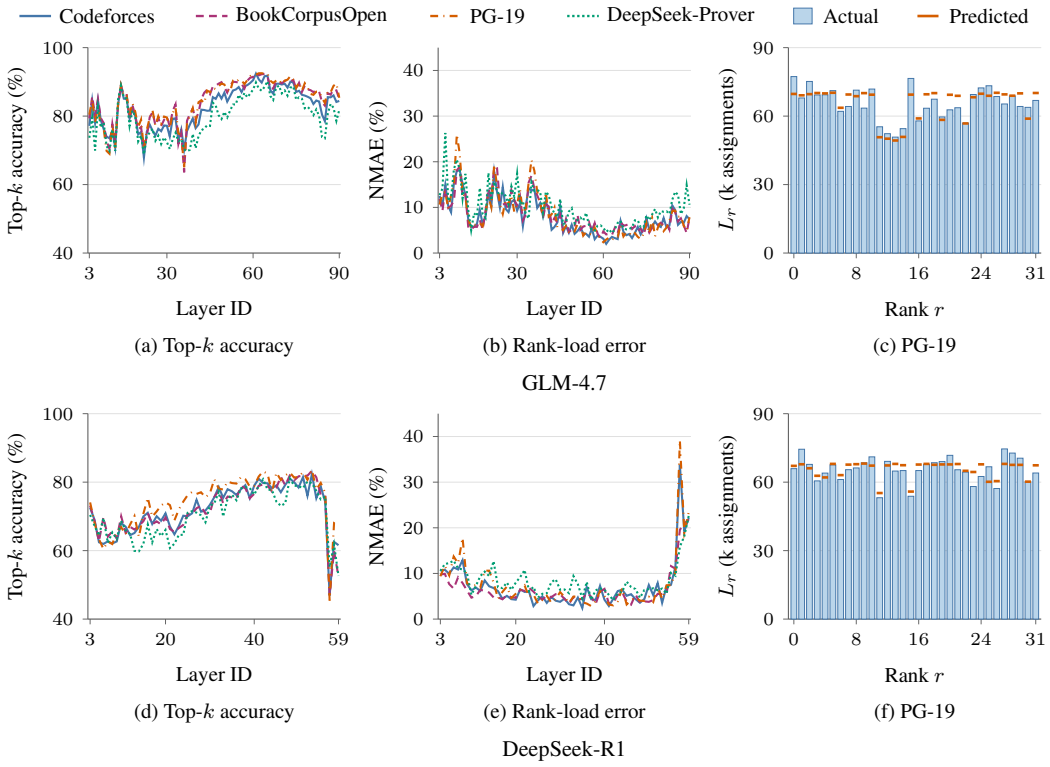


Figure 8: Next-layer prediction under Albus at $\alpha = 2$ with 8K tokens per source rank. (a,d) Top-k accuracy ($k = 8$) and (b,e) rank-load NMAE across four datasets. Layer ID denotes the MoE layer predicting its successor. (c,f) Single-layer examples on PG-19: blue bars show $L_r(D; A)$ and orange ticks show $L_r(D; \hat{A})$, in thousands of assignments. Indices are zero-based.

Predicted and actual loads use the same mapping D. Figure 8 reports accuracy and NMAE as percentages. Across the four datasets, layer-averaged top-k accuracy ranges from 79.8–84.4% for GLM-4.7 and 70.5–74.5% for DeepSeek-R1; the corresponding mean NMAE ranges are 7.9–10.1% and 6.4–8.3%. Means weight layers equally. For the single-layer PG-19 examples, NMAE is 5.02% and 4.14%, respectively. Prediction quality varies with depth, particularly near R1's final layers.

D LAYERWISE REDUNDANT EXPERT PLACEMENT RESULTS

Figure 9 compares redundant expert placement across the MoE layers of GLM-4.7 and DeepSeek-R1 on PG-19. We compare native SGLang inference with a baseline that adds locality enhancement at $\alpha = 2$ without replicas. EPLB, Libra++, and Albus use $\alpha = 2$. At the first MoE layer (Layer ID 3), Albus and Libra++ use the same home dispatch as Baseline at $\alpha = 2$. Albus Oracle reports the predicted-demand imbalance $\mathcal{I}(D; \hat{A})$ from the second MoE layer onward; no prediction result exists at the first layer.

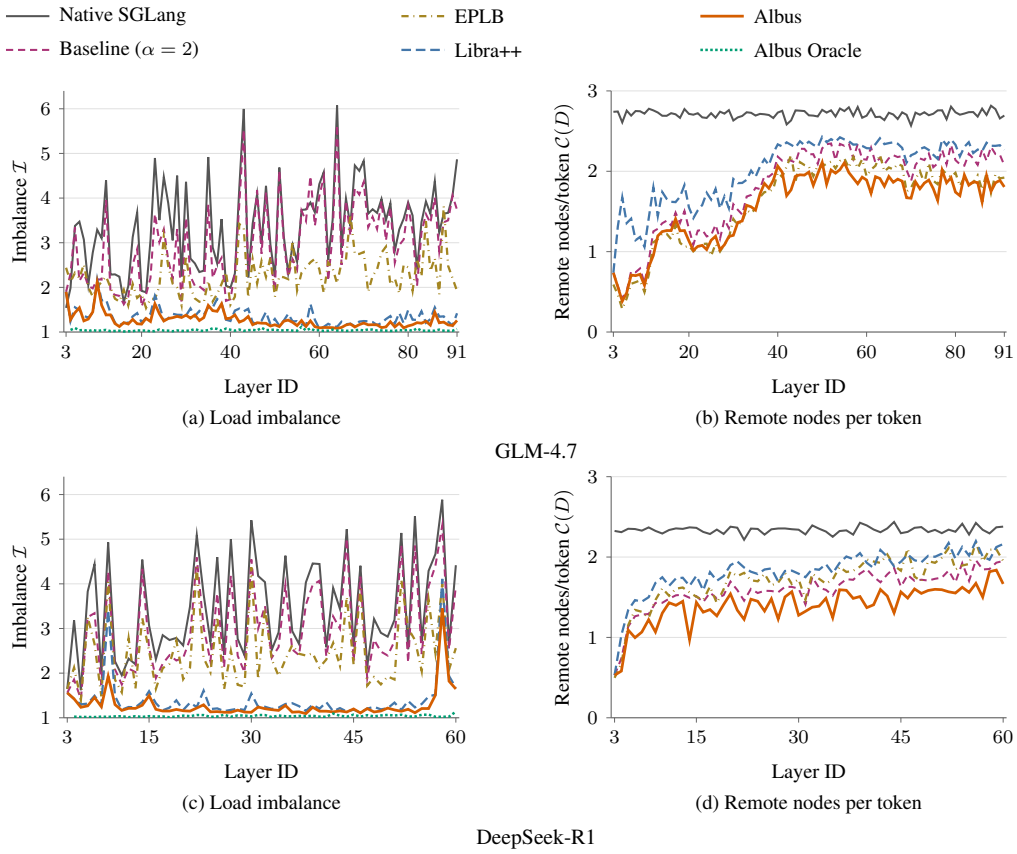
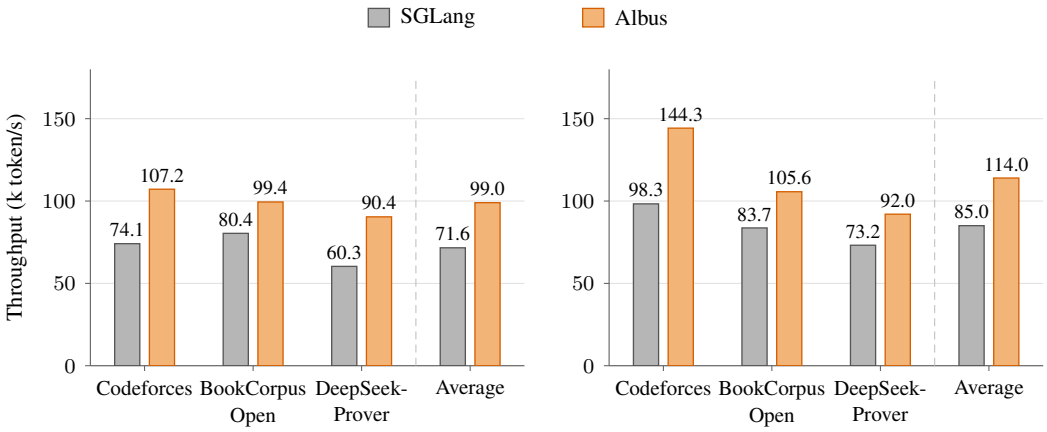


Figure 9: Layerwise redundant expert placement results on PG-19, equally averaged over 4K, 8K, 12K, and 16K tokens per source rank. (a,c) Actual imbalance $\mathcal{I}(D; A)$ and the Albus Oracle predicted-demand reference $\mathcal{I}(D; \hat{A})$. (b,d) Mean distinct remote nodes per token $\mathcal{C}(D)$ for five actual-routing configurations, excluding the source node. Both baseline settings are shown; other methods use $\alpha = 2$. At L3, Albus and Libra++ use home dispatch; Oracle starts at L4. Layer IDs are zero-based.

Averaged equally over layers, Albus achieves $\mathcal{I}(D; A) = 1.27$ on GLM-4.7 and 1.28 on DeepSeek-R1, compared with 1.35 and 1.39 for Libra++, and 2.26 and 2.47 for EPLB. Its mean $\mathcal{C}(D)$ is 1.57 and 1.40, respectively, 22.8% and 23.2% below Libra++.

E THROUGHPUT ACROSS DATASETS

We compare Albus at $\alpha = 2$ with native SGLang inference on Codeforces, BookCorpusOpen, and DeepSeek-Prover, using GLM-4.7 and DeepSeek-R1. Both models are evaluated at EP32, with at most 8K tokens per GPU. Figure 10 reports throughput for both models. Albus delivers higher throughput than native SGLang across all six model–dataset configurations. Across the three datasets, mean throughput increases from 71.6K to 99.0K tokens/s on GLM-4.7 (1.38×) and from 85.0K to 114.0K tokens/s on DeepSeek-R1 (1.34×).



(a) GLM-4.7 (b) DeepSeek-R1

Figure 10: Throughput across three datasets, comparing native SGLang inference with Albus. Average is the arithmetic mean of the three dataset throughputs for each method. Bar labels give throughput in thousands of tokens/s.

F ROUTING PARAMETER CALIBRATION

We calibrate α, β, and p on WikiText-103 using Qwen3.6-35B-A3B at EP4. Figure 11 reports relative increases in mean perplexity (PPL). Choosing $\alpha = 2$ increases PPL by 2.53% relative to $\alpha = 1$, while larger values produce progressively greater increases. With $\alpha = 2$ fixed, we use a 5% PPL increase threshold to identify a conservative region in the (β, p) grid. The setting $(\beta, p) = (4.5, 6)$ used in the small-model experiments lies within this region, with an additional 2.05% increase relative to the $\alpha = 2$ reference.

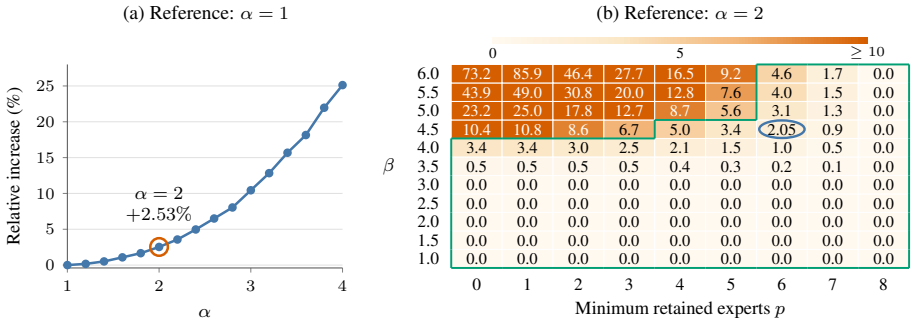


Figure 11: Calibration increases (%) relative to $\alpha = 1$ in (a) and $\alpha = 2$ without additional skipping in (b). Circles mark the selected settings; the green outline encloses tested settings with increases $\leq 5\%$.

G Ablation Studies

G.1 Small-Model Validation

Table 7 isolates the roles of local tail replacement and minimum expert retention. With $\alpha = 2$, $\beta = 4.5$, and $p = 6$, Albus obtains average accuracies of 81.0 and 83.9 on Qwen3-30B-A3B and Qwen3.6-35B-A3B, compared with baseline values of 81.1 and 83.6. Skip Tail drops displaced experts without adding local replacements, reducing these averages to 80.6 and 80.5; at $\alpha = 3$, they fall further to 76.6 and 71.7. Although tail experts may have low routing weights, removing them can still degrade model quality; replacing them with local experts effectively mitigates this loss. Without Limit removes the minimum-expert constraint and reaches only 76.7 and 78.0 at $\alpha = 2$. Threshold-only pruning can leave too few active experts in some layers, motivating the per-token minimum. The full policy largely preserves average quality at $\alpha = 2$, although stronger locality bias can still degrade accuracy.

Table 7: Small-model accuracy (%) on eight datasets at EP4. Ours uses $\beta = 4.5$, a minimum of $p = 6$ experts per token, and $\alpha \in \{2, 3\}$. Skip Tail drops the original experts selected for replacement without adding local substitutes. Without Limit uses the same α and β as Ours but removes the minimum-expert constraint. Avg is the mean accuracy across the eight datasets.

Model	Method	α	C-Eval	MMLU	MMLU-Pro	LIMO	IFEval	SciQ	RTE	TruthfulQA	Avg
Qwen3-30B-A3B	**Baseline**	–	**82.1**	**82.9**	**72.2**	92.7	**83.9**	96.5	82.3	**56.2**	**81.1**
	Skip Tail	2	81.9	82.6	71.1	**93.3**	82.4	96.4	82.3	54.5	80.6
		3	78.8	80.4	65.9	79.0	76.2	95.7	82.3	54.4	76.6
	Without Limit	2	78.9	80.4	65.7	76.4	78.7	96.4	83.4	53.9	76.7
		3	80.8	81.8	69.6	88.7	82.3	**97.0**	83.4	55.1	79.8
	Ours	2	81.9	82.7	71.4	92.4	83.6	96.6	**83.8**	55.8	81.0
		3	81.5	82.0	70.1	90.1	81.9	96.4	82.7	55.6	80.0
Qwen3.6-35B-A3B	**Baseline**	–	89.7	**90.5**	84.9	**84.0**	83.6	**96.8**	**74.7**	64.4	83.6
	Skip Tail	2	89.7	90.0	84.1	75.5	79.9	94.7	67.2	63.1	80.5
		3	83.8	86.8	78.6	35.0	71.0	92.4	65.0	61.2	71.7
	Without Limit	2	88.6	89.5	82.7	56.2	78.2	94.4	70.0	64.2	78.0
		3	89.2	89.8	83.8	79.1	81.7	96.0	71.5	64.1	81.9
	Ours	2	**90.1**	90.4	**85.0**	**84.0**	**86.0**	96.6	74.4	**64.7**	**83.9**
		3	89.5	89.7	83.9	79.9	82.3	96.2	71.8	64.0	82.2

G.2 Effect of the Communication Guarantee

We ablate the communication guarantee in the redundant expert placement solver, which preserves source-node service when a local expert copy is available. Table 8 compares Albus with and without the guarantee against native SGLang. Enabling the guarantee provides 1.12–1.17× prefill speedups over disabling it, averaging 1.14× across the four per-rank loads. These gains support preserving node-local service during load balancing.

Table 8: GLM-4.7 prefill latency (ms): PG-19, EP32. Both Albus variants use $\alpha = 2$. w/o guarantee means without the communication guarantee.

Per-rank load	Baseline	Albus w/o guarantee	Albus
4K	1786.50	1414.52	**1210.81**
8K	3791.71	2547.34	**2228.94**
12K	5959.44	3809.95	**3416.07**
16K	7987.02	5249.11	**4705.75**