J-MoE: Attention-Aware Joint Sample–Expert Placement for MoE RL Training

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning (RL) post-training has become central to improving large language models, which increasingly adopt mixture-of-experts (MoE) architectures. However, variable-length rollout samples and skewed expert routing cause substantial load imbalance across GPUs in three major cost components: dense attention computation, sparse expert computation, and routed All-to-All communication. Balancing these costs requires coordinating two tightly coupled decisions: sample placement and expert placement. Sample placement determines both attention workloads and the sources of routed traffic, while expert placement determines both expert workloads and traffic destinations. Existing load-balancing techniques address only subsets of these costs, without jointly optimizing all three through coordinated sample and expert placement.

We present J-MoE, an MoE RL training system that jointly plans sample and expert placement to trade off among all three costs at once. J-MoE leverages routing replay, in which training reuses the routing decisions recorded during rollout, making each sample's expert demand known exactly before training begins. Guided by this information and a unified cost model of all three components, a two-stage solver first assigns the samples of each optimizer step to microbatches and GPUs, and then jointly optimizes sample and expert placement for each microbatch. To execute these plans efficiently, J-MoE redistributes samples to their assigned GPUs and overlaps expert reconfiguration with attention computation, without altering routing or dropping tokens. Across multiple MoE models and RL datasets, J-MoE achieves up to $1.93\times$ training speedup over existing training systems.

1 Introduction

Reinforcement learning (RL) post-training has become a widely adopted paradigm for large language models (Sheng et al., 2025; Yu et al., 2025), and it is increasingly applied to mixture-of-experts (MoE) models (DeepSeek-AI, 2024; Guo et al., 2025; Qwen Team, 2025), which scale model capacity without a proportional increase in computation (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). Each RL step first rolls out responses for a batch of prompts and then updates the policy on them using expert parallelism (EP), where MoE experts are distributed across GPUs and routed tokens are exchanged through All-to-All communication, as shown in Fig. 1. This training stage suffers from two forms of load imbalance. First, dynamic routing sends uneven numbers of tokens to different experts, so GPUs hosting heavily loaded experts become stragglers. Second, rollout responses differ widely in length, so different samples incur very different attention workloads.

Fig. 3 quantifies both forms of imbalance. Expert load distributions not only deviate from a uniform distribution but also differ across microbatches, with different microbatches exhibiting different hot experts, as illustrated in Fig. 3(a) and (b). This variation makes it difficult for a single batch-level expert placement to match the expert demand of every microbatch. Response lengths also vary widely, and this variation persists across datasets and model configurations, as shown in Fig. 3(c–e), so samples incur highly variable attention workloads.

These two forms of imbalance affect three major components of training: attention computation, expert computation, and All-to-All communication. The attention workload of each GPU depends on the samples it processes, while its expert workload depends on the experts it hosts and their routing demand. Communication connects the two: routed tokens must be dispatched from the GPUs

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

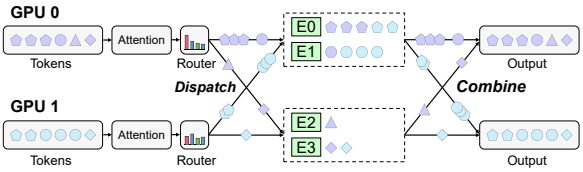


Figure 1: MoE training with expert parallelism.

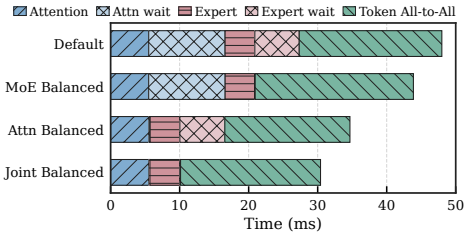
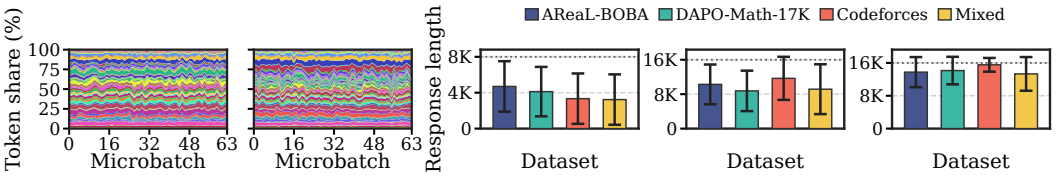


Figure 2: Per-layer forward time breakdown under three balancing strategies.



(a) DAPO-Math-17K (b) Mixed (c) Qwen3 Instruct, 8K (d) Qwen3, 16K (e) Qwen3.5, 16K

Figure 3: Expert-load and response-length variation in MoE RL training. (a,b) Per-expert token shares across microbatches on different datasets. (c–e) Mean and standard deviation of response lengths per dataset under three model configurations; dotted lines mark the maximum response length.

holding their samples to the GPUs hosting their selected experts and combined afterwards. Balancing these costs therefore involves two coupled decisions: assigning samples to microbatches and GPUs (*sample placement*) and assigning experts to GPUs (*expert placement*).

These two decisions are tightly coupled. Relocating a sample shifts attention load between GPUs, moves the source of its routed tokens, and, if it crosses microbatches, changes the aggregate expert demand of both microbatches. Likewise, relocating an expert redistributes expert load across GPUs and changes the destinations of routed tokens. Optimizing either placement while holding the other fixed may therefore miss a better joint configuration. Fig. 2 illustrates this opportunity. Compared with the default placement, *MoE Balanced* keeps sample placement fixed and balances expert load, but GPUs still wait for the slowest attention computation. *Attn Balanced* balances attention through sample placement but leaves expert computation imbalanced. *Joint Balanced* coordinates both placements, trades off among all three cost components, and achieves the shortest critical path.

Exploiting this opportunity requires knowing each sample's routing demand in advance. Expert-balancing methods developed for MoE pretraining instead rely on historical expert-load statistics to determine expert placement or replication (Nie et al., 2023; Zhai et al., 2023; Liu et al., 2026), but the microbatch-level variation in Fig. 3(a) and (b) makes such statistics unrepresentative of the upcoming microbatch. MoE RL training removes the need for such prediction: because rollout and training process the same samples, the routing decisions recorded during rollout can be replayed during training (Ma et al., 2025), exposing the exact routing demand of each optimizer step before it starts. Recent MoE RL systems exploit this property: ReLibra (Jin et al., 2026b) first performs batch-level expert reordering and then adjusts sample placement, ForeMoE (Zhou et al., 2026) performs fine-grained expert reconfiguration without optimizing sample placement, and RoutePack (Shen et al., 2026) optimizes sample placement and expert placement using routing demand and sequence length while overlooking communication cost. None of them, however, decides the two placements jointly under a single objective that covers all three cost components.

Based on these observations, we present J-MoE, a system that jointly optimizes sample and expert placement for MoE RL training. Using the exact routing demand exposed by routing replay, J-MoE scores both placements with a unified cost model that accounts for attention computation, expert computation, and All-to-All communication. Its two-stage solver first assigns the samples of each optimizer-step batch to microbatches and GPUs, and then alternately refines sample and expert placement within each microbatch. To execute the resulting placement, J-MoE redistributes complete samples before training and prefetches expert parameters while attention computation is running. Across multiple MoE models and datasets, J-MoE achieves up to a 1.93× training speedup over existing training systems.

In summary, this paper makes the following contributions:

- We characterize expert-load imbalance and response-length imbalance in MoE RL training and show how they translate into microbatch-level imbalance in attention computation, expert computation, and All-to-All communication.
- We formulate sample and expert placement as a single optimization problem under a unified cost model covering all three components, and design a two-stage solver that first builds a batch-level sample placement and then alternately refines both placements within each microbatch.
- We build a runtime that applies the resulting placement by redistributing samples before each optimizer step and overlapping expert reconfiguration with attention computation. Across multiple MoE models and datasets, J-MoE achieves up to 1.93× speedup over existing training systems.

2 Background and Related Work

2.1 MoE Training and RL Post-Training

MoE training. Mixture-of-experts (MoE) layers route each token to its top-k experts (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022), scaling model parameters while keeping the activated computation per token nearly unchanged. Under expert parallelism (EP), experts are partitioned across GPUs, as illustrated in Fig. 1. Token activations are dispatched to the GPUs hosting their selected experts through an All-to-All collective, and the expert outputs are returned through another All-to-All. Routing is typically skewed, so both expert loads and All-to-All traffic differ across GPUs, and each layer proceeds at the pace of its slowest GPU.

RL post-training and routing replay. RL training consists of a rollout stage that generates responses and a training stage that updates the policy on the generated samples (Sheng et al., 2025; Yu et al., 2025; Zhou et al., 2026). We refer to one prompt and its generated response as a sample. Each optimizer step processes a batch of samples, which is split into microbatches that are executed sequentially with gradient accumulation to bound activation memory. For MoE policies, rollout and training process the same generated samples, so the token-to-expert routing decisions recorded during rollout can be replayed during training, which aligns the training and inference routers and stabilizes optimization (Ma et al., 2025; Jin et al., 2026b). As a result, the exact routing demand of each optimizer step is known before training starts, rather than predicted from historical load statistics.

2.2 Related Work

Algorithmic load balancing methods. To mitigate expert load imbalance, one line of work reshapes routing itself. Auxiliary balancing losses encourage more balanced routing (Lepikhin et al., 2021; Fedus et al., 2022), capacity constraints and token dropping limit the number of tokens assigned to each expert (Lepikhin et al., 2021; Rajbhandari et al., 2022), and other methods redesign the routing algorithm (Lewis et al., 2021; Wang et al., 2024b). Because these methods change or constrain routing decisions, they cannot rebalance a training step whose routing is inherited from rollout through routing replay.

System-level load balancing methods. The other line of work rebalances load at the system level without changing routing. Most existing works reconfigure expert placement and replication based on historical load statistics (Nie et al., 2023; Zhai et al., 2023; DeepSeek-AI, 2025; Liu et al., 2026), assuming that expert demand is stable over time; in RL training, however, it shifts from one microbatch to the next. NetMoE optimizes sample placement to improve communication locality while keeping expert placement fixed (Liu et al., 2025). Recent works exploit routing replay for MoE RL training: ReLibra performs expert reordering, expert replication, and sample reordering (Jin et al., 2026b); ForeMoE performs fine-grained expert reconfiguration (Zhou et al., 2026); and RoutePack considers both routing demand and sequence length during sample placement and expert placement while overlooking communication cost (Shen et al., 2026). Each of them optimizes only a subset of attention computation, expert computation, and All-to-All communication, rather than all three jointly.

Computation and communication optimization. A broad line of work improves distributed MoE execution. For expert computation, MegaBlocks (Gale et al., 2023) introduces block-sparse kernels

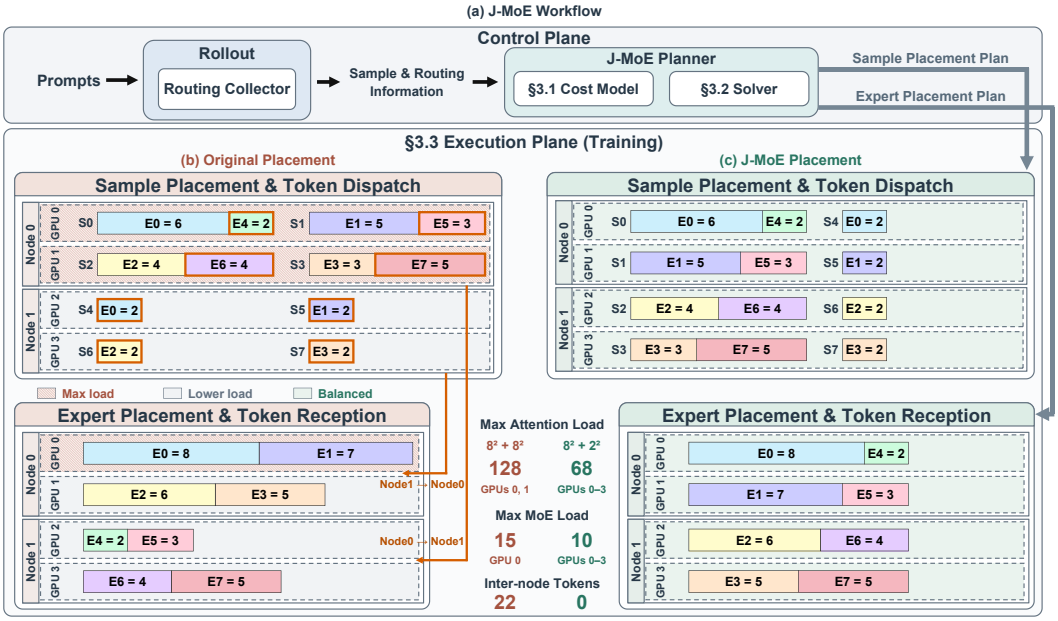


Figure 4: Overview of J-MoE. (a) J-MoE collects sample-level routing demand during rollout, jointly plans sample and expert placement, and applies the resulting placement during training. (b) The original placement exhibits imbalanced attention and expert loads and incurs inter-node token communication, shown in orange. (c) J-MoE coordinates sample and expert placement to balance attention and expert loads while reducing inter-node token communication.

Table 1: Notation.

Symbol	Description
B, M	Global batch size; microbatch count
N, L, E	GPU count; MoE-layer count; expert count
$\mathcal{S}, \mathcal{E}$	Sample set; expert set
n	Sample sequence lengths
d	Per sample routing demand
r	Per microbatch aggregated expert demand
$\mathcal{P}^S, \mathcal{P}^E$	Sample and expert placements

Table 2: Evaluated model configurations.

Model	Layers	Experts	Top-k	Response	EP
Qwen3 Instruct	48	128	8	8K	16, 32
Qwen3	48	128	8	16K	32
Qwen3.5	40	256	8	16K	32

and SonicMoE (Guo et al., 2026) overlaps memory IO with computation. For communication, DeepEP (Zhao et al., 2025) optimizes token dispatch and combine, while FSMoE (Pan et al., 2025), COMET (Zhang et al., 2025), and MegaScale-MoE (Jin et al., 2026a) improve computation-communication overlap through execution scheduling and communication-efficient parallelism. These efforts optimize MoE training under a given placement and are complementary to J-MoE, which decides the placement itself by jointly planning samples and experts within each optimizer step.

Resource scheduling in RL post-training systems. Recent RL post-training systems improve end-to-end efficiency through resource scheduling and execution optimizations across rollout and training. AReaL, StreamRL, and LlamaRL improve resource utilization through disaggregated or asynchronous execution (Fu et al., 2025; Zhong et al., 2025a; Wu et al., 2025), while RLHFuse overlaps different stages of the RL pipeline (Zhong et al., 2025b) and Seer accelerates synchronous rollout (Qin et al., 2026). These systems optimize the execution and scheduling of the RL pipeline and are orthogonal to J-MoE, which focuses on the training stage and addresses the load imbalance introduced by MoE execution and variable-length rollout samples.

3 Design

In this section, we present J-MoE, which jointly plans sample and expert placement within each optimizer step. Unlike prior systems that optimize one placement while keeping the other fixed,

J-MoE treats the two as a single decision, scoring them together under one cost model that covers attention computation, expert computation, and All-to-All communication. Fig. 4(a) shows the workflow. During rollout, a routing collector gathers the sequence length of every sample and its per-layer token-to-expert routing decisions. Because routing replay reuses these decisions during training, the collected data describes the upcoming optimizer step exactly. Guided by the cost model (Sec. 3.1), a two-stage joint solver (Sec. 3.2) first assigns each sample to a microbatch and a GPU, and then jointly refines the sample and per-layer expert placement within each microbatch. Finally, the placement executor (Sec. 3.3) redistributes samples before each optimizer step and overlaps expert reconfiguration with attention computation during training. Tab. 1 summarizes the frequently used notation.

3.1 Problem Formulation

Problem statement. Each optimizer step is given a batch of B samples $\mathcal{S} = \{s_1, \ldots, s_B\}$, where B is the global batch size, and the expert sets $\mathcal{E}_l = \{e_{l,1}, \ldots, e_{l,E}\}$ of the L MoE layers, where E is the number of experts per layer. The batch is processed as M microbatches on N GPUs. The routing collector provides the sequence lengths $n \in \mathbb{N}^B$ and the routing demand $d \in \mathbb{N}^{B\times L\times E}$, where $d_{s,l,e}$ is the number of tokens from sample s routed to expert e at layer l. J-MoE jointly solves for two placements. The *sample placement* $\{\mathcal{P}^S_{m,g}\}$ partitions the samples across M microbatches and N GPUs of each microbatch, where $\bigcup_{m,g} \mathcal{P}^S_{m,g} = \mathcal{S}$. The *expert placement* $\{\mathcal{P}^E_{m,l,g}\}$ distributes the experts of every layer across N GPUs, separately for each microbatch, where $\bigcup_g \mathcal{P}^E_{m,l,g} = \mathcal{E}_l$. Let $[K] = \{1, \ldots, K\}$. Since microbatches and layers execute sequentially and each is bottlenecked by its slowest GPU, J-MoE plans the two placements by solving

$$\begin{aligned} \min_{\mathcal{P}^S,\,\mathcal{P}^E} \quad & F_{\text{joint}} = \sum_{m\in[M]} F_{\text{m}} = \sum_{m\in[M]} \sum_{l\in[L]} \Big(T^{\text{attn}}_{m,l}(\mathcal{P}^S, n) + T^{\text{expert}}_{m,l}(\mathcal{P}^S, \mathcal{P}^E, d) + T^{\text{comm}}_{m,l}(\mathcal{P}^S, \mathcal{P}^E, d) \Big) \\ \text{s.t.} \quad & \bigcup_{m,g} \mathcal{P}^S_{m,g} = \mathcal{S}, \qquad \bigcup_{g} \mathcal{P}^E_{m,l,g} = \mathcal{E}_l \quad \forall (m,l) \in [M]\times[L], \end{aligned} \tag{1}$$

where $T^{\text{attn}}_{m,l}$ and $T^{\text{expert}}_{m,l}$ are the attention and expert-computation costs of layer l in microbatch m, each determined by the most heavily loaded GPU, and $T^{\text{comm}}_{m,l}$ is the All-to-All communication cost of that layer, determined by its bottleneck channel.

Necessity of joint placement. To make the benefit of joint placement concrete, Fig. 4(b) and (c) give an example with top-1 routing and a simplified cost model, in which the attention cost of a sample is its squared sequence length, the expert cost is the number of tokens routed to an expert, and the communication cost is the total number of inter-node routed tokens. In the original placement of Fig. 4(b), the two samples with $n = 8$ share one GPU, giving an attention bottleneck of $8^2+8^2 = 128$; the two most loaded experts E_0 and E_1 are colocated, giving an expert bottleneck of 15; and the orange boxes mark 22 inter-node tokens. Planning both placements against the actual load yields the placement in Fig. 4(c): each long sample is paired with a short one to balance attention, the heavily loaded experts are spread across GPUs so that every GPU receives the same total demand, and each sample is placed on the node that hosts the experts it routes to. The attention and expert bottlenecks drop to $8^2 + 2^2 = 68$ and 10, and the number of inter-node tokens drops to 0.

These improvements involve distinct but coupled placement decisions. Within a microbatch, moving samples changes attention load and the source GPUs of routed tokens, while moving experts changes expert load and the destination GPUs. Communication therefore depends on both placements: a sample assignment that provides good locality under one expert placement may incur substantial inter-node traffic under another. Allowing samples to move across microbatches introduces an additional dependency, since changing microbatch membership also changes the aggregate demand of each expert. Optimizing either placement without accounting for these interactions can miss better joint configurations, and a one-pass sequential strategy has the same limitation, as it never revisits its first decision after optimizing the second. This motivates J-MoE to score both placements under a unified objective and to refine them alternately, so that each adapts to changes in the other.

Cost model. We now model the three terms of Eq. 1. Following prior work (Pan et al., 2025; Chen et al., 2024; 2025), we adopt the α-β model (Sarvotham et al., 2001) to estimate computation and communication overheads, where α denotes a size-independent startup cost and β the cost per unit of

work; both are calibrated through profiling. The *attention cost* depends only on the sample placement and is determined by the most heavily loaded GPU:

$$T^{\text{attn}}_{m,l}(\mathcal{P}^S) = \max_{g\in[N]} \sum_{s\in\mathcal{P}^S_{m,g}} \left(\beta_1 n_s^2 + \beta_2 n_s\right) + \alpha_1, \tag{2}$$

where the quadratic term captures attention-score computation and the linear term the projections that scale with sequence length. The *expert-computation cost* depends on both placements. The sample placement fixes the total number of tokens each expert receives in a microbatch: let $r_{m,l,e}(\mathcal{P}^S) = \sum_{s\in\mathcal{P}^S_m} d_{s,l,e}$ be the aggregate demand of expert e at layer l of microbatch m. The expert placement then decides how that demand is distributed across GPUs. The cost is likewise determined by the most heavily loaded GPU:

$$T^{\text{expert}}_{m,l}(\mathcal{P}^S, \mathcal{P}^E) = \beta_3 \max_{g\in[N]} \sum_{e\in\mathcal{P}^E_{m,l,g}} r_{m,l,e}(\mathcal{P}^S) + \alpha_2. \tag{3}$$

The *All-to-All communication cost* comprises the dispatch and combine collectives and also depends on both placements: the sample placement determines the source GPU of every routed token, and the expert placement its destination. This communication traverses two channels, NVLink and RDMA, and its cost is the larger of the two. In practice RDMA has substantially lower bandwidth and is therefore the bottleneck (Wang et al., 2024a; Yao et al., 2024; Singhania et al., 2026), so we model inter-node traffic only. Specifically, the number of tokens sent from GPU g to GPU g' at layer l of microbatch m is $f_{m,l,g,g'} = \sum_{s\in\mathcal{P}^S_{m,g}} \sum_{e\in\mathcal{P}^E_{m,l,g'}} d_{s,l,e}$. Since combine returns the same tokens in the reverse direction, the communication cost is

$$T^{\text{comm}}_{m,l}(\mathcal{P}^S, \mathcal{P}^E) = 2 \times \beta_4 \sum_{(g,g')\in\mathcal{N}^{\text{inter}}} f_{m,l,g,g'} + \alpha_3, \tag{4}$$

where $\mathcal{N}^{\text{inter}} = \{(g, g') \mid \text{nd}(g) \neq \text{nd}(g')\}$ is the set of GPU pairs connected over RDMA and $\text{nd}(g)$ denotes the node hosting GPU g.

For models that interleave different attention types, such as Qwen3.5, we calibrate the attention coefficients per type; Appendix A gives the details.

3.2 Two-stage Joint Solver

Eq. 1 has to be solved before every optimizer step, but solving it exactly is NP-hard and thus infeasible within the available time. J-MoE therefore adopts a two-stage heuristic solver, as shown in Alg. 1. It first constructs a sample placement over the whole optimizer-step batch (lines 1–11), and then jointly refines sample and expert placement within each microbatch (lines 13–17).

Batch-level sample placement. The first stage assigns samples one at a time to a microbatch and a GPU within it. A simple observation guides the order of assignment: samples placed earlier have more feasible destinations, whereas those placed later are constrained by the remaining slots. We therefore place samples in decreasing order of sequence length, since long samples dominate the attention cost and are the hardest to accommodate once the microbatches fill up. Throughout the stage, the solver maintains $r \in \mathbb{N}^{M\times L\times E}$, where $r_{m,l,e}$ is the aggregate demand of expert e at layer l accumulated by the samples already committed to microbatch m. We also precompute a token capacity C, the largest number of tokens a GPU can hold under its memory budget. For each sample s, the solver enumerates the destination slots (m, g) that still have room, i.e., those satisfying $\sum_{p\in\mathcal{P}^S_{m,g}} n_p + n_s \leq C$. Placing s in microbatch m raises its expert demand from $r_{m,l,\cdot}$ to $r_{m,l,\cdot} + d_{s,l,\cdot}$, so the solver runs the Longest Processing Time (LPT) algorithm (Graham, 1969) on the updated demand of every layer to obtain a conditional expert placement $\mathcal{P}'^E_m$. LPT sorts experts by decreasing demand and assigns each to the currently least-loaded GPU with an available expert slot. Each candidate slot is then scored with F_{m} in Eq. 1, which weighs attention, expert, and All-to-All communication costs jointly, and s is committed to the cheapest slot. Once all samples are committed, the initial sample placement $\mathcal{P}^S$ is fixed, and LPT is applied once more to every layer of every microbatch to obtain the initial expert placement $\mathcal{P}^E$.

Joint refinement. After initialization, each microbatch is refined independently and in parallel. For microbatch m, J-MoE alternates Sample-KL and Expert-KL, two passes that apply the same

Algorithm 1 Two-stage joint placement in J-MoE.

Require: Sequence lengths $\{n_s\}$, routing demands $\{d_{s,l,e}\}$, token capacity C
Ensure: Sample placement $\mathcal{P}^S$ and expert placement $\mathcal{P}^E$

// Stage 1: batch-level sample placement
sort samples by decreasing n_s
$\mathcal{P}^S \leftarrow \emptyset$; $r \leftarrow \mathbf{0}$
for each sample s in this order **do**
  BestCost $\leftarrow \infty$
  for each (m, g) with a slot satisfying C **do**
    ${\mathcal{P}'}^E_m \leftarrow \mathrm{LPT}(r_{m,l,\cdot} + d_{s,l,\cdot})$ for every layer l
    Cost $\leftarrow F_m$ under $s \mapsto (m, g)$ and ${\mathcal{P}'}^E_m$
    if Cost $<$ BestCost **then**
      BestCost $\leftarrow$ Cost, $(m^\star, g^\star) \leftarrow (m, g)$
  $\mathcal{P}^S_{m^\star,g^\star} \leftarrow \mathcal{P}^S_{m^\star,g^\star} \cup \{s\}$
  $r_{m^\star,l,\cdot}$ += $d_{s,l,\cdot}$
$\mathcal{P}^E_{m,l} \leftarrow \mathrm{LPT}(r_{m,l,\cdot})$ for every layer l
// Stage 2: joint refinement within each microbatch
for all microbatches m **in parallel do**
  repeat
    $\mathcal{P}^S_m \leftarrow$ SAMPLE-KL$(\mathcal{P}^S_m, \mathcal{P}^E_m)$
    $\mathcal{P}^E_m \leftarrow$ EXPERT-KL$(\mathcal{P}^S_m, \mathcal{P}^E_m)$
  until neither pass reduces F_m
return $(\mathcal{P}^S, \mathcal{P}^E)$

Kernighan–Lin-style pair-swap refinement (Kernighan & Lin, 1970) to the sample and expert placements, respectively, and score every candidate swap directly with F_m.

A KL pass explores a sequence of swaps rather than greedily committing one improving swap at a time. Starting from the current placement, the solver repeatedly selects the highest-gain pair among the remaining unlocked items, provisionally applies the swap, locks the two selected items for the rest of the pass, and recomputes the remaining swap gains under the updated placement. Individual swaps need not improve F_m, allowing the search to pass through temporarily worse placements to reach a better combination of subsequent swaps. The pass then commits the prefix of the swap sequence with the largest cumulative gain, or retains the original placement if no prefix improves on it.

Sample-KL swaps pairs of samples on different GPUs while keeping $\mathcal{P}^E_m$ fixed. A sample swap changes the attention load and the source GPUs of routed tokens, but leaves the aggregate expert demand $r_{m,l,e}$ unchanged. Expert-KL swaps pairs of experts of the same MoE layer on different GPUs while keeping $\mathcal{P}^S_m$ fixed. An expert swap changes the expert load and token destinations of that layer, but leaves attention computation unchanged. The two passes alternate until neither further reduces F_m. Candidate swaps are evaluated incrementally by updating only the affected attention, expert-computation, and communication costs; Appendix B gives more details.

3.3 System Implementation

We implement J-MoE on top of verl (Sheng et al., 2025), with vLLM (Kwon et al., 2023) for rollout and Megatron-LM (Narayanan et al., 2021) for training. During rollout, we record the token-to-expert routing decisions of each sample and aggregate them into the routing demand used for placement planning. The solver is implemented in C++ with OpenMP (Dagum & Menon, 1998). Placement planning starts once the rollout routing information is available and overlaps with post-rollout processing, keeping it off the training critical path.

Before each optimizer step, J-MoE redistributes complete samples to their planned GPUs. During training, the executor applies the planned expert placement layer by layer. For each layer, the expert parameters required by its MoE computation are asynchronously prefetched on CUDA streams while its attention computation is running, whereas optimizer states remain on their home ranks. During backward, expert gradients are returned to their home ranks concurrently with attention computation. More details are given in Appendix C. These operations change only sample and expert placement and preserve the token-to-expert routing decisions recorded during rollout. J-MoE neither drops tokens nor alters routing, so the training semantics are unchanged and model accuracy is unaffected.

4 Evaluation

4.1 Experimental Setups

Testbed. We conduct all experiments on a 4-node cluster with 8 NVIDIA H20 GPUs per node. GPUs within a node are connected by NVLink with 450 GB/s peak unidirectional bandwidth, and nodes are connected by InfiniBand with 8×200 Gbps per node.

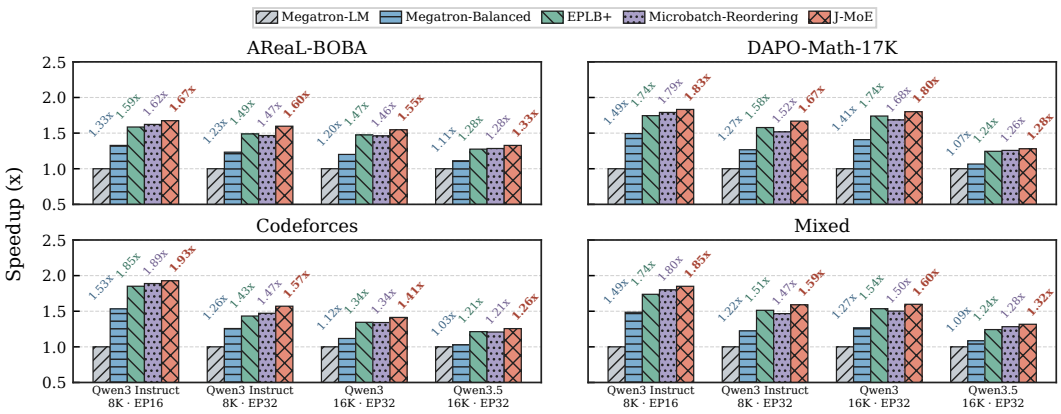


Figure 5: Training speedup over Megatron-LM across different workloads.

Models and datasets. We evaluate three MoE models from the Qwen3 and Qwen3.5 families (Qwen Team, 2025; 2026). Tab. 2 summarizes their model and parallelism configurations. We use four RL datasets: AReaL-BOBA (xDAN-datasets, 2025), DAPO-Math-17K (Yu et al., 2025), Codeforces (Penedo et al., 2025), and *Mixed*, which combines prompts from IF-RLVR (Pyatkin et al., 2025), DAPO-Math-17K, and Codeforces. Each optimizer step contains 128 prompts with 4 responses per prompt. For each configuration, all methods train on the same generated responses.

Baselines. We compare J-MoE with four baselines. **Megatron-LM** (Narayanan et al., 2021) uses the standard expert placement without MoE load balancing. **Megatron-Balanced** uses the default static expert placement as Megatron-LM and enables verl's built-in sequence-length balancing, which partitions samples across DP ranks and microbatches to balance their token counts. **EPLB$^+$** strengthens EPLB (DeepSeek-AI, 2025) by replacing historical expert-load statistics with the exact batch-level routing demand obtained from routing replay, and optimizes expert placement accordingly while keeping sample placement fixed. **Microbatch-Reordering** follows the expert and sample placement strategies of ReLibra (Jin et al., 2026b), applied at microbatch granularity. For each microbatch, it determines expert placement from the replayed routing demand, and then reorders samples while keeping expert placement fixed. EPLB$^+$ and Microbatch-Reordering enable the same sequence-length balancing method as Megatron-Balanced, whereas J-MoE determines sample placement with its joint solver. All methods use the same training runtime and parallelism configuration.

Metrics. We report training-stage time, as J-MoE optimizes training execution and is orthogonal to rollout optimization. After a 10-step warm-up, we report the average over the next 10 optimizer steps.

4.2 Experimental Results

End-to-end training performance. Fig. 5 reports the training speedup over Megatron-LM across different models, datasets, and training configurations. J-MoE achieves the highest speedup in every configuration, up to 1.93× over Megatron-LM and up to 1.32× over Megatron-Balanced.

EPLB$^+$ determines expert placement from the aggregate routing demand of the optimizer-step batch while keeping sample placement unchanged. Although routing replay removes the uncertainty of history-based load estimation, a single batch-level expert placement cannot adapt to the expert load distribution of individual microbatches. In contrast, J-MoE jointly optimizes sample and expert placement at microbatch granularity while accounting for attention computation, expert computation, and All-to-All communication, achieving up to 1.10× speedup over EPLB$^+$.

Compared with Microbatch-Reordering, J-MoE introduces two additional degrees of freedom. First, samples can move across microbatches within the same optimizer step. Second, sample and expert placement are jointly refined rather than optimized sequentially. These differences provide a further speedup of up to 1.10×. Microbatch-Reordering occasionally underperforms EPLB$^+$ because its MoE-only objective ignores attention balance: its sample reordering can reintroduce attention imbalance, whose cost may outweigh its gains on the MoE path.

Ablation study. Fig. 6 isolates the contribution of the two solver stages on Qwen3.5 (16K, EP32), where Stage 1 is the batch-level sample placement and Stage 2 is the KL-style joint refinement.

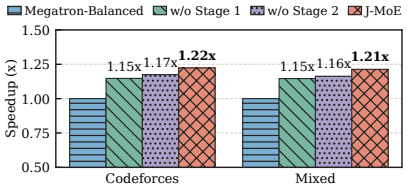
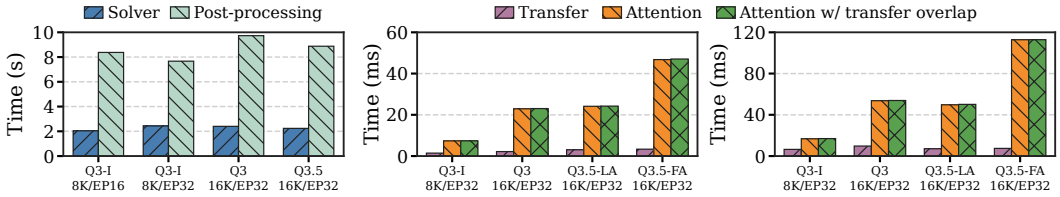


Figure 6: Ablation study.

Table 3: Load balance and inter-node communication of different placement strategies. Lower is better for all metrics.

Method	Attn. Imbal.	Expert Imbal.	Traffic (GiB)
Megatron-LM	2.716	1.772	2795.04
Megatron-Balanced	1.034	1.815	2789.96
EPLB$^+$	1.034	1.259	2787.66
Microbatch-Reordering	1.107	1.102	2605.01
J-MoE	**1.015**	**1.057**	**2238.28**



(a) Solver overlap (b) Parameter-transfer overlap (c) Gradient-transfer overlap

Figure 7: Planning and expert-reconfiguration overheads. Q3-I, Q3, and Q3.5 denote Qwen3 Instruct, Qwen3, and Qwen3.5; LA and FA denote linear and full attention.

Relative to Megatron-Balanced, applying Stage 2 alone (w/o Stage 1) speeds up training by $1.15\times$ on Codeforces and $1.15\times$ on Mixed, showing that joint pair-swap refinement further improves an attention-balanced placement. Applying Stage 1 alone (w/o Stage 2) yields larger speedups of $1.17\times$ and $1.16\times$, indicating that a good initialization matters at least as much as the refinement. Combining both stages gives the full J-MoE and the highest speedups, $1.22\times$ and $1.21\times$.

Case study. To understand how the placement strategies affect execution, we examine the load distribution and inter-node traffic on Mixed with Qwen3 Instruct (8K, EP32). Across all microbatches and layers, we measure attention and expert imbalance as the ratio of the per-GPU maximum load to the per-GPU mean load (1 means perfectly balanced), together with the total inter-node traffic, as shown in Tab. 3. Megatron-Balanced cuts attention imbalance from 2.716 to 1.034, but expert imbalance rises slightly from 1.772 to 1.815 because it balances attention only. EPLB$^+$ lowers expert imbalance to 1.259 but cannot go further, as a single batch-level expert placement cannot track the demand variation across microbatches. Microbatch-Reordering reduces expert imbalance to 1.102 by acting per microbatch, but raises attention imbalance to 1.107 because attention is absent from its objective. In contrast, J-MoE achieves the lowest attention and expert imbalance (1.015 and 1.057) and cuts inter-node traffic to 2238 GiB, 14% less than Microbatch-Reordering and 20% less than Megatron-LM, which is exactly what the joint objective in Eq. 1 targets.

Solver performance. Fig. 7(a) compares the solver time with the post-rollout processing window. Since stage 1 is a single greedy pass with polynomial cost and stage 2 runs independently across microbatches, the solver completes within the post-processing window in all evaluated configurations, so placement planning does not extend the training critical path.

Expert reconfiguration performance. Figs. 7(b) and (c) report the overlap between expert reconfiguration and attention computation. Both the BF16 parameter prefetch in the forward pass and the FP32 gradient return in the backward pass are fully hidden by attention computation in all configurations shown, illustrating that expert reconfiguration does not extend the per-layer critical path.

5 Conclusion

We present J-MoE, a system that jointly plans sample and expert placement for MoE RL training. Exploiting the exact routing demand that routing replay exposes before each optimizer step, J-MoE scores the two placements under a single cost model covering attention computation, expert computation, and All-to-All communication. A two-stage solver first builds a batch-level sample placement and then refines both sample and expert placements against each other within every microbatch, and a runtime applies the resulting plan by redistributing samples and overlapping expert reconfiguration with attention computation. Across multiple MoE models and datasets, J-MoE achieves up to a $1.93\times$ training speedup over existing training systems.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing, survey the literature, and debug code. We have reviewed all AI-assisted work and we take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Jinfan Chen, Shigang Li, Ran Guo, Jinhui Yuan, and Torsten Hoefler. AutoDDL: Automatic Distributed Deep Learning With Near-Optimal Bandwidth Cost. *IEEE Transactions on Parallel and Distributed Systems*, 35:1331–1344, 2024.

Tiancheng Chen, Ales Kubicek, Langwen Huang, and Torsten Hoefler. CrossPipe: Towards Optimal Pipeline Schedules for Cross-Datacenter Training. In *USENIX Annual Technical Conference (USENIX ATC)*, pp. 1089–1108, 2025.

Leonardo Dagum and Ramesh Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science & Engineering*, 5:46–55, 1998.

DeepSeek-AI. DeepSeek-V3 Technical Report. *CoRR*, abs/2412.19437, 2024.

DeepSeek-AI. Expert Parallelism Load Balancer (EPLB). `https://github.com/deepseek-ai/EPLB`, 2025.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23:120:1–120:39, 2022.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AREAL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

Ronald L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, Hao Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, Tao Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan

Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

Wentao Guo, Mayank Mishra, Xinle Cheng, Ion Stoica, and Tri Dao. SonicMoE: Accelerating MoE with IO and Tile-aware Optimizations. In *International Conference on Learning Representations (ICLR)*, 2026.

Chao Jin, Ziheng Jiang, Zhihao Bai, Zheng Zhong, Juncai Liu, Xiang Li, Ningxin Zheng, Xi Wang, Cong Xie, Qi Huang, Wen Heng, Yiyuan Ma, Wenlei Bao, Size Zheng, Xuegui Zheng, Yanghua Peng, Haibin Lin, Xuanzhe Liu, Xin Jin, and Xin Liu. MegaScale-MoE: Large-Scale Communication-Efficient Training of Mixture-of-Experts Models in Production. In *European Conference on Computer Systems (EuroSys)*, pp. 366–382, 2026a.

Chao Jin, Xinming Wei, Yinmin Zhong, Chengxu Yang, Bingyang Wu, Ruidong Zhu, Zili Zhang, Yuliang Liu, and Xin Jin. ReLibra: Routing-Replay-Guided Load Balancing for MoE Training in Reinforcement Learning. *CoRR*, abs/2605.08639, 2026b.

Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, 1970.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 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 *International Conference on Learning Representations (ICLR)*, 2021.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE Layers: Simplifying Training of Large, Sparse Models. In *International Conference on Machine Learning (ICML)*, pp. 6265–6274, 2021.

Xinyi Liu, Yujie Wang, Fangcheng Fu, Xupeng Miao, Shenhan Zhu, Xiaonan Nie, and Bin Cui. NetMoE: Accelerating MoE Training through Dynamic Sample Placement. In *International Conference on Learning Representations (ICLR)*, 2025.

Xinyi Liu, Yujie Wang, Fangcheng Fu, Xuefeng Xiao, Huixia Li, Jiashi Li, and Bin Cui. LAER-MoE: Load-Adaptive Expert Re-layout for Efficient Mixture-of-Experts Training. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, volume 2, pp. 1055–1072, 2026.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE Reinforcement Learning by Aligning Training and Inference Routers. *CoRR*, abs/2510.11370, 2025.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using megatron-LM. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 58, 2021.

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. *Proceedings of the ACM on Management of Data*, 1:110:1–110:19, 2023.

Xinglin Pan, Wenxiang Lin, Lin Zhang, Shaohuai Shi, Zhenheng Tang, Rui Wang, Bo Li, and Xiaowen Chu. FSMoE: A Flexible and Scalable Training System for Sparse Mixture-of-Experts Models. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, volume 1, pp. 524–539, 2025.

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.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing Verifiable Instruction Following. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ruoyu Qin, Weiran He, Weixiao Huang, Yangkun Zhang, Yikai Zhao, Bo Pang, Xinran Xu, Yingdi Shan, Yongwei Wu, and Mingxing Zhang. Seer: Online context learning for fast synchronous llm reinforcement learning. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 883–901. USENIX Association, 2026.

Qwen Team. Qwen3 Technical Report. *CoRR*, abs/2505.09388, 2025.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents. `https://qwen.ai/blog?id=qwen3.5`, 2026.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *International Conference on Machine Learning (ICML)*, pp. 18332–18346, 2022.

Shriram Sarvotham, Rudolf H. Riedi, and Richard G. Baraniuk. Connection-level analysis and modeling of network traffic. In *ACM SIGCOMM Internet Measurement Workshop*, pp. 99–103, 2001.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations (ICLR)*, 2017.

Yibo Shen, Xudong Han, Xiaowei Zhu, Gen Li, and Zhenxuan Pan. RoutePack: Expert Placement and Attention-Aware Data Packing for MoE Reinforcement Learning. *arXiv preprint arXiv:2608.12146*, 2026.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A Flexible and Efficient RLHF Framework. In *European Conference on Computer Systems (EuroSys)*, pp. 1279–1297, 2025.

Prajwal Singhania, Siddharth Singh, Lannie Dalton Hough, Akarsh Srivastava, Harshitha Menon, Charles Fredrick Jekel, and Abhinav Bhatele. Understanding and Improving Communication Performance in Multi-node LLM Inference. In *ACM Conference on AI and Agentic Systems (CAIS)*, pp. 1070–1083, 2026.

Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Xiaoxia Wu, Connor Holmes, Zhewei Yao, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. ZeRO++: Extremely Efficient Collective Communication for Large Model Training. In *International Conference on Learning Representations (ICLR)*, 2024a.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts. *CoRR*, abs/2408.15664, 2024b.

Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, Xiaocheng Tang, Yundi Qian, Beibei Zhu, and Rui Hou. LlamaRL: A Distributed Asynchronous Reinforcement Learning Framework for Efficient Large-scale LLM Training. *CoRR*, abs/2505.24034, 2025.

xDAN-datasets. xDAN-datasets/AReaL-boba-Data. https://huggingface.co/datasets/xDAN-datasets/AReaL-boba-Data, 2025.

Jinghan Yao, Quentin Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. Exploiting Inter-Layer Expert Affinity for Accelerating Mixture-of-Experts Model Inference. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 915–925, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Mingshu Zhai, Jiaao He, Zixuan Ma, Zan Zong, Runqing Zhang, and Jidong Zhai. SmartMoE: Efficiently Training Sparsely-Activated Models through Combining Offline and Online Parallelization. In *USENIX Annual Technical Conference (USENIX ATC)*, pp. 961–975, 2023.

Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, Quan Chen, and Xin Liu. COMET: Fine-grained Computation-communication Overlapping for Mixture-of-Experts. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

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, 2025.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, Hongyu Zhou, Yimin Jiang, Yibo Zhu, and Daxin Jiang. StreamRL: Scalable, Heterogeneous, and Elastic RL for LLMs with Disaggregated Stream Generation. *CoRR*, abs/2504.15930, 2025a.

Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, and Xin Jin. Optimizing RLHF Training for Large Language Models with Stage Fusion. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 489–503, 2025b.

Yuming Zhou, Haoyang Li, Sheng Lin, Yanfeng Zhao, Tong Zhao, Xupeng Miao, Jie Jiang, Fangcheng Fu, and Bin Cui. Harnessing Routing Foresight for Micro-step-level MoE load balancing in RL Post-training. *CoRR*, abs/2606.11867, 2026.

A Cost Model Details

This appendix describes the details of the cost model. All coefficients in the cost model are fitted by profiling under the evaluated model, training, and hardware configuration, and each coefficient accounts for the forward pass, the backward pass, and recomputation when it is enabled.

Per-type attention calibration. The attention cost in Eq. 2 uses a single set of coefficients $(\alpha_1, \beta_1, \beta_2)$ for all layers. This suffices when every layer uses the same attention type, as in Qwen3. Qwen3.5 instead interleaves linear-attention and full-attention layers (Qwen Team, 2026). The cost of a linear-attention layer grows linearly with sequence length, whereas a full-attention layer also has a quadratic term, so a single calibration cannot fit both layer types. As a result, we calibrate the two attention types separately. For attention type $a \in \{\mathrm{lin}, \mathrm{full}\}$, the per-GPU cost of one layer takes the same form as Eq. 2:

$$T^a = \sum_s \left(\beta_1^a n_s^2 + \beta_2^a n_s\right) + \alpha_1^a, \tag{5}$$

where the sum ranges over the samples on the GPU. Since linear attention has no quadratic term, we fix $\beta_1^{\mathrm{lin}} = 0$. We denote the fitted coefficients by $\theta^a = (\alpha_1^a, \beta_1^a, \beta_2^a)$.

Shared expert. Qwen3.5 also contains a shared expert that processes every token. The shared expert executes on the GPU hosting the sample, so its cost depends on the sample placement but not on the expert placement. We profile it in the same way using a fixed-plus-linear model, i.e., θ^{shared} has a zero quadratic coefficient, and fold it into the sample-side term $T_{m,l}^{\mathrm{attn}}$.

Per-layer attention coefficients. The evaluated Qwen3.5 configuration has 40 layers, of which 30 use linear attention and 10 use full attention. Since the attention type of each layer is known, the planner directly applies the corresponding calibrated coefficients to each layer. Specifically, a linear-attention layer uses θ^{lin}, while a full-attention layer uses θ^{full}. The shared-expert cost is added to the sample-side computation term for every layer. Thus, Qwen3.5 is evaluated with per-layer attention coefficients rather than a single set of coefficients averaged across layer types.

B KL-Style Pair-Swap Refinement

This appendix details the joint refinement stage of J-MoE, i.e., lines 13–17 of Alg. 1. The refinement runs independently for each microbatch m, so different microbatches are refined in parallel. Let $\mathcal{P}_m = (\mathcal{P}_m^S, \mathcal{P}_m^E)$ denote the sample and expert placements of microbatch m, $\mathcal{S}_m$ the samples in m, and $\mathcal{E}_l$ the experts of layer l. Alg. 2 gives the round-level procedure, and Alg. 3 gives the Kernighan–Lin-style pass (Kernighan & Lin, 1970) shared by Sample-KL and Expert-KL.

Refinement rounds. The refinement starts from the placement produced by the first stage, i.e., the sample placement $\mathcal{P}_m^S$ and the LPT expert placement $\mathcal{P}_m^E$. Each round first copies the current placement into a provisional placement $\widetilde{\mathcal{P}}_m$. It then runs one Sample-KL pass over $\mathcal{S}_m$ with the expert placement fixed, followed by one Expert-KL pass for each layer l with the sample placement fixed. If the round reduces F_m by more than a tolerance ϵ, the provisional placement is committed and the next round begins; otherwise, it is discarded and the refinement stops. This matches the termination condition in Alg. 1, with the tolerance ϵ and a round budget R added. Since each committed round reduces F_m by more than ϵ, the refinement is monotone and always terminates.

KL pass. A KL pass takes a placement, a set $\mathcal{X}$ of movable items, and a swap budget K. The items are the samples $\mathcal{S}_m$ in Sample-KL and the experts $\mathcal{E}_l$ of one layer in Expert-KL. All items are unlocked at the start of the pass. At each step, the pass enumerates the feasible pairs of unlocked items on different GPUs and selects the pair whose swap yields the lowest F_m, even if this value exceeds the current one. It applies the swap to the working placement and locks both items for the rest of the pass, so each item moves at most once per pass. The pass records the best placement seen so far and replaces it only when a swap lowers F_m by more than ϵ. It stops when the budget is exhausted or no feasible pair remains, and returns the recorded placement. The returned placement corresponds to the best prefix of the swap sequence, which may be empty, so a pass never returns a placement worse than its input. The Sample-KL budget $\lfloor |\mathcal{S}_m|/2 \rfloor$ allows every sample to be swapped once, whereas the Expert-KL budget H caps the number of swaps per layer. When two candidates score within ϵ of each other, the tie is broken by the canonical indices of the swapped items, making each pass deterministic.

Algorithm 2 Refinement rounds for microbatch m.

Require: Placement $\mathcal{P}_m$ from the first stage, round budget R, expert-swap budget H, tolerance ϵ
Ensure: Refined placement $\mathcal{P}_m$

for $r = 1, \ldots, R$ **do**
  $\widetilde{\mathcal{P}}_m \leftarrow \mathcal{P}_m$ ▷ Provisional placement
  $\widetilde{\mathcal{P}}_m \leftarrow \text{KLPASS}(\widetilde{\mathcal{P}}_m, \mathcal{S}_m, \lfloor |\mathcal{S}_m|/2 \rfloor)$ ▷ Sample-KL
  for $l = 1, \ldots, L$ **do**
    $\widetilde{\mathcal{P}}_m \leftarrow \text{KLPASS}(\widetilde{\mathcal{P}}_m, \mathcal{E}_l, H)$ ▷ Expert-KL
  if $F_m(\mathcal{P}_m) - F_m(\widetilde{\mathcal{P}}_m) > \epsilon$ **then**
    $\mathcal{P}_m \leftarrow \widetilde{\mathcal{P}}_m$ ▷ Commit the round
  else
    break ▷ Discard the round
return $\mathcal{P}_m$

Algorithm 3 KL pass with best-prefix retention.

Require: Placement $\mathcal{P}$, movable items $\mathcal{X}$, swap budget K
Ensure: Best placement $\mathcal{P}^\star$ along the swap sequence

function $\text{KLPASS}(\mathcal{P}, \mathcal{X}, K)$
  $\widetilde{\mathcal{P}} \leftarrow \mathcal{P}$; $\mathcal{P}^\star \leftarrow \mathcal{P}$; $\mathcal{U} \leftarrow \mathcal{X}$
  for $k = 1, \ldots, K$ **do**
    $\mathcal{C} \leftarrow \text{FEASIBLEPAIRS}(\widetilde{\mathcal{P}}, \mathcal{U})$
    if $\mathcal{C} = \varnothing$ **then**
      break
    $(x_i, x_j) \leftarrow \arg\min_{(x_i,x_j)\in\mathcal{C}} F_m\big(\text{Swap}(\widetilde{\mathcal{P}}, x_i, x_j)\big)$
    $\widetilde{\mathcal{P}} \leftarrow \text{Swap}(\widetilde{\mathcal{P}}, x_i, x_j)$
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{x_i, x_j\}$ ▷ Lock both items
    if $F_m(\widetilde{\mathcal{P}}) < F_m(\mathcal{P}^\star) - \epsilon$ **then**
      $\mathcal{P}^\star \leftarrow \widetilde{\mathcal{P}}$
  return $\mathcal{P}^\star$

Sample-KL and Expert-KL. In Sample-KL, a candidate swaps two samples of $\mathcal{S}_m$ on different GPUs while keeping $\mathcal{P}_m^E$ fixed. The swap preserves the number of samples on each GPU but changes their token counts, so a candidate is feasible only if both GPUs stay within the token capacity C. It changes the attention loads of the two GPUs and the source GPUs of the two samples' routed tokens at every layer, but leaves the aggregate expert demand $r_{m,l,e}$ unchanged. The evaluator therefore updates only these attention loads and traffic counters. In Expert-KL, a candidate swaps two experts of layer l on different GPUs while keeping $\mathcal{P}_m^S$ fixed. The swap preserves the number of expert slots on each GPU and keeps a single copy of every expert. It changes only the expert loads of the two GPUs and the destinations of the tokens routed to the two experts at layer l, which the evaluator updates from the cached demand of each source GPU for each expert. In both cases, the evaluator then recomputes the affected terms of F_m, including the maxima over GPUs, so the score reflects shifts of the bottleneck rather than only the local load change.

C Expert Reconfiguration Overlap Analysis

As described in Sec. 3.3, J-MoE overlaps expert reconfiguration with attention computation: the BF16 parameters of the experts that a GPU will execute are prefetched during forward attention, and their FP32 gradients are returned to the home GPUs during backward attention. This appendix estimates when attention computation is long enough to hide these transfers. We first derive the attention FLOPs and the transfer volume, then compare the resulting times, and finally instantiate the condition for our experimental configuration. The analysis uses the compute rate of hardware and the profiled All-to-All bandwidth for expert transfers; the measured overlap is reported in Sec. 4.

Attention computation. Consider one GPU that processes q sequences of lengths $n_1, \ldots, n_q$ in a microbatch, and let $n_{\text{tot}} = \sum_s n_s$ and $u = \sum_s n_s^2$. Let h be the hidden size, H_Q and

H_{KV} the numbers of query and key/value heads, and d_h the head dimension, with $d_Q = H_Q d_h$ and $d_{KV} = H_{KV} d_h$. Counting a multiply-add as two FLOPs, the query, key, value, and output projections cost $4 n_{\text{tot}} h (d_Q + d_{KV})$. Under causal masking, $QK^\top$ and AV each cover about half of the n_s^2 query–key pairs of every sequence for all H_Q query heads, which costs $2 d_Q u$ in total. Ignoring softmax, normalization, and positional operations, and approximating the backward pass as twice the forward work, the attention times at compute throughput F are

$$T_{\text{attn}}^{\text{fwd}} = \frac{4 n_{\text{tot}} h (d_Q + d_{KV}) + 2 d_Q u}{F}, \qquad T_{\text{attn}}^{\text{bwd}} = 2\, T_{\text{attn}}^{\text{fwd}}. \tag{6}$$

The quadratic term depends on u rather than n_{tot}^2 because sequences do not attend to one another.

Expert transfer. A SwiGLU expert with intermediate size h_{ff} has three projection matrices and thus $3hh_{\text{ff}}$ parameters. Its BF16 parameters occupy $S_{\text{param}} = 6hh_{\text{ff}}$ bytes, and its FP32 gradients occupy $S_{\text{grad}} = 12hh_{\text{ff}} = 2S_{\text{param}}$ bytes. Optimizer states stay on the home GPUs, so only parameters and gradients are transferred. We treat each reconfiguration as an All-to-All over the EP group. In the forward pass, each GPU sends and receives the BF16 parameters of E/N experts, i.e., $(E/N)\, S_{\text{param}}$ bytes; in the backward pass, the FP32 gradients of these experts are returned in the reverse direction, i.e., $(E/N)\, S_{\text{grad}}$ bytes per GPU. Using the same effective All-to-All bandwidth BW_{a2a} for both transfers, defined as the bytes each GPU sends divided by the completion time, the transfer times are

$$T_{\text{param}} = \frac{(E/N)\, S_{\text{param}}}{\text{BW}_{\text{a2a}}}, \qquad T_{\text{grad}} = \frac{(E/N)\, S_{\text{grad}}}{\text{BW}_{\text{a2a}}} = 2\, T_{\text{param}}. \tag{7}$$

Overlap condition. Assume that each transfer starts when its attention window begins and runs concurrently with it. Complete overlap then requires $T_{\text{param}} \le T_{\text{attn}}^{\text{fwd}}$ in the forward pass and $T_{\text{grad}} \le T_{\text{attn}}^{\text{bwd}}$ in the backward pass. Since both the gradient volume and the backward attention work are twice their forward counterparts, the two conditions reduce to the same inequality:

$$\frac{(E/N) \cdot 6hh_{\text{ff}}}{\text{BW}_{\text{a2a}}} \le \frac{4 n_{\text{tot}} h (d_Q + d_{KV}) + 2 d_Q u}{F}. \tag{8}$$

Instantiation. We instantiate the condition for the Qwen3 configuration with 16K responses and EP32 (Tab. 2). Qwen3-30B-A3B (Qwen Team, 2025) has $h = 2048$, $h_{\text{ff}} = 768$, $E = 128$, $H_Q = 32$, $H_{KV} = 4$, and $d_h = 128$, so $d_Q = 4096$ and $d_{KV} = 512$. For attention, we use the H20 BF16 peak throughput $F = 148$ TFLOP/s. With $E/N = 4$, each GPU transfers $(E/N)\, S_{\text{param}} = 36$ MiB of BF16 parameters in the forward pass and 72 MiB of FP32 gradients in the backward pass.

The maximum sequence length is $L = 2{,}048 + 16{,}384 = 18{,}432$ tokens, and each GPU processes 2 sequences per microbatch. Taking the average sequence length to be $L/2$, a GPU holds $n_{\text{tot}} = L$ tokens per microbatch. Since J-MoE places at most three sequences on a GPU, we use the Cauchy–Schwarz lower bound $u \ge n_{\text{tot}}^2/3$, which gives the shortest attention window for this token count. The forward and backward attention windows are then at least 10.97 ms and 21.94 ms. By Eq. 8, complete overlap requires $\text{BW}_{\text{a2a}} \ge 3.44$ GB/s, which is satisfied by the bandwidth in our cluster. The same estimate applies to Qwen3.5, whose linear-attention layers have no quadratic term and cost about 7.05×10^7 FLOPs per token, mostly in projections. With $E/N = 8$ experts per GPU, i.e., 48 MiB of BF16 parameters, and the same n_{tot} and u, the forward attention windows are 13.06 ms for full-attention layers and 8.78 ms for linear-attention layers. Complete overlap therefore requires $\text{BW}_{\text{a2a}} \ge 3.85$ GB/s and 5.73 GB/s, respectively, both well below the nominal link.