Balancing and Beyond: Communication-Centric Optimizations in Expert Parallelism

Jiamin Cao*
Alibaba Cloud
China

Qingxu Li*
Alibaba Cloud
China

Yaozhong Liu*
Alibaba Cloud
China

Jiaqi Gao
Alibaba Cloud
China

Yan Zhang
Alibaba Cloud
China

Shangfeng Shi
Alibaba Cloud
China

Zian Chen
Alibaba Cloud
China

Yizhi Wang
Alibaba Cloud
China

Jun Zhang
Alibaba Cloud
China

Kunling He
Alibaba Cloud
China

Ennan Zhai
Alibaba Cloud
China

Jianbo Dong
Alibaba Cloud
China

Binzhang Fu
Alibaba Cloud
China

Dennis Cai
Alibaba Cloud
China

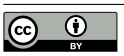
Abstract

The Mixture-of-Experts (MoE) architecture scales large language models (LLMs) to trillions of parameters by activating only a small subset of experts per token. In practice, MoE inference is commonly deployed with Expert Parallelism (EP), which places whole experts on different GPUs to preserve kernel efficiency. However, production EP deployments often suffer from two bottlenecks: (1) *expert workload imbalance*, which creates computation and communication stragglers, and (2) *communication inefficiency*, where inter-GPU transfers dominate latency even after balancing. We present EPIC, an experience-driven EP inference system that addresses these issues progressively for real deployments. EPIC mitigates imbalance via performance-aware expert migration and runtime expert activation, and then improves communication with topology-adaptive transport kernels and fine-grained computation–communication overlap. EPIC has been deployed at scale across O(10K) GPUs in our online inference service for both open-source models (e.g., Qwen3-Coder and DeepSeek-R1) and internal models, reducing communication time and per-token latency by up to 40% and 21%, respectively.

CCS Concepts

• **Networks** → **Data path algorithms**; **Data center networks**; • **Computing methodologies** → **Machine learning**.

*Co-first authors with equal contribution.



SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829201

Keywords

Mixture-of-Experts, Expert Parallelism, Load Balancing, Collective Communication, LLM Inference

ACM Reference Format:
Jiamin Cao, Qingxu Li, Yaozhong Liu, Jiaqi Gao, Yan Zhang, Shangfeng Shi, Zian Chen, Yizhi Wang, Jun Zhang, Kunling He, Ennan Zhai, Jianbo Dong, Binzhang Fu, and Dennis Cai. 2026. Balancing and Beyond: Communication-Centric Optimizations in Expert Parallelism. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3789240.3829201

1 Introduction

Large language models (LLMs) are rapidly scaling from billions to trillions of parameters, bringing strong generalization across a wide range of tasks [3, 15, 20, 35]. In our online inference service, this scale directly translates to higher memory footprint, higher per-token cost, and tighter latency requirements, where users expect interactive responses and operators must meet strict SLOs under fluctuating traffic. To make trillion-parameter inference practical, our production deployment adopts the Mixture-of-Experts (MoE) architecture [22, 30, 40], which activates only a small subset of experts per token and keeps per-token compute manageable. As a result, MoE has become the default choice for serving many state-of-the-art LLMs in our fleet, spanning both open-source models and internal proprietary models.

A key enabler for serving MoE at scale is *Expert Parallelism (EP)* [39, 46], which places entire experts onto different GPUs. Compared to tensor parallelism, EP preserves GEMM/kernel efficiency and avoids fine-grained partitioning overhead, making it attractive for latency-sensitive online inference. However, after operating EP-based MoE inference in production, we found that EP frequently fails to deliver the expected latency/throughput benefits because of two main bottlenecks, *i.e.,* end-to-end decoding latency: **expert**

workload imbalance and **communication inefficiency**. These bottlenecks persist across models and traffic patterns, and become more pronounced as we scale out to more nodes.

Bottleneck #1: expert workload imbalance in production. In EP, tokens are dynamically routed to experts by a gating module, and the resulting token distribution is often highly skewed [18]. In our online traces, a small fraction of experts frequently become "hot" and receive disproportionately more tokens than others. This imbalance creates stragglers in both computation and communication: GPUs hosting hot experts experience longer compute and communication times, while GPUs hosting cold experts remain underutilized but must still wait at synchronization points. Across production traffic, we repeatedly observe large gaps between the busiest expert and the average expert within the same decoding step, which directly increases time-per-output-token (TPOT).

Existing systems like Expert Parallelism Load Balancing (EPLB) [8] mitigate imbalance via periodic reshuffling experts based on historical statistics. In our deployment, however, these approaches reveals practical limitations. First, expert workload regularly deviate from historical averages, and periodic rebalancing is unable to adjust in real-time. Second, they only optimize compute balance between GPUs but ignore the balancing of network traffic. As a result, imbalance remains a dominant source of tail latency in our service.

Bottleneck #2: communication inefficiency after balancing. Existing MoE communication optimizations largely follow two directions. One is to leverage high-bandwidth transports to alleviate bandwidth bottlenecks. For example, DeepEP [7] uses NVLink intra-node and RDMA inter-node (IBGDA [1]). However, a fixed transport policy may not fit all deployment scenarios. In our multi-node clusters, we find DeepEP's design to be inefficient: traffic can be pushed through congested rails/spine links. When multiple top-k experts for a token reside on the same node, identical token payloads are transmitted multiple times over the NIC, wasting bandwidth. The other direction tries to overlap computation and communication, such as two-batch overlap (TBO) [21]. While effective for very large batches, in our online setting where batch sizes are often small to medium, splitting batches reduces compute efficiency, yielding limited or even negative gains.

These production lessons motivate EPIC, an experience-driven system for EP-based MoE inference. A key takeaway from our deployment is that the two bottlenecks are coupled in practice: eliminating imbalance is necessary to avoid stragglers, but once the workload becomes balanced, communication efficiency determines the achievable latency. Therefore, EPIC addresses EP bottlenecks progressively: it first establishes a balanced baseline and then applies communication-centric optimizations to approach the hardware limits.

To mitigate imbalance, EPIC extends EPLB with mechanisms that co-design periodical expert placement and real-time adjustments. At the placement level, EPIC performs topology aware reshuffling to balance both compute and network loads. To react to short-term workload variations observed in production, EPIC supports lightweight, intra-host expert migration using high-bandwidth interconnects (*e.g.,* NVLink), enabling rapid adaptation without violating latency budgets. Together, these mechanisms reduce stragglers and improve utilization under real traffic patterns.

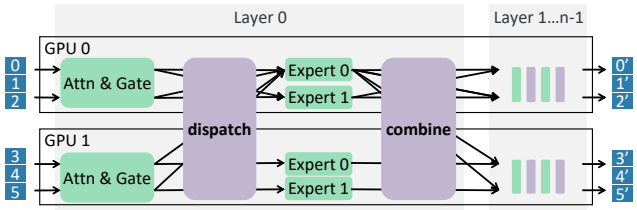


Figure 1: EP Inference workflow of a MoE model.

Building on the balanced workload, EPIC further improves communication efficiency. First, EPIC provides topology-aware transport kernels and de-redundancy mechanisms to eliminate duplicate transfers. Second, EPIC enables per-expert fine-grained computation that hides communication without reducing GEMM efficiency. In reality, EPIC selects the most fitting combination of optimization policies catered to each scenario to achieve the best performance.

EPIC has been deployed at scale in our online inference service, serving both open-source MoE models (*e.g.,* Qwen3-Coder [35] and DeepSeek-R1 [20]) and internal proprietary models, and achieves up to 21% lower TPOT and 27% higher TPS while maintaining service stability. This work contributes: (1) production insights on EP-based MoE inference and two latency-dominant bottlenecks; (2) an imbalance-mitigation framework that integrates topology-aware placement with low-cost migration to balance compute and communication; (3) communication optimizations that are adaptively applied to match workload requirements; and (4) validation at production scale across models, hardware, and configurations. *This work does not raise any ethical issues.*

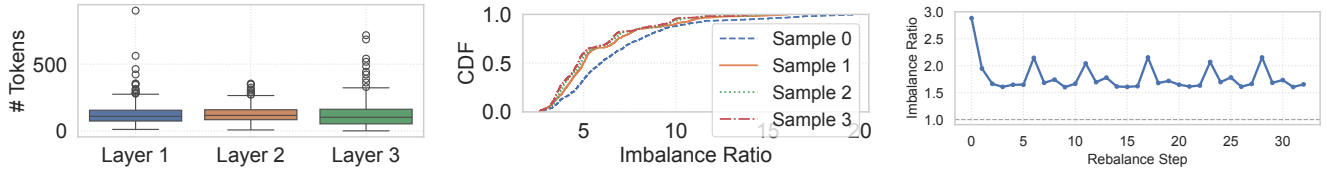
2 Background and Motivation

We operate a large, multi-tenant online model inference service. Over the past few years, we have observed two clear shifts in production: models have moved from dense LLMs to Mixture-of-Experts (MoE), and serving has moved from single-GPU to multi-node EP inference.

From dense to MoE model. Today, almost all of our served LLM capacity (by tokens and GPU-hours) is MoE, including both open-source and in-house ones. As model sizes continue to grow, dense LLMs are expensive to serve under tight SLOs. In contrast, MoE models scale capacity by activating only a small subset of experts per token, keeping per-token compute affordable while enabling trillion-parameter models.

From single-GPU to multi-node EP. As model sizes increase, a single GPU cannot accommodate a whole model. Currently, almost all of our LLM inference traffic runs on multi-GPU, often multi-node, deployments (ranging from 8 GPUs within a node to clusters spanning up to 8 nodes). MoE serving typically relies on EP [39, 46], which places each expert's parameters local to a single GPU (as shown in Figure 1). Compared to TP with the same scale, EP greatly reduces communication volume and improves performance.

As an inference service provider, our objective is to deliver both throughput (TPS) and latency (TPOT) guarantees, while minimizing the cost. Different scenarios pin TPOT and TPS at different targets. For example, agentic workflows may require ultra-low TPOT (*e.g.,* 10-30 ms), while offline generation tolerates looser latency. In general, increasing batch size (*i.e.,* requests per step) raises TPOT (more concurrent work) but improves TPS (better compute efficiency). In production, we select batch size per scenario to satisfy its SLOs.



(a) Expert workload distribution in sampled layers. (b) Imbalance ratio distribution. (c) Imbalance ratio at each EPLB rebalance step.

Figure 2: Expert workload imbalance distribution and temporal variations in DeepSeek-R1 inference deployment.

Config	Real (Imbalanced) Hot GPU	Real (Imbalanced) Cold GPU	Theoretical (Balanced)
Dispatch (μs)	155.7	74.2	86.3
Expert GEMM (μs)	191.0	127.4	147.0
Combine (μs)	231.0	376.1	187
TPOT (ms)	65.2	65.2	54.2

Table 1: Performance comparison between balanced (theoretically-optimal) and imbalanced (real) gating.

Given this batch size, EPIC targets the dominant bottleneck in MoE inference: EP communication. In production, two issues consistently limit performance in EP:

2.1 Expert workload imbalance

Before serving a model, we estimate a *theoretical* inference step time for a given batch size by decomposing one step into constituent kernels, benchmarking them in isolation, and summing their latencies. In production, however, the measured TPOT is consistently higher than these estimates. The dominant cause is expert workload imbalance.

Expert workload imbalance. MoE gating assigns uneven numbers of tokens to experts, creating hot experts that receive disproportionate traffic. GPUs hosting these experts must transfer and process more data, inflating both communication time and expert-side compute. Figure 2(a) shows a sampled decode step across three layers, plotting tokens received per expert. In all three, loads vary widely, with the busiest expert receiving 74 times the layer average. We quantify this skew with the imbalance ratio, defined per layer as the maximum tokens received by any expert divided by the layer mean. This metric captures the bottleneck because per-step latency is gated by the slowest expert. Figure 2(b) reports layer-wise imbalance ratios over 500 steps across four sampled production traces. Even the most balanced intervals remain 2.5, while the most imbalanced one reaches 10.6.

Table 1 (DeepSeek-R1, unoptimized; EP16 on 16×H20) profiles hot and cold GPUs under imbalanced versus balanced routing. The hot GPU incurs longer dispatch time (more incoming tokens) and longer expert compute (more tokens processed), while the cold GPU's kernels complete sooner but stall at the combine barrier waiting for the hot GPU. Compared to a balanced routing baseline, real skew inflates the slowest-GPU dispatch+combine path from 273.3 to 450.3 μs (the cold GPU is dominated by its combine-barrier wait) and hot-GPU expert compute from 147.0 to 191.0 μs, raising TPOT from 54.2 to 65.2 ms.

Existing EPLB cannot fully resolve expert imbalance. State-of-the-art systems adopt EPLB [8], periodically (*e.g.,* every 10 minutes) reshuffling experts across GPUs based on historical workload statistics. By co-locating hot and cold experts, EPLB smooths aggregate load and achieves GPU-level balance.

However, EPLB cannot adapt in real-time when workloads shift between rebalance intervals. Figure 2(c) plots the imbalance ratio measured immediately after each periodic rebalance (one every 200 decode steps) in a DeepSeek-R1 deployment. Despite repeated reshuffling, the ratio remains 1.6–2.9× and never approaches 1 (perfect balance), because the historical statistics used by EPLB quickly become stale within the next 200 steps. This motivates fine-grained mechanisms to mitigate imbalance at per-step timescales. In our experience, combining on-demand, dynamic migration with periodic EPLB yields better balance.

In addition, existing EPLB algorithms optimize for compute balance while overlooking communication, which can induce network bottlenecks in practice. Because EPLB is *topology-blind* to the GPU↔NIC mapping, it can place the two heaviest experts on any pair of GPUs—including the pair that shares one NIC on our H20 hosts.

2.2 Communication is not one-size-fits-all

As in Figure 1, EP's dominant communication patterns are dispatch and combine. Requests are batched and processed in each layer as following: after attention computation, the gate selects *top-k* experts per token, which are then dispatched correspondingly. Then, expert computation executes locally on the hosting GPUs, and results are combined back on the originating GPUs. This repeats across layers and steps.

From NCCL to DeepEP. Conventional CCLs (*e.g.,* NCCL) may incur large overhead when directly used for EP. First, they require collect sending and receiving metrics so ranks learn who to receive from. Second, they encode communication shapes as kernel parameters that hinder CUDA Graph capture, which is critical for reducing CPU launch overhead in online inference. Third, their kernels occupy SMs until completion, limiting compute–communication overlap.

DeepEP [7] addresses these constraints with specifically designed dispatch/combine kernels. As shown in Figure 3, it pre-allocates sufficiently large send/receive buffers so all tokens can be transmitted concurrently, minimizing latency. For each token's top-k destinations, transport is locality-aware: same-GPU copies stay local, same-host traffic uses NVLink load/store, and inter-node transfers use GPU-driven RDMA (IBGDA [1]). With IBGDA, network I/O is offloaded to NICs. The GPU only rings doorbells and can release SMs for overlapped computation immediately after issuing all sends. Building on this asynchronous path, DeepEP provides a two-batch overlap (TBO) mode (Figure 4) that splits a batch into two halves, computing on one while communicating the other. These make DeepEP the de facto CCL for MoE inference.

Limitations of DeepEP. Despite its strengths, DeepEP's default strategies are not universally optimal.

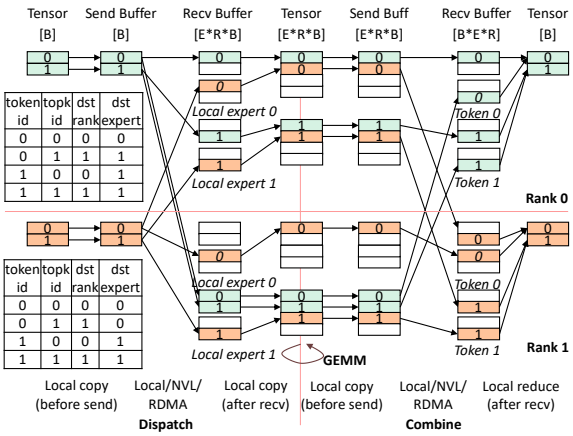


Figure 3: Dispatch and combine patterns in DeepEP (B = batch size, E = # local experts per rank, R = # GPUs).

Batch Size	Attention	Expert GEMM	Dispatch	Combine
8	134	68	50	51
16	139	69	88	64
32	157	70	88	80
64	190	86	70	110
128	256	117	166	203

Table 2: Kernel time (μs) under different batch sizes for a Qwen3++ model with EP16 on H20 GPUs.

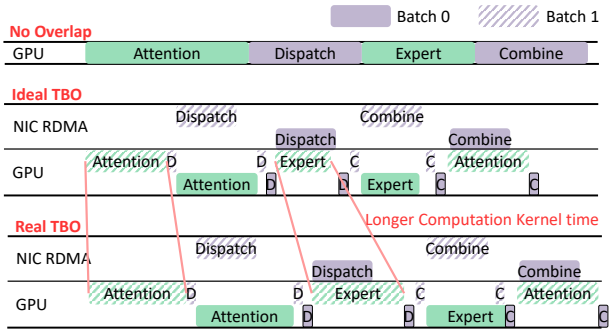


Figure 4: Two batch overlap in DeepEP compromises computation kernel efficiency.

(1) Redundant inter-host transmissions and fabric hot spots. DeepEP sends each token concurrently to all top-k destination experts, even when multiple experts reside on the same host or GPU.[1] This maximizes concurrency but can waste bandwidth. That choice fits DeepEP's original environment, *i.e.,* large EP scale across many hosts, where destinations are typically spread across hosts and redundancy is limited. They also use H800 clusters with high inter-host bandwidth (*e.g.,* 8×400 Gbps per host). Our production fleets differ: deployments scale span 1–8 nodes under different bandwidth constraints. For example, our H20 clusters have fewer NICs thus lower inter-host bandwidth (4×400 Gbps per host). As a result, the inter-host network becomes the bottleneck.

(2) Overlap efficacy at small batches. TBO is effective only when the batch is large enough. At small or medium batches, splitting the batch hurts performance: as shown in Table 2, compute kernel latencies change little at small batch sizes because expert kernels are predominantly memory-bandwidth bound (weight loads dominate).

[1]We specifically refer to DeepEP's low-latency kernel designed for decode.

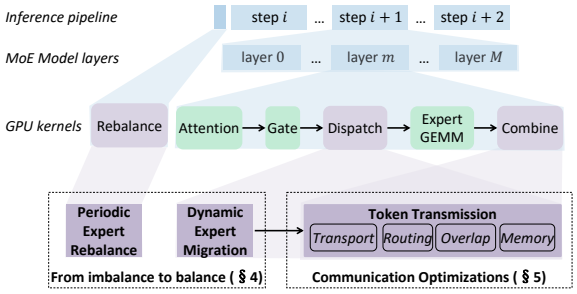


Figure 5: EPIC's system architecture overview

For example, splitting a batch of 16 into two 8s nearly doubles total compute time (Expert GEMM doubles from 69μs to 2×68μs, and attention doubles from 139μs to 2×134μs), adding approximately 196μs of computation and 50μs of additional communication, which is larger than the theoretical overlap ceiling (*i.e.,* total communication time, 202μs), so TBO yields a slowdown. In our production environment, TBO hardly provide benefits since the batch is usually small (*e.g.,* ≤128) under tight SLOs. For smaller batches, we need overlap strategies that preserve kernel efficiency and avoid extra weight loads.

(3) Large memory footprint. DeepEP achieves low latency by preallocating generous send/receive buffers so dispatch and combine can proceed without blocking, even under worst-case all-to-all. The footprint scales with model shape (hidden size H), total expert count X = E×R (E experts per GPU, R GPUs), and per-GPU batch size B. Concretely, DeepEP provisions approximately B×H and B×H×X elements for dispatch send/receive, and B×H×X elements for both send and receive during combine. A DeepSeek-R1 model with batch size 128 requires roughly 1.1 GB of extra memory per GPU. On 80-GB GPUs, model weights already consume most of the memory, leaving limited headroom for the KV cache. Long-context workloads exacerbate this constraint: reduced KV-cache capacity forces smaller batches, which in turn lowers throughput and increases TPOT. Hence, we need to minimize buffer footprint while preserving performance.

(4) Prone to failures. EP communication is globally synchronized: in dispatch and combine, each GPU exchanges data with everyone; if any single GPU fails, this round of communication fails. Current DeepEP implementations do not handle such anomalies and raise exceptions directly, corrupting the entire inference pipeline. A large EP region binds more GPUs into a synchronized communication group, which makes the entire inference system more prone to failures. In production, failures are unavoidable. Therefore, it's necessary for the CCL to detect, isolate, and resolve failures.

3 Design Overview

EPIC tackles two core bottlenecks in MoE inference: workload imbalance and communication inefficiency. The design follows a progressive principle: *first establish balance, then optimize communication.* Imbalance creates stragglers that dominate latency regardless of communication efficiency; therefore, EPIC first eliminates skew to expose the true communication bottleneck, and then applies targeted communication optimizations to reduce the remaining latency.

For workload imbalance, complementing periodic EPLB, EPIC rebalances experts and adjusts replica activation in real time based on

the layer's top-k routing pattern. For communication inefficiency, EPIC propose multiple transport strategies and a finer grained overlapping strategy as an alternative to DeepEP's fixed strategies and applies them based on the workload. Additionally, EPIC implements memory optimization techniques and fault-tolerant mechanisms for EP communication that are critical to large-scale operational deployment. We detail the real-time rebalance mechanism in §4 and the communication optimizations in §5.

4 From Imbalance to Balance

This section introduces how EPIC solves the imbalance problem by a combination of coarse-grained periodic load-balancing and fine-grained real-time expert rebalancing.

4.1 Problem formulation

Consider an MoE model with E experts served across G GPUs. Each GPU provisions S slots to host expert weights, one expert per slot. When $S * G > E$, some experts are mapped to multiple slots.

Given the current step's token activations we aim to map E experts onto $N * G$ GPU slots to minimize the maximum per-GPU workload, including the communication (dispatch + combine) and computation (GEMM) time. Suboptimal placements co-locate hot experts on one GPU, creating a compute hotspot (Figure 6(a)). Even when compute is balanced, placing hot GPUs behind the same NIC creates a network hotspot (Figure 6(b)). Effective placements (Figure 6(c)) jointly balance compute and communication by spreading hot experts across GPUs and NICs.

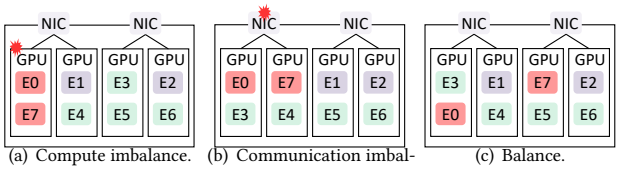
Minimizing the makespan across resources, *i.e.,* the maximum of per-GPU compute load and per-NIC inter-host traffic, yields a generalized assignment problem with cardinality and shared-resource constraints, which is NP-hard. Additionally, the migration overhead should also be considered. Thus, computing and applying a global optimum at each step is infeasible for low-latency inference; We therefore decompose the rebalancing into a periodic global shuffle based on historical statistics, and an instant node-local adjustment based on real-time routing results. Periodic shuffling is triggered based on real-time accumulated statistics and executes *between* decoding steps. Real-time migration, in contrast, operates *within* the current step, performing intra-host expert swaps on the fly. EPLB completes its placement update before the next step begins.

4.2 Periodic EPLB algorithm

The periodic EPLB algorithm collects historical statistics and shuffles experts across the EP group to minimize imbalanced workload. EPIC considers both communication and compute, and uses a hierarchical bin-packing algorithm to solve this.

In our production deployment, one inference job runs on homogeneous GPUs with identical compute capacity. Leveraging this symmetry, EPIC decouples compute and communication objectives, by first balancing per-GPU compute by distributing workloads evenly across GPUs, and then mapping GPUs to server/NIC positions to balance communication:

- *Collect workload statistics.* We collect a load vector $c = [c_0, c_1, \ldots, c_{|E|-1}]$, where c_i is the total number of tokens processed by expert i in the past window.



(a) Compute imbalance. (b) Communication imbalance. (c) Balance.

Figure 6: Expert placement examples. $E = 7$ experts across $G = 4$ GPUs. Each GPU provisions $S = 2$ slots.

- *Decide expert redundancy.* Atop c, we use greedy search to compute a redundancy vector $r = [r_0, r_1, \ldots, r_{|E|-1}]$, where r_i denotes the number of redundant slots allocated to expert i, to minimize the maximum effective load $\max_i \frac{c_i}{1+r_i}$.
- *Assigning experts to GPUs.* Let N_g denote the slots hosted by GPU g. The objective is to minimize the maximum GPU load $\max_{g \in G} \sum_{n \in N_g} c_{f(n)}$, where $c_{f(n)}$ is the load of the expert assigned to slot n. We use a greedy strategy: experts are sorted by their computation loads, and each expert is assigned to the GPU with the lowest current total load.
- *Assigning GPUs to server.* For the case where multiple GPUs share the same NIC, we need to minimize the maximum communication time across NICs. We adopt a greedy placement strategy: at each step, the system places a GPU onto the machine position that leads to the smallest increase in the global maximum per-NIC communication volume. By scattering hot experts across GPUs attached to different NICs, this step prevents communication hotspots and balances bandwidth utilization.

EPIC's EPLB provides a baseline upon which lightweight dynamic migration can further adapt to short-term workload fluctuations.

4.3 Real-time expert migration

EPIC 's real-time migration targets short-term, rapidly fluctuating workloads. Per-step latency budgets make low migration overhead and fast decision-making the primary design constraint. For a typical expert weight size (e.g. ≈ 42 MB for DeepSeek-R1), on H20/H800, NVLink at 450/200 GB/s moves one expert in 92/210 µs, whereas RDMA at 200/400 Gbps needs 1.7/0.8 ms. Given O(10)ms-level TPOT budgets and thus O(100)μs-level per-layer budgets, we restrict real-time rebalance to intra-host transfers.

EPIC performs rebalance immediately after gating and fuses it into a single kernel with dispatch communication. The kernel first runs a short *rebalance* phase to apply expert swaps, then enters the *dispatch* phase with the updated placement. To enable fast decisions, EPIC utilize a simple greedy algorithm to calculate the rebalancing plan:

(1) Per-GPU workload statistics. Each GPU counts per-GPU activations for its local input tokens, *i.e.,* the number of tokens activating experts resident on each GPU, and then performs AllReduce-Sum to obtain global per-GPU loads $\{L_g\}$. Critically, this AllReduce and the subsequent plan generation are overlapped with the mandatory BF16→FP8 quantization that precedes every dispatch.

(2) Plan generation. On each host, GPUs are sorted by L_g and greedily paired(heaviest with lightest, second heaviest with second lightest, etc.). For each pair (g_h, g_l), we consider swapping one expert from g_h with one from g_l. We try to select the swap

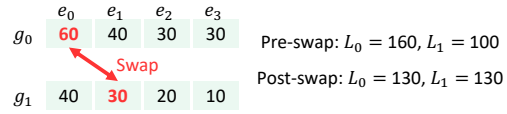


Figure 7: Dynamic rebalance example. Swapping e_0 on g_0 with e_1 on g_1 reduces max. workload from 160 to 130.

that reduces the maximum workload the most. Figure 7 shows an example, by swapping e_0 on g_0 with e_1 on g_1, the maximum workload decreases from 160 to 130. We accept this swap only if it lowers the pair's max load by at least a threshold τ (in tokens), where $\tau = t_{swap}/t_{tok}$, $t_{swap} = w/v_{comm}$, and t_{tok} is the profiled per-token compute time, w is the expert weight size, and v_{comm} is the communication bandwidth. This ensures the expected reduction in compute time exceeds the swap cost.

(3) In-kernel swap. The fused kernel applies the planned swaps: it performs device-to-device copies to exchange the selected experts between slots on (g_h, g_l) and updates the device-resident placement map.

(4) Token dispatch. After rebalancing, the kernel proceeds to the *dispatch* phase and routes tokens using the updated mapping, so benefits materialize within the same launch.

This design delivers fast responsiveness to fluctuation with negligible overhead.

Deployment effect. In production across EP8–EP64 deployments of open-source models such as DeepSeek-R1 and Qwen-Coder, our imbalance-oriented mechanisms reduce the imbalance ratio by about 40% versus naive EPLB and cut TPOT by 10–18%. However, once workload is well balanced, the bottleneck shifts to dispatch/combine: communication still accounts for 12–22% of end-to-end TPOT, and several default design choices in DeepEP prove suboptimal for our workloads and topology. These observations motivate the communication-centric optimizations presented next to further reduce end-to-end latency.

5 Communication Optimizations

DeepEP's direct all-to-all approach to communication and the TBO strategy fits its original environment with large EP scale and high inter-host bandwidth clusters. However, they present issues in our deployments with smaller scale EP and higher NVLink bandwidth as introduced in §2.2.

From a general optimization perspective, we should: (1) Reduce transmitted bytes and prefer high-bandwidth paths. (2) Balance link utilization to avoid NIC and cross-rail hotspots. (3) Preserve compute efficiency under overlap by minimizing SM occupancy for communication. In practice, the importance of these factors varies across workloads and topologies. On top of DeepEP's original techniques, we need a broader portfolio of strategies and the ability to switch among them at runtime to optimize EP in our environment.

In this section, we introduce EPIC's communication optimizations: topology-adaptive routing (§5.1) and per-expert overlap (§5.2), which together reshape how dispatch/combine traffic is routed and scheduled. Two further mechanisms that are essential for production deployment but *orthogonal* to the communication algorithms themselves—reducing the EP memory footprint and tolerating in-instance GPU failures—are deferred to §6 and §7 respectively.

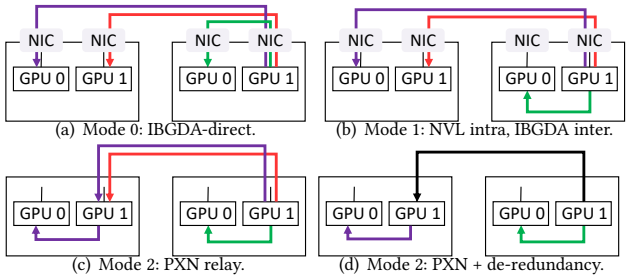


Figure 8: Communication modes in EPIC: (a) Mode 0: IBGDA-direct (SM-free intra/inter-node); (b) Mode 1: NVLink intra-node with IBGDA inter-node; (c)–(d) Mode 2: NVLink intra-node with PXN relay inter-node, without and with de-redundancy that eliminates duplicate inter-host copies.

5.1 Communication techniques and trade-offs

We implement four communication modes (Figure 8) that embody different points along different trade-offs:

- Mode=0: SM-free intra-node and inter-node (IBGDA-direct). Use IBGDA for both intra- and inter-node transfers to avoid SM contention. Favorable when overlap is critical and communication should not perturb GEMM kernels.
- Mode=1: NVLink intra-node, IBGDA inter-node (direct). Exploit NVLink for higher bandwidth, while keeping inter-node on IBGDA. Improves communication efficiency but reduces overlap headroom due to NVLink kernel SM usage.
- Mode=2: NVLink intra-node with PXN relay inter-node (topology-aware). Alleviate cross-rail/NIC hotspots via relaying. For dispatch, first send inter-node to the same-index GPU on the remote host, then forward intra-node via NVLink to the final destination (inter→intra). For combine, reverse the order (intra→inter): locally gather to one GPU, then send inter-node. This spreads load across NICs and avoids many-to-1 or 1-to-many congestion in imbalanced workload with hot GPUs. In our multi-rail clusters, it eliminates cross-rail traffic when index-aligned GPUs share a rail—traffic that would otherwise traverse the spine switch. Based on PXN, we further eliminate duplicate inter-node transmissions. For dispatch, if multiple target experts live on the same host, send a single copy inter-node and multicast intra-node. For combine, we apply the symmetric reverse (aggregate intra-node and issue one inter-node send). This directly cuts inter-node volume.

5.2 Per-expert overlap

Two-batch overlap (TBO) splits a batch into two sub-batches and overlaps their workflows, but fragments kernels and effectively doubles matrix weight loads, making it viable only when batches are large enough. For medium and small batches, we need finer-grained overlap that hides communication latency without compromising compute efficiency.

Key insight: overlap along the expert dimension, not the batch dimension. Prior overlap approaches (TBO) partition the *batch* dimension. GroupedGEMM execution time comprises two parts: loading each expert's weight matrix from global memory and the arithmetic on the loaded tiles. Halving the batch halves the arithmetic per expert but still requires loading *all* expert weights in each sub-batch, so weights are loaded twice overall—doubling the

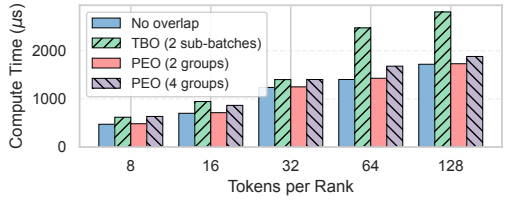
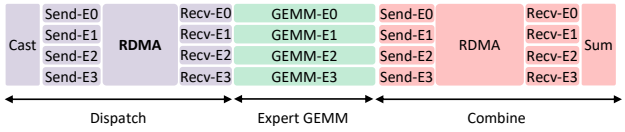
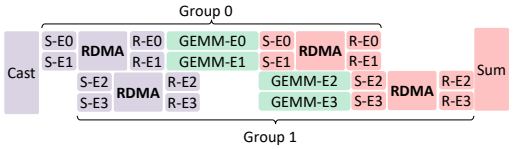


Figure 9: GroupedGEMM time under equivalent total workload (DeepSeek-R1, H20, 8 experts/GPU).



(a) DeepEP communication workflow.



(b) Per-expert overlap workflow.

Figure 10: DeepEP vs Per-expert overlap of EPIC.

memory-bandwidth cost and pushing the kernel into the bandwidth-bound regime. We instead partition the *expert* dimension: it groups experts and pipelines dispatch/compute/combine across groups, but each group's GEMM operates on the *full batch* of tokens destined for those experts. Each group loads only its subset of expert weights, while per-expert token count (and thus computation workload) stays unchanged.

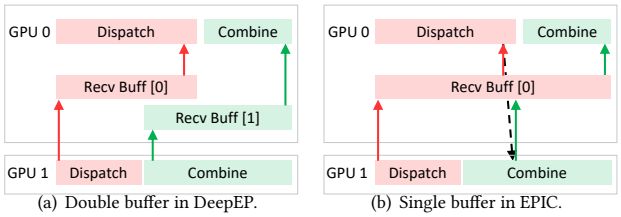
Figure 9 validates this with a GroupedGEMM microbenchmark: Splitting a batch into two sub-batches incurs 30–99% higher total compute time than the non-split baseline due to redundant weight loads, whereas splitting with 2 or 4 expert groups adds only 1–8% overhead at moderate-to-large token counts.

Insight: expert workloads are naturally independent. Figure 10(a) illustrates DeepEP's bulk-synchronous design. It enforces phase barriers, *i.e.,* GEMMs wait for full dispatch and combine waits for all local experts before starting, creating head-of-line blocking.

Critically, experts have no inter-dependencies after gating: each expert processes its own tokens independently. A local expert can launch GEMM as soon as its inputs for that expert arrive via dispatch and, upon completion, immediately return outputs to source GPUs during combine.

Our design: group-wise pipelining of experts. Leveraging this independence, EPIC adapts the classic pipeline principle (PipeDream, GPipe)—which distributes weights across GPUs and micro-batches the input—to a *single-device, expert-dimension* setting: it partitions each GPU's local experts into multiple groups and pipelines dispatch, compute, and combine across groups, overlapping communication of one group with computation of another without fragmenting per-group GEMM size. Figure 10(b) shows an example with two groups: E0, E1 and E2, E3.

Within each group, stages run in a fixed order and the group's experts are processed together to preserve kernel size and avoid fragmentation: (1) The GPU first issues dispatch sends for the current group only (*e.g.,* the first two experts across all GPUs). (2) Once



(a) Double buffer in DeepEP. (b) Single buffer in EPIC.

Figure 11: EPIC reduces buffer with signal notification.

the required tokens for the group's experts have arrived, GEMMs for all experts in the group are launched. (3) As soon as those GEMMs finish, outputs from the group's experts are returned to source GPUs without waiting for other groups.

Across groups, we stagger these stages to form a pipeline, allowing different groups to occupy different phases concurrently while enforcing minimal dependencies to prevent NIC and SM contention. (1) Dispatch staging: the next group begins dispatch only after the current group's dispatch completes, ensuring early groups can receive tokens without interference from later groups. (2) Compute staging: the next group's GEMMs are scheduled after the current group's GEMMs completion, eliminating possible SM contention between GEMMs of different groups.

Choosing the number of groups. The number of groups m presents a trade-off between kernel efficiency and overlap ratio. We select m empirically based on the workload and profiling results and find $m = 2$ or $m = 4$ typically balances overlap and kernel efficiency well.

PEO is fully compatible with the communication optimizations in §5.1: each group's dispatch/combine can use PXN relay just like a full all-to-all, and de-redundancy still applies across groups—if different tokens are destined for different experts on the same remote GPU (regardless of whether those experts belong to the same group), only one inter-host transfer is needed.

6 Reducing EP Memory Footprint

As discussed in §2.2, DeepEP's communication buffers incur large memory overhead (hundreds of MB to GB). This becomes prohibitive in scenarios with long contexts (large KVCache) or offline generation with large batches. We introduce two optimizations that substantially reduce footprint with negligible performance impact.

(1) Single-buffer reuse instead of double buffering. DeepEP allocates two receive buffers so each sender can start the next communication even if the receiver has not finished reading the previous payload. This avoids read–write races across GPUs but doubles memory usage. We replace it with a lightweight synchronization: for each sender–receiver pair, the receiver flips a device-visible "consumed" flag after finishing reads. The sender begins writing the next epoch only after observing this flag. In practice, this halves comm buffer memory with only a minor performance impact ($< 10\mu s$).

(2) Reducing the size for combine buffers. DeepEP provisions combine receive space as batch_size × num_experts to simplify address calculation, but each rank only receives at max batch_size × top-k tokens. Thus, we resize combine buffers to batch_size × top-k. Then, during dispatch we piggyback a compact top-k id (*e.g.,* 1 byte when k≤256) alongside the token header. During combine, the receiver writes results directly into the k-sized slot for that token. This eliminates over-provisioning and reduces memory by a

factor of roughly E/K (*e.g.,* 32× at E=256, K=8). The extra metadata is tiny relative to payloads and has negligible runtime overhead.

Policy selection. The optimal policy depends on the scenario (*e.g.,* model, hardware, and SLO). For a new deployment, EPIC performs a brief sweep over candidate combinations and constructs a per-batch-size policy table that selects the one minimizing TPOT under SLO constraints. At runtime, EPIC picks the policy matching the current batch size.

7 Failure Handling

Availability is critical to online serving: a single GPU crash that brings down the entire inference instance severely violates SLOs. While DeepEP cannot progress if any GPU in the EP group fails, EPIC provides a non-disruptive failure-handling mechanism that masks out the failed GPU and continues serving within the same decode step.

In-band failure detection. EPIC uses *in-band* detection rather than slow out-of-band hardware monitors. During each communication round, every GPU maintains a per-peer timer; if the expected data does not arrive before the timeout, the peer is marked unreachable in a device-resident *bitmask*. Once a failure is detected, subsequent dispatch/combine operations in the same step bypass the failed rank via the updated bitmask, preventing cascading stalls. The mechanism is entirely GPU-local and CUDA-graph compatible: timeouts and bitmask updates execute inside the captured graph, and the CPU inspects the bitmask only after graph completion to trigger recovery.

Expert-weight recovery. After the failed rank is isolated, EPIC recovers the lost experts through a tiered strategy: (1) If redundant copies of the lost experts exist on surviving GPUs, no reconstruction is needed—tokens are simply routed to surviving replicas. (2) Otherwise, lost expert weights are rebuilt from a distributed CPU-side weight cache—which is pre-loaded with all expert parameters at instance startup. EPIC exploits the sequential execution of transformer layers to pipeline weight recovery with ongoing computation: while the model computes layer ℓ, the expert weights of layer ℓ+1 are restored in the background; a layer blocks only if its recovery has not yet completed. This layer-wise overlap hides most recovery latency behind normal inference computation.

8 Deployment

Our system is a CCL built on DeepEP [7], exposing `dispatch` and `combine` APIs for real-time EP communication. Production inference engines based on SGLang [12] and vLLM [13] call these interfaces directly. Before 2025, our EP stack used NCCL [6] all-to-all; since early 2025, we have migrated to DeepEP and have since extended, hardened, and continuously optimized it for our workloads.

We deploy this stack for both real-time inference (*e.g.,* chat and agentic workflows) and offline generation (*e.g.,* reinforcement-learning pipelines), serving hundreds of thousands of customers across external enterprise tenants, consumer products, and internal teams running open-source MoE models (*e.g.,* DeepSeek-R1, Qwen3-Coder) as well as in-house models. Our inference fleet spans principal clusters across NVIDIA, AMD, and in-house GPU families, interconnected by scale-out fabrics ranging from 4×400,Gbps to 8×800,Gbps per host, with autoscaling for diurnal and bursty demand. Over the past year, our CCL has supported inference jobs across O(10K) GPUs; all online-serving scenarios run on our CCL, selecting optimizations per SLO tier.

SLO tiers shape deployment choices, including host provisioning, batch sizing, transport selection, and overlap strategies. Interactive/agentic workloads target 10–30,ms TPOT with batch sizes of 8–32, mid-latency tiers typically use 32–128, and offline tiers prioritize TPS with larger batches. KV-cache footprints span short contexts (tens of tokens) to long contexts (tens of millions), constraining GPU memory headroom and influencing communication buffer sizing.

For each model family and GPU type, we run configuration sweeps to tune parameters before rollout. On lower inter-host bandwidth clusters (*e.g.,* H20), de-redundancy and relay forwarding reduce peak NIC load and improve communication time and TPOT by up to 40% and 21%, respectively. In small/medium batches, fine-grained per-expert overlap avoids the inefficiency of two-batch overlap and improves TPOT by up to 17.6%. Overall, these techniques deliver up to 27% higher TPS in production.

We have open-source part of EPIC code (including per-expert overlap and failure-handling) at *(URL omitted for blind review).*

9 Evaluation

In this section, we first evaluate the end-to-end performance of EPIC across diverse models, hardware, scales, and batch sizes(§9.2). We then quantify the impact of periodic and dynamic expert rebalance (§9.3), followed by an assessment of the communication-centric optimizations (§9.4).

9.1 Evaluation setup

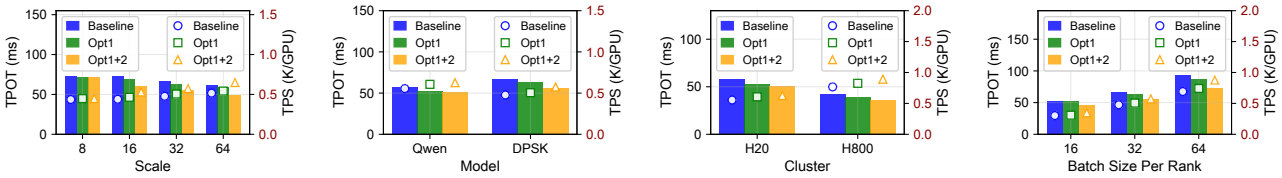
Testbed. We report results from our NVIDIA H20 and H800 clusters. Each server hosts eight GPUs behind NVSwitch. H20 provides 450 GB/s per-GPU NVLink bandwidth and four 400 Gb/s NICs per server, while H800 provides 200 GB/s per-GPU NVLink and eight 400 Gb/s NICs per server.

Models and workload. We evaluate EPIC on two open-source MoE models, Qwen-3-Coder (480B parameters) and DeepSeek-R1 (685B parameters). The model configuration details are listed in Table 3. Across all experiments, weights and activations use FP8, while the KV cache uses BF16. The workload is drawn from our production traffic. The median input and output lengths are 2,396 and 512 tokens, respectively. We employ data parallelism for attention module and expert parallelism for expert module.

Baselines. We benchmark EPIC against the state-of-the-art EP CCL (DeepEP [7]), the expert rebalance scheme EPLB [8], and the widely used TBO scheme [10]. For the DeepEP baseline, we configure it with the default transport mode (NVLink intra-node + IBGDA inter-node, *i.e.,* Mode 1) and TBO overlap where applicable. PXN-like forwarding and de-redundancy are not available in the baseline. For end-to-end comparisons, EPIC and these baselines are integrated into vLLM with identical model, kernel, and configurations.

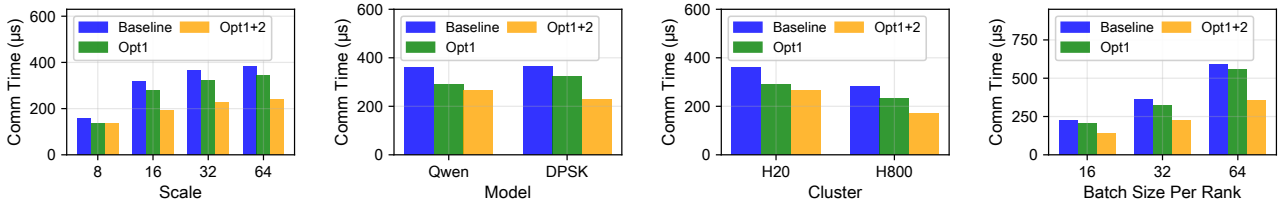
Model	#Layers	#Experts	top-k	Hidden Size	MoE intermediate Size
DeepSeek-R1	61	256	8	7168	2048
Qwen3-Coder	62	160	8	6144	2560

Table 3: Model parameters.



(a) DeepSeek, H20, BS32 (b) EP32, H20, BS32 (c) Qwen, EP32, BS32 (d) DeepSeek, EP32, H20

Figure 12: End-to-end improvement.



(a) DeepSeek, H20, BS32 (b) EP32, H20, BS32 (c) Qwen, EP32, BS32 (d) DeepSeek, EP32, H20

Figure 13: Communication time improvement.

Metrics. For end-to-end evaluation, we report decoding TPOT (latency) and TPS (throughput). We also report communication time to attribute end-to-end gains, defined as the sum of dispatch and combine per step, and its kernel component (send+recv) to reflect GPU-side occupancy and overlap headroom. Unless noted, results are collected in steady-state decode (excluding prefill), averaged over multiple steps.

9.2 End-to-end effect

End-to-end improvement. We report end-to-end TPOT/TPS and per-step communication across model, scale, cluster, and batch size (Figures 12, 13). Baseline is the unoptimized system (DeepEP+EPLB). Opt1 is our system with imbalance optimizations in §4 including both communication-aware EPLB and dynamic expert migration, and Opt1+2 add our communication optimizations in §5.

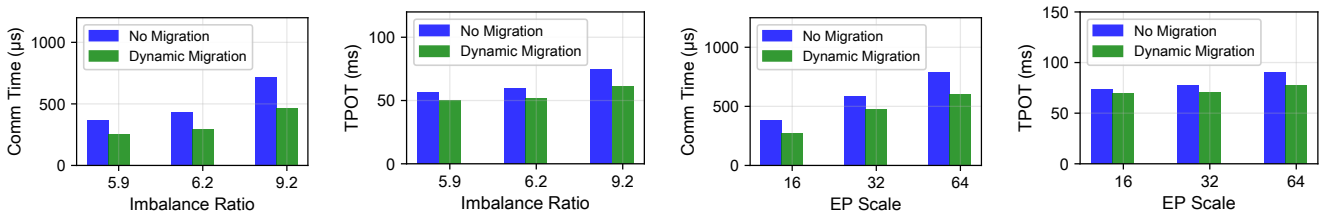
Figure 12 summarizes end-to-end latency (TPOT) and throughput (TPS) across batch size, model, and cluster settings. By scale (Figure 12(a), DeepSeek/BS32/H20), we can see the TPOT decreases as the scale grows up, because the expert computation is more spread out. At 8 H20 GPUs, Opt1+2 has no benefit over Opt1 because all communication goes via NVLink, but the gain starts to show immediately when we deploy the model to 2 servers with 16 GPUs. By model (Figure 12(b), EP32/H20/BS32), comparing the gain of Opt1, Qwen achieves larger end-to-end gains than DeepSeek because fewer experts concentrate tokens and create stronger skew. However, we see more TPOT gains on DeepSeek when we apply more communication optimizations; again, it's because DeepSeek has more experts and the traffic is more spread out. By cluster (Figure 12(c), Qwen/EP32/BS32), H20 benefits more than H800 because inter-host bandwidth is the bottleneck on H20; reducing NIC hotspots translates more directly to end-to-end latency reduction and TPS increase. On H800 the improvements are present but smaller. By batch size (Figure 12(d), DeepSeek/EP32/H20), TPOT decreases and TPS increases as our communication-aware optimizations are applied, with gains growing at larger batches where per-step traffic is higher. This matches the communication trends and the overlap results: lightweight kernels (Mode 0/1) benefit more from overlap at high batch size *BS*, while PXN+de-redundancy (Mode 2) yields the lowest raw communication without overlap.

Communication time improvement. Figure 13 shows the corresponding communication time reductions, which explain the end-to-end trends. By scale (Figure 13(a)), we see a 13% performance gain when we apply expert balancing via Opt1. For multi-server settings, we receive up to 40% gains. By model (Figure 13(b)), Qwen exhibits larger communication reductions than DeepSeek under the same EP/cluster because stronger skew increases many-to-one and one-to-many contention; periodic EPLB alleviates these hotspots. But DeepSeek gains more communication reductions when we apply Opt2 on top because of the higher expert count. By cluster (Figure 13(c)), on H800, the baseline is less bandwidth-limited, so the relative improvement is smaller. By batch size (Figure 13(d)), per-step communication grows with batch size, and our optimizations cut a larger absolute share accordingly. PXN+de-redundancy (Mode 2) achieves the lowest communication without overlap; with overlap, modes with shorter kernels (Mode 0/1) hide more communication behind compute at a large batch size.

9.3 Imbalance optimization

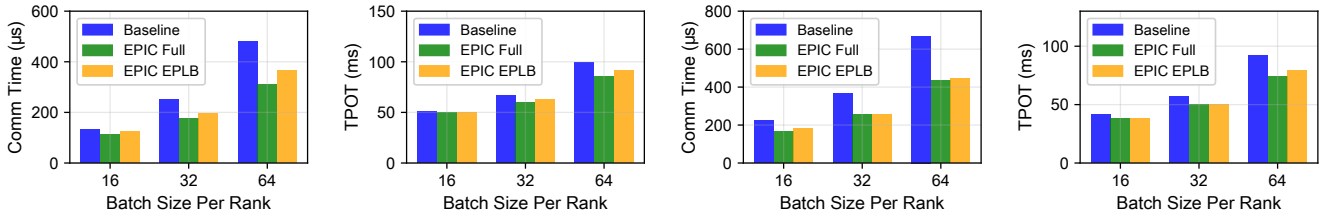
Periodic EPLB. We evaluate our communication-aware periodic EPLB against two baselines: random placement and a compute-only EPLB [8]. As shown in Figure 16, our method reduces dispatch/combine times by 28.2–31.9% vs. random and 5.0–11.8% vs. compute-only EPLB, and shortens end-to-end decode-step latency by 13.8–16.9% and 4.8–6.3%, respectively. Gains grow with batch size because per-step traffic scales with B and top-k. Compute-only EPLB leaves NIC hotspots that become the bottleneck under higher load, while our placement spreads traffic across NICs/hosts.

Dynamic expert migration. Figure 14(a)(b) shows the effect of dynamic expert migration under different imbalance ratios we extracted from production traces. As the imbalance ratio increases, total communication without migration grows from 367μs to 432.6μs (+17.9%) and 714.1μ (+65.1%), while with dynamic migration it rises more gently from 254.3μs to 292.4μ (+15.0%) and 467.6μs (+59.9%).



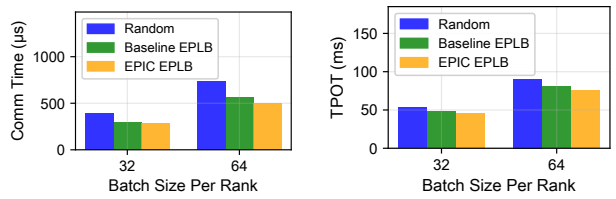
(a) Comm time (ratio) (b) TPOT (ratio) (c) Comm time (EP scale) (d) TPOT (EP scale)

Figure 14: Effect of dynamic expert migration under different imbalance ratios and EP scales [Qwen, H20, BS32].



(a) DeepSeek, Comm time (b) DeepSeek, TPOT (c) Qwen, Comm time (d) Qwen, TPOT

Figure 15: Effect of improved EPLB algorithm combined with dynamic expert migration under different batch sizes [EP32, H20].



(a) Comm time (b) TPOT

Figure 16: Effect of improved EPLB algorithm under different batch sizes [EP32, Qwen, H20].

The resulting reduction versus no migration strengthens from 30.7% to 32.4% and 34.5%. TPOT shows the same pattern. Overall, higher imbalance severely degrades performance, whereas dynamic migration shortens TPOT by 12–18%.

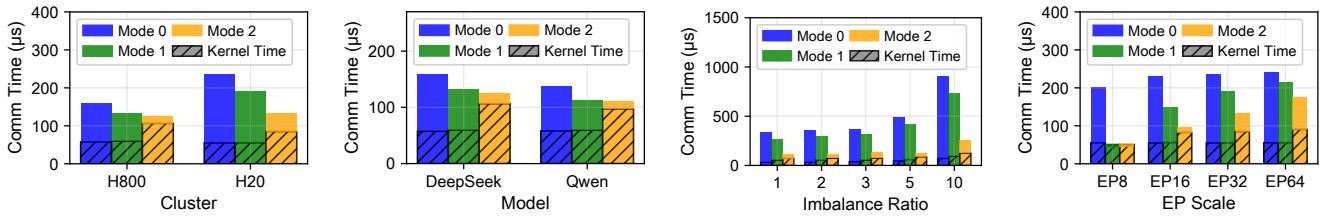
Figure 14(c)(d) compares dynamic migration against no migration across EP scales 16, 32, and 64. Without migration, total communication grows from 386μs→585μs→ 790μs and TPOT 74.0ms→ 77.4ms→ 90.5ms With migration, they rise more slowly (278μs→ 476μs→ 607μs, 70.1ms→ 70.5ms→ 77.4ms). Migration cuts communication by 18.6–28.0% and TPOT by 5.3–14.5%, with larger gains at higher EP because more experts amplify skew and many-to-one traffic patterns; migration spreads experts and traffic across GPUs/NICs to relieve hotspots.

Dynamic expert migration combined with period EPLB. Figure 15 compares the baseline EPLB (computed on a prior window) against EPIC EPLB (communication-aware periodic reshuffle) and EPIC EPLB + Migration (adding intra-host migration at runtime) on Qwen and DeepSeek across batch sizes from 16 to 64. We can see that EPIC EPLB already cuts communication and TPOT noticeably, and adding dynamic intra-host migration yields further gains—up to roughly 35% lower communication and 20% lower TPOT at larger batches. The gains grow with batch sizes because per-step traffic increases while stale placements leave NIC hotspots; migration spreads hot experts intra-host to relieve these hotspots. We observe that Qwen benefits more than DeepSeek (*e.g.*, at batch size 64, TPOT drops 19.5% for Qwen vs. 14.5% for DeepSeek) since fewer experts lead to higher per-expert concentration and stronger skew at the same EP scale.

9.4 Communication optimization

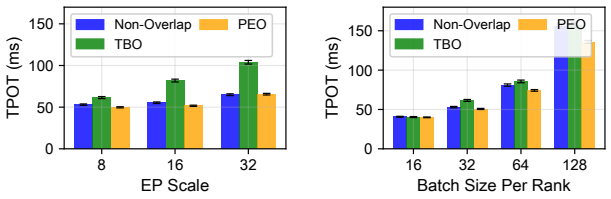
Effect of communication kernel optimizations. We compare the communication kernels in §5.1: Mode 0 (IBGDA-direct, SM-free), Mode 1 (DeepEP: NVLink intra-node + IBGDA inter-node), and Mode 2 (PXN relay + de-redundancy). Across settings (Figure 17), total communication time consistently orders as Mode 2 ≤ Mode 1 < Mode 0, while kernel time (send+recv) orders as Mode 0 < Mode 1 ≤ Mode 2. Furthermore, the effect differs depending on the scenario:

- Clusters (Figure 17(a)): Mode 2 helps more on H20 where inter-host bandwidth is the bottleneck compared with H800. On H20, Mode 2 reduces communication from 235.48 to 134.00μs vs. Mode 0 (-43.1%) and from 190.21 to 134.00μs vs. Mode 1 (-29.6%). On H800 the gains are smaller (-20.9% vs. Mode 0 and -4.9% vs. Mode 1).
- Models (Figure 17(b)): DeepSeek benefits more than Qwen (Mode 1→Mode 2: 131.78→125.33μs, -4.9% vs. 112.07→110.08μs, -1.8%), mainly because DeepSeek's larger hidden size drives higher communication volume.
- EP scale (Figure 17(d)): Mode 2's advantage over Mode 1 shrinks with scale: EP16: 148.66→97.69μs (-34.3%), EP32: 190.21→134.00μs (-29.6%), EP64: 214.50→176.52μs (-17.7%). This is because per-host duplication decreases as experts spread out. EP8 is intra-node only, so Mode 1 = Mode 2 and both outperform Mode 0.
- Imbalance (Figure 17(c)): with growing imbalance, DeepEP (Mode 1) rises from 316.84 to 823.14μs (+160%), whereas Mode 2 grows from 190.98 to 379.17μs (+99%), yielding 39–56% lower time than Mode 1 across ratios. PXN relieves many-to-one/one-to-many contention and de-redundancy removes duplicate inter-host sends.



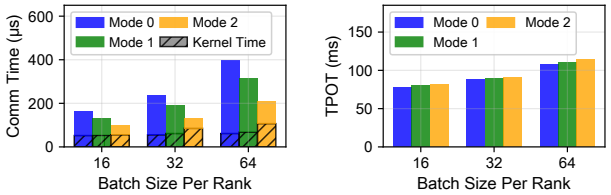
(a) DeepSeek, BS32, EP32 (b) H800, BS32, EP32 (c) DeepSeek, H20, EP32 (d) DeepSeek, H20, BS32

Figure 17: Effect of communication kernel optimizations. Mode 1 aligns with Standard DeepEP.



(a) DeepSeek, BS32, H20 (b) Varying batch size

Figure 18: Non-overlap vs TBO vs PEO.



(a) Without overlap (b) With overlap

Figure 19: Kernel optimizations combined with compute communication overlap [DeepSeek, H20, EP32].

The trade-off is higher kernel time (*e.g.*, at ratio 1: Mode 0/1/2 = 32.63/54.10/72.68μs).

Effect of comp communication overlap. We compare Non-Overlap, TBO, and PEO across batch sizes and EP scales (Figure 18). For batch size $\leq$ 128, PEO consistently lowers TPOT by 0.9–17.6% and 1.8–8.3% compared to TBO and Non-Overlap, respectively. We observe that on our H20 cluster, TBO behaves worse in all of our cases, at all batch sizes. At batch size 32 across EP scales, PEO is always better than TBO and mostly better than Non-Overlap: relative to TBO it reduces TPOT by 18.9-36.9%; relative to Non-Overlap it is lower at EP8 (-6.1%) and EP16 (-6.6%), but slightly higher at EP32 (+1.0%). As EP size increases, each GPU hosts fewer experts and the per-GPU compute workload shrinks. In this regime, PEO's partitioning of the expert computation into multiple groups elongates the effective compute time (due to reduced per-group arithmetic intensity and additional kernel-launch overhead), which eventually exceeds the overlap benefit and leads to a net negative effect. In short, TBO only pays off at very large batches, whereas PEO delivers gains even at small–medium batches and smaller EP scales.

Effect of kernel optimization combined with overlap. Without overlap (Figure 19(a)), Mode 2 consistently achieves the lowest communication time across batch sizes. With overlap enabled (Figure 19(b)), Mode 0/1 benefit more than Mode 2 because their kernels occupy fewer SMs. From batch size 16 to 64, 64, Mode 0 yields the smallest TPOT under overlap (lower by 2.2–3.3% vs. Mode 1 and 3.3–5.3% vs. Mode 2 (81.5/91/114 ms).

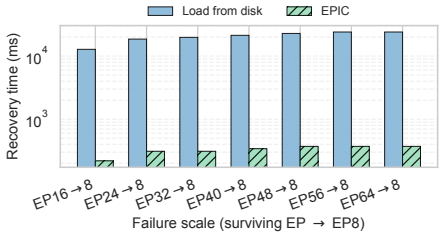


Figure 20: Failure recovery latency after node-level failures.

9.5 Failure recovery

Failover recovery latency. We measure how long it takes to recover from a node failure causes the EP group to shrink to EP8 (§7). We compare *Load from disk*, which reloads the lost expert weights from local disk onto surviving GPUs, with *EPIC*, which fetches the weights from the distributed CPU-side cache. Figure 20 reports the exposed recovery time on Qwen across failure scales (one to seven failed nodes, i.e., EP16→EP8 through EP64→EP8). Disk reload stalls decoding for 12.8–24.1 s, dominated by serialized disk reads onto the surviving GPUs. EPIC brings the exposed recovery time down to 219–371 ms across all scales—roughly two orders of magnitude faster. Combined with in-band failure detection that completes within one collective round (<1 ms via bitmask propagation, §7), end-to-end recovery stays well below half a second even when seven nodes are lost at once, so a peer failure manifests as a brief TPOT spike rather than an instance restart.

10 Lessons-Learned

Lesson #1: CCLs can—and should—transmit anything the workload demands. A CCL is not limited to transporting *tokens* (activations) for distributed parallel computation. In EPIC, we also transmit expert *weights* across GPUs to rebalance load via real-time migration (§4), and transfer auxiliary *metadata* (*e.g.*, load statistics) to coordinate placement decisions and failure recovery without CPU-side round-trips. The boundary between computation and communication is not fixed: the CCL should be designed around what the workload actually needs to move, rather than a rigid set of predefined primitives.

Lesson #2: there is no one-fit-all configuration for diverged workloads. No single communication strategy fits all workloads. In online inference, heterogeneous traffic patterns and SLO tiers

demand different optimizations; early vanilla DeepEP exposed issues uncommon in its original setting, including redundant inter-node transfers and large memory footprint. We addressed these via topology-aware de-redundancy, tighter CCL memory budgeting, and engine-level scheduling (orthogonal to this paper).

We also support pretraining (in-house and external) using a different CCL tuned for training. While patterns are similar, constraints differ: (i) large batches tighten the CCL memory budget, requiring aggressive buffer reuse; (ii) compute dominates step time, making small prep kernels (*e.g.,* permutations) negligible, so the CCL should prioritize collective throughput over latency.

Lesson #3: CCL optimization must be driven by both hardware capabilities and end-to-end workload bottlenecks. Two keys guide effective CCL design: (1) *fully exploiting underlying hardware*, and (2) *optimizing from the actual end-to-end bottleneck of the upper-layer workload*. On the hardware side, DeepEP leverages GPU-driven IBGDA to post and progress network operations directly—eliminating CPU proxy threads and their associated control-plane latency—and leverages TMA [2] as a communication engine to free SMs. On the workload side, the CCL must reason about the *overall* end-to-end bottleneck—not just communication latency in isolation. For instance, load imbalance is a system-level bottleneck that manifests as compute stragglers; the CCL can address it by migrating expert weights at the cost of slight extra communication, yet the end-to-end TPOT improves because the dominant compute straggler is eliminated. A hardware capability's headline benefit on one benchmark does not predict its deployment payoff on another—CCL design must take a stance on *which* bottleneck it targets.

Still, some hardware features are ill-suited to dynamic EP. NVSwitch multicast and in-network reduction favor static collectives, while MoE's per-step top-K routing makes group membership unpredictable. Future hardware could better support variable-size messages and programmable, ephemeral fan-out/fan-in groups. Likewise, many zero-SM copy engines require host-side synchronization/submission, reintroducing CPU overhead and negating benefits in our trials. Fully GPU-managed communication and QoS mechanisms that isolate GEMM from communication traffic would better match dynamic MoE workloads.

11 Related Work

LLM serving, disaggregation, and heterogeneous inference. Prior work improves LLM inference through batching/scheduling and KV-cache management. Orca proposes iteration-level scheduling for autoregressive decoding [44], vLLM reduces KV-cache fragmentation with PagedAttention [29], and Sarathi-Serve refines schedules (e.g., chunked prefills) to balance throughput and latency under SLOs [14]. At the cluster level, Llumnix migrates workloads to mitigate imbalance [41], ServerlessLLM improves elasticity via fast loading/offload [23], and dLoRA orchestrates multi-tenant adapter serving [43]. For MoE, MegaScale-Infer disaggregates attention and experts during decoding and customizes data movement for expert parallelism [46]; related heterogeneous pipelines include FastDecode [25] and Lamina [17]. Overall, MoE serving introduces additional challenges beyond dense models, notably expert hot spots and routing-dependent communication.

MoE optimization. MoE models [20, 28, 35] scale to trillions of parameters while keeping per-token compute low. Expert Parallelism (EP) [27, 36] maps experts to GPUs and preserves kernel efficiency, but suffers *expert workload imbalance* from skewed token routing. Prior approaches add balancing terms or per-expert capacity limits during training [22, 30, 36], improving balance but often reducing accuracy. Serving-time EPLB reshuffles or replicates experts using historical load [8], yet stale statistics can miss rapid shifts and leave residual imbalance. Training-time real-time expert migration [24, 26] can improve balance, but its state-transfer and synchronization costs are typically too high for online inference latency targets. Recent systems further address MoE efficiency from complementary angles. Janus [32] co-designs routing and all-to-all schedules for training clusters; Lina [31] optimizes communication by exploiting expert locality; Brainstorm [19] dynamically partitions experts for heterogeneous clusters; SmartMoE [45] searches for optimal parallelism strategies for MoE layers; and AMoE [34] proposes adaptive expert selection to mitigate load imbalance. EPIC differs from these by targeting *online inference* latency (not training throughput) with a progressive design that combines NIC-aware EPLB, per-step intra-host migration, and topology-adaptive communication.

Collective Communication Optimization. Collective communication is fundamental to distributed ML. Traditional libraries [4–6, 11] and schedule optimizations [16, 33, 37, 38, 42] mainly optimize bandwidth-bound primitives (`AllReduce`, `Broadcast`, `AllGather`) and offer limited support for the irregular, fine-grained traffic of token-level MoE dispatch in EP inference. Specialized libraries such as DeepEP [7] and PPLX [9] therefore introduce explicit `dispatch`/`combine` primitives for token-to-expert `top-k` routing. EPIC builds on DeepEP by adding topology-aware routing, bandwidth de-redundancy, and fine-grained communication–computation overlap to better exploit hardware in large-scale MoE inference.

12 Conclusion

We presented EPIC, an optimization framework for EP-based MoE inference addressing two key bottlenecks: expert workload imbalance and communication inefficiency. EPIC reduce workload imbalance via communication-aware period expert rebalance and lightweight intra-host migration, and further boosts performance with various communication optimization mechanisms. EPIC has been deployed in our online serving system and greatly improves inference performance in various scenarios.

Acknowledgments

We would like to thank anonymous reviewers and our shepherd for their insightful comments.

References

[1] 2022. ibgda. https://docs.nvidia.com/nvshmem/api/using.html.
[2] 2022. NVIDIA Hopper Architecture In-Depth. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/.
[3] 2024. DeepSpeed. https://github.com/microsoft/DeepSpeed.
[4] 2024. Gloo. https://github.com/facebookincubator/gloo.
[5] 2024. MSCCL-EXECUTOR-NCCL. https://github.com/Azure/msccl-executor-nccl.
[6] 2024. NCCL. https://github.com/NVIDIA/nccl.
[7] 2025. DeepEP. https://github.com/deepseek-ai/DeepEP.

[8] 2025. Expert Parallelism Load Balancer (EPLB). https://github.com/deepseek-ai/EPLB.

[9] 2025. pplx-kernels. https://github.com/perplexityai/pplx-kernels.

[10] 2026. Dual batch overlap. https://docs.vllm.ai/en/latest/design/dbo/.

[11] 2026. rccl. https://github.com/ROCm/rccl.

[12] 2026. SGLang. https://github.com/sgl-project/sglang.

[13] 2026. vllm. https://github.com/vllm-project/vllm.

[14] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/agrawal

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). https://arxiv.org/abs/2005.14165

[16] Jiamin Cao, Shangfeng Shi, Jiaqi Gao, Weisen Liu, Yifan Yang, Yichi Xu, Zhilong Zheng, Yu Guan, Kun Qian, Ying Liu, et al. 2025. SyCCL: Exploiting Symmetry for Efficient Collective Communication Scheduling. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 645–662.

[17] Shuo Chen et al. 2024. Efficient Heterogeneous Large Language Model Decoding with Model-Attention Disaggregation. *arXiv preprint arXiv:2405.01814* (2024). https://arxiv.org/abs/2405.01814

[18] Zixiang Chen, Yihe Deng, Yue Wu, Quanquan Gu, and Yuanzhi Li. 2022. Towards Understanding the Mixture-of-Experts Layer in Deep Learning. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 23049–23062. https://proceedings.neurips.cc/paper_files/paper/2022/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf

[19] Weihao Cui, Zhenhua Han, Lingji Ouyang, Yichuan Wang, Ningxin Zheng, Lingxiao Ma, Yuqing Yang, Fan Yang, Jilong Xue, Lili Qiu, Lidong Zhou, Quan Chen, Haisheng Tan, and Minyi Guo. 2023. Optimizing Dynamic Neural Networks with Brainstorm. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association.

[20] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, 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, Chenyu Zhang, Chong Ruan, Damai Dai, 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, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, 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, 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, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, 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, Yanhong Xu, 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. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[21] 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. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[22] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[23] Yichao Fu et al. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. *arXiv preprint arXiv:2401.14351* (2024). https://arxiv.org/abs/2401.14351

[24] Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. 2023. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems* 5 (2023), 288–304.

[25] Jiaao He and Jidong Zhai. 2024. FastDecode: High-Throughput GPU-Efficient LLM Serving using Heterogeneous Pipelines. *arXiv preprint arXiv:2403.11421* (2024). https://arxiv.org/abs/2403.11421

[26] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. 2022. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 120–134.

[27] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. 2023. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems* 5 (2023), 269–287.

[28] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Comput.* 3, 1 (1991), 79–87. https://doi.org/10.1162/NECO.1991.3.1.79

[29] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. *arXiv preprint arXiv:2309.06180* (2023). https://arxiv.org/abs/2309.06180

[30] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Max Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[31] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. 2023. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association.

[32] Juncai Liu, Jessie Hui Wang, and Yimin Jiang. 2023. Janus: A Unified Distributed Training Framework for Sparse Mixture-of-Experts Models. In *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, Henning Schulzrinne, Vishal Misra, Eddie Kohler, and David A. Maltz (Eds.). ACM, 486–498. https://doi.org/10.1145/3603269.3604869

[33] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. 2024. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. In *Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM 2024, Sydney, NSW, Australia, August 4-8, 2024*. ACM, 16–37. https://doi.org/10.1145/3651890.3672249

[34] Xiaonan Nie, Pinxue Zhao, Xupeng Miao, Jiangfei Duan, and Bin Cui. 2025. AMoE: Adaptive Mixture-of-Experts for Efficient Load Balancing in Serving. *arXiv preprint arXiv:2505.08944* (2025).

[35] Alibaba Cloud Qwen Team. 2025. Qwen3-Coder. https://github.com/QwenLM/Qwen3-Coder

[36] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*. PMLR, 18332–18346.

[37] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. 2022. Themis: a network bandwidth-aware collective scheduling policy for distributed training of DL models. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, Valentina Salapura, Mohamed Zahran, Fred Chong, and Lingjia Tang (Eds.). ACM, 581–596. https://doi.org/10.1145/3470496.3527382

[38] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, Mahesh Balakrishnan and Manya Ghobadi (Eds.). USENIX Association, 593–612. https://www.usenix.org/conference/nsdi23/presentation/shah

[39] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=B1ckMDqlg

[40] Noam Shazeer, Azalia Mirhoseini, Piotr Maziarz, Andy Davis, Quoc V Le, Geoffrey Hinton, and Jeffrey Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[41] Biao Sun et al. 2024. Llumnix: Dynamic Scheduling for Large Language Model Serving. *arXiv preprint arXiv:2406.03243* (2024). https://arxiv.org/abs/2406.03243

[42] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil R. Devanur, and Ion Stoica. 2020. Blink: Fast and Generic Collectives for Distributed ML. In *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/paper_files/paper/2020/hash/cd3a9a55f7f3723133fa4a13628cdf03-Abstract.html

[43] Bingyang Wu et al. 2024. dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/wu-bingyang

[44] Gyeong-In Yu, Younggyu Jeong, Geon-Woo Kim, Soojeong Lee, Hyeonwoo Kim, Joo Seong Lee, Byeonghyeon Lee, Jaehoon Sohn, Yongsub Oh, et al. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi22/presentation/yu

[45] Zhihao Zhang and Yibo Zhu. 2023. SmartMoE: Efficiently Training Sparsely-Activated Models through Combining Offline and Online Parallelism. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association.

[46] Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A. Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, Jianzhe Xiao, Xinyi Zhang, Lingjun Liu, Haibin Lin, Li-Wen Chang, Jianxi Ye, Xiao Yu, Xuanzhe Liu, Xin Jin, and Xin Liu. 2025. MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism. *arXiv preprint arXiv:2504.02263* (2025). https://arxiv.org/abs/2504.02263