MIXTURE-OF-KITTENS: MOE MEGAKERNEL FOR NVL72S

Anonymous authors
Paper under double-blind review

ABSTRACT

AI accelerator systems are rapidly consolidating into scale-up architectures, where tens to thousands of GPUs communicate over high-bandwidth, single-hop fabrics. We find that existing Mixture-of-Experts (MoE) training systems, optimized for conventional scale-out networks, transfer poorly to this setting, often running slower than a naive baseline built with PyTorch and NCCL. With industry roadmaps pointing toward even larger scale-up domains, understanding the performance tradeoffs of this hardware regime is increasingly important. We present Mixture-of-Kittens (MoK), an MoE training system designed for Nvidia NVL72. MoK builds on three insights that unlock performance on scale-up domains: (1) choosing push- or pull-based communication per operator, (2) restructuring the computation-communication overlap, and (3) fully eliminating CPU-GPU synchronization. MoK distills these insights into a single deterministic training megakernel that fuses token dispatch, shared and routed expert FFNs, and token combine. Across MoE layer shapes from four widely used open-weight models, MoK delivers up to $2.37\times$ the throughput of the strongest publicly available baseline. In a production run on 512 GPUs spanning multiple GB300 NVL72 racks, MoK improves end-to-end training throughput by $1.41\times$.

1 INTRODUCTION

The hardware for training large language models (LLMs) is rapidly shifting toward scale-up architectures. Rather than having small groups of accelerators with a commodity scale-out network, vendors now place tens to thousands of accelerators inside a single high-bandwidth scale-up domain: examples include Nvidia's GB200/GB300 NVL72 (Nvidia, 2024; 2025b), AMD's MI455X-based Helios (AMD, 2026), Google's TPU Pod (Jouppi et al., 2023), and AWS Trn2 UltraServers (Amazon Web Services, 2024). In these platforms, the interconnect stops behaving like a network and starts behaving like one large memory system, opening new opportunities for AI systems design while invalidating assumptions that existing designs were built on.

We present a study of this opportunity, focusing on Mixture-of-Experts (MoE) (Shazeer et al., 2017) training on Nvidia NVL72 systems (Nvidia, 2025b). MoE has become a core component of frontier AI models and has been shown to significantly improve their capabilities (Moonshot AI, 2026; GLM-5-Team et al., 2026; Qwen Team, 2026; DeepSeek-AI et al., 2026). However, when scaled across multiple GPUs, MoE becomes both communication- and compute-intensive, consequently consuming more than half of end-to-end execution time during large-scale training as reported by previous work (Pan et al., 2025; Yan et al., 2026). Meanwhile, Nvidia's roadmap points toward progressively larger scale-up domains, including NVL144, 576, and 1152 (Bhargava et al., 2026). As the first generation in this family, NVL72 provides a timely platform for studying system tradeoffs that will become increasingly important in future hardware.

Prior work often accelerates distributed MoE by overlapping token communication with expert computation (Zhao et al., 2025b; Zhang et al., 2025; Yu et al., 2026). However, we find that existing approaches either do not support scale-up platforms or become suboptimal on them. In nearly half of the configurations we evaluate, a naive baseline built from PyTorch and NCCL outperforms every evaluated alternative (Section 4.1). These results suggest that current MoE system designs do not transfer cleanly to the cost structure of scale-up hardware.

To understand this gap, we analyze MoE performance characteristics on NVL72s and distill our

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.

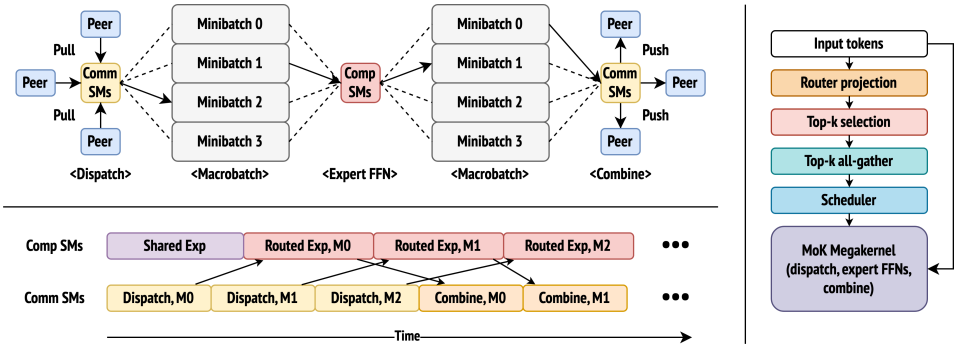


Figure 1: Overview of MoK. Top left: communication SMs pull tokens from peer GPUs into a fixed-size macrobatch ring buffer, expert FFNs process each minibatch, and communication SMs push the outputs back. Bottom left: MoK partitions SMs and pipelines minibatches (M0, M1, ...) to overlap computation with communication. Right: before the megakernel launch, MoK performs GPU-side router projection, top-k selection and all-gather, and dispatch scheduling.

findings into three key insights that unlock performance on scale-up architectures:

1. **Choosing the right communication direction** (Section 3.1). Unlike scale-out domains, scale-up fabrics make pull-based communication as efficient as push-based. This allows us to rethink each MoE operator's communication *direction*, push or pull. We observe that prior designs make suboptimal direction choices that incur avoidable overheads in fine-grained communication, spending up to 18% of its latency on completion signaling and 13% of MoE runtime on pre-dispatch scheduling. By selecting the right set of directions (e.g., pull dispatch and push combine), we can keep signaling overhead under 1%, build the schedule with up to 2.2× less overhead, and share one schedule table across all operators.
2. **Restructuring the computation-communication overlap** (Section 3.2). The scale-up domain enables computation-communication overlap across all 72 GPUs at granularities that were impractical over InfiniBand or RoCE. However, if the granularity is too fine, the tensor cores never saturate, while too coarse a granularity leaves dispatch and combine latency exposed. We find that the throughput-maximizing granularity ranges from 512 to 32,768 tokens across our workloads. By making granularity a first-class tunable parameter, and designing the GPU-side schedule, overlapping scheme, and megakernel fusion around it, we achieve up to 3.53× the throughput of a design fixed to a suboptimal granularity.
3. **Fully eliminating CPU-GPU synchronization** (Section 3.3). The integrated CPUs on NVL72s run slower than those on DGX systems, with execution of common PyTorch operations up to 2.97× slower. It is thus critical to remove CPU-GPU synchronization by moving memory buffer management to the device side. To enable this, we cycle tokens through a fixed-size ring buffer with overlapped writes and reads, and integrate a ring-aware forward replay into the backward pass to recover overwritten tokens. Compared to over-allocation, our ring buffering adds only 1.7% latency on average across our workloads.

Based on these insights, we introduce **Mixture-of-Kittens (MoK)**, an MoE training system built from first principles for NVL72 platforms. MoK fuses all MoE computation and communication (i.e., dispatch, expert GEMMs, gated activation, and combine) into a single persistent megakernel (Spector et al., 2025; Cheng et al., 2026). Beyond demonstrating the above three insights, MoK is a fully production-ready system. It provides features critical for real-world MoE training, including determinism, RDMA overlap for FSDP, MXFP8 support with fused quantization, fused router weight gradient computation, and tunable SM partitioning (Appendix D).

We evaluate MoK on four widely used open-weight model shapes: Kimi K2.7, GLM 5.2, Qwen 3.5-397B-A17B, and DeepSeek V4 Pro. Against publicly available implementations that support NVL72s, MoK outperforms the strongest baseline by up to 2.37× (MXFP8 forward), 1.78× (MXFP8 backward), 1.92× (BF16 forward), and 1.58× (BF16 backward). In our production training stack, on a 512-GPU run across multiple GB300 NVL72 racks, MoK improves end-to-end training throughput by 1.41× in tokens per second per GPU over the previous DeepEP-based backend.

MoK is open-sourced and powers the production training of a frontier LLM across tens of thousands of GPUs. MoK is integrated into Nvidia NeMo AutoModel as the MoE backend (Nvidia, 2025a).

In summary, we make the following contributions:

- We present MoK, an open-source MoE training system that fuses all MoE computation and communication into a deterministic, production-ready megakernel (Section 3).
- We identify three key factors for efficient MoE execution on scale-up platforms: communication direction, overlap granularity, and CPU-GPU synchronization (Sections 3.1–3.3).
- We show that MoK achieves up to 2.37× speedup over the strongest baselines (Section 4.1) and a 1.41× end-to-end throughput improvement in production training (Section 4.2).

2 Background

We review MoE and expert parallelism, GPUs and scale-up architectures, and related work.

2.1 MoE and Expert Parallelism

For each input token, the MoE operation uses a router projection to select the top-k routed experts and assign each a router weight. The shared expert and each selected routed expert then apply a feed-forward network (FFN), consisting of up and gate projections, a gated activation, and a down projection. MoE sums the outputs, weighted by the normalized router weights:[1]

$$\mathrm{MoE}(x, s) = f_{\mathrm{shared}}(x) + \sum_{i \in \mathcal{E}} \frac{s_i}{\sum_{j \in \mathcal{E}} s_j} f_i(x),$$

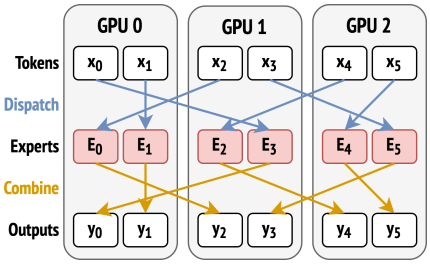


Figure 2: Expert parallelism example with EP degree 3 and top-1 routing.

where $x \in \mathbb{R}^D$ is the input token with hidden dimension D, $\mathcal{E}$ is the set of k selected routed experts, $s \in \mathbb{R}^{|\mathcal{E}|}$ are their router weights, and $f_i(x) = W^{(i)}_{\mathrm{down}}\left(\phi(W^{(i)}_{\mathrm{gate}}x) \odot W^{(i)}_{\mathrm{up}}x\right)$ is expert i's FFN, with activation ϕ, intermediate dimension I, and weights $W^{(i)}_{\mathrm{up}}, W^{(i)}_{\mathrm{gate}} \in \mathbb{R}^{I \times D}$ and $W^{(i)}_{\mathrm{down}} \in \mathbb{R}^{D \times I}$.

Expert parallelism (EP) (Lepikhin et al., 2021; Fedus et al., 2022) partitions the routed experts across multiple GPUs, or *ranks*. We call the number of ranks that collectively hold all expert weights the *EP degree*. Figure 2 shows an example: with 6 routed experts and an EP degree of 3, each rank holds 2 routed experts. Tokens must therefore be transferred across GPUs according to the router selection. We refer to the transfer before the expert FFN as *dispatch*, and after it as *combine*.

Many systems optimize EP by *overlapping* dispatch and combine with expert FFNs (Zhao et al., 2025b; Zhang et al., 2025; Aimuyo et al., 2025; Yu et al., 2026), communicating one chunk of tokens while computing the FFN on another chunk. MoK is one variant of these schemes.

2.2 GPUs and Scale-Up Architectures

GPUs load data from local or remote HBM, perform computation, and write the results back to local or remote HBM. To do this, a GPU kernel executes tens of thousands of hardware threads across over a hundred *streaming multiprocessors* (SMs), which issue instructions to different execution units concurrently. Because these units saturate at different rates, achieving peak performance requires overlapping their use to hide non-critical paths while maximizing the throughput of critical ones.

The GPU memory hierarchy trades latency for capacity as it moves farther from the SMs. Each SM contains 65,536 32-bit registers, accessible every clock cycle, and 228 KB on-chip shared memory (≈40 TB/s). All SMs share a 129 MB L2 cache (≈22 TB/s) backed by 279 GB HBM (8 TB/s), and can access a remote GPU's HBM over NVLink (1.8 TB/s) or InfiniBand/RoCE (200 GB/s).[2]

Scale-up architectures like NVL72 extend this hierarchy by connecting far more GPUs through a high-bandwidth, non-blocking fabric with uniform, single-hop any-to-any connectivity. This changes the tradeoff for parallelism strategies like EP. Parallelism axes that previously had to traverse

[1] We use the DeepSeek V3-style MoE architecture (DeepSeek-AI et al., 2024; Dai et al., 2024) as an example here, but the principles generalize to other MoE variants.

[2] We use the numbers for the Nvidia GB300 NVL72 (Nvidia, 2025b) as a representative example.

the lower-bandwidth scale-out network can now stay entirely within the scale-up domain, raising the performance ceiling and enabling more aggressive overlapping.

2.3 Related Work

Computation-Communication Overlap. A large body of work overlaps GPU computation and communication for various AI workloads. Some systems overlap host-triggered inter-GPU memory transfers with device kernels (Wang et al., 2024; Chang et al., 2024; Zhao et al., 2025b; Thakkar et al., 2017), while others propose on-device schedulers that issue fine-grained, device-initiated communication (Zhang et al., 2025; Aimuyo et al., 2025). Some works instead provide general abstractions and programming primitives that enable flexible overlap strategies (Zheng et al., 2025; Sul et al., 2026). These techniques span individual operators such as attention (Liu et al., 2024) and MoE (Zhang et al., 2025), as well as entire models (Shoeybi et al., 2019; Spector et al., 2025). MoK builds on this body of work, specifically targeting distributed MoE training on scale-up platforms.

Efficient MoE Kernels. MoK is inspired by three classes of systems that optimize MoE execution. First, some systems accelerate expert FFNs via block-sparse reformulation (Gale et al., 2023), fused token permutation (Tan et al., 2024), low-precision expert-grouped GEMM (Zhao et al., 2025a), or IO-efficient algorithmic redesign (Guo et al., 2026). MoK reuses some of these techniques for local computation, such as SonicMoE's router gradient fusion (Appendix D).

Second, many systems optimize EP communication and its overlap with computation via hierarchical all-to-all (Rajbhandari et al., 2022), adaptive pipelining of all-to-all with expert FFNs (Hwang et al., 2023), dedicated dispatch and combine kernels (Zhao et al., 2025b), or fine-grained device-initiated communication fused with expert FFNs (Zhang et al., 2025; Aimuyo et al., 2025). Closest to MoK is MegaMoE, a fused EP MoE megakernel (Zhao et al., 2025a). However, MegaMoE is an inference-only forward kernel for DeepSeek-V4 serving in FP8×FP4, whereas MoK supports full training on scale-up architectures with training-preferred precisions like BF16 and MXFP8.

Third, several end-to-end frameworks (Shoeybi et al., 2019; Rasley et al., 2020; Nvidia, 2023; Kwon et al., 2023; Zheng et al., 2024; Zhu et al., 2025) integrate efficient per-operator kernels and optimization techniques into full training or inference stacks. MoK is complementary to these systems and can serve as their MoE backend, as it already does for Nvidia NeMo AutoModel (Nvidia, 2025a).

Megakernels. MoK adopts the megakernel approach (Spector et al., 2025; Cheng et al., 2026), in which a single persistent kernel executes multiple operators to remove per-kernel launch latency, achieve inter-operator overlap, and minimize straggler tasks. In MoK, dispatch, expert FFNs, and combine are integrated into a single megakernel. Megakernels, however, only provide a mechanism for combining kernels at operator boundaries and do not prescribe operator fusion or computation-communication overlap, which is where MoK's main design decisions lie.

3 Mixture-of-Kittens

MoK's goal is to minimize end-to-end MoE latency C. At a high level, this decomposes as:

$$C = \max(T_{\text{comp}}, T_{\text{mem}}, T_{\text{comm}}) + T_{\text{non-overlap}} + T_{\text{sync}} + T_{\text{host}}. \quad (1)$$

Here, T_{comp}, T_{mem}, and T_{comm} denote computation, intra-GPU memory access, and inter-GPU communication times, respectively. If these components overlap perfectly, their joint cost is their maximum. $T_{\text{non-overlap}}$ accounts for the non-overlapped portions of these operations, such as the initial dispatch and the final combine. T_{sync} covers the intra- and inter-GPU synchronization time spent on completion signals and memory fences. T_{host} captures host-side overheads like CPU-GPU synchronization, memory allocation, and kernel launch latency. Figure 3 illustrates these terms.

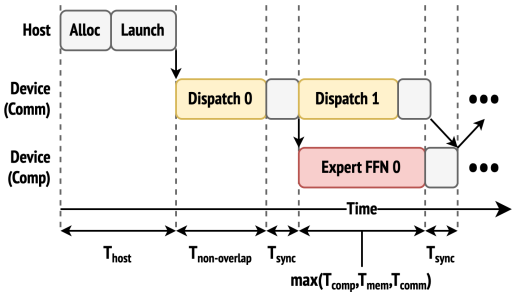
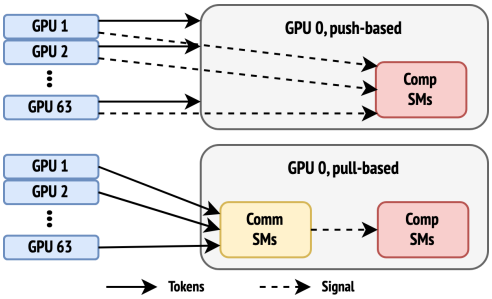


Figure 3: Cost model terms on a naive overlapped MoE execution. Unlabeled boxes denote synchronization. Arrows mark execution dependencies.



	Latency (μs)		Signal overhead (μs)	
EP degree	Pull	Push	Pull	Push
4	475	489	1	39
8	559	574	1	51
16	603	633	4	66
32	638	704	-1	119
64	638	744	-2	135

Figure 4: Signaling under push and pull dispatch (left) and its latency and signal overhead on NVL72 (right) for 7,168 BF16 tokens of width 7,168 per GPU, 4 local experts, and 4 random routes per token. Signal overhead is the latency added by in-kernel completion signaling.

Each of T_{comp}, T_{mem}, and T_{comm} must also be minimized individually by driving its hardware unit to peak utilization. For T_{comp}, this means keeping the tensor cores fed by pipelining data operands. For T_{comm}, it means saturating the network lanes with outstanding transfers and spreading traffic evenly.

The remainder of this section presents three design choices that minimize C. In addition, Appendix D discusses features and optimizations that improve production readiness and performance.

3.1 Choosing the Right Communication Direction

This section explains why communication direction matters for fine-grained MoE overlap, which direction MoK chooses for each operator, and how we optimize for the chosen directions.

Under fine-grained computation-communication overlap, signaling (T_{sync}) becomes a bottleneck, because each task (e.g., a tensor-core matrix multiply) cannot begin until its input data is known to have arrived. The direction of communication determines how much signaling is required. Figure 4 (left) illustrates this for dispatch. With push-based dispatch, every peer must send a completion signal each time it pushes a chunk of tokens, whereas with pull-based dispatch a local signal suffices once the destination GPU has issued all of its loads. As Figure 4 (right) shows, push-based dispatch spends 39–135 μs on signaling, or 8–18% of dispatch time, and this cost grows with EP degree. On the other hand, pull-based signaling stays within 4 μs. A fine-grained megakernel incurs this overhead many times per execution, so the cost compounds.

Prior EP MoE systems predominantly use push-based transfers (Zhao et al., 2025b; Mao et al., 2026; Yu et al., 2026), since push outperforms pull for fine-grained RDMA transfers (Wei et al., 2018; Ma et al., 2026). On scale-up platforms, however, the entire EP group can fit within a scale-up domain, which frees us to choose each MoE operator's communication direction independently.

MoK therefore chooses each operator's direction so that the GPU receiving the signal also issues the remote loads or stores. For the forward pass, each GPU pulls its inputs (pull-based dispatch), signals its local computation workers, and pushes its outputs (push-based combine) once they signal completion. The backward pass mirrors this with pull-based reverse-combine and push-based reverse-dispatch. The handoff between communication and computation thus never leaves the GPU.

Scheduling. Once the communication direction is chosen, we must decide the token transfer order, so that it both saturates the network links (T_{comm}) and keeps the receiving tensor cores busy (T_{comp}).

Let us consider pull-based dispatch first. At each step, a GPU must decide (1) which peer to pull from, (2) which token to pull from that peer, and (3) where in local memory to store it. For (1), peers should be selected round-robin so that all network lanes carry roughly the same number of outstanding requests. For (2), tokens should arrive in non-decreasing local-expert order so that the per-expert matrix multiplies are not fragmented. For (3), tokens should fill the local buffer contiguously so that no local memory movement is needed before they are fed to the tensor cores.

MoK satisfies all three conditions through a separate kernel that builds a schedule table, which is then passed to the megakernel (Figure 1, right). The table has two columns, {`src_rank`, `src_index`}, and its row index is the token's position in the destination buffer. Dispatch walks the table row by row. Algorithm 1 (Appendix B) details how the table is built. Conceptually, it sorts the routes by the key (local expert ℓ, per-source-rank ordinal o, source rank r), where o counts the

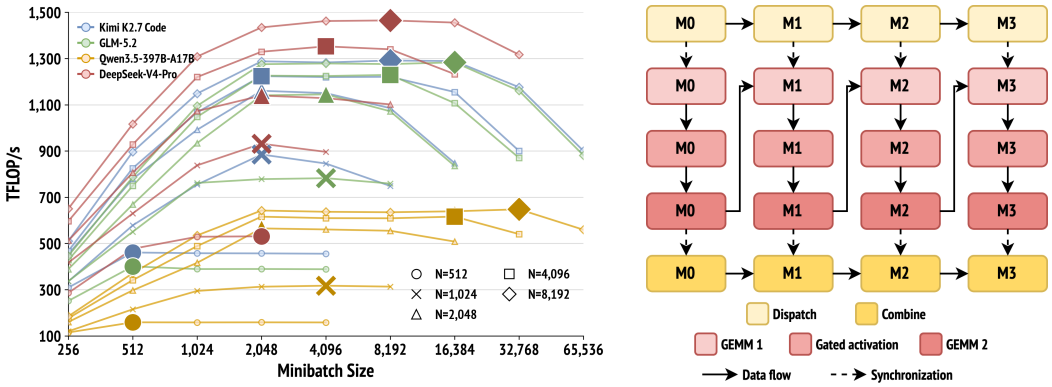


Figure 5: Left: MoK MXFP8 forward throughput on 16 GPUs as a function of minibatch size. N is input tokens per GPU, and enlarged symbols mark each curve's throughput-maximizing minibatch size, which yields 1.38×–3.53× the throughput of the worst. Right: maximally deferred synchronization over minibatches M0–M3. Communication and computation signal at the dashed arrows.

earlier routes from the same source rank to the same expert. Sorting by ℓ first gives each expert a contiguous region, and sorting by (o, r) within a region interleaves the source ranks round-robin. MoK's GPU implementation parallelizes this and builds the schedule in 2.1–7.7% of MoE runtime across our workloads, a 1.4–2.2× smaller overhead than Comet's (Zhang et al., 2025; Appendix C).

Finally, the schedule can be built once and reused by all four communication operators. Push-based combine, for instance, can reinterpret {`src_rank, src_index`} as {`dst_rank, dst_index`}, and the pushes will remain evenly spread across peers. The table occupies only a few megabytes in the worst case, so retaining it across operators costs negligible memory.

3.2 Restructuring the Computation-Communication Overlap

This section describes computation-communication overlap granularity, shows that the optimal granularity depends on the workload, and explains how MoK supports all granularities efficiently.

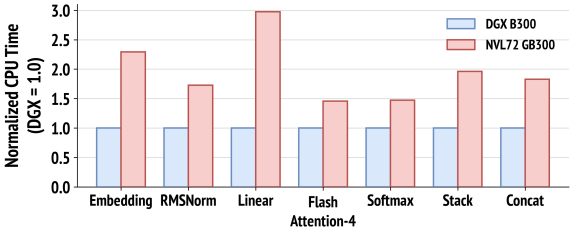
Overlapping computation and communication requires exchanging a readiness signal for each chunk of tokens transferred and computed. We call this chunk a *minibatch*, and its size sets the *granularity* of the overlap. At one extreme, the overlap is fine-grained, at just enough tokens to fill a single matrix-multiply-accumulate instruction, as in Comet (Zhang et al., 2025). At the other, it is coarse-grained, at tens of thousands of tokens. This is a common usage of DeepEP (Zhao et al., 2025b).

Neither extreme is optimal. Let N_{routed} denote the number of routed tokens, b the minibatch size, and w the workload shape, including hidden and intermediate dimensions. A small b never saturates the tensor cores and pays a per-minibatch cost $C_{\text{ineff}}(b, w)$ of underutilized compute and synchronization overhead, giving a fine-grained penalty $P_{\text{fine}}(b, w) \approx \lceil N_{\text{routed}}/b \rceil\, C_{\text{ineff}}(b, w)$. A large b leaves the first minibatch's dispatch and the last minibatch's combine unoverlapped, giving a coarse-grained penalty $P_{\text{coarse}}(b) \approx T_{\text{dispatch}}(b) + T_{\text{combine}}(b)$. Increasing b lowers the first term and raises the second, so b should be chosen per workload to minimize their sum, $P_{\text{fine}}(b, w) + P_{\text{coarse}}(b)$.

Figure 5 (left) shows this tradeoff empirically. Throughput first rises with minibatch size as tensor-core utilization improves and per-minibatch overhead amortizes. It then plateaus or declines as coarser minibatches expose communication and tensor-core bubbles. The throughput-maximizing minibatch size ranges from 512 to 32,768 tokens across model shapes and token counts.

Thus, MoK makes granularity a first-class tunable parameter and keeps its overhead low at any granularity, instead of fixing one at design time. MoK achieves this through (1) maximally deferring the computation-communication synchronization and (2) fusing all operators into one megakernel.

Maximally Deferred Synchronization. MoK lets computation and communication run as far ahead as data dependencies allow. Figure 5 (right) illustrates this. Communication proceeds *breadth-first* over minibatches, dispatching all tokens continuously without waiting for computation to complete, whereas computation proceeds *depth-first*, running one minibatch through the entire expert FFN before moving on to the next. The two sides signal each other asynchronously only at the dashed arrows, where computation waits for a minibatch to arrive and combine waits for its outputs.



MoE Shape	Local tokens per rank 1,024	2,048	4,096
Kimi K2.7 Code	0.3%	1.2%	0.7%
GLM-5.2	7.4%	4.6%	1.9%
Qwen3.5-397B	0.7%	1.3%	2.5%
DeepSeek-V4-Pro	0.1%	−0.5%	1.0%

Figure 6: Left: host-side invocation time of common PyTorch operations on GB300 NVL72, normalized to DGX B300. Right: percentage increase in MoK latency when using a ring buffer that can hold half the routed tokens versus an over-allocated buffer that can hold all of them.

Kernel Fusion. MoK fuses all computation and communication into a single megakernel (Spector et al., 2025) and runs them concurrently through inter-SM overlapping (Sul et al., 2026). Some SMs are assigned exclusively to the expert FFNs and others to dispatch and combine. The two groups signal each other through a GPU-local barrier. Without a megakernel, the schedule in Figure 5 (right) would require launching several kernels per minibatch and overlapping them through multiple streams. This would incur kernel launch and stream synchronization costs (T_{host}) of tens of microseconds per minibatch, and it would leave the interleaving to the hardware scheduler, which does not guarantee concurrent execution of computation and communication (Nvidia, 2026b).

3.3 Eliminating CPU-GPU Synchronization

This section quantifies the cost of CPU-GPU interaction on the NVL72 platform, describes MoK's token ring buffering, and explains how it stays efficient in both the forward and backward passes.

Host-side execution on GB300 NVL72 systems is slow relative to conventional DGX systems. In Figure 6 (left), we compare the host-side invocation time of common PyTorch operations, and find that the NVL72 host path is $1.46\times$ to $2.97\times$ slower. At production scale, we observe that GPU execution is thus far more likely to be bottlenecked by host work, and CPU-GPU synchronization becomes much more costly, since it prevents the GPU stream from running ahead of the CPU thread.

One common source of such synchronization in EP execution is memory allocation. Because the number of tokens routed to each expert is not known until runtime, allocating exact-sized destination buffers requires copying these counts to the CPU before the allocation (Gale et al., 2023). An alternative is to drop tokens that exceed a fixed buffer capacity (Lepikhin et al., 2021; Fedus et al., 2022), but this degrades training quality (Gale et al., 2023).

MoK avoids both CPU-GPU synchronization and token dropping by managing a token ring buffer entirely on the device. We call this ring buffer a *macrobatch* and expose its size as a tunable parameter. The ring buffer enables a producer-consumer pipeline between dispatch and the expert FFNs, and between the expert FFNs and combine. Each producer writes into one region of the macrobatch while its consumer works on another. For example, in Figure 1 (top left), dispatch writes tokens to the third slot while the expert FFNs process the second slot and combine reads from the first slot. The same pipelining continues when the ring wraps around, so the ring boundary adds almost no overhead. Compared to over-allocating a buffer that holds every routed token, which is impractical in production yet shows the best achievable performance, ring buffering adds only 1.7% latency in geometric mean across our workloads (Figure 6, right).

Ring-Aware Forward Replay. In production training, device memory cannot hold all intermediate activations of the forward pass, so the backward pass must recompute some of them through forward replay. However, naively rerunning the MoK forward pass preserves only the activations of the last macrobatch, since ring buffering overwrites everything before it, and the forward megakernel recomputes activations that the backward pass never consumes (e.g., the down-projection output).

MoK therefore fuses a ring-aware forward replay into the backward megakernel, where it recovers only the missing intermediates and is overlapped with the backward. We further minimize this replay by maximizing the tokens that survive in the ring after the forward. If the forward processes macrobatches in ascending order, only the last macrobatch survives, which is usually partial unless all tokens fit in one macrobatch. MoK's forward pass instead processes tokens in *reverse macrobatch order*, so that it ends with a completely full ring and the backward pass replays as little as possible.

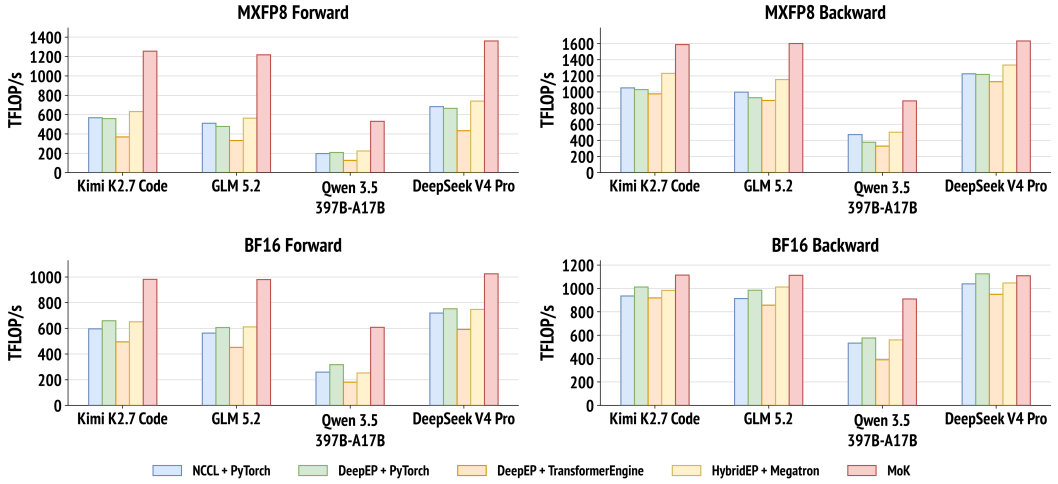


Figure 7: MoE layer throughput per GPU on a GB300 NVL72. The x-axis is the MoE shape and the y-axis is the effective expert-FFN throughput in TFLOP/s. We use an EP degree of 64 and 2,048 input tokens per GPU before routing. Higher is better.

4 Experiments

Section 3 introduced MoK and its three key design choices: communication direction, overlap granularity, and ring buffering. To evaluate their effectiveness, we benchmark MoK at two levels: (1) the effective compute throughput of standalone MoE forward and backward passes (Section 4.1), and (2) the end-to-end training throughput on a production training stack with 512 GPUs (Section 4.2). All experiments were run on GB300 NVL72 racks, each containing 72 B300 GPUs fully interconnected via fifth-generation NVLink (Nvidia, 2025b), using CUDA 13.0, Python 3.13, and PyTorch 2.13. We made a best-effort attempt to tune every baseline for maximum performance, and all final configurations are in our open-source repository.

4.1 MoE Layer Benchmarks

We benchmark the complete execution of a single MoE layer, including global token scheduling, dispatch, routed and shared expert FFNs, combine, and the router-weighted sum. Router logits and model inputs are generated from a standard normal distribution, and every implementation receives bitwise-identical copies of them. We measure the forward and backward passes separately in both BF16 and MXFP8, timing 100 executions after 500 warmup iterations.

We record the median slowest-rank latency over the 100 executions and convert it to effective expert-FFN throughput. For N input tokens, hidden dimension D, expert intermediate dimension I, top-k routing, and measured time t, forward throughput is $6N(k+1)DI/t$, where $k+1$ accounts for the k routed experts and one shared expert. The backward pass uses twice this FLOP count.

We compare against the publicly available baselines that (1) support both MoE forward and backward passes and (2) support execution on NVL72 systems: NCCL (Nvidia, 2015) + PyTorch, DeepEP (Zhao et al., 2025b) + PyTorch, DeepEP + TransformerEngine (Nvidia, 2022), and HybridEP (Yu et al., 2026)[3] + Megatron (Shoeybi et al., 2019).

We evaluate the MoE layer shapes of four widely used open-weight models, each with one shared expert, E routed experts, hidden dimension D, intermediate dimension I, and k routes per token: Kimi K2.7 Code (E=384, D=7168, I=2048, top-k=8), GLM-5.2 (E=256, D=6144, I=2048, top-k=8), Qwen3.5-397B-A17B (E=512, D=4096, I=1024, top-k=10), and DeepSeek-V4-Pro (E=384, D=7168, I=3072, top-k=6).

Figure 7 shows MoE layer throughput for every combination of model shape, pass (forward and backward), and precision, using EP degree 64 and 2,048 tokens per GPU before dispatch. Relative to the fastest baseline for each combination, MoK achieves speedups of up to 2.37× for MXFP8 forward, 1.78× for MXFP8 backward, 1.92× for BF16 forward, and 1.58× for BF16 backward.

[3]Megatron also supports DeepEP, but Nvidia recommends HybridEP for optimal performance on NVL72.

Table 1: MoK sensitivity to local token count. "Tokens" is local tokens per GPU before dispatch. "Best" is the lowest latency among the four baselines per workload, with the subscript identifying the baseline (N: NCCL+PyTorch, H: HybridEP+Megatron). "Speedup" is "Best" divided by "MoK".

		Forward (ms)		Backward (ms)		Speedup	
MoE Shape	Tokens	MoK	Best	MoK	Best	Fwd	Bwd
Kimi K2.7 Code	1,024	1.016	2.600_{N}	1.555	2.295_{H}	**2.56**×	**1.48**×
	2,048	1.293	2.570_{H}	2.045	2.636_{H}	**1.99**×	**1.29**×
	4,096	2.524	4.002_{H}	3.871	4.709_{H}	**1.59**×	**1.22**×
GLM-5.2	1,024	0.989	2.706_{H}	1.612	2.287_{H}	**2.74**×	**1.42**×
	2,048	1.142	2.471_{H}	1.737	2.414_{H}	**2.16**×	**1.39**×
	4,096	2.171	3.354_{H}	3.294	4.147_{H}	**1.54**×	**1.26**×
Qwen3.5-397B-A17B	1,024	0.995	2.567_{N}	1.442	2.348_{H}	**2.58**×	**1.63**×
	2,048	1.068	2.527_{H}	1.273	2.267_{H}	**2.37**×	**1.78**×
	4,096	1.830	2.978_{H}	2.137	2.959_{H}	**1.63**×	**1.38**×
DeepSeek-V4-Pro	1,024	0.981	2.670_{N}	1.483	2.470_{H}	**2.72**×	**1.67**×
	2,048	1.392	2.564_{H}	2.321	2.839_{H}	**1.84**×	**1.22**×
	4,096	2.594	3.796_{N}	4.346	5.112_{H}	**1.46**×	**1.18**×

We observe that MoK's relative gains are larger for MXFP8 than for BF16, and larger for the forward pass than for the backward pass. This is because peak BF16 tensor-core throughput on GB300 is half that of MXFP8 (Nvidia, 2025b), and the backward pass performs roughly twice as many floating-point operations as the forward pass. Both factors make execution more compute-bound, spending a smaller fraction of time on the inter-GPU communication that MoK effectively overlaps or eliminates. We expect such gains to grow on future hardware; for example, Nvidia Vera Rubin (Alvarez et al., 2026) raises FP8 compute throughput by 3.5× but scale-up network bandwidth by only 1.67× compared to GB300, making execution more communication-bound.

Sensitivity to Local Token Count. Table 1 reports MXFP8 latency and speedup at EP degree 64 for 1,024, 2,048, and 4,096 tokens per GPU. MoK achieves larger speedups at smaller local token counts, where communication and scheduling overheads account for a larger fraction of execution time; the same trend holds at other EP degrees as well, and the full table is available in Appendix A. This trend is important in large-scale training, because the global batch size cannot be increased indefinitely without diminishing optimization returns. Scaling to more GPUs must therefore reduce the tokens processed per GPU (McCandlish et al., 2018; Shallue et al., 2019; Kaplan et al., 2020), making MoK increasingly effective as deployments scale.

4.2 End-to-End Benchmarks

We also evaluated MoK in our internal production training stack. We compared against the previous version of the stack, which overlapped DeepEP-based MoE communication with our previously open-sourced MXFP8 expert FFN kernels. Varying only the MoE implementation, we measured end-to-end training throughput on 512 GB300 GPUs across multiple NVL72 racks, with an EP degree of 32, MXFP8 routed experts, and a BF16 shared expert. MoK increased mean tokens per second per GPU from 761.0 to 1,070.2, which is a 41% improvement over the previous DeepEP-based implementation. In practice, our production training runs use far more GPUs and fewer tokens per GPU than this experiment, which further increases MoK's gain, as explained in Section 4.1.

5 Conclusion

We studied MoE training on NVL72 and identified three insights that unlock performance on scale-up domains: choosing the communication direction per operator, restructuring the computation-communication overlap around a tunable granularity, and eliminating CPU-GPU synchronization through ring buffering. Mixture-of-Kittens (MoK) embodies these insights in a single deterministic megakernel, fusing MoE computation and communication with features for production readiness. MoK delivers up to 2.37× the throughput of the strongest publicly available baselines across widely used open-weight model shapes and a 1.41× end-to-end speedup in production-scale LLM training.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on research methodology or experiments, supporting qualitative and thematic data analysis, and interpreting results. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing research methodology or experiments, implementing methods, assisting with translation, or cleaning and reformatting datasets.

Additionally, we used generative AI tools for modifying scientific figures or images, creating or editing software code, drafting parts of a research paper, summarizing or analyzing existing literature, brainstorming, sourcing/searching for information, editing a research paper to improve readability, identifying relevant literature, and formatting references.

We have reviewed all AI-assisted work. All AI-generated or AI-edited code was thoroughly reviewed line by line and tested for correctness by the authors. AI-assisted analyses and interpretations of results were verified against the measured data. In general, all analyses in this work went through many rounds of human feedback. AI-assisted literature summaries and references were manually checked against the original sources, and all AI-drafted or AI-edited text and figures were reviewed by the authors and many peers.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have released the complete MoK implementation and the full benchmark suite as open source. The suite produces every result reported in Section 4.1 and Appendix A, and the released code is identical to the code used to obtain those results. No internal variant was used. The hardware and software environment is documented in Section 4 and in the repository, each benchmark script records the exact version of the baseline against which it was run, and each script contains the tuned configuration used in our experiments. These configurations are the best we found after several days of best-effort tuning per baseline. All aspects of MoK described in this paper are implemented in the released code, which is written to be readable and organized so that the algorithms and insights presented here can be recognized directly in the code. The end-to-end result in Section 4.2 is the one exception. It cannot be reproduced externally because our production training stack cannot be released, but we include it because end-to-end training throughput is the metric that ultimately matters, and we describe the experimental setup in that section as completely as possible. Links to the code will be included in the camera-ready version.

REFERENCES

Osayamen Aimuyo, Byungsoo Oh, and Rachee Singh. FlashMoE: Fast distributed MoE in a single kernel. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Proceedings of the 39th Annual Conference on Neural Information Processing Systems*, volume 38, pp. 100676–100699, San Diego, California, USA, December 2025. Curran Associates, Inc. doi: 10.52202/085713-3368. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/918d938bd209e5b56072777366f8a211-Paper-Conference.pdf.

Eduardo Alvarez, Vishal Mehta, and Farshad Ghodsian. Inside NVIDIA Rubin GPU architecture: Powering the era of agentic AI. https://developer.nvidia.com/blog/inside-nvidia-rubin-gpu-architecture-powering-the-era-of-agentic-ai/, July 2026. Accessed 2026-09-12.

Amazon Web Services. Amazon EC2 Trn2 instances and UltraServers. https://aws.amazon.com/ec2/instance-types/trn2/, 2024. Accessed 2026-08-20.

AMD. AMD Helios rackscale solution. https://www.amd.com/en/products/rackscale-solutions/helios.html, 2026. Accessed 2026-08-20.

Rohil Bhargava, Taylor Allison, and Harry Petty. NVIDIA Vera Rubin POD: Seven chips, five rack-scale systems, one AI supercomputer. https://developer.nvidia.com/blog/nvidia-vera-rubin-pod-seven-chips-five-rack-scale-systems-one-ai-supercomputer/, March 2026. Accessed 2026-09-03.

Liwen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Xuanrun Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, and Xin Liu. FLUX: Fast software-based communication overlap on GPUs through kernel fusion. *arXiv preprint arXiv:2406.06858v1*, June 2024. URL https://arxiv.org/abs/2406.06858v1.

Xinhao Cheng, Zhihao Zhang, Yu Zhou, Jianan Ji, Jinchen Jiang, Zepeng Zhao, Ziruo Xiao, Zihao Ye, Yingyi Huang, Ruihang Lai, Hongyi Jin, Bohan Hou, Mengdi Wu, Yixin Dong, Anthony Yip, Zihao Ye, Songting Wang, Wenqin Yang, Xupeng Miao, Tianqi Chen, and Zhihao Jia. MPK: A compiler and runtime for Mega-Kernelizing tensor programs. In *Proceedings of the 20th USENIX Symposium on Operating Systems Design and Implementation*, pp. 1909–1926, Seattle, Washington, USA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL https://www.usenix.org/conference/osdi26/presentation/cheng.

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

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J.L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R.J. Chen, R.L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S.S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W.L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X.Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y.K. Li, Y.Q. Wang, Y.X. Wei, Y.X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z.F. Wu, Z.Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, December 2024. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya

Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, April 2026. URL https://arxiv.org/abs/2606.19348.

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

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of the 6th Annual Conference on Machine Learning and Systems*, volume 5, pp. 288–304, Miami Beach, Florida, USA, June 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/5a54f79333768effe7e8927bcccffe40-Paper-mlsys2023.pdf.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song

Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, February 2026. URL https://arxiv.org/abs/2602.15763.

Wentao Guo, Mayank Mishra, Xinle Cheng, Ion Stoica, and Tri Dao. SonicMoE: Accelerating MoE with IO and tile-aware optimizations. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *Proceedings of the 14th International Conference on Learning Representations*, volume 2026, pp. 67814–67839, Rio de Janeiro, Brazil, April 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/6ed5bf446f59e2c6646d23058c86424b-Paper-Conference.pdf.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of the 6th Annual Conference on Machine Learning and Systems*, volume 5, pp. 269–287, Miami Beach, Florida, USA, June 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/5616d34cf8ff73942cfd5aa922842556-Paper-mlsys2023.pdf.

Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David A. Patterson. TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, Orlando, Florida, USA, June 2023. Association for Computing Machinery. ISBN 9798400700958. doi: 10.1145/3579371.3589350. URL https://doi.org/10.1145/3579371.3589350.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, January 2020. URL https://arxiv.org/abs/2001.08361.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, pp. 611–626, Koblenz, Germany, October 2023. Association for Computing Machinery. ISBN 979-8-4007-0229-7. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

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 *Proceedings of the 9th International Conference on Learning Representations*, Virtual Conference, May 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Hao Liu, Matei Zaharia, and Pieter Abbeel. RingAttention with blockwise transformers for near-infinite context. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *Proceedings of the 12th International Conference on Learning Representations*, volume 2024, pp. 3992–4008, Vienna, Austria, May 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/1119587863e78451f080da2a768c4935-Paper-Conference.pdf.

Yijun Ma, Siyuan Shen, Tiancheng Chen, Akhil Langer, Jiri Kraus, Benjamin Glick, Craig Belusar, Jeff Hammond, and Torsten Hoefler. Demystifying NVSHMEM: A system-level analysis on symmetric memory and device-initiated operations in GPU communication. *arXiv preprint arXiv:2606.05951*, June 2026. URL https://arxiv.org/abs/2606.05951.

Ziming Mao, Yihan Zhang, Chihan Cui, Zhen Huang, Kaichao You, Zhongjie Chen, Zhiying Xu, Zhenyu Gu, Scott Shenker, Costin Raiciu, Yang Zhou, and Ion Stoica. UEP: Portable Expert-Parallel communication. In *Proceedings of the 20th USENIX Symposium on Operating Systems Design and Implementation*, pp. 1107–1123, Seattle, Washington, USA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL https://www.usenix.org/conference/osdi26/presentation/mao-ziming-uep.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, December 2018. URL https://arxiv.org/abs/1812.06162.

Moonshot AI. Kimi K2.7 Code. https://huggingface.co/moonshotai/Kimi-K2.7-Code, June 2026. Accessed 2026-08-20.

Nvidia. NCCL: Optimized primitives for collective multi-GPU communication. https://github.com/NVIDIA/nccl, 2015.

Nvidia. TransformerEngine: A library for accelerating Transformer models on Nvidia GPUs, including using 8-bit and 4-bit floating point (FP8 and FP4) precision on Hopper, Ada and Blackwell GPUs, to provide better performance with lower memory utilization in both training and inference. https://github.com/NVIDIA/TransformerEngine, 2022.

Nvidia. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2023.

Nvidia. Nvidia GB200 NVL72. https://www.nvidia.com/en-us/data-center/gb200-nvl72/, March 2024. Accessed 2026-08-20.

Nvidia. NeMo AutoModel: DTensor-native SPMD library for scalable and efficient training. https://github.com/NVIDIA-NeMo/Automodel, May 2025a.

Nvidia. Nvidia GB300 NVL72. https://www.nvidia.com/en-us/data-center/gb300-nvl72/, March 2025b. Accessed 2026-08-20.

Nvidia. Work stealing with Cluster Launch Control. https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/cluster-launch-control.html, 2026a. Accessed 2026-08-29.

Nvidia. Green Contexts. https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/green-contexts.html, May 2026b. Accessed 2026-08-29.

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 *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 524–539, Rotterdam, Netherlands, March 2025. Association for Computing Machinery. ISBN 9798400706981. doi: 10.1145/3669940.3707272. URL https://doi.org/10.1145/3669940.3707272.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026. Accessed 2026-08-20.

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 Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 18332–18346, Baltimore, Maryland, USA, July 2022. PMLR. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, Virtual Conference, August 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, Stosic Dusan, Venmugil Elango, Maximilian Golub, Alexander Heinecke, Phil James-Roxby, Dharmesh Jani, Gaurav Kolhe, Martin Langhammer, Ada Li, Levi Melnick, Maral Mesmakhosroshahi, Andres Rodriguez, Michael Schulte, Rasoul Shafipour, Lei Shao, Michael Siu, Pradeep Dubey, Paulius Micikevicius, Maxim Naumov, Colin Verrilli, Ralph Wittig, Doug Burger, and Eric Chung. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, October 2023. URL https://arxiv.org/abs/2310.10537.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. URL https://jmlr.org/papers/v20/18-789.html.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France, April 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, September 2019. URL https://arxiv.org/abs/1909.08053.

Benjamin Spector, Jordan Juravsky, Stuart H. Sul, Owen Dugan, Dylan Lim, Dan Fu, Simran Arora, and Christopher Ré. Look ma, no bubbles! Designing a low-latency megakernel for Llama-1B. https://hazyresearch.stanford.edu/blog/2025-05-27-no-bubbles, May 2025. Accessed 2026-08-20.

Stuart H. Sul, Simran Arora, Benjamin F. Spector, and Christopher Ré. ParallelKittens: Systematic and practical simplification of multi-GPU AI kernels. In A. Chowdhery and Z. Jia (eds.), *Proceedings of the 9th Annual Conference on Machine Learning and Systems*, volume 8, pp. 1076–1089, Bellevue, Washington, USA, May 2026. MLSys. URL https://proceedings.mlsys.org/paper_files/paper/2026/file/ff997469ac66cf893c4183efeb22212a-Paper-Conference.pdf.

Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. Scattered mixture-of-experts implementation. In *Proceedings of the 1st Conference on Language Modeling*, Philadelphia, Pennsylvania, USA, October 2024. URL https://openreview.net/forum?id=YDZ7GeFLxq.

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Aditya Atluri, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish

Gupta. CUTLASS: CUDA templates and Python DSLs for high-performance linear algebra. https://github.com/NVIDIA/cutlass, 2017.

Yifu Wang, Horace He, Less Wright, Luca Wehrstedt, Tianyu Liu, and Wanchao Liang. Introducing async tensor parallelism in PyTorch. https://discuss.pytorch.org/t/distributed-w-torchtitan-introducing-async-tensor-parallelism-in-pytorch/209487/1, September 2024. Accessed 2026-08-20.

Xingda Wei, Zhiyuan Dong, Rong Chen, and Haibo Chen. Deconstructing RDMA-enabled distributed transactions: Hybrid is better! In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*, pp. 233–251, Carlsbad, California, USA, October 2018. USENIX Association. URL https://www.usenix.org/conference/osdi18/presentation/wei.

Zijie Yan, Hongxiao Bai, Xin Yao, Dennis Liu, Tong Liu, Hongbin Liu, Pingtian Li, Evan Wu, Shiqing Fan, Li Tao, Robin Zhang, Yuzhong Wang, Shifang Xu, Jack Chang, Xuwen Chen, Kunlun Li, Yan Bai, Gao Deng, Nan Zheng, Vijay Anand Korthikanti, Abhinav Khattar, Ethan He, Soham Govande, Sangkug Lym, Zhongbo Zhu, Qi Zhang, Haochen Yuan, Xiaowei Ren, Deyu Fu, Tailai Ma, Shunkang Zhang, Jiang Shao, Ray Wang, Vasudevan Rengasamy, Rachit Garg, Santosh Bhavani, Xipeng Li, Chandler Zhou, David Wu, Yingcan Wei, Ashwath Aithal, Michael Andersch, Mohammad Shoeybi, Jiajie Yao, and June Yang. Scalable training of mixture-of-experts models with Megatron Core. *arXiv preprint arXiv:2603.07685*, March 2026. URL https://arxiv.org/abs/2603.07685.

Fan Yu, Tong Liu, and Kai Sun. Optimizing communication for mixture-of-experts training with hybrid expert parallel. https://developer.nvidia.com/blog/optimizing-communication-for-mixture-of-experts-training-with-hybrid-expert-parallel/, February 2026. Accessed 2026-08-20.

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 M. Zaharia, G. Joshi, and Y. Lin (eds.), *Proceedings of the 8th Annual Conference on Machine Learning and Systems*, volume 7, Santa Clara, California, USA, May 2025. MLSys. URL https://proceedings.mlsys.org/paper_files/paper/2025/file/e27ea0cd50b798ff8942caf9203f0992-Paper-Conference.pdf.

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

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Proceedings of the 38th Annual Conference on Neural Information Processing Systems*, volume 37, pp. 62557–62583, Vancouver, British Columbia, Canada, December 2024. Curran Associates, Inc. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Size Zheng, Wenlei Bao, Qi Hou, Xuegui Zheng, Jin Fang, Chenhui Huang, Tianqi Li, Haojie Duanmu, Renze Chen, Ruifan Xu, Yifan Guo, Ningxin Zheng, Ziheng Jiang, Xinyi Di, Dongyang Wang, Jianxi Ye, Haibin Lin, Li-Wen Chang, Liqiang Lu, Yun Liang, Jidong Zhai, and Xin Liu. Triton-distributed: Programming overlapping kernels on distributed AI systems with the Triton compiler. *arXiv preprint arXiv:2504.19442*, April 2025. URL https://arxiv.org/abs/2504.19442.

Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Tian Tang, Qinyu Xu, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Ziren Wang, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. NanoFlow: Towards optimal large language model serving throughput. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation*, pp. 749–765, Boston, Massachusetts, USA, July 2025. USENIX Association. ISBN 978-1-939133-47-2. URL https://www.usenix.org/conference/osdi25/presentation/zhu-kan.

APPENDIX

A SENSITIVITY TO EP DEGREE AND LOCAL TOKEN COUNT

This section extends Table 1 in Section 4.1. Below, Table 2 reports MXFP8 forward and backward latency and speedup for EP degrees 16 and 64 across the four open-weight model shapes and 1,024, 2,048, and 4,096 local tokens per GPU.

Table 2: MoK sensitivity to EP degree and local token count. "Tokens" is local tokens per GPU before dispatch. "Best" is the lowest latency among the four baselines per workload, with the subscript identifying the best baseline (N: NCCL+PyTorch, D: DeepEP+PyTorch, H: HybridEP+Megatron). "Speedup" is "Best" divided by "MoK". Lower latency and higher speedup are better.

			Forward (ms)		Backward (ms)		Speedup	
EP	MoE Shape	Tokens	MoK	Best	MoK	Best	Fwd	Bwd
16	Kimi K2.7 Code	1,024	0.901	2.334_{N}	1.786	2.413_{N}	**2.59**×	**1.35**×
		2,048	1.332	2.581_{N}	2.425	3.329_{N}	**1.94**×	**1.37**×
		4,096	2.555	3.308_{H}	4.320	5.053_{H}	**1.29**×	**1.17**×
	GLM-5.2	1,024	0.849	2.321_{N}	1.284	2.070_{N}	**2.73**×	**1.61**×
		2,048	1.180	2.367_{N}	1.965	2.762_{N}	**2.01**×	**1.41**×
		4,096	2.051	2.913_{H}	3.514	4.147_{H}	**1.42**×	**1.18**×
	Qwen3.5-397B-A17B	1,024	0.857	2.316_{N}	1.440	1.915_{N}	**2.70**×	**1.33**×
		2,048	0.901	2.307_{N}	1.296	2.129_{N}	**2.56**×	**1.64**×
		4,096	1.705	2.777_{N}	2.180	3.005_{H}	**1.63**×	**1.38**×
	DeepSeek-V4-Pro	1,024	0.992	2.319_{N}	2.321	2.571_{N}	**2.34**×	**1.11**×
		2,048	1.560	2.612_{N}	3.139	3.343_{N}	**1.67**×	**1.06**×
		4,096	2.762	3.555_{H}	5.157	5.462_{D}	**1.29**×	**1.06**×
64	Kimi K2.7 Code	1,024	1.016	2.600_{N}	1.555	2.295_{H}	**2.56**×	**1.48**×
		2,048	1.293	2.570_{H}	2.045	2.636_{H}	**1.99**×	**1.29**×
		4,096	2.524	4.002_{H}	3.871	4.709_{H}	**1.59**×	**1.22**×
	GLM-5.2	1,024	0.989	2.706_{H}	1.612	2.287_{H}	**2.74**×	**1.42**×
		2,048	1.142	2.471_{H}	1.737	2.414_{H}	**2.16**×	**1.39**×
		4,096	2.171	3.354_{H}	3.294	4.147_{H}	**1.54**×	**1.26**×
	Qwen3.5-397B-A17B	1,024	0.995	2.567_{N}	1.442	2.348_{H}	**2.58**×	**1.63**×
		2,048	1.068	2.527_{H}	1.273	2.267_{H}	**2.37**×	**1.78**×
		4,096	1.830	2.978_{H}	2.137	2.959_{H}	**1.63**×	**1.38**×
	DeepSeek-V4-Pro	1,024	0.981	2.670_{N}	1.483	2.470_{H}	**2.72**×	**1.67**×
		2,048	1.392	2.564_{H}	2.321	2.839_{H}	**1.84**×	**1.22**×
		4,096	2.594	3.796_{N}	4.346	5.112_{H}	**1.46**×	**1.18**×

B DISPATCH SCHEDULE CONSTRUCTION

This section details how MoK constructs the pull-based dispatch schedule table described in Section 3.1. For the current rank c, Algorithm 1 counts the routes from each source rank to each local expert (M), computes the start of each expert's contiguous region in the destination buffer (S), and then places every route destined for rank c at the position given by its per-source-rank ordinal o and

source rank r within that region. Push-based combine and the two backward operators reuse the same table by reinterpreting its columns.

Algorithm 1 Pull-based dispatch schedule construction

Require: R ranks, each with N tokens and L experts; top-K routes $A \in \{0, \ldots, RL-1\}^{R \times NK}$, where $A_{r,j}$ is route j's expert on rank r; current rank c; local expert ℓ is global expert $cL + \ell$
Ensure: schedule Φ, where $\Phi_p = (r, j)$ means local buffer position p receives route j from rank r

$M_{r,\ell} \leftarrow \left|\{\, j : A_{r,j} = cL + \ell \,\}\right| \quad \forall\, r, \ell$ ▷ route count per (source rank, local expert)
$S_\ell \leftarrow \sum_{\ell' < \ell} \sum_r M_{r,\ell'} \quad \forall\, \ell$ ▷ start of expert ℓ's buffer region
for all local expert ℓ and routes (r, j) with $A_{r,j} = cL + \ell$ **do**
  $o \leftarrow \left|\{\, j' < j : A_{r,j'} = A_{r,j} \,\}\right|$
  $p \leftarrow \sum_{r'=0}^{R-1} \min(M_{r',\ell},\, o) + \left|\{\, r' < r : M_{r',\ell} > o \,\}\right|$
  $\Phi_{S_\ell + p} \leftarrow (r,\ j)$
end for

C Scheduling Overhead

This section compares the scheduling overhead of MoK and Comet (Zhang et al., 2025). In MoK, scheduling involves the all-gather of the local top-k routes followed by the dispatch schedule construction described in Appendix B. In Comet, scheduling involves the all-gather of the local top-k routes followed by the per-expert token count and scatter index computation, the source-rank sort of the routes, and the host-side GEMM tile scheduling.

We measure BF16 forward passes at EP degree 4 with the workloads, inputs, and local token counts used in Section 4.1, timing 100 iterations after 500 warmup iterations and reporting the median slowest-rank latency. Absolute times are not comparable because MoK requires Blackwell GPUs and Comet supports GPUs only up to Hopper. Thus, we compare each system's relative scheduling overhead. Because the forward pass is faster on MoK, this comparison is biased *against* MoK and favors Comet. Table 3 reports the results.

Table 3: Scheduling overhead of MoK and Comet for BF16 forward passes at EP degree 4. "Tokens" is local tokens per GPU before dispatch. "Sched." is the time spent building the schedule before dispatch begins, "Total" is "Sched." plus the forward latency, both in ms, and "Share" is "Sched." divided by "Total".

		MoK (GB300)			Comet (H100)		
MoE Shape	Tokens	Sched.	Total	Share	Sched.	Total	Share
Kimi K2.7 Code	1,024	0.068	2.250	**3.04%**	0.331	4.892	**6.76%**
	2,048	0.093	2.571	**3.60%**	0.389	5.640	**6.89%**
	4,096	0.133	4.507	**2.96%**	0.515	10.537	**4.89%**
GLM-5.2	1,024	0.064	1.411	**4.50%**	0.294	3.268	**9.00%**
	2,048	0.077	2.050	**3.74%**	0.349	4.774	**7.30%**
	4,096	0.107	3.360	**3.20%**	0.439	7.751	**5.67%**
Qwen3.5-397B-A17B	1,024	0.076	1.228	**6.20%**	0.321	2.630	**12.22%**
	2,048	0.101	1.316	**7.65%**	0.387	2.970	**13.03%**
	4,096	0.156	2.342	**6.66%**	0.473	5.165	**9.16%**
DeepSeek-V4-Pro	1,024	0.068	3.195	**2.14%**	0.325	6.878	**4.72%**
	2,048	0.087	3.468	**2.50%**	0.361	7.614	**4.75%**
	4,096	0.115	5.201	**2.22%**	0.504	11.927	**4.23%**

D Additional Features and Optimizations

This section describes the additional features and optimizations, beyond the three design choices in Section 3, that MoK implements for production MoE training.

Determinism. To enable training ablations and on-policy reinforcement learning, MoK is fully deterministic. The order of floating-point operations is fixed across executions, so the same input produces bitwise-identical output regardless of hardware scheduling. The main source of nondeterminism in GPUs is atomic operations. MoK either forces atomics to the same address to execute in a fixed order through software scheduling, or replaces them with writes to a temporary buffer that another worker reduces in a deterministic order.

RDMA Overlap via Cluster Launch Control. MoK schedules its tasks through Cluster Launch Control (CLC) (Nvidia, 2026a), a hardware-native work-stealing mechanism introduced in Nvidia Blackwell. A CLC-based kernel can naturally yield SMs to kernels on a higher-priority stream without having to fully complete. This matters because, during training, we often need to overlap intra-rack communication and computation with inter-rack communication, such as FSDP all-gathers, which needs SMs of its own. Without CLC, that inter-rack communication would serialize behind the megakernel, causing performance degradation.

MXFP8 Support. For better hardware utilization, MoK supports both BF16 and MXFP8 (Rouhani et al., 2023) execution. In MXFP8 mode, the shared expert FFN stays in BF16 for training stability. One added cost of MXFP8 is that tensors must be quantized before they reach the tensor cores. To minimize this overhead, MoK (1) provides an optimized MXFP8 quantization kernel for pre-quantizing the weights, and (2) fuses activation quantization into the dispatch, the expert-grouped GEMMs, and the gated activation computation.

Router Weight Gradient Computation. The MoK backward pass also computes the router weight gradients, using the SonicMoE-style calculation (Guo et al., 2026). Rather than materializing the full down-projection output tensor, MoK computes the router weight gradients from the inner product of the gated activation and the down-projection activation gradient. This computation is fused into the gated activation backward to avoid additional global memory traffic.

Shared Expert. Many open-weight MoE architectures include a shared expert that every token passes through regardless of routing. The shared expert's load is uniform across EP ranks and it does not depend on dispatch or combine. Therefore, MoK fits the shared expert FFN in the first dispatch window, where the computation SMs would otherwise sit idle.

Computation-Communication SM Tuning. The optimal split between computation and communication SMs depends on the workload (Sul et al., 2026). Local token count, model shape, and hardware compute and network bandwidth all change how long dispatch and combine take relative to the expert FFN tasks. Even within the same workload, the optimum differs between the forward and backward passes. MoK therefore exposes the number of communication SMs as a tunable parameter separately for the forward and backward passes.