X-MoE-4D: Breaking the Scaling Barrier for Trillion Parameter MoEs on Hierarchical HPC Systems

Zixian Wang
University of Illinois Urbana-Champaign
Urbana, USA
zixianw4@illinois.edu

Liheng Jing
University of Illinois Urbana-Champaign
Urbana, USA
lihengj2@illinois.edu

Emily Herron
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
herronej@ornl.gov

Sajal Dash
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
dashs@ornl.gov

Feiyi Wang
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
fwang2@ornl.gov

Minjia Zhang
University of Illinois Urbana-Champaign
Urbana, USA
minjiaz@illinois.edu

***Abstract*—We present X-MoE-4D, a distributed training system for scaling expert-specialized Mixture-of-Experts models on hierarchical HPC platforms. We observe that existing expert-parallel approaches suffer from severe scalability limits due to cross-node all-to-all communication. To address this, X-MoE-4D adopts a pipeline-global, expert-local execution model that confines expert computation within nodes while scaling across nodes via pipeline parallelism. X-MoE-4D further introduces a Minimax Pipeline Planner that jointly optimizes pipeline partitioning and activation checkpointing to minimize bottleneck stage execution time under memory constraints. Together, these techniques eliminate cross-node expert routing and mitigate MoE Pipeline Skew. We evaluate X-MoE-4D on the Frontier supercomputer, demonstrating the first efficient 1-trillion-parameter expert-specialized MoE training on an AMD-based HPC platform. X-MoE-4D achieves 2.24× higher aggregate throughput than X-MoE at 1-trillion-parameter scale (2.4× per GPU throughput) while using fewer GPUs (960 vs. 1,024), and sustains a 3.24× speedup when strong-scaling the 173B model to 2,304 GPUs. Code available at https://github.com/Supercomputing-System-AI-Lab/X-MoE.**

I. Introduction

Mixture-of-Experts (MoE) has emerged as a powerful approach for scaling large language models to unprecedented model sizes by activating only a subset of experts per token, allowing the capacity of the model to grow without proportional increase in computation [1], [2]. As a result, MoE models are increasingly adopted in large-scale AI systems [3]–[5], as well as emerging AI4Science models (e.g., SciDFM [6]). However, efficiently training trillion-parameter MoE models on modern HPC systems introduces significant system challenges, particularly due to the communication and memory constraints inherent in distributed MoE training. To address this, existing systems primarily rely on expert parallelism (EP), which distributes experts across devices and performs token dispatch via *all-to-all* communication.

While effective for earlier MoE models with a small number of large experts [7]–[10], this approach is increasingly constrained by recent architectural trends. Modern MoE designs, such as Kimi K3 [11], DeepSeek-V4 [12], Qwen3-Next [5], and Llama4 [4], have shifted toward fine-grained architectures with large top-k routing, which increase the number of experts significantly while reducing per-expert size to improve expert specialization. This shift substantially increases routing-related activation memory that scales with both the number of experts and the top-k routing degree. This can dominate per-layer memory footprint and leads to highly uneven memory and computation costs across layers.

To address the resulting communication and memory pressure, recent systems such as X-MoE [13] improve memory efficiency and reduce redundant communication through topology-aware routing and hierarchical dispatch, enabling training of large expert-specialized models on HPC platforms such as Frontier [14]. However, these optimizations do not fully eliminate the scaling bottlenecks. For example, X-MoE achieves only around 10.4% model FLOPs utilization (MFU) at the trillion-parameter scale, indicating diminishing efficiency at massive scale.

The root cause of this limitation lies in a deeper mismatch between modern MoE architectures and existing distributed training paradigms. Under expert parallelism (EP), tokens are dynamically routed to remote experts and gathered back across devices, requiring *all-to-all* communication among participating GPUs. As the number of experts and devices increases, this communication pattern incurs rapidly growing cross-node traffic and becomes increasingly constrained by the hierarchical bandwidth of HPC interconnects (e.g., Dragonfly [15]). Consequently, communication overhead grows disproportionately with scale and begins to dominate execution time.

A natural alternative is to reduce cross-node communication

SC26, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-4789-7/26/$31.00 ©2026 IEEE

by localizing expert computation within nodes and scaling across nodes via pipeline parallelism. However, this design is not directly feasible under existing MoE training systems. Expert-specialized MoEs incur substantial activation and routing overhead, making it difficult to host a full MoE layer's activations within a single node under standard execution, particularly at large and trillion-parameter scales. As a result, prior systems rely on expert parallelism to distribute memory footprint across devices. Moreover, simply adopting pipeline parallelism is insufficient. Unlike dense transformer models, MoE layers exhibit both significantly larger and highly uneven activation memory, as well as imbalanced compute costs due to routing and fine-grained expert execution. When combined with pipeline execution, these effects induce a new scaling pathology, which we term *MoE Pipeline Skew (MPS)*: fine-grained experts amplify activation memory and introduce imbalanced compute costs, while pipeline execution prolongs activation lifetimes in early stages. Together, this creates a strong lifetime asymmetry that magnifies layer-wise differences, disproportionately inflating the cost of early pipeline stages and making them the bottleneck for end-to-end training.

To address this challenge, we present X-MoE-4D, a distributed training system for scalable expert-specialized MoE training on hierarchical HPC clusters. X-MoE-4D combines a topology-aligned execution paradigm with a skew-aware pipeline planning strategy to address both communication and imbalance bottlenecks. At the core of X-MoE-4D is *Expert-Localized Minimax Pipeline Parallelism (ELM-PP)*, a skew-aware pipeline planning formulation that directly targets the bottleneck stage induced by MPS. ELM-PP jointly optimizes stage partitioning and activation checkpoint placement to minimize the execution time of the slowest stage under per-GPU memory constraints. Unlike prior approaches, ELM-PP jointly optimizes stage partitioning and checkpointing to directly minimize bottleneck stage latency. It transforms deep, skewed pipelines into efficient execution schedules, enabling scaling of MoE training without incurring cross-node communication overheads.

We implement X-MoE-4D on the Frontier supercomputer and evaluate it at scales comparable to prior MoE training systems. At the 537B-parameter scale, the largest MoE workload previously demonstrated on Frontier, X-MoE-4D achieves 35.4% MFU, delivering a 3.03× improvement over X-MoE, a strong AMD-based baseline for large-scale MoE training. We further scale X-MoE-4D to a 1-trillion parameter expert-specialized MoE, demonstrating, to our knowledge, *the first efficient trillion-scale expert-specialized MoE training on an AMD-based GPU platform*. Notably, X-MoE-4D trains the 1T model using 960 MI250X GPUs (GCDs), achieving 24.8% MFU and 2.24× higher aggregate training throughput (2.4× per GPU throughput) than X-MoE at the trillion-parameter scale. When strong-scaling the 173B model to 2,304 GPUs, X-MoE-4D continues to outperform EP-centric approaches, achieving up to 3.24× higher throughput than X-MoE. This paper makes the following contributions:

1) *MoE Pipeline Skew: a new scaling pathology.* We identify MPS, a previously overlooked source of inefficiency in pipeline-parallel MoE training, caused by the interaction of amplified activation memory, fine-grained expert computation, and pipeline execution, which breaks the assumption underlying existing pipeline parallelism.
2) *A topology-aligned execution and planning paradigm.* We propose pipeline-global, expert-local execution together with Expert-Localized Minimax Pipeline Parallelism, which jointly optimizes stage partitioning and selective checkpointing to eliminate cross-node expert routing and minimize bottleneck stage latency.
3) *Trillion-scale MoE training on hierarchical HPC systems.* We demonstrate efficient training up to 1T parameters model and up to 2,304 AMD GPUs, achieving up to 3.24× higher throughput than prior EP-centric systems.

II. Background

Training expert-specialized MoE models at trillion-parameter scale on modern HPC systems introduces significant system challenges, driven by the interaction between model architecture and hierarchical hardware constraints.

A. Distributed MoE Training Systems

To scale Mixture-of-Experts (MoE) models, modern frameworks primarily rely on expert parallelism (EP), which distributes experts across devices and routes tokens via *all-to-all* communication [16]–[23]. GShard [7] establishes EP as a fundamental scaling strategy for MoEs. Subsequent systems improve its efficiency through better memory and parallelism management. DeepSpeed-MoE [9] integrates ZeRO-DP [24] to reduce optimizer and parameter memory, while DeepSpeed-TED [25] combines tensor parallelism with EP to balance computation and communication. Tutel [10] further reduces overhead by dynamically switching between EP and hybrid TP+EP configurations. MegaBlocks [21] reformulates expert computation as block-sparse operations to improve GPU utilization. Recent work explores scaling MoE training through multi-dimensional parallelism. Megatron-Core [26], [27] and MegaScale-MoE [28] extend parallelism to improve scalability, but are tightly coupled to specific software and hardware stacks (e.g., NVIDIA GPUs). In contrast, X-MoE [13] targets AMD-based HPC platforms, optimizing memory by using ERI-based sparse data layouts to eliminate the expensive routing mask and employing a topology-aware pilot mechanism to route minimal tokens inter-node at scale. Despite these advances, existing MoE training systems primarily rely on *all-to-all* based EP for scaling. This communication pattern maps poorly to HPC platforms with hierarchical network topology, where cross-node bandwidth is limited and non-uniform. As model scale and expert count increase, *all-to-all* traffic grows rapidly and becomes a dominant bottleneck, limiting scalability at large cluster sizes.

B. Pipeline Parallelism for Distributed Training

Pipeline parallelism (PP) avoids expensive global collectives entirely. It partitions model layers into sequential stages and

restricts communication to point-to-point transfers between adjacent stages. GPipe [29] uses a memory-heavy all-forward-all-backward (AFAB) schedule, whereas PipeDream [30] introduces 1F1B to interleave forward and backward passes, reducing the number of in-flight micro-batches and lowering peak activation memory. Subsequent works further minimize pipeline bubbles (interleaved 1f1b [31], Zero-Bubble [32]), overlap communication via bidirectional execution (Chimera [33], DualPipe [3]), or support graph-structured dependencies (GraphPipe [34]). Methods like BPipe [35] and Mario [36] target memory imbalance and intra-stage compute, respectively, but suffer from communication overhead [37] or fail to resolve stage-level imbalance. Crucially, prior works target dense transformers with uniform layers, where model states dominate memory costs. In contrast, we address MoE Pipeline Skew in expert-specialized MoEs on HPC systems, where routing-induced activations become the dominant bottleneck and cause severe stage-level imbalance.

C. Hierarchical HPC Network Constraints

Modern HPC systems are built on hierarchical network topologies (e.g., Dragonfly [15]), which combines fast intra-node communication and bandwidth-constrained inter-node links across switches and groups. This hierarchy creates significant variation in communication cost depending on distance and placement. Consequently, the global *all-to-all* required by expert parallelism suffers from severe cross-node bottlenecks and network contention at scale. These effects significantly limit MoE scalability. We use the Frontier supercomputer [14] as a representative system to evaluate these constraints.

III. Observations and Opportunities

To understand the challenges of scaling expert-specialized MoE training on hierarchical HPC systems, we examine the scaling limitations of expert parallelism. Our analysis reveals a systematic MoE Pipeline Skew (MPS) phenomenon.

A. Motivation: Scaling Limitations of Expert Parallelism

Expert parallelism (EP) is the dominant approach for scaling MoE training by distributing experts across devices and routing tokens via *all-to-all* communication [7], [9], [10], [13], [25]. While effective on systems with high-bandwidth and relatively uniform interconnects [38], it becomes increasingly inefficient on hierarchical HPC networks (e.g., Dragonfly [15]) when scaling toward trillion-parameter MoE models, where global *all-to-all* communication must traverse bandwidth-constrained cross-node links and quickly dominate execution time.

To quantify this effect, we benchmark a single MoE layer with 1024 experts, matching the hidden dimension and expert sizes of the trillion-parameter model in Table I. We fix the micro-batch size per GPU and scale EP from 16 to 512 GPUs. Fig. 1 shows that, as EP scales, the latency of the MoE layer increases substantially, driven primarily by the growth in dispatch and combine *all-to-all* communication. Starting at EP=128, the dispatch and combine A2A time

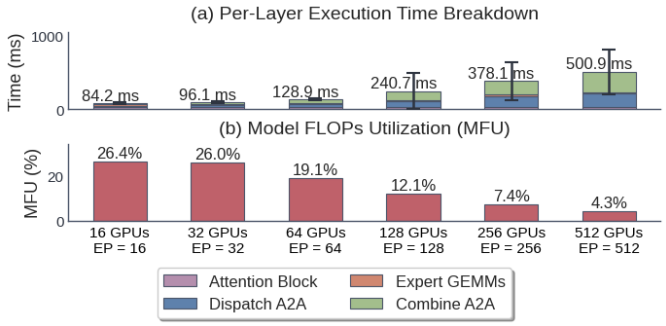


Fig. 1. **Impact of scaling expert parallelism (EP).** As EP increases, dispatch and combine *all-to-all* communication becomes a dominant component of MoE execution time, reducing MFU and increasing runtime variability.

dominate overall execution. Continuing through larger scales (EP=256–512), the *all-to-all* communication accounts for over 81.8%–90.8% of per-layer execution time. As a result, model FLOPs utilization (MFU) drops from 26.4% at 16 GPUs to 4.3% at 512 GPUs. Moreover, runtime variability increases significantly with scale. The per-layer execution time exhibits a standard deviation of 244.20 ms (mean: 241.58 ms, std/mean: 1.01) at EP=128 and 300.45 ms (mean: 500.90 ms, std/mean: 0.60) at EP=512, indicating highly variable cross-node A2A communication. This variability is consistent with observations in prior large-scale MoE training systems on HPC platforms [13].

Insight 1: Scaling expert parallelism on hierarchical HPC systems leads to rapidly increasing all-to-all overhead and variability, becoming an execution bottleneck that ultimately limits trillion-parameter MoEs scalability on large clusters.

B. Memory Imbalance in Expert-Specialized MoE

The communication bottleneck in EP motivates the use of pipeline parallelism (PP) for cross-node scaling. However, PP often assumes that model layers can be partitioned into stages with the same computational and roughly balanced memory costs [30], [31]. This assumption largely holds for dense transformer models, where each layer requires the same amount of computation and relatively small activation consumption. Expert-specialized MoEs break this assumption. They employ a large number of fine-grained experts with large top-k routing [3], significantly amplifying activation memory due to routing buffers and aggregated expert activations. As a result, MoE layers exhibit substantially larger memory footprints than surrounding attention layers, leading to highly uneven resource demands across layers.

To quantify this effect, we profile the per-stage memory consumption under pipeline parallelism across three configurations: (1) a dense transformer model, (2) a coarse-grained MoE with a small number of large experts, and (3) an expert-specialized MoE with many fine-grained experts. All configurations share the same base architecture and total FLOPs to ensure a fair comparison. Fig. 2 shows that expert-specialized MoEs exhibit significantly higher variance in per-stage memory consumption. Compared to the dense transformer baseline,

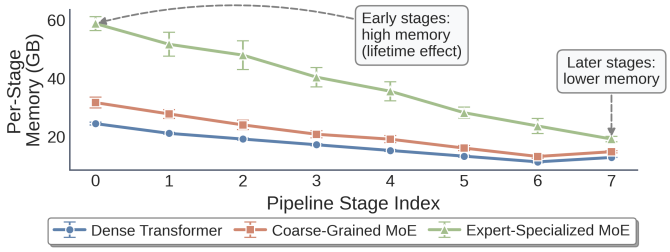


Fig. 2. **Memory imbalance across pipeline stages**. Expert-specialized MoE exhibits much larger memory imbalance than dense and coarse-grained MoE.

the expert-specialized MoE exhibits substantially higher stage-level memory imbalance, with the gap between the earliest and latest pipeline stages reaching about 40 GB, versus about 12 GB for the dense model. Compared to classical MoEs, this fine-grained scaling significantly increases aggregated expert activations and routing buffer overhead, resulting in a highly uneven memory distribution across pipeline stages.

Insight 2: Expert-specialized MoE layers consume significantly more activation memory than surrounding attention layers due to routing buffer and aggregated expert activations, breaking the uniformity assumption of pipeline parallelism and creating imbalanced pipeline stages.

C. Increasing Pipeline Depth Amplifies Stage Imbalance

Building on the memory imbalance introduced by expert-specialized MoE layers (§ III-B), increasing pipeline depth further amplifies stage-level imbalance through activation lifetime asymmetry. Under pipeline execution (e.g., 1F1B [31]), activations produced during the forward pass must be retained until their corresponding backward pass completes. As a result, in a pipeline with p stages, stage s holds activations for approximately $(p - s)$ in-flight micro-batches, causing earlier stages to retain significantly more activations than later stages. This creates a *lifetime asymmetry* across pipeline stages: the memory footprint of early stages grows proportionally with pipeline depth, while later stages experience significantly lower pressure. When combined with the already high activation cost of MoE layers, this effect disproportionately inflates the memory consumption of early stages, making them the primary bottleneck in deep pipelines.

To quantify this effect, we measure per-stage memory under different pipeline depths using the same expert-specialized MoE configuration as in § III-B. We vary pipeline depth (PP = 4, 6, 8) while keeping the number of layers per stage fixed ($L_{\text{stage}} = 4$) to isolate the impact of pipeline depth. Fig. 3 shows that as pipeline depth increases, memory consumption in early stages grows significantly, while later stages remain largely unchanged. This behavior follows the $(p-s)$ activation lifetime pattern: increasing pipeline depth increases the number of in-flight micro-batches retained by early stages, disproportionately inflating their memory footprint. As a result, the gap between early and late stages widens with pipeline depth, making balanced pipeline execution increasingly difficult.

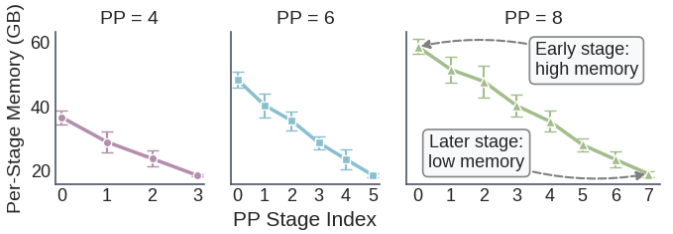


Fig. 3. **Memory amplification with increasing pipeline depth.** Under fixed layers per stage, increasing pipeline depth enlarges activation lifetimes, causing early stages to consume more memory and amplifying stage-level imbalance. All results use EP = 8.

Insight 3: Increasing pipeline depth amplifies stage-level imbalance through activation lifetime asymmetry, disproportionately increasing the memory footprint of early pipeline stages and making them the bottleneck in deep pipelines.

D. Fragmented Computation and Utilization Variability

In addition to memory imbalance, expert-specialized MoEs introduce inefficiencies in computation that further exacerbate stage imbalance. Fine-grained MoEs increase the number of experts while reducing the dimension of each expert, resulting in smaller and more fragmented matrix multiplications. While this preserves theoretical FLOPs, it reduces hardware efficiency in practice. Smaller expert dimensions lead to underutilized GPU compute units, and the increased number of experts introduces additional kernel launch overhead. As a result, the effective execution time of MoE layers becomes significantly higher than predicted by theoretical FLOPs.

To understand the impact on hardware efficiency, we measure the achieved TFLOPS of expert GEMM kernels by sweeping the expert dimension (H_{exp}) and routed tokens per expert (C) under a fixed configuration (model dimension D=4096, sequence length S=4096, and intra-node EP=8). This produces a normalized efficiency landscape that isolates the effect of expert granularity on kernel performance. Fig. 4 shows the resulting efficiency landscape. We annotate representative configurations inspired by recent MoE architectures [8], [39]–[46], based on their expert dimension H_{exp} and routing patterns (which determine C). Note that these annotations only approximately correspond to the actual GEMM shapes in those models, as real models may use different model dimension (D). The upper-right region corresponds to coarse-grained expert with large H_{exp} and C, achieving high TFLOPS, while the lower-left region corresponds to the fine-grained MoEs, where smaller GEMMs lead to significantly reduced efficiency. This inefficiency translates into higher execution time for MoE layers. For example, in a representative 63B model configuration (Table I), fine-grained expert settings incur higher per-layer execution time than coarse-grained counterparts (e.g., around a 30% increase), despite similar theoretical FLOPs. This gap arises from reduced hardware efficiency in fine-grained MoE: smaller expert dimensions lead to underutilized GPU compute units, while the increased number of experts introduces fragmented execution and higher kernel launch overhead. Under pipeline-based scaling with

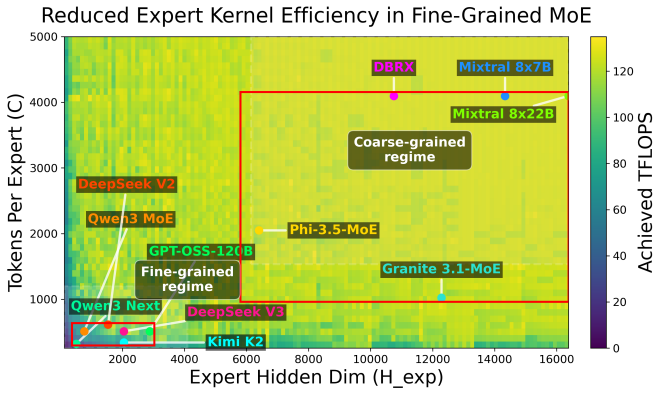


Fig. 4. **Reduced expert kernel efficiency in fine-grained MoE.** Expert GEMM utilization is high for large expert dimensions and tokens per expert (upper right), but drops significantly for fine-grained experts (lower left), indicating reduced hardware efficiency.

localized expert placement, more experts are co-located on each GPU, further amplifying these effects.

Insight 4: Fine-grained expert specialization reduces hardware efficiency and increases effective computation cost, disproportionately slowing MoE-heavy pipeline stages and further amplifying stage-level imbalance.

Summary of Observations. These observations reveal that the inefficiency of expert-specialized MoE training is not solely due to communication overhead, but arises from a structural imbalance introduced by the interaction of MoE architecture and pipeline execution. This phenomenon, which we term *MoE Pipeline Skew*, limits conventional pipeline parallelism just as cross-node *all-to-all* limits expert parallelism (§ III-A). Consequently, addressing MPS requires jointly optimizing how MoE computation is mapped to hardware and how pipeline execution is scheduled.

IV. System Design

Building on the observations in § III, X-MoE-4D removes the cross-node all-to-all bottleneck through a topology-aligned execution model, mitigates the MoE Pipeline Skew through a skew-aware planning strategy, and recovers the kernel efficiency that expert co-location exposes. As illustrated in Fig. 5, the system consists of three key components: (i) *Pipeline-global, expert-local execution*, which keep expert routing within nodes while scaling across nodes using pipeline parallelism (§ IV-A); (ii) *Minimax pipeline planning (ELM-PP)*, which minimizes bottleneck stage execution time under GPU memory constraints through joint stage partitioning and checkpointing (§ IV-B); (iii) *Efficient expert-local execution*, which improves utilization by mitigating kernel fragmentation under localized experts (§ IV-C).

A. Pipeline-Global, Expert-Local Execution

Existing MoE training systems rely on expert parallelism (EP) [9], [10], [13], [25], which incurs significant cross-node communication on hierarchical HPC systems and limits scalability. To eliminate this bottleneck, X-MoE-4D adopts

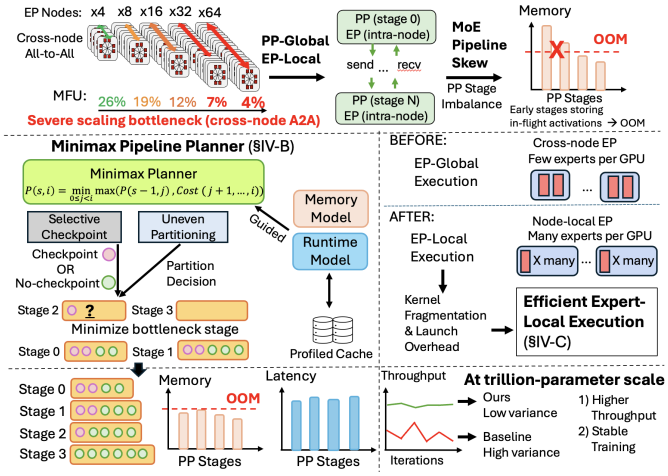


Fig. 5. **Overview of X-MoE-4D.** EP-based scaling (top-left) suffers from cross-node all-to-all bottlenecks. X-MoE-4D adopts pipeline-global, expert-local execution (top-right) to eliminate cross-node routing, but introduces MoE Pipeline Skew (MPS) due to stage imbalance. ELM-PP (left) mitigates MPS by balancing pipeline stages via joint partitioning and checkpointing. Efficient expert-local execution (right) improves utilization by reducing kernel fragmentation. Together, these components enable stable, high-throughput training of trillion-parameter MoEs on hierarchical HPC systems.

a *pipeline-global, expert-local* execution paradigm. Expert parallelism is kept within each node, ensuring that all expert routing occurs over high-bandwidth intra-node interconnects, while cross-node scaling is achieved through pipeline parallelism.

This design aligns MoE execution with the underlying hardware topology and eliminates cross-node expert routing. This design is feasible because the aggregate memory within a node is often sufficient to host a full MoE layer even at trillion scale (e.g., 512 GB across 8 GCDs on Frontier). However, this shift fundamentally changes how the model is scaled. Supporting trillion-parameter models now requires deep pipelines across nodes, whose stages hold MoE layers of markedly uneven compute cost and memory. This unevenness motivates the skew-aware pipeline planning strategy described next.

Choice of Pipeline Schedule. While recent schedulers aim to reduce pipeline bubbles, they introduce trade-offs that are problematic for memory-constrained MoE training. Interleaved 1F1B [31] increases peak activation memory due to additional virtual stages; DualPipe [3], [33] doubles parameter memory and expands the activation footprint across stages; and ZB1P [32] preserves the activation memory profile of 1F1B but interleaves forward, backward, and update phases more tightly, making checkpointing decisions no longer independent across stages. Therefore, X-MoE-4D builds on the 1F1B schedule as the execution backbone, leveraging two key properties. First, 1F1B induces a structured activation footprint. Under 1F1B, stage p_{id} retains activations for $(p - p_{\text{id}})$ in-flight micro-batches, yielding a peak activation memory of $(p - p_{\text{id}})\, A_{\text{mbs}}$. This creates a monotonically descending per-stage peak activation memory, where earlier stages incur significantly higher memory pressure than later ones. This *predictable asymmetry* enables effective balancing of mem-

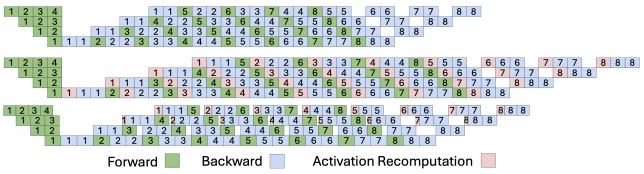


Fig. 6. **1F1B scheduling and checkpointing under pipeline execution.** Top: standard 1F1B scheduling. Middle: uniform checkpointing reduces memory but introduces recomputation overhead without addressing stage imbalance. Bottom: skew-aware planning (ELM-PP) jointly optimizes checkpointing and partitioning to balance stage memory and execution time.

ory and execution across stages. Second, its execution order provides well-defined stage-local behavior, allowing memory and recomputation decisions to be applied per stage without introducing complex cross-stage interactions. These properties make 1F1B well suited for pipeline planning under memory constraints. Fig. 6 illustrates how 1F1B scheduling induces activation lifetime asymmetry and how different checkpointing strategies interact with pipeline execution, motivating the need for skew-aware planning.

B. Expert-Localized Minimax Pipeline Planning (ELM-PP)

Under MoE Pipeline Skew, existing pipeline parallelism (PP) strategies and activation checkpointing (AC) techniques fall short for different reasons. PP schedules, designed primarily for dense transformer models with relatively uniform layer structure and modest activation memory [31], [36], [47], do not account for MoE-specific characteristics such as large routing-induced activation memory and fragmented expert computation, leading to imbalanced pipeline stages. AC techniques, originally designed for single-device execution, optimize memory–compute trade-offs without considering pipeline structure [48]–[51]. When applied directly to pipeline stages, they do not ensure globally balanced execution. As a result, combining PP and AC without coordination fails to eliminate bottleneck stages under MPS.

However, coordinating these factors is non-trivial, as they are tightly coupled. Stage memory depends on both layer assignment and checkpointing decisions, while partitioning determines how compute and memory are distributed across stages. Reducing memory via recomputation increases execution time, whereas redistributing layers changes both memory footprint and compute balance.

To jointly optimize partitioning and checkpointing under MPS, X-MoE-4D formulates pipeline planning as a minimax optimization problem that minimizes the execution time of the bottleneck stage under per-device memory constraints. In this formulation, stage cost is determined by memory usage and recomputation, while partitioning determines how this cost is distributed across stages, enabling balanced compute and memory across the pipeline. This optimization is combinatorial, as it requires deciding both layer partitioning across stages and compute–memory trade-offs within each stage. To solve it efficiently, X-MoE-4D adopts a dynamic programming formulation that exploits the optimal substructure of pipeline partitioning to minimize bottleneck stage latency.

Let $P(s, i)$ denote the minimum possible execution time of the bottleneck stage when partitioning the first i layers across s pipeline stages. The key insight is that an optimal solution, adhering to the Bellman Optimality Principle [52], exhibits optimal substructure: an optimal partition of the first i layers across s stages must consist of an optimal partition of a prefix of layers across the first $s-1$ stages, followed by a final stage containing the remaining layers.

To construct the optimal partition, the planner considers all possible split points $j < i$, where layers $1{:}j$ are assigned to the first $s-1$ stages and layers $j+1{:}i$ form the s-th stage. The objective then is to minimize the maximum execution time across all stages under per-GPU memory constraints:

$$P(s,i) = \min_{0 \le j < i} \max\left(P(s-1,j), \text{Cost}(j+1,\ldots,i)\right) \quad (1)$$

Here, j is the split point, which is the index of the last layer of $s-1$ stage. $\text{Cost}(j+1,\ldots,i)$ represents the execution time of a stage containing layers $j+1$ to i, taking into account both its memory usage and compute cost. Memory feasibility is enforced through $\text{Cost}(\cdot)$.

To evaluate the cost, the planner determines the optimal compute–memory trade-off within the stage. Let a stage contain l layers, of which c are recomputed. Each recomputed layer reduces memory usage but increases execution time. The cost of a stage is therefore defined as:

$$\text{Cost}(l) = \min_{\substack{0 \le c \le l \\ \text{Mem}(l,c) \le M_{\text{limit}}}} \{(l-c)\, t_{\text{uc}} + c\, t_{\text{c}}\} \quad (2)$$

where t_{uc} and t_{c} denote the per-layer execution time without and with recomputation, respectively, and $\text{Mem}(l,c)$ models the peak memory usage of the stage. If no checkpointing configuration satisfies the per-GPU memory constraint for layers $j+1{:}i$, the corresponding stage cost is set to infinity.

This formulation jointly optimizes cross-stage workload distribution and stage-level compute–memory trade-offs. The recurrence in Eq. 1 balances the execution time across stages by minimizing the bottleneck stage latency, while the cost function in Eq. 2 captures stage-specific memory usage and recomputation decisions under per-GPU memory constraints. Infeasible configurations that violate the memory constraint are excluded by assigning infinite cost. Together, these components enable the planner to systematically explore the design space and identify a balanced, memory-feasible pipeline configuration that mitigates MPS.

The effectiveness of this formulation depends on accurately modeling the cost of each pipeline stage. In X-MoE-4D, stage cost is determined by two complementary dimensions: (i) stage-level compute–memory trade-offs and (ii) cross-stage workload distribution. We instantiate these through selective checkpointing and uneven partitioning, respectively.

1) Stage-Level Memory Control via Selective Checkpointing. Instead of uniform checkpointing, X-MoE-4D applies checkpointing at the granularity of individual layers within each stage. For a stage with l layers, where c are checkpointed, recomputation reduces activation memory but increases execution time, providing fine-grained control over the memory–compute trade-off. The planner selects c to satisfy

memory constraints while minimizing stage execution cost, applying recomputation only where necessary. This avoids the inefficiency of uniform checkpointing: stages that fit in memory incur no recomputation, while only memory-constrained stages do. However, selective checkpointing alone cannot eliminate pipeline imbalance: early stages may remain slower due to recomputation, while later stages are underutilized.

2) Cross-Stage Load Balancing via Uneven Partitioning. To address residual imbalance, X-MoE-4D allows stages to contain different numbers of layers rather than enforcing equal partitioning. Under 1F1B scheduling, later stages retain fewer in-flight micro-batches and thus have lower memory pressure (§III-C). The planner exploits this asymmetry by shifting layers from memory-constrained early stages to later stages, reducing recomputation overhead while improving utilization and balancing execution across stages.

Checkpointing and partitioning play complementary roles in the planner. Checkpointing controls the memory–compute trade-off within each stage, while partitioning determines how workload is distributed across stages. Neither alone is sufficient: checkpointing reduces memory but cannot resolve cross-stage imbalance, whereas partitioning redistributes workload but does not eliminate intra-stage memory pressure. Their combination enables the planner to minimize bottleneck stage latency under memory constraints.

3) Stage Cost Modeling. To evaluate each configuration's costs, X-MoE-4D first decomposes the pipeline execution overheads, then instantiates the stage cost function in Eq. 2 using lightweight runtime and memory models.

Runtime derivation: For a p-stage pipeline (e.g., 1F1B), execution consists of warm-up, steady-state, and cool-down phases:

$$T_{\text{total}} = T_{\text{warm-up}} + T_{\text{steady}} + T_{\text{cool-down}} \tag{3}$$

Warm-up and cool-down depend on the cumulative execution time of all stages, while steady-state throughput is determined by the slowest stage across micro-batches:

$$\begin{aligned} T_{\text{warm-up}} + T_{\text{cool-down}} &= t_{\text{sum}}, \\ T_{\text{steady}} &= (N_{\text{mbs}} - 1)\, t_{\text{max}} \end{aligned} \tag{4}$$

where stage time is the sum of per-layer costs:

$$t_s = \sum_{l \in s} t_l, \qquad t_{\text{sum}} = \sum_{s \in p} t_s, \qquad t_{\text{max}} = \max_{s \in p} t_s \tag{5}$$

The pipeline execution time can be further divided into an ideal processing time and a bubble time. The ideal processing time for a balanced 1F1B pipeline is a stage's cost accumulated over all micro-batches: $T_{\text{id}} = N_{\text{mbs}}\, t_{\text{s}}$. A stage's per-micro-batch cost is $t_{\text{s}} = t_{\text{fwd}} + t_{\text{bwd}} + t_{\text{recomp}}$, where t_{recomp} is the activation recomputation time if checkpointing and is 0 otherwise. The bubble time is the $p-1$ stages fill-drain latencies, $T_{\text{bb}} = (p-1)\, t_{\text{s}}$, with bubble fraction as: $f_{\text{bb}} = T_{\text{bb}}/T_{\text{id}} = (p-1)/N_{\text{mbs}}$ [31]. T_{bb} and T_{id} together constitute the warm-up, steady, and cool-down states of Eq. 4:

$$\begin{aligned} T_{\text{total}} = T_{\text{id}} + T_{\text{bb}} &= (N_{\text{mbs}} + p - 1)\, t_{\text{s}} \\ &= \underbrace{(N_{\text{mbs}} - 1)\, t_{\text{max}}}_{T_{\text{steady}}} + \underbrace{t_{\text{sum}}}_{T_{\text{warm-up}} + T_{\text{cool-down}}}, \end{aligned} \tag{6}$$

with $t_{\text{max}} = t_{\text{s}}$ and $t_{\text{sum}} = p\, t_{\text{s}}$.

When stages are unevenly partitioned, the steady state's execution is paced by the slowest stage that has processing time $N_{\text{mbs}}\, t_{\text{max}}$. Under Eq. 4, the remainder, $T_{\text{total}} - N_{\text{mbs}}\, t_{\text{max}} = t_{\text{sum}} - t_{\text{max}}$, is the bottleneck stage's fill-drain cost. The remaining stages, busy for only $N_{\text{mbs}}\, t_s$, further idle while being paced by the slowest stage, adding a load-imbalance term. The bubble time of stage s is

$$\begin{aligned} T_{\text{bb}}^{(s)} &= T_{\text{total}} - N_{\text{mbs}}\, t_s \\ &= \underbrace{\big(t_{\text{sum}} - t_s\big)}_{\text{fill–drain}} + \underbrace{(N_{\text{mbs}} - 1)\big(t_{\text{max}} - t_s\big)}_{\text{load-imbalance}}, \end{aligned} \tag{7}$$

a one-time fill-drain bubble in addition to a load-imbalance bubble accrued over the $N_{\text{mbs}} - 1$ steady state periods. Summing Eq. 7 over all stages and normalizing by the total compute time $N_{\text{mbs}}\, t_{\text{sum}}$ give the end-to-end bubble fraction:

$$f_{\text{bb}} = \frac{p-1}{N_{\text{mbs}}} + \frac{(N_{\text{mbs}} - 1)\big(p\, t_{\text{max}} - t_{\text{sum}}\big)}{N_{\text{mbs}}\, t_{\text{sum}}} \tag{8}$$

The fill-drain term, identical to the balanced-pipeline bubble, grows linearly with pipeline depth p and hence with more device count in scaling PP dimension, but it is amortized with the number of micro-batches N_{mbs} increases. The load-imbalance term grows with t_{max} and, unlike the fill–drain term, is not amortized as N_{mbs} increases. It is therefore the component that persists at scale, and it is precisely what minimizing t_{max} in Eq. 1 resolves through selective checkpointing and uneven partitioning.

Runtime modeling: Unlike compilation-based planners [34], [53] that require full model instantiation for each parallelism configuration (taking tens of minutes to hours at scale), X-MoE-4D decomposes the runtime overheads into computation and communication costs.

Let T_{gemm} denote the aggregated computation cost of a layer and T_{comm} denote its dominant communication cost (e.g., MoE's A2A token routing). The P2P transfer costs across pipeline stage boundaries are excluded as they are minor compared to per-layer computation and MoE token routing time. Thus, we model forward and backward as:

$$\begin{aligned} T_f &= T_{\text{gemm}} + T_{\text{comm}} \\ T_b &= 2T_{\text{gemm}} + T_{\text{comm}} \end{aligned} \tag{9}$$

where the backward pass incurs additional computation for gradients with respect to both inputs and parameters. The resulting per-layer execution cost without checkpointing is:

$$t_{\text{uc}} = 3T_{\text{gemm}} + 2T_{\text{comm}} \tag{10}$$

With recomputation, the forward pass is re-executed during backward, yielding:

$$t_{\text{c}} = 4T_{\text{gemm}} + 3T_{\text{comm}} \tag{11}$$

To capture hardware-specific behavior, we profile per-layer T_{gemm} and T_{comm} once and derives per-stage cost using Eq. 10–11. This modular design avoids recalibration across configurations, unlike direct per-stage profiling, which depends on stage partitioning and checkpointing policy. This is also particularly important for MoE layers, where fragmented expert execution deviates from theoretical FLOPs. ELM-PP leverages T_{gemm} and T_{comm} to find the highest throughput configuration across micro-batch sizes, parallelism degrees, and checkpointing strategies given any GPU device budgets.

Memory modeling. To enforce per-device memory constraints

in Eq. 2, we estimate peak memory per stage as:

$$\text{Mem}(l, c) = \text{Mem}_{\text{state}} + \text{Mem}_{\text{activation}} + A_{\text{overhead}} \quad (12)$$

With mixed precision [54], model states scale linearly with parameter count ($\approx 16P$ with FP32 optimizer states and BF16 weights and gradients). Activation memory depends on layer composition and checkpointing decisions; routing buffers and aggregated expert activations dominate, scaling with token count, expert count, and in-flight micro-batches. We include a fixed overhead term A_{overhead}, estimated empirically, to account for framework-level allocator behavior and fragmentation.

Configurations violating $\text{Mem}(l, c) \leq M_{\text{limit}}$ are excluded. The planner evaluates a small set of candidate micro-batch sizes and selects the highest throughput configuration; each is solved in seconds even for 1T model with 60-stage pipelines.

C. Efficient Expert-Local Execution

While the above design addresses pipeline-level inefficiency, expert-local execution introduces fragmented computation due to many fine-grained experts (Insight 4), leading to excessive kernel launches and low GEMM utilization. Prior work [28], [41], [55], [56] has developed GroupGEMM optimizations, mostly NVIDIA-specific stacks. Although GroupGEMM itself is not a novel contribution of this work, it becomes crucial under X-MoE-4D's pipeline-global, expert-local execution: by confining experts within each node, up to tens of experts (e.g. 64) are co-located per GPU, significantly amplifying kernel fragmentation compared to EP-centric designs that distribute experts across devices. We leverage two GroupGEMM backends: Primus-Turbo's Composable Kernel (CK) [57] for optimized execution on AMD GPUs, and a Triton-based implementation for cross-platform support. Both adopt a persistent kernel design that aggregates per-expert workloads into larger execution units, reducing kernel launch overhead and improving hardware utilization. This ensures that expert-local execution remains efficient and does not become a bottleneck.

V. Evaluation

A. Experimental Setup

Platform and Hardware. We evaluate on Frontier [58], an AMD-based exascale HPC system composed of MI250X GPUs interconnected via the Dragonfly network. Each MI250X GPU provides a theoretical peak of 191 TFLOPS in BF16. We report results on scales of up to 2,304 GPUs.

Implementation. We implement X-MoE-4D on top of PyTorch and DeepSpeed [9], extending X-MoE [13] with 1F1B pipeline parallelism and our skew-aware planning mechanism.

Model Configurations. We consider a range of model configurations ranging from 10B to 1T parameters (denoted as Small, Medium, Large, Super, and Ultra), as summarized in Table I. The 10B–537B settings follow the model scales used in prior work such as X-MoE [13], which enables direct comparison with existing training systems. We further extend the evaluation to 1T (Ultra) model, which goes beyond the scale demonstrated by prior AMD-based MoE training systems. All configurations follow recent expert-specialized MoE models [3], [5], [39], [59].

TABLE I
Model configurations ranging from 10B to 1T parameters.

Models	**Small**	**Medium**	**Large**	**Super**	**Ultra**
H	2048	5120	7168	7168	7168
H_{FFN}	1408	1536	2048	2560	2560
Experts	64	128	256	256	512
Top-k	6	6	8	8	8
Seq. Len.	2048	4096	4096	4096	4096
Layers	24	32	24	60	60
Param.	**10B**	**63B**	**173B**	**537B**	**1T**

Baselines. We compare X-MoE-4D against state-of-the-art MoE training systems that support AMD-based platforms including DeepSpeed-MoE [9], DeepSpeed-TED [25], Tutel [10], TorchTitan [60], and X-MoE [13]. Systems that rely on CUDA-specific implementations (e.g., Megatron-LM [26], [31], MegaBlocks [21]) are excluded due to lack of stable ROCm support on AMD GPUs (MI250) at scale. To ensure a fair comparison, we search for the highest throughput configuration for each system. The search space differs between EP-centric systems, including DS-MoE, DS-TED, Tutel, and X-MoE, and PP-centric systems, including TorchTitan and X-MoE-4D. For the EP-centric systems, we fix PP = 1, and grid search at every scale over TP {1, 2, 4, 8}, micro-batch size {1, 2}, activation checkpointing {none, full activation recomputation from per-layer inputs}. EP is searched over a per-scale candidate set bounded by the GPU allocation: {32, 64} (63B), {64, 128, 256} (173B), {128, 256} (537B), {128, 256, 512} (1T). For the PP-centric systems, we fix EP = 8 and TP = 1. TorchTitan searches micro-batch size {1, 2}, activation checkpointing {none, full}, and PP in {4, 8} (63B), {8, 12} (173B), {15, 30, 60} (537B), {30, 60} (1T). X-MoE-4D searches the same PP candidates as TorchTitan, and its layer partitioning, per-stage checkpointing, and micro-batch size are selected by ELM-PP. All systems use ZeRO-1 except for TorchTitan's FSDP. The DP degrees of non-MoE and MoE layers are determined by the number of GPUs and the selected TP, EP, and PP degrees. The micro-batch candidates are kept small because model states already consume a substantial fraction of the per-GCD 64 GB memory for fine-grained MoE models, leaving limited headroom for larger activation footprints. The global batch size is set to 4096. All reported results correspond to the highest throughput achieved by each method without exceeding per-device memory limits, and Table II lists the selected configuration for each system. We report mean throughput and standard deviation across post-warm-up training iterations.

Metrics. We report Model FLOPs Utilization (MFU) and end-to-end training throughput. MFU measures hardware utilization relative to theoretical hardware peak performance, excluding recomputation overhead from activation checkpointing.

B. Main Results

TABLE II
PER-SCALE CONFIGURATIONS ACROSS SYSTEMS. "–" INDICATES THAT NO SEARCHED CONFIGURATION WAS TRAINABLE. D DENOTES DP_NON_MOE AND M DENOTES DP_MOE.

Scale	System	GCDs	TP	PP	EP	MBS	AC	D/M	Error
63B	DS-MoE DS-TED Tutel	64	1	1	64	1	full	64/1	N/A
	TorchTitan	64	1	8	8	1	full	8/1	N/A
	X-MoE	64	1	1	64	1	none	64/1	N/A
	X-MoE-4D	64	1	4	8	1	planner	16/2	N/A
173B	DS-MoE Tutel	256	2	1	128	1	full	128/2	N/A
	DS-TED	256	2	1	128	1	full	128/1	N/A
	TorchTitan	256	–	–	–	–	–	–	Collective timeout at init
	X-MoE	256	2	1	128	1	none	128/2	N/A
	X-MoE-4D	256	1	8	8	1	planner	32/4	N/A
537B	DS-MoE DS-TED Tutel	512	–	–	–	–	–	–	OOM at every searched
	TorchTitan	480	–	–	–	–	–	–	Collective timeout at init
	X-MoE	512	4	1	128	1	full	128/4	N/A
	X-MoE-4D	480	1	30	8	1	planner	16/2	N/A
1T	DS-MoE DS-TED Tutel	1024	–	–	–	–	–	–	OOM at every searched
	TorchTitan	960	–	–	–	–	–	–	Collective timeout at init
	X-MoE	1024	4	1	256	1	full	256/4	N/A
	X-MoE-4D	960	1	60	8	2	planner	16/2	N/A

Fig. 7 shows the end-to-end training performance of X-MoE-4D across model scales up to 1T parameters on Frontier. X-MoE-4D consistently achieves the highest efficiency and throughput, reaching up to 2.4× higher per-GPU throughput than the strongest prior system while using fewer GPUs at large scales.

Trainability. We first examine whether existing systems can train large-scale expert-specialized MoE models under realistic hardware constraints. As shown in Fig. 7, widely used frameworks, such as DeepSpeed-MoE [9], DeepSpeed-TED [25], and Tutel [10], fail to scale to large models on Frontier. These systems were primarily designed for coarse-grained MoEs and do not address the memory bottlenecks introduced by expert-specialized MoEs, where activation and routing overhead dominate. TorchTitan [60] with an FSDP backend [61] has recently introduced support for MoE training using PP+EP parallelism. However, in our experiments, it encounters nondeterministic collective deadlocks at scale on Frontier, preventing stable execution[1]. As a result, at 537B and 1T scales, these systems either run out of memory or fail to execute. X-MoE [13] remains the only baseline that successfully scales to these regimes on AMD-based HPC systems, enabled by its memory-efficient sharding design that directly addresses the memory bottleneck in expert-specialized MoEs. In contrast, X-MoE-4D successfully scales to the 1T Ultra model on Frontier using 960 GPUs. This is enabled by two mechanisms. First, pipeline-global, expert-local execution

[1]Observed with ROCm 7.0.2, torch 2.11 and TorchTitan 0.2.0.

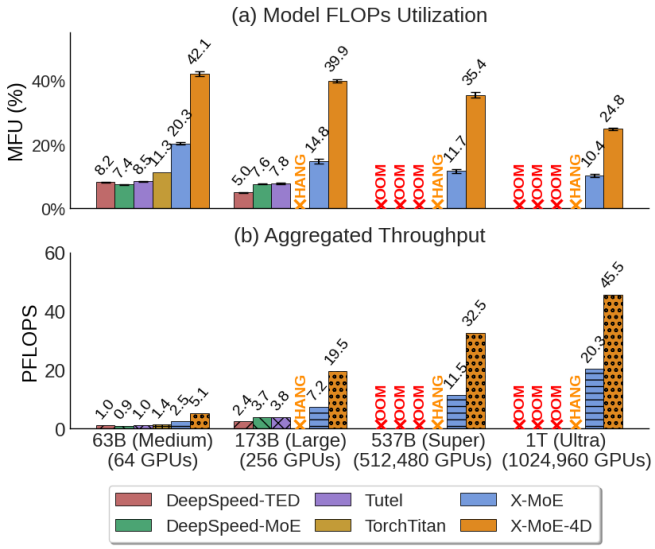


Fig. 7. Training performance across model scales (63B–1T) on Frontier. (a) Model FLOPs Utilization (MFU). (b) Aggregate throughput (PFLOPS). X-MoE-4D achieves the highest MFU and throughput across all scales.

localizes expert computation within nodes and avoids cross-node expert parallelism, reducing routing buffer overhead. In addition, pipeline partitioning distributes model layers across devices, lowering the per-device model state footprint. Second, deep pipeline execution introduces activation imbalance, where early stages retain many in-flight activations and become the memory bottleneck. The minimax pipeline planner addresses this by jointly optimizing activation checkpointing and stage partitioning decisions, ensuring that peak memory at every stage remains within the 64 GB device limit. **Training Throughput.** We compare the end-to-end training throughput of X-MoE-4D against state-of-the-art MoE training systems under comparable hardware budget (e.g., 960 vs. 1024). As shown in Fig. 7, X-MoE-4D consistently outperforms prior approaches across all model scales. At the 1T scale, X-MoE-4D achieves 24.8% MFU, compared to 10.4% for X-MoE, corresponding to 2.4 × higher per-GPU throughput. Despite using 6% fewer GPUs (960 vs. 1024), X-MoE-4D reaches an aggregated 45.5 PFLOPS throughput, 2.24× higher than X-MoE's 20.3 PFLOPS. The throughput gains of X-MoE-4D stem from two key design choices. First, by confining expert parallelism within nodes, X-MoE-4D avoids cross-node expert routing, eliminating a major communication bottleneck in EP-based approaches at scale. Second, X-MoE-4D employs minimax pipeline planning to balance stage computation, reducing pipeline imbalance and improving hardware utilization. Together, these enable X-MoE-4D to sustain high training throughput at large scale.

These results demonstrate X-MoE-4D not only enables trillion-scale MoE training but also utilizes available compute resources significantly more efficiently, achieving a record-high efficiency for MoE training on AMD-based HPC systems.

C. Scalability Results

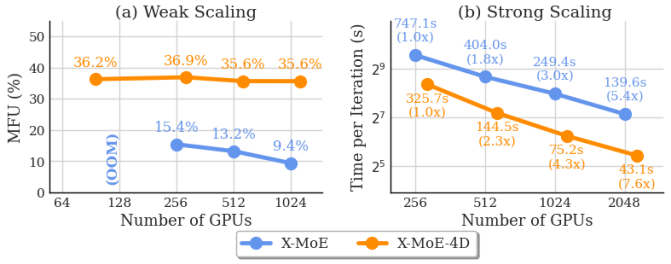


Fig. 8. **Scalability of X-MoE-4D vs. X-MoE on the 173B model.** (a) Weak scaling. (b) Strong scaling. X-MoE-4D maintains high efficiency and achieves faster iteration time. The performance gap widens with scale.

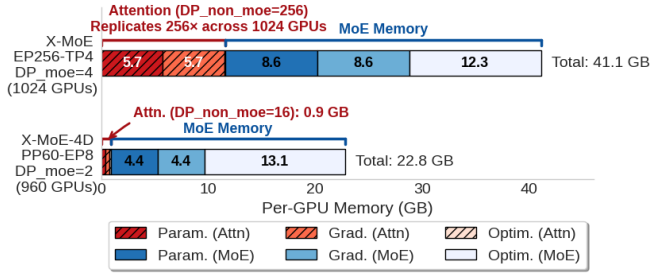


Fig. 9. **Per-GPU model state memory at 1T scale**. Comparison of the highest throughput X-MoE and X-MoE-4D configurations. X-MoE-4D reduces per-device model-state memory from 41.1 GB to 22.8 GB.

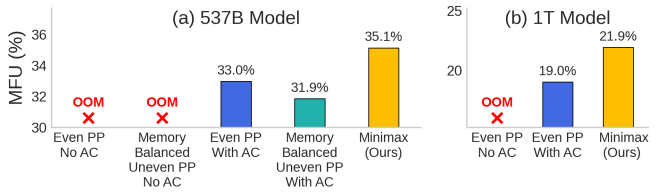


Fig. 10. **ELM-PP gain.** Run with 480 GPUs at 537B, 1T. No AC, all baselines OOM. Even partitioning and memory-balanced uneven partitioning yield sub-optimal MFU, while ELM-PP achieves the highest utilization.

We evaluate the scalability of X-MoE-4D using both weak and strong scaling on the 173B model, enabling comparison with the baseline system across a wide range of GPU counts by scaling along DP dimension. For comparison, X-MoE sets EP = 128, TP = 2, and no activation checkpointing along with mbs = 1 are selected. X-MoE-4D sets PP = 12, and checkpointing strategies and micro-batch sizes are selected by ELM-PP.

Weak Scaling. We fix the per-node batch size 133 and increase the number of GPUs proportionally with the global batch size. As shown in Fig. 8(a), X-MoE-4D maintains a stable 36% MFU as the system scales to 1,152 GPUs, achieving up to 3.79× higher throughput than X-MoE. This stable efficiency arises from two factors. First, X-MoE-4D eliminates cross-node expert routing, avoiding the growth of *all-to-all* communication that dominates EP-based approaches at scale. Second, minimax pipeline planning maintains balanced computation and memory across pipeline stages, preventing stage-level bottlenecks from emerging as the system scales. Together, these ensure that both communication and computation remain well-behaved, enabling consistent efficiency on hierarchical HPC networks.

Strong Scaling. We fix the global batch size (GBS=19,200) and increase the number of GPUs, reducing per-device workload. As shown in Fig. 8(b), X-MoE-4D scales efficiently up to 2,304 GPUs, reducing iteration time from 325.7 to 43.1s, while X-MoE decreases from 747.1s to 139.6s. This corresponds to up to 3.24× speedup over X-MoE at the largest scale. The performance gap widens with scale, growing from 2.3× at the smallest configuration to 3.24× at the largest. Already at 576 GPUs, X-MoE-4D matches the iteration time of X-MoE with 2,048 GPUs. As scale increases, X-MoE remains limited by cross-node communication overhead, while X-MoE-4D eliminates cross-node expert routing and shifts the bottleneck to pipeline imbalance. Under strong scaling, reduced per-device workload amplifies stage-level imbalance. X-MoE-4D mitigates this effect through minimax pipeline planning, which balances computation and memory across stages, reducing idle time propagation and maintaining high utilization even as parallelism increases.

D. Analysis Results

Why does X-MoE-4D require fewer GPUs at 1T scale? We compare per-device memory usage at the 1T scale using each system's highest throughput configuration under comparable GPU counts. As shown in Fig. 9, X-MoE runs on 1024 GPUs with EP256-TP4, which induces DP_{moe}=4 for MoE layers and $DP_{non\text{-}moe}$=256 for non-MoE layers, while X-MoE-4D runs on 960 GPUs with PP60-EP8, which induces DP_{moe}=2 and $DP_{non\text{-}moe}$=16. Under these configurations, X-MoE requires 41.1 GB per-device for model states alone, whereas X-MoE-4D uses only 22.8 GB, a 1.8× reduction. This gap comes primarily from how non-MoE states are distributed. Under X-MoE, attention parameters and gradients are replicated across $DP_{non\text{-}moe}$=256 ranks, consuming 11.4 GB per device and approximately 2.67 TB system-wide. This high replication arises from TP=4 in non-MoE layers to achieve the best throughput. Increasing TP would reduce per-device memory but leads to higher communication overhead and lower efficiency in practice. In contrast, X-MoE-4D shards non-MoE layers across pipeline stages, reducing attention-state memory to 0.9 GB per device. Intuitively, pipeline parallelism distributes non-MoE layers across devices, reducing their replication compared to EP-centric designs. This corresponds to a lower replication degree ($DP_{non\text{-}moe}$=16). Although X-MoE-4D has a slightly larger MoE-state footprint per device due to lower replication, its total model-state memory remains substantially lower. This reduced per-device memory footprint leaves sufficient headroom for activations under the 64 GB device limit, enabling 1T training with as few as 480 GPUs without running OOMs.

How does ELM-PP improve performance? We evaluate ELM-PP with 480 GPUs at two scales. At 537B (PP30-EP8-DP2), 30 stages with 2 layers each allow joint optimization of selective checkpointing and layer redistribution. We compare against (i) even partitioning and (ii) memory-balanced uneven partitioning (equalizing peak memory without optimizing execution time), each with and without full AC. At 1T (PP60-EP8-DP1), one layer per stage leaves only per-stage checkpointing

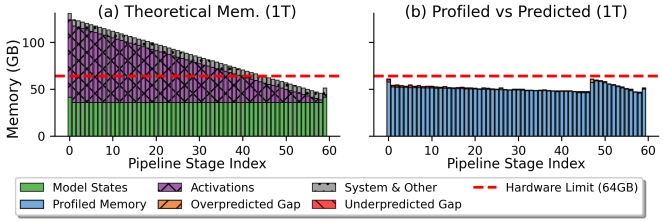


Fig. 11. **Memory model validation**. 1T scale (pp60-ep8, 480 GPUs), (a) Per-stage memory breakdown shows MoE Pipeline Skew due to 1F1B scheduling: stage p_{id} holds $(p-p_{id})$ in-flight micro-batches. (b) Predicted vs. profiled peak memory. The model achieves MAE = 0.975 GB and MAPE = 1.86%.

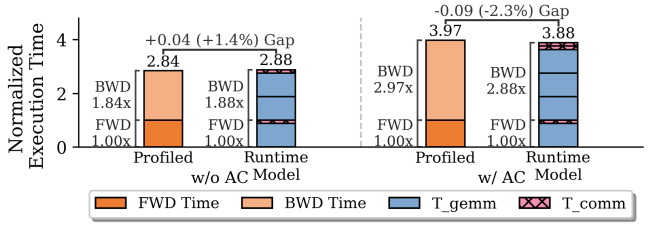


Fig. 12. **Runtime model accuracy.** Comparison of stage-level profiled execution time and model prediction for 1T model. The model achieves 1.4% relative error without activation checkpointing and 2.3% with checkpointing.

to optimize; we compare against full AC and no AC. At 537B (Fig. 10(a)), without AC, both baselines OOM. With full AC, even partitioning achieves 33.0% MFU. Memory-balanced uneven partitioning redistributes layers to equalize peak memory across stages, but introduces stage imbalance that degrades performance to 31.9% MFU. In particular, some stages become heavier than others, increasing their execution time and dominating $t_{\max}$. Minimax planner jointly optimizes both dimensions by selectively checkpointing layers only at memory-constrained stages while adjusting partitioning to minimize $t_{\max}$, achieving 35.1% MFU, a 2.1 point (6.4%) improvement over even partitioning with AC. At 1T (Fig. 10(b)), early stages exceed 64 GB without AC due to 1F1B lifetime asymmetry. Applying full AC to all 60 stages resolves OOM but degrades to 19.0% MFU, as stages that already fit within the memory budget incur unnecessary recomputation overhead. Minimax planner selectively checkpoints only memory-constrained stages, avoiding unnecessary recomputation while satisfying memory constraints, and achieves 21.9% MFU, a 15.3% relative improvement.

How accurate are the analytical models used for planning? Fig. 11(a) shows the predicted per-stage memory breakdown for the 1T model (PP60-EP8, 480 GPUs). Model states remain constant, while activation memory decreases with stage index due to 1F1B's descending in-flight micro-batch count. Without checkpointing, early stages exceed the 64 GB limit. Fig. 11(b) compares predicted and profiled peak memory under the optimal configuration from ELM-PP, achieving MAE = 0.975 GB and MAPE = 1.86%. At 537B scale (PP30-EP8, 240 GPUs), the model achieves MAE = 0.917 GB and MAPE = 1.72%, confirming accuracy generalizes across scales. The primary error source is stochastic MoE routing, handled via average token assignment and a conservative buffer ($A_{\text{overhead}} \approx 7$ GB).

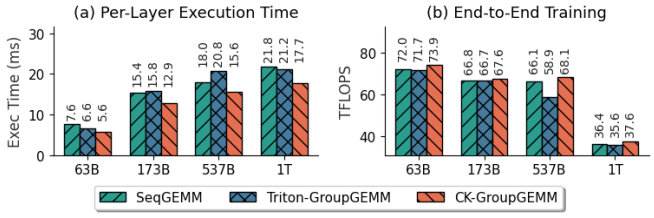


Fig. 13. **GroupGEMM Results.** (a) Per-layer expert execution time, run under 1-node EP=8 scenario across multiple individual runs. (b) End-to-end training throughput (with GBS=2048). CK-GroupGEMM yields consistent improvements across all scales.

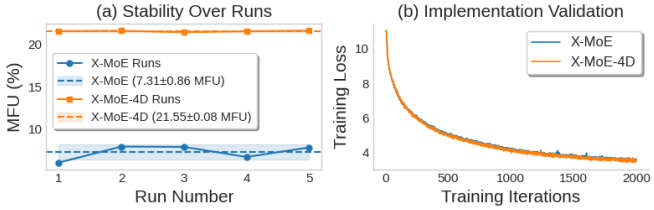


Fig. 14. **(a) Coefficient of variation (std/mean) in runs.** X-MoE has high variability (CV 11.8%). X-MoE-4D remains stable (CV = 0.4%). **(b) Implementation validation.** X-MoE-4D preserves training convergence as X-MoE.

Critically, the prediction consistently matches the profiled bottleneck, ensuring the correctness property required by the minimax formulation. Fig. 12 evaluates the runtime prediction accuracy. Without AC, the model achieves 1.4% relative error compared to measured stage execution time. With AC, backward time increases by up to 2.97× due to recomputation overhead, yet prediction error remains low at 2.3%. These results demonstrate robust accuracy across configurations.

How does expert execution efficiency impact performance? Fig. 13(a) shows that GroupGEMM consistently reduces per-layer execution time compared to sequential GEMM, with larger gains at smaller scales where expert fragmentation is more prominent. As shown in Fig. 13(b), the end-to-end throughput improvement is modest, as overall performance at large scale is primarily dominated by pipeline imbalance rather than intra-stage execution. Among implementations, the CK-based backend delivers the best performance, which shows the high-performance of hardware-optimized kernels. On the other hand, Triton-based GroupGEMM provides portable benefits across different hardware platforms. Nevertheless, GroupGEMM remains essential: without it, fragmented expert execution would degrade utilization and lead to performance loss under expert-local execution.

How stable is training at the 1T scale? We evaluate training stability at the 1T scale across five independent training runs. To stress-test communication stability, we use a smaller global batch size (GBS=1792), increasing the frequency of pipeline flushes compared to the main evaluation (GBS=4096). As shown in Fig. 14, X-MoE-4D exhibits significantly more stable performance, with coefficient of variation (CV) of 0.4% compared to 11.8% for X-MoE. This instability in X-MoE correlates with its reliance on large-scale cross-node *all-to-all* communication across 1024 GPUs, exhibiting sub-

stantial latency variability at scale (Fig. 1). Such variability directly propagates to iteration-level performance due to the synchronization requirements of MoE dispatch and combine operations. In contrast, X-MoE-4D confines *all-to-all* communication to intra-node scope. This significantly reduces the variability of communication, resulting in consistently high hardware utilization across iterations.

Implementation validation. We validate implementation correctness by training the 10B model for 2,000 iterations at GBS=160 under both systems. As shown in Fig. 14(b), the loss curves remain nearly identical throughout: at iterations 400/1,200/2,000, X-MoE reaches 5.415/3.961/3.583 while X-MoE-4D reaches 5.350/3.907/3.543, with no growing gap. The partitioning and recomputation decisions made by ELM-PP therefore do not alter convergence.

VI. Conclusion

We presented X-MoE-4D, a system for scaling expert-specialized MoE training on hierarchical HPC systems. Our study reveals that the primary bottleneck at scale is not communication alone, but MoE Pipeline Skew (MPS), a structural imbalance arising from the interaction between MoE architectures and pipeline execution. Our results show that addressing MPS requires jointly reasoning about execution mapping and pipeline scheduling. By combining pipeline-global, expert-local execution with skew-aware planning, X-MoE-4D achieves efficient scaling up to 1T parameters and 2,304 GPUs. More broadly, our findings suggest that scaling sparse MoEs on hierarchical HPC systems requires topology-aligned execution and skew-aware pipeline planning to address stage imbalance.

Acknowledgments

We sincerely appreciate the anonymous reviewers and shepherd. Their insightful feedback helped significantly improve the quality of the paper. We also gratefully acknowledge Junqi Yin and Hao Lu from ORNL for their valuable discussions. This research was supported by the National Science Foundation (NSF) under Grant No. 2441601. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Advanced Scientific Computing Research programs in the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The work also utilized the Delta and DeltaAI system at the National Center for Supercomputing Applications (NCSA) and Jetstream2 at Indiana University through allocation CIS240055 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. The Delta advanced computing resource is a collaborative effort between the University of Illinois Urbana-Champaign and NCSA, supported by the NSF (award OAC 2005572) and the State of Illinois. UIUC SSAIL Lab is supported by research funding from Google, Amazon, Microsoft, IBM, and AMD.

References

[1] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1701.06538

[2] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2101.03961

[3] DeepSeek-AI, A. Liu, B. Feng, B. Xue, and et al., "Deepseek-v3 technical report," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.19437

[4] A. Adcock, A. Srivastava, A. Dubey, A. Jauhri, A. Pande, A. Pandey, A. Sharma, A. Kadian, A. Kumawat, A. Kelsey *et al.*, "The llama 4 herd: Architecture, training, evaluation, and deployment notes," *arXiv preprint arXiv:2601.11659*, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2601.11659

[5] Qwen Team, "Qwen3-next: Towards ultimate training & inference efficiency," Qwen Blog, sep 2025, accessed: 2025-12-13. [Online]. Available: https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd

[6] L. Sun, D. Luo, D. Ma, Z. Zhao, B. Chen, Z. Shen, S. Zhu, L. Chen, X. Chen, and K. Yu, "Scidfm: a large language model with mixture-of-experts for science," *arXiv preprint arXiv:2409.18412*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.18412

[7] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2006.16668

[8] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.04088

[9] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International conference on machine learning*. PMLR, 2022, pp. 18 332–18 346. [Online]. Available: https://doi.org/10.48550/arXiv.2201.05596

[10] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram *et al.*, "Tutel: Adaptive mixture-of-experts at scale," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 269–287, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2206.03382

[11] K. Team, T. Bai, Y. Bai, Y. Bao, J. Cai, X. Cai, P. Cao, Y. Cao, Z. Chai, Y. Charles *et al.*, "Kimi k3: Open frontier intelligence," *arXiv preprint arXiv:2607.24653*, 2026.

[12] A. Xu, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong, C. Ling *et al.*, "Deepseek-v4: Towards highly efficient million-token context intelligence," *arXiv preprint arXiv:2606.19348*, 2026.

[13] Y. Yuan, A. Gupta, J. Li, S. Dash, F. Wang, and M. Zhang, "X-moe: Enabling scalable training for emerging mixture-of-experts architectures on hpc platforms," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 1315–1331. [Online]. Available: https://doi.org/10.1145/3712285.3759886

[14] V. Rajaraman, "Frontier—world's first exaflops supercomputer," *Resonance*, vol. 28, no. 4, pp. 567–576, 2023. [Online]. Available: https://doi.org/10.1007/s12045-023-1583-7

[15] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 77–88, 2008. [Online]. Available: https://doi.org/10.1145/1394608.1382129

[16] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," *arXiv preprint arXiv:2103.13262*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2103.13262

[17] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "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*, 2022, pp. 120–134. [Online]. Available: https://doi.org/10.1145/3503221.3508418

[18] X. Nie, P. Zhao, X. Miao, T. Zhao, and B. Cui, "Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system,"

arXiv preprint arXiv:2203.14685, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.14685

[19] J. Liu, J. H. Wang, and Y. Jiang, "Janus: A unified distributed training framework for sparse mixture-of-experts models," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 486–498. [Online]. Available: https://doi.org/10.1145/3603269.3604869

[20] X. Pan, W. Lin, L. Zhang, S. Shi, Z. Tang, R. Wang, B. Li, and X. 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*, 2025, pp. 524–539. [Online]. Available: https://doi.org/10.1145/3669940.3707272

[21] T. Drozd, S. Rao, S. Gelada, A. Harlap, D. Narayanan, M. Chowdhury, and M. Zaharia, "Megablocks: Efficient sparse training with mixture-of-experts," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 6, 2024, pp. 240–255. [Online]. Available: https://doi.org/10.48550/arXiv.2211.15841

[22] X. Nie, X. Miao, Z. Wang, Z. Yang, J. Xue, L. Ma, G. Cao, and B. Cui, "Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement," *Proc. ACM Manag. Data*, vol. 1, no. 1, May 2023. [Online]. Available: https://doi.org/10.1145/3588964

[23] C. Jiang, Y. Tian, Z. Jia, S. Zheng, C. Wu, and Y. Wang, "Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2404.19429

[24] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16. [Online]. Available: https://doi.org/10.48550/arXiv.1910.02054

[25] S. Singh, O. Ruwase, A. A. Awan, S. Rajbhandari, Y. He, and A. Bhatele, "A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 203–214. [Online]. Available: https://doi.org/10.48550/arXiv.2303.06318

[26] D. Liu, Z. Yan, X. Yao, T. Liu, V. Korthikanti, E. Wu, S. Fan, G. Deng, H. Bai, J. Chang *et al.*, "Moe parallel folding: Heterogeneous parallelism mappings for efficient large-scale moe model training with megatron core," *arXiv preprint arXiv:2504.14960*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.14960

[27] Z. Yan, H. Bai, X. Yao, D. Liu, T. Liu, H. Liu, P. Li, E. Wu, S. Fan, L. Tao *et al.*, "Scalable training of mixture-of-experts models with megatron core," *arXiv preprint arXiv:2603.07685*, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2603.07685

[28] C. Jin, Z. Jiang, Z. Bai, Z. Zhong, J. Liu, X. Li, N. Zheng, X. Wang, C. Xie, Q. Huang *et al.*, "Megascale-moe: Large-scale communication-efficient training of mixture-of-experts models in production," *arXiv preprint arXiv:2505.11432*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2505.11432

[29] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1811.06965

[30] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3341301.3359646

[31] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2021, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3458817.3476209

[32] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero bubble pipeline parallelism," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.10241

[33] S. Li and T. Hoefler, "Chimera: efficiently training large-scale neural networks with bidirectional pipelines," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14. [Online]. Available: https://doi.org/10.1145/3458817.3476145

[34] B. Jeon, M. Wu, S. Cao, S. Kim, S. Park, N. Aggarwal, C. Unger, D. Arfeen, P. Liao, X. Miao *et al.*, "Graphpipe: Improving performance and scalability of dnn training with graph pipeline parallelism," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025, pp. 557–571. [Online]. Available: https://doi.org/10.1145/3669940.3707220

[35] T. Kim, H. Kim, G.-I. Yu, and B.-G. Chun, "Bpipe: memory-balanced pipeline parallelism for training large language models," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023.

[36] W. Liu, M. Li, G. Tan, and W. Jia, "Mario: Near zero-cost activation checkpointing in pipeline parallelism," in *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 197–211. [Online]. Available: https://doi.org/10.1145/3710848.3710878

[37] M. Huang, C. Wang, C. Ma, Y. Zhang, P. Zhang, and L. Yu, "Re-evaluating the memory-balanced pipeline parallelism: Bpipe," *arXiv preprint arXiv:2401.02088*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.02088

[38] G. He, Y. Jiang, W. Xiao, K. Jiang, S. Wang, J. Wang, Z. Du, Z. Jiang, X. Zhang, B. Yuan *et al.*, "Efficient pre-training of llms via topology-aware communication alignment on more than 9600 gpus," *arXiv preprint arXiv:2509.15940*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2509.15940

[39] A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.04434

[40] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2505.09388

[41] QwenLM, "Qwen3: Think deeper, act faster," https://qwenlm.github.io/blog/qwen3/, 2025, official Blog.

[42] T. Kimi, Y. Bai, Y. Bao, G. Chen, and et al., "Kimi k2: Open agentic intelligence," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2507.20534

[43] OpenAI, "gpt-oss-120b & gpt-oss-20b model card," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.10925

[44] Microsoft, "Announcing the availability of phi-3.5 moe in azure ai studio and github," https://techcommunity.microsoft.com/blog/azure-ai-foundry-blog/announcing-the-availability-of-phi-3-5-moe-in-azure-ai-studio-and-github/4256278, 2024, microsoft Tech Community.

[45] I. Granite, "Granite 3.1 language models," https://github.com/ibm-granite/granite-3.1-language-models, 2024, gitHub repository.

[46] D. The Mosaic Research Team, "Introducing dbrx: A new state-of-the-art open llm," https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, 2024, databricks Blog, March 27, 2024.

[47] Z. Sun, H. Cao, Y. Wang, G. Feng, S. Chen, H. Wang, and W. Chen, "Adapipe: Optimizing pipeline parallelism with adaptive recomputation and partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 86–100. [Online]. Available: https://doi.org/10.1145/3620666.3651359

[48] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1604.06174

[49] P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, J. Gonzalez, K. Keutzer, and I. Stoica, "Checkmate: Breaking the memory wall with optimal tensor rematerialization," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 497–511, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.1910.02653

[50] V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *URL https://arxiv. org/abs/2205.05198*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2205.05198

[51] O. Beaumont, L. Eyraud-Dubois, J. Herrmann, A. Joly, and A. Shilova, "Optimal re-materialization strategies for heterogeneous chains: How to train deep neural networks with limited memory," *ACM Trans. Math. Softw.*, vol. 50, no. 2, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3648633

[52] R. Bellman, *Dynamic Programming*. Princeton University Press, 1947.

[53] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing *et al.*, "Alpa: Automating inter- and intra-operator parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 559–578. [Online]. Available: https://doi.org/10.48550/arXiv.2201.12023

[54] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1710.03740

[55] S. Zhang, N. Zheng, H. Lin, Z. Jiang, W. Bao, C. Jiang, Q. Hou, W. Cui, S. Zheng, L.-W. Chang *et al.*, "Comet: Fine-grained computation-communication overlapping for mixture-of-experts," *arXiv preprint arXiv:2502.19811*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.19811

[56] Y. Qian, F. Li, X. Ji, X. Zhao, J. Tan, K. Zhang, and X. Cai, "Eps-moe: Expert pipeline scheduler for cost-efficient moe inference," *arXiv preprint arXiv:2410.12247*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.12247

[57] AMD Advanced GPGPU-Infra Team, "Primus-Turbo: A tool for maximizing GPU utilization for training large models," https://github.com/AMD-AGI/Primus-Turbo, 2024, accessed: 2026-01-10.

[58] S. Abraham, "Frontier Architecture Overview," Oak Ridge Leadership Computing Facility (OLCF), 2024, accessed: 2024-12-15. [Online]. Available: https://www.olcf.ornl.gov/wp-content/uploads/Frontier-Architecture-Overview_Abraham.pdf

[59] D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu *et al.*, "Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models," *arXiv preprint arXiv:2401.06066*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.06066

[60] W. Liang, T. Liu, L. Wright, W. Constable, A. Gu, C.-C. Huang, I. Zhang, W. Feng, H. Huang, J. Wang *et al.*, "Torchtitan: One-stop pytorch native solution for production ready llm pre-training," *arXiv preprint arXiv.2410.06511*, vol. 6511, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.06511

[61] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.11277