DLRover-LM: LLM Pre-Training Framework with Thousands of Accelerators in AntGroup

Ziling Huang†, Zhengmao Ye†, Qingsong Cai†, Zelong Huang†,
Bo Sang*, Haitao Zhang*, Jian Sha‡, Tingfeng Lan§,
Hui Lu‖, Yuanchun Zhou¶, Mingjie Tang†
*Ant Group †Sichuan University ‡Tsinghua University
§University of Virginia ¶Chinese Academy of Sciences ‖The University of Texas at Arlington
{youxingling1, yezhengmaolove, qingsong1010, zelonghuang05, xuanying2019, t12345}@gmail.com
shaj24@mails.tsinghua.edu.cn, erc8gx@virginia.edu, hui.lu@uta.edu, zyc@cnic.cn, tangrock@gmail.com

2026 IEEE 42nd International Conference on Data Engineering (ICDE) | 979-8-3315-8365-1/26/$31.00 ©2026 IEEE | DOI: 10.1109/ICDE65706.2026.00271

***Abstract*—We introduce DLROVER-LM, an out-of-the-box production system designed for efficiently and reliably training MoE-based LLMs at scales of up to 3,000 accelerators. Training models of this magnitude presents significant challenges in efficiency, performance, and stability. To address these, DLROVER-LM first employs a practical and highly effective hybrid parallelism strategy (i.e., FSDP+EP), along with advanced optimizations such as computation-communication overlap and operator fusion, substantially improving training efficiency. Furthermore, DLROVER-LM develops a multi-agent fault diagnosis framework with node health check and flash checkpointing mechanisms, enabling rapid detection and recovery from failures. In production environment at AntGroup with a 290B MoE-based LLM across 3,072 accelerators, DLROVER-LM achieves a Model FLOPs Utilization (MFU) of 24%, comparable to state-of-the-art (i.e., Megatron-LM), while greatly reducing the downtime due to faults to only 5% of the total training time – all without complex manual configurations.**

***Index Terms*—LLM, Distributed System, Parallel Training, Fault Tolerance**

I. INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable generation across a wide range of natural language and reasoning tasks, such as text2sql [1]–[5], machine translation [6], [7], and intelligent Q&A [8], [9]. Recent advances [10] in LLMs have been largely driven by substantial increases in both the size of model parameters and the scale of training data [11]–[13]. However, as the model and training data scale continue to expand, the associated computational resources for training these models increase rapidly.

Mixture-of-Experts (MoE) models [14], which activate only a subset of parameters per input, thus requiring far fewer computational resources than dense models while achieving the same performance, have emerged as a promising approach and received widespread attention. Despite their efficiency, MoE models still demand large clusters with thousands of high-performance accelerators (e.g., GPUs); achieving efficient large-scale training for MoE models remains challenging [15], particularly due to two key bottlenecks discussed below.

Parallel Training. Because LLMs can exceed the memory capacity of individual GPUs, their parameters (as well as activations and optimizer states) are typically partitioned across multiple nodes/devices to enable distributed training [16]. Collective communication (e.g., `AllReduce` and `AllGather`) required for gradient synchronization and parameter exchange often dominates the critical path, leading to substantial training slowdowns [17]. Mitigating communication overhead requires topology-aware partitioning strategies that align tensor and pipeline splits with interconnect bandwidth and latency characteristics. As shown in Table I, under the state-of-practice 3D parallelism [16] the model parameters are partitioned across accelerators in three dimensions: data [18], tensor [19], and pipeline parallelism [20]. While 3D parallelism can potentially achieve near-linear scaling on massive clusters, as both the model size and the cluster scale grow, the search space for finding the optimal 3D parallelism execution plan expands combinatorially. For example, Megatron-LM [19], a widely used training framework, requires days of tuning effort to determine the best parallel configuration when training a 290 billion model across 3,000 accelerators.

High Failure Rate. Large-scale training mostly relies on tightly coupled synchronous updates, even a single-node failure can stall the global process until recovery, significantly extending the end-to-end training time [21]. As summarized in Table II, we have observed in AntGroup's data center that node failures are common and diverse, ranging from software/hardware failures to misconfigurations. Even if the failure rate of a single node is low, the overall failure rate of the cluster increases as the number of nodes increases [22].

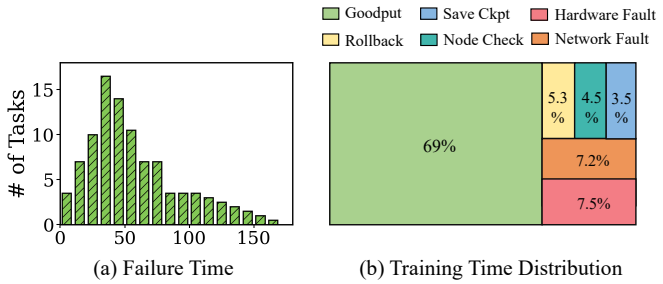
When node failure occurs, the training system must quickly detect anomalies, identify the root cause, and take the appropriate action to minimize disruption. Unfortunately, due to the complexity and diversity of failures, training systems often struggle to accurately determine the root cause and apply corresponding remedies. As a result, manual intervention becomes necessary, further increasing training time. As shown in Figure 1(a), only 2% of the training tasks in the Ant Group's cluster did not encounter failures. Additionally, as depicted in Figure 1(b), only 69% of the total time in the cluster was actually spent on model training. Similarly, Meta reported that during the two-month training process of the OPT-175B model using 992 A100 GPUs (80GB) [23], at least 35 manual restarts were required due to failures. These disruptions resulted in

2375-026X/26/$31.00 ©2026 IEEE
DOI 10.1109/ICDE65706.2026.00271

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISONS OF HFU USING 3D PARALLELISM WITH DIFFERENT CONFIGURATIONS. HFU MEASURES HARDWARE UTILIZATION.

Model Size	#GPUs	TP	PP	DP	HFU
7B	8	2	2	2	49.9%
		1	2	4	61.0%
20B	256	4	2	32	41.7%
		2	4	32	42.1%
		2	2	64	44.9%
80B	1856	8	8	29	17.1%
		4	16	29	22.7%
		4	8	58	27.4%
		...	...	...	...



(a) Failure Time (b) Training Time Distribution

Fig. 1. (a) Distribution of the failure time from multiple LLM tasks. (b) Distribution of the running time for the GLM-65B model training task on thousands of accelerators. The elapsed time includes effective training time, detailed checkpointing (i.e., ckpt) time, and different fault recovery time.

more than 20 days of wasted training time and increased computational costs by tens of millions of dollars.

To address these challenges, we introduce DLROVER-LM, a distributed training framework designed for efficient and reliable training of MoE-based LLMs across thousands of accelerators. Instead of 3D parallelism, DLROVER-LM employs a hybrid parallel strategy that integrates Fully Sharded Data Parallel (FSDP) [24] with Expert Parallelism (EP) [20], [25]. This design choice reduces parallel configuration complexity by combining FSDP's parameter sharding with EP's expert distribution, reducing the reliance on explicit, complex tensor and pipeline partitioning and thus substantially narrowing the search space for optimal parallel configurations. DLROVER-LM demonstrates that, by (1) minimizing communication overhead through effective computation-communication overlapping and (2) constructing high-performance fused operators via expert-guided local compilation, this hybrid parallel approach achieves training performance comparable to that of 3D parallelism. In addition, DLROVER-LM introduces a proactive, multi-agent-driven self-healing mechanism that entails node health monitoring, collaborative diagnostics, and flash checkpointing techniques. Together, these mechanisms enable DLROVER-LM to achieve high training efficiency and robust fault tolerance, *without* requiring complex manual configuration, exhaustive parallel tuning, or frequent human intervention during fault recovery.

DLROVER-LM has been deployed in AntGroup's data center for training LLMs of various parameter scales. In summary, the main contributions of DLROVER-LM are as follows:

- We advocate a **hybrid FSDP + EP** strategy for training large-scale MoE-based LLMs on moderate-scale clusters (e.g., around 3,000 accelerators). By leveraging communication–computation overlapping and operator fusion techniques, our approach achieves efficient end-to-end training performance while remaining easy to configure and extend.[1]
- We design a multi-agent diagnosis and recovery assistance framework. This framework aims to accurately identify the root cause of faults and achieve an efficient, automated fault recovery process through multi-agent interaction and collaborative reasoning with the computing cluster. Furthermore, we optimize the node health check mechanism and *flash checkpointing* technology to ensure rapid fault detection and recovery during large model training.
- We conducted comprehensive evaluation of DLROVER-LM on a production cluster at AntGroup. When training a 290B parameter MoE model on 3,072 accelerators, DLROVER-LM achieved a model floating point operation utilization (MFU) of 24%, with performance comparable to Megatron-LM that requires expensive fine-tuning of the parallel configuration. Additionally, in long-term production clusters, DLROVER-LM quickly and autonomously recovered from 20+ failures, ultimately achieving 95% of training time.
- We have open-sourced DLROVER-LM and shared a wealth of best practices for training LLMs. These resources enable AI researchers to efficiently train LLMs out of the box.

II. BACKGROUND AND MOTIVATION

In this section, we first provide the background of the MoE architecture as well as main parallelism strategies for LLMs training, and then discuss two key challenges encountered when training large-scale MoE models.

A. Preliminaries

Mixture of Experts (MoE) is an architecture that employs multiple specialized sub-networks (e.g., MLP), known as experts. As illustrated in Figure 2 (I), in transformer architectures, MoE typically enhances the standard Feed-Forward Network (FFN) layers. Instead of utilizing a single dense FFN, the MoE structure replaces it with multiple parallel expert sub-networks. A dedicated gating network (or router) within each MoE layer evaluates input representations to select and combine the outputs from the most suitable experts, effectively capturing specialized knowledge or skills across diverse inputs. This architecture substantially increases model capacity and versatility without proportionally raising computational cost.

3D Parallelism, or three-dimensional parallelism, refers to the integrated use of tensor [19], pipeline [20], and data [18] parallelism to scale LLM training efficiently across thousands of GPUs. Specifically, model parameters are split across devices within a layer (tensor parallelism), layers are distributed across sequential pipeline stages (pipeline parallelism), and

[1]For large-scale production training, distributed configurations can be manually tuned using Megatron-LM to enable further scaling.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

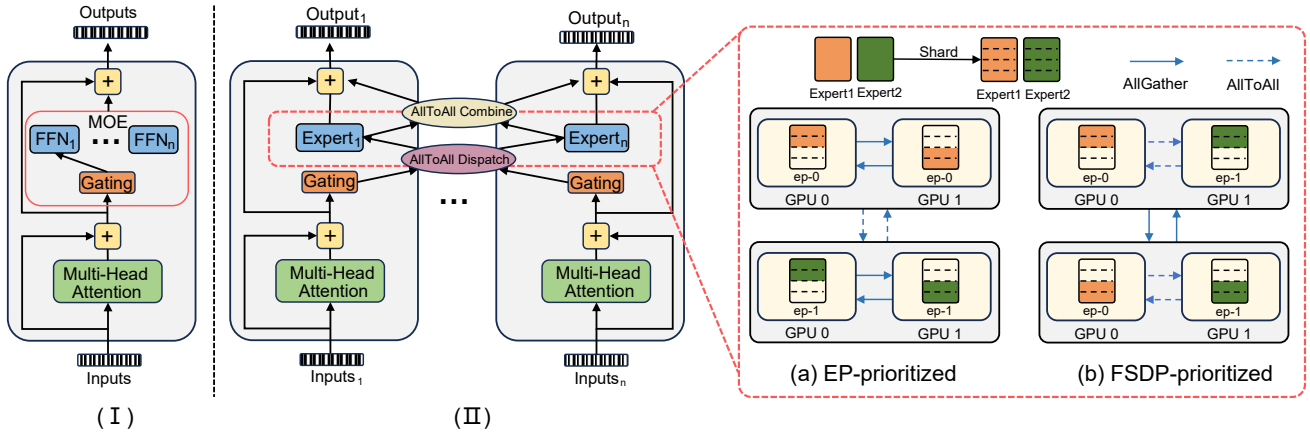


Fig. 2. (I) shows the LLM with MoE architecture. (II) depicts the expert parallelism strategy employed by the experts. (a) and (b) illustrate the different parallel execution plans when using the FSDP + EP hybrid parallel strategy.

multiple model replicas process different input batches (data parallelism). Each dimension introduces unique synchronization patterns, e.g., intra-layer `AllReduce` for TP, activation `send/recv` between PP stages, and gradient `AllReduce` across DP replicas, whose combined overhead can become the primary bottleneck at scale. While 3D parallelism can potentially achieve near-linear scaling on massive clusters, it demands careful orchestration of interconnect topology, stage balancing, and fault recovery, making its implementation and tuning highly complex in practice.

Fully Sharded Data Parallelism (FSDP) [24], [26] eliminates redundant model replication across data-parallel workers by sharding parameters, gradients, and optimizer states evenly among devices. Specifically, FSDP stores only a local shard of each tensor and reconstructs full parameters on demand during computation: Before executing a layer's forward or backward pass, FSDP performs an `AllGather` to materialize the relevant parameters across ranks, then immediately frees them after use by `ReduceScatter`ing gradients back to their owning shards. This dynamic gathering–sharding workflow reduces peak memory consumption while maintaining mathematical equivalence to standard data parallel training. The approach, however, introduces heavy collective communication overhead and synchronization complexity due to `AllGather`/`ReduceScatter` operations at every layer.

Expert Parallelism (EP) [27] is a specialized form of model parallelism tailored for MoE architectures, where only a small subset of experts is activated per input token. In EP, individual experts – typically independent feed-forward networks – are distributed across different GPUs, and tokens are dynamically routed to their assigned experts using a learned gating function. This routing requires two `AllToAll` collective operations per MoE layer: one to dispatch tokens to the correct experts and another to gather the processed outputs back to their originating ranks. While this design enables efficient scaling of sparsely activated models by distributing experts across devices, it introduces substantial communication overhead and potential load imbalance among experts. Consequently,

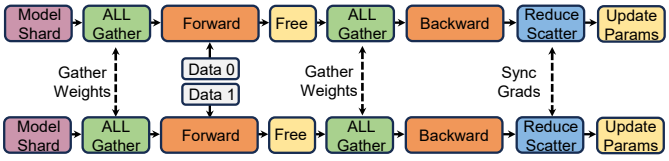


Fig. 3. FSDP partitions the model parameters across multiple accelerators and uses collective communication operations to exchange parameters and synchronize gradient updates during training.

achieving high training efficiency depends on optimizing the `AllToAll` operations (e.g., via fused collectives).

B. Key Challenges

Complexities in Parallel Training. While 3D parallelism enables fine-grained partitioning of both data and model components to scale training across thousands of accelerators, its implementation introduces deep systems-level challenges. The interplay between tensor, pipeline, and data parallelism creates nontrivial dependencies among communication, memory, and scheduling domains, making holistic optimization difficult. In practice, as summarized in Table I, different parallel configurations can yield Hardware FLOPs Utilization (HFU) variations ranging from 17% to 60%, largely driven by inter-stage pipeline bubbles, all-reduce synchronization delays, and uneven parameter partitioning across accelerators.

Efficiently realizing 3D parallelism requires sophisticated resource orchestration to balance memory usage, minimize idle time, and coordinate collective operations. However, memory fragmentation often arises because model shards differ in size across layers, while pipeline stages exhibit imbalanced compute intensities that produce stragglers. Furthermore, overlapping collectives such as `AllReduce` (DP), `AllGather` (TP), and point-to-point sends (PP) can easily saturate the interconnect fabric, especially in heterogeneous clusters. As the scale of training increases, identifying a globally optimal partitioning and communication schedule becomes combina-

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

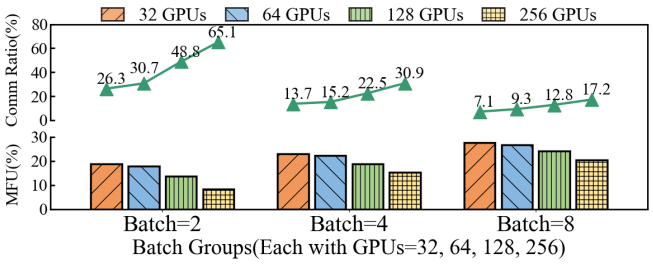


Fig. 4. When training Qwen2-57B-A14B, the efficiency of FSDP gradually decreases as the training scale increases, primarily due to the growth in communication latency.

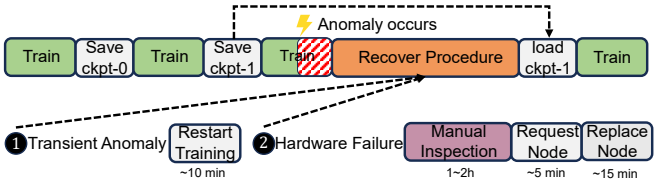


Fig. 5. The training process is usually divided into multiple phases and checkpoint-saving operations occur at the end of each phase. Recovering from a failure using checkpoints (e.g., for the GLM-65B model training) takes a significant amount of time depending on the types of failures.

torially complex, often requiring extensive tuning to approach near-optimal performance, e.g., days of tuning effort [19]

Such intricate configuration and tuning requirements become a major bottleneck for industrial teams that must rapidly iterate on new model architectures, parameter sizes, and training recipes. As a result, the demand for "ease-of-use" has driven growing interest in simpler training strategies such as FSDP, which abstracts away the need for multi-dimensional configuration and enables single-command scaling across clusters. While such simplicity greatly lowers the engineering barrier – through automatic end-to-end sharding of model states – it introduces new communication bottlenecks. In particular, at large scale, FSDP's layer-wise synchronization collectives (e.g., `AllGather`/`ReduceScatter`) dominate end-to-end training time. As shown in Figure 4, FSDP's training efficiency decreases with larger GPU counts, as Model FLOPs Utilization (MFU) declines and communication ratio rises. This tension between usability and scalability motivates us to develop new parallelization strategies that preserve FSDP's ease of deployment while mitigating its communication overhead through more efficient sharding and collective scheduling (§IV-A).

Difficulties in Fault Handling. Modern training systems, as shown in Figure 5, must periodically save model states and coordinate recovery across hundreds of nodes.

When anomalies occur, such as stalled workers or node failures, the system typically pauses and then triggers a recovery procedure that may involve process restarts or checkpoint rollbacks. Specifically, for transient errors, training can resume by restarting the process. In contrast, hardware failures, such as faulty compute nodes, commonly require manual inspection

TABLE II
CLASSIFICATION OF LLM TRAINING TASK ANOMALIES.

Type	Taxonomy	Symptoms
Immediate Fault	Algorithm Bugs	Training Failure
	Infra Errors	
	Hardware Errors	
Gradual Fault	Algorithm Bugs	Training Hang
	Hardware Aging/Overheat	
	Inappropriate Configuration	

by an expert. After diagnosing the issue, the system requests a healthy compute node to replace the faulty one and resumes training at the most recent checkpoint, i.e., a process that can take 1 to 2 hours, as shown in Figure 5.

Worse, accurately distinguishing between transient faults and persistent hardware failures is notoriously difficult. For example, in our year-long observation of hundreds of production-scale LLM training jobs (see Table II), we found that GPU-related issues accounted for 48.4% of all failures, often manifesting as inconsistent or misleading symptoms across ranks. These faults, together with software anomalies and communication hangs, expose the limitations of the common two-step recovery scheme: "restart first, replace later": (1) The system frequently misclassifies anomalies, leading to ineffective recovery actions such as repeated, futile restarts. (2) It lacks mechanisms for early risk detection – e.g., training hangs often require a timeout before being recognized as failures, delaying recovery by 15–30 minutes.

Fundamentally, fault handling at this scale is hard because training involves asynchronous collectives, heterogeneous hardware health signals, and incomplete observability across nodes. An effective fault tolerance framework must therefore fuse multi-source diagnostic signals – e.g., from logs, metrics, and health monitors – and perform adaptive recovery based on real-time inference of fault type and severity (§IV-B).

III. System Overview

We introduce DLRover-LM, an efficient and reliable platform for LLM training, designed to achieve two key objectives: (1) *Ease-of-use with high performance.* DLRover-LM provides simple access to end-users, allowing them to focus on the algorithms without delving into system optimizations, while maintaining high throughput and hardware utilization. (2) *Fault tolerance.* DLRover-LM can quickly and accurately identify faults and automatically perform recovery operations to ensure the continuity of large-scale distributed training and minimize the impact of training interruptions.

System Architecture. Figure 6 shows the architecture of DLRover-LM. First, DLRover-LM provides a thin Trainer API, serving two primary functions: it enables the configuration of the computational cluster and allows users to submit training models and training datasets. The simplicity of this API allows users to focus exclusively on algorithm

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

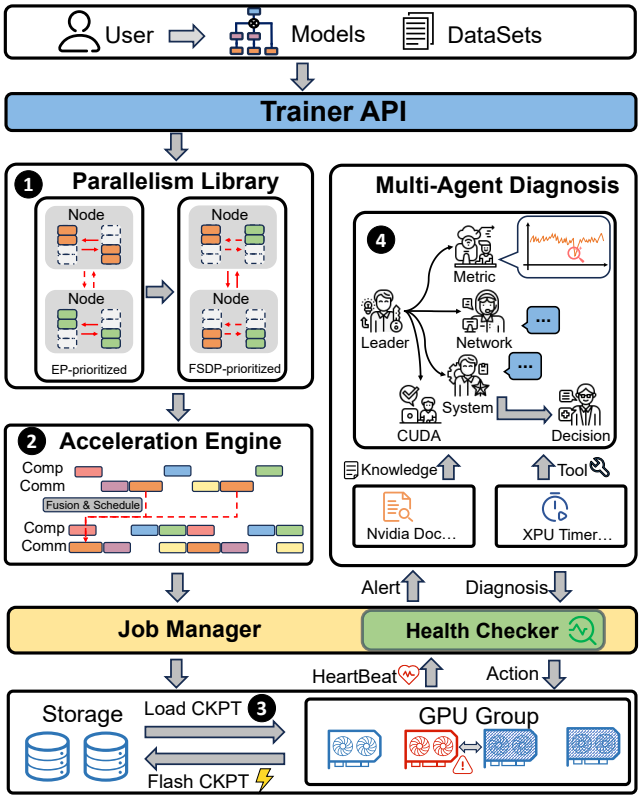


Fig. 6. Overview of DLRover-LM.

development, eliminating concerns regarding the complexities of system configuration and optimization. Upon submission of the user's algorithm-focused code, DLRover-LM employs a hybrid parallelism approach (detailed in §IV-A), partitioning the model into segments distributed across multiple GPUs. This approach optimizes resource utilization and accelerates the training process. To further improve efficiency, DLRover-LM's training engine incorporates optimizations at both communication (§IV-A2) and computational operator levels (§IV-A3), thereby significantly reducing computation latency and communication overhead.

During the training of a model training task, the health checker module is tasked with monitoring the cluster's status and collecting information from the cluster (§IV-B1). This information is then relayed to the job manager. Based on the collected metrics and the training status, the job manager decides on task management strategies, such as triggering efficient node replacements and checkpoint recovery (§IV-B3). When an anomaly occurs, the job manager notifies the LLM-based multi-agent system (§IV-B2) of the anomaly information. The multi-agent system then conducts an in-depth analysis of the fault, relying on the information provided by the job manager, and provides a precise solution.

IV. Design of DLRover-LM

We first introduce DLRover-LM's hybrid parallel strategy with the combination of FSDP and EP to train LLMs with the MoE architecture (§IV-A). Next, we present DLRover-LM's self-healing mechanism (§IV-B) that achieves efficient fault detection and recovery through proactive fault detection, multi-agent anomaly diagnosis, and fast checkpointing.

A. Hybrid FSDP + EP Parallelism Strategy

As discussed in Section §II-B, hybrid parallelism offers a flexible design space for scaling LLM training, yet selecting and tuning the optimal combination remains nontrivial. We therefore focus on a hybrid strategy combining FSDP and EP, motivated by two key observations. First, EP naturally decomposes the global communication domain into smaller local groups, reducing the synchronization overhead that limits FSDP scalability. Second, both FSDP and EP expose compact and well-defined execution plan spaces, simplifying automatic configuration relative to 3D parallelism. In our production experiments (§V-B), this hybrid FSDP + EP scheme achieves comparable throughput to carefully tuned 3D parallel setups while requiring substantially less tuning effort.

1) **EP- vs. FSDP-Prioritized Hybrid Strategy:** When using a hybrid parallel strategy with FSDP and EP, we need to consider two schemes: *(i) EP-prioritized* strategy (Figure 2(a)): All experts in the MoE layer of the model are first allocated to different computational nodes for expert-level parallel computation. Subsequently, within each node, the assigned experts are partitioned across different accelerators. *(ii) FSDP-prioritized* strategy (Figure 2(b)): All experts in the MoE layer are first sliced into multiple segments. Then, a portion of the segments from each expert is selected and placed at the same node for computation. Both schemes incur non-trivial communication overhead for the MoE layer computation.

In the *EP-prioritized* scheme, tokens are first routed to their assigned experts with an *inter-node* `AlltoAll` operation, followed by an *intra-node* `AllGather` operation that assembles fragmented expert parameters (distributed across accelerators in the same node) for expert computation. Once completed, the output tokens are routed back to their original nodes using another *inter-node* `AlltoAll` operation. In contrast, the *FSDP-prioritized* scheme first routes tokens using an *intra-node* `AlltoAll` operation. Then, it aggregates fragmented expert parameters through an *inter-node* `AllGather` operation, forming complete expert parameter sets for computation. Finally, outputs are redistributed back to their original accelerators via another *intra-node* `AlltoAll` operation.

The relative efficiency of the two strategies depends primarily on communication cost, which in turn varies with hardware topology and model scale [28]. We formalize the communication cost as follows:

Communication Cost. We define the communication volume of `AllGather` as: $C_\alpha = P \times (N-1)/N$, where P is the number of parameters per MoE layer and N represents the number of accelerators. The volume of `AllToAll` is: $C_\beta = D \times K \times H \times (N-1)/N$, where D is the size of training data, K is the number of experts to which each token is routed, and H represents the hidden size of the FFN layer among the experts.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

Therefore, when employing the FSDP + EP hybrid parallelism strategy, the total communication volume is: $C = C_\alpha + 2C_\beta$.

We use B_a to denote the *inter-node* communication rate and B_i to denote the *intra-node* rate; communication latency is the volume of communication divided by the communication rate. Hence, for the *EP-prioritized* scheme, the latency is represented as: $T_{ep} = C_\alpha/B_i + 2C_\beta/B_a$, while for the *FSDP-prioritized* scheme, the latency is: $T_{fsdp} = C_\alpha/B_a + 2C_\beta/B_i$.

Discussions. In our production environment (§V-B), we observe that (1) the communication volume of `AllGather` (C_α) is typically smaller than that of `AlltoAll` (C_β) and (2) inter-node communication bandwidth is significantly lower than intra-node bandwidth. For instance, in representative commercial GPU clusters at AntGroup, NVLink provides up to 900 GB/s intra-node bandwidth, whereas InfiniBand interconnects offer only 50 GB/s inter-node bandwidth. Under these conditions, the FSDP-prioritized hybrid strategy is preferable because it confines most `AllToAll` operations within nodes, minimizing cross-node communication overhead. Consequently, DLROVER-LM adopts the *FSDP-prioritized* hybrid parallel strategy by default to ensure training efficiency and scalability. Nonetheless, when inter-node bandwidth becomes higher or model size grows such that C_β dominates intra-node traffic, the EP-prioritized variant becomes more advantageous. In such cases, DLROVER-LM can switch to EP-prioritized operation based on hardware profiling, ensuring adaptability across diverse cluster environments.

Although the FSDP-prioritized strategy confines most collective operations within nodes to improve communication locality, its native implementation still exhibits noticeable performance gaps – primarily from inter-node `AllGather` communication latency and inefficient operator execution. These findings motivate two targeted optimizations presented next: (1) hiding communication cost through overlapped scheduling of computation and communication, and (2) constructing high-performance operators to enhance kernel-level efficiency.

2) **Hiding Communication Cost:** Inter-node communication overhead due to relatively low cross-node network bandwidth remains a major bottleneck in FSDP-prioritized LLM training. To mitigate this, we introduce an optimized overlap strategy that hides `AllGather` latency by scheduling communication concurrently with preceding computations.

As shown in Figure 7(a), consider the interaction between the operations $E1$ and AG_{E1}. Here, the $E1$ operation cannot commence until the `AllGather` operation completes, since $E1$ relies directly on the results from AG_{E1}. Consequently, there is no overlap between these two operations. In contrast, the communication operation AG_{E1} is independent of its preceding computational operations and can thus be started earlier. By scheduling AG_{E1} concurrently with the preceding computational operation $A1$, as depicted in Figure 7(b), we create an effective overlap between communication and computation, effectively masking the overhead incurred by the `AllGather` communication operation.

However, a key challenge arises: the `AllGather` operations introduce additional memory overhead for temporarily storing communication data, which cannot be released until the dependent computational operations finish execution. Consequently, careful scheduling of `AllGather` operations is essential to ensure that peak memory usage stays within the device's available capacity. Moreover, multiple `AllGather` communication operations can be fused into a single, more efficient `AllGather` operation. This fusion reduces the overhead associated with initiating numerous collective communications and enhances network bandwidth utilization by batching data transfers. In essence, the primary challenge is striking an optimal balance between memory limitations and maximizing the fusion of `AllGather` operations to minimize overall communication overhead.

To determine the optimal overlap pattern, we formulate communication–computation scheduling as a constraint-based optimization problem that minimizes total execution time while satisfying dependency and memory-capacity constraints [29]. We solve this model using Google OR-Tools [30], an off-the-shelf operations research solver, to derive the optimal start times and fusion groupings for `AllGather` operations. The resulting execution schedule is then integrated into DLROVER-LM as a static runtime pattern to enforce efficient overlapping during training.

Constraint-based Scheduling Model. Specifically, we first profile the latency O^l of each operator O under the non-overlapping baseline, as illustrated in Figure 7(a). We then introduce a decision variable O^s, representing the start time of each operator, constrained by dependency and resource availability relationships. The model, defined in Equation 1, captures when independent operators can execute concurrently while respecting memory and ordering constraints. Therefore, the optimization objective is to minimize the completion time of the final operator Cn^s, thus to reduce the total pipeline execution time.

$$
\begin{aligned}
&Ei^s \geq Di^s + Di^l, && i = 1, \cdots, n \\
&Ei^s \geq AG^s_{Ei} + AG^l_{Ei}, && i = 1, \cdots, n \\
&Ai^s \geq C^s_{i-1} + C^l_{i-1}, && i = 2, \cdots, n \\
&Ai^s \geq AG^s_{Ai} + AG^l_{Ai}, && i = 2, \cdots, n \\
&Gi^s \geq Ai^s + Ai^l, && i = 1, \cdots, n \\
&Di^s \geq Gi^s + Gi^l, && i = 1, \cdots, n \\
&Ci^s \geq Ei^s + Ei^l, && i = 1, \cdots, n
\end{aligned}
\tag{1}
$$

Based on the optimized operator start times obtained from the scheduling model, we fuse consecutively executed `AllGather` operators into a single, more efficient collective. We then validate whether the resulting overlap pattern fits within device memory limits; if not, we iteratively adjust the schedule to avoid out-of-memory conditions. In practice, this optimization yields an overlap configuration that effectively hides 100% of the communication latency of `AllGather` operations, as shown in Figure 7(b). More specifically, we fuse the `AllGather` operations for retrieving parameters of the nth layer Expert and the $(n+1)th$ layer Attention, overlapping them with the attention and gate computation of the nth layer.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

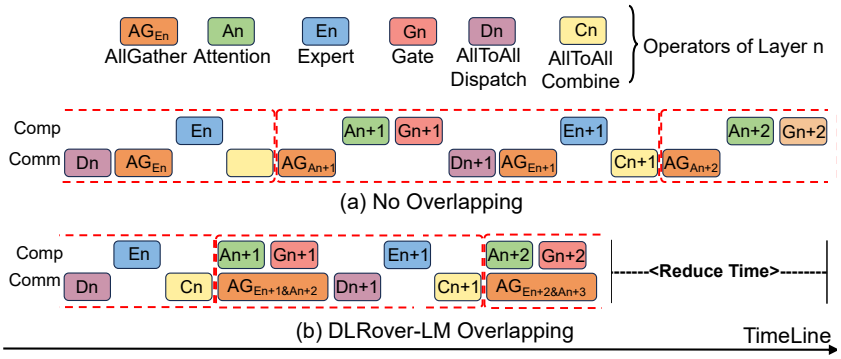


Fig. 7. Timelines of communication operator scheduling w/ and w/o overlapping – the length of each operator represents its time cost, while multiple rows indicate parallel CUDA streams.

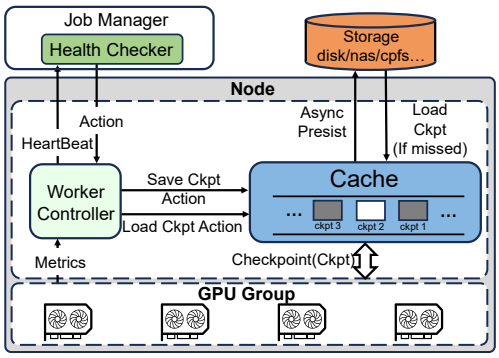


Fig. 8. DLROVER-LM's mode health monitoring and fault recovery mechanisms.

Benefit Analysis. The overlapping pattern in DLROVER-LM substantially reduces communication latency by overlapping `AllGather` operations and fusing two consecutive collectives into a single launch. Compared to the non-overlapping baseline, this design eliminates most communication idle time and cuts collective-launch overhead by roughly half.

3) **Constructing High-Performance Operators:** Optimizing computational operators is essential for improving overall training performance. While experts can manually craft highly optimized custom kernels like FlashAttention [31], [32], this approach lacks scalability and struggles to keep pace with rapidly evolving model architectures in production environments. Conversely, relying solely on full-graph compilation using tools like XLA [33] often proves impractical due to excessive compilation times (e.g., over 30 minutes for a 7B model) and the need for frequent recompilation upon input changes, hindering rapid development cycles.

To bridge this gap, we adopt an *expert-guided local compilation strategy*. Experts first leverage performance profiling tools (e.g., Nsight Systems [34]) to pinpoint computational bottlenecks. Instead of attempting whole-graph optimization, they selectively annotate only these critical sub-graph regions for localized compilation by XLA. The resulting optimized operators can then be further fine-tuned by experts for precise adaptation. For example, we identified that individual Dropout, Add, and LayerNorm operators underutilized accelerator resources. After expert annotation, XLA generated a fused operator integrating these three operations, reducing kernel launch overhead and improving memory access efficiency.

Complementing XLA's graph-level strengths, we integrate Triton [35] for rapid prototyping and debugging of custom CUDA kernels at the operator level. This is particularly beneficial as XLA typically generates C++ code, which can have high maintenance costs and limited portability across different accelerator types. We leverage Triton for secondary development of key operators like Cross-entropy and RMSNorm, enhancing code portability and reducing maintenance overhead. This synergistic approach, combining Triton's kernel-level agility with XLA's localized graph optimization capabilities, delivers efficient and scalable operator performance without sacrificing development velocity.

Summary. DLROVER-LM's FSDP-prioritized hybrid parallelism effectively balances scalability, efficiency, and ease of deployment. By prioritizing intra-node communication locality and introducing overlap and operator optimizations, it hides most inter-node `AllGather` stalls and achieves near-optimal utilization, thus improving training throughput. In production-scale runs, this design achieves performance comparable to carefully tuned 3D-parallel systems while requiring far less configuration complexity and tuning effort (§V).

B. Multi-Agent, Collaborative Self-Healing

To achieve fast and efficient fault detection and recovery, we present a self-healing runtime system for DLROVER-LM based on a multi-agent architecture, as illustrated in Figure 6. The system comprises two main components: a *health check* module, which continuously monitors node states and pre-validates candidate replacements to avoid cascading failures (§IV-B1); and a *multi-agent diagnosis system*, which coordinates distributed agents to localize faults and select recovery strategies (§IV-B2). Additionally, a lightweight flash-checkpointing mechanism (§IV-B3) enables rapid saving and restoration of model states, reducing downtime during recovery. Together, these components provide automated, collaborative fault detection and repair, thus maintaining high training availability with minimal manual intervention.

Workflow. As shown in Figure 6, DLROVER-LM employs a hierarchical, multi-agent workflow for real-time fault detection and recovery. *(i)* Each GPU group periodically reports runtime metrics and heartbeats to the job manager's health check module, which performs continuous monitoring and anomaly detection. Missing or irregular heartbeats trigger an alert to the multi-agent diagnosis system. *(ii)* Upon receiving an alert, distributed diagnosis agents analyze runtime traces and auxiliary telemetry to identify potential root causes and propose recovery actions. *(iii)* The multi-agent coordinator selects and issues an appropriate self-healing plan, such as isolating a faulty node or reallocating workloads, to the Job Manager. *(iv)* Finally, the Job Manager executes the recovery procedure, e.g., with the flash-checkpointing mechanism to restore model states and resume training with minimal disruption.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

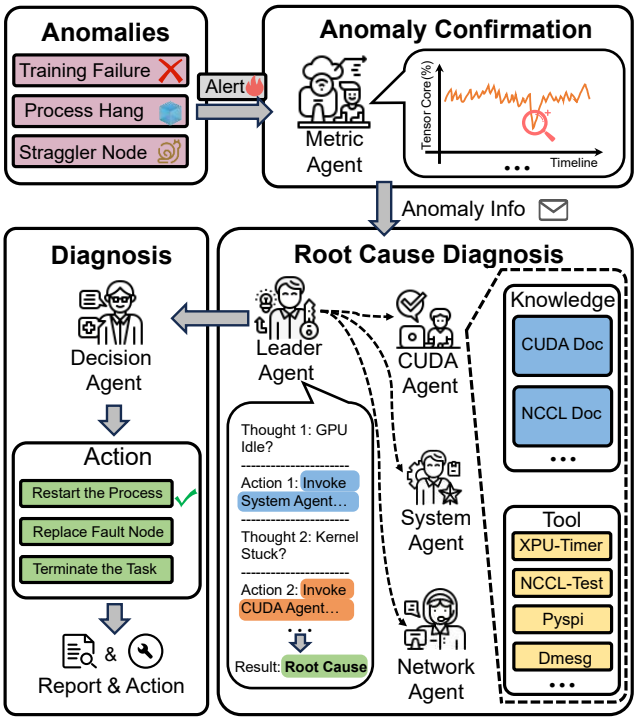


Fig. 9. DLROVER-LM's multi-agent, collaborative self-healing system.

1) **Node Health Monitoring and Fault Detection:** To ensure timely fault detection in large-scale clusters, DLROVER-LM employs a two-tier monitoring design combining both passive and proactive validation.

At the passive layer, each GPU group periodically (every few seconds) sends heartbeat packets containing hardware and system metrics – such as GPU temperature, SM utilization, and NCCL/CUDA logs – to its worker controller, which forwards them to the health check module, as illustrated in Figure 8. The module flags nodes as failed when heartbeats are missing or abnormal (e.g., fault-related information within the heartbeat payload), and reports fault metadata to the multi-agent diagnosis system for analysis.

Because simple heartbeat loss cannot capture gradual performance degradation (e.g., performance degradation in communication or computation due to accelerator or network card issues), DLROVER-LM adds a *pre-check mechanism* for proactive validation. First, during cluster initialization, nodes perform lightweight compute (e.g., `Gemm`) and communication (e.g., `AllGather`) probes in groups to verify readiness. Suspect nodes undergo secondary verification with known-healthy peers before being marked faulty. Further, during training, idle nodes are pre-validated with the same tests before replacing failed ones, ensuring immediate readiness. The pre-check mechanism also runs periodic speed and performance audits to detect latent degradation, maintaining long-term cluster reliability. This way, these dual mechanisms enable early fault discovery and stable training continuity, while minimizing false detections and recovery delay.

2) **Multi-Agent Anomaly Diagnosis:** As discussed in Section II-B, diagnosing training anomalies in large-scale LLM clusters is notoriously difficult, because a single symptom (e.g., a training hang) may stem from different sources (e.g., transient network congestion, GPU hardware degradation, or CUDA kernel deadlocks). Traditional rule-based diagnosis tools struggle with such complexity, since static rules cannot effectively distinguish overlapping causes or evolving system states. Manual inspection, in contrast, is too slow and costly at production scale. To address these limitations, DLROVER-LM develops a multi-agent anomaly diagnosis system inspired by collaborative human debugging (similar in spirit to the D-Bot system [36], yet distinct in scope, scale, and purpose).

Each agent represents a specialized "domain expert", focusing on a subsystem such as Network, System, or CUDA Kernel. Agents collect and analyze subsystem-specific signals (e.g., metrics, logs, and traces) using dedicated diagnostic tools (e.g., NCCL traces, CUDA logs, XPUTimer [37]). A Leader Agent coordinates this team, synthesizing multi-modal evidence into a unified root-cause hypothesis that no single agent could infer alone. As illustrated in Figure 9, the diagnosis proceeds in three collaborative phases:

Step 1. Anomaly identification and team formation: Upon receiving an anomaly alert from the health check module, the Metric Agent performs quick triage, examining tensor-core utilization, GPU temperature, and communication throughput to verify that the anomaly is genuine. The Leader Agent then reviews historical records (e.g., recent NCCL or GPU failure logs) to hypothesize likely fault domains and forms a task-specific agent team. Because domain-specific diagnostic data are scarce, DLROVER-LM fine-tunes its internal LLM models on curated diagnostic documents and log annotations, enabling the agents to reason over highly technical artifacts beyond what generic LLMs can interpret.

Step 2. Collaborative expert analysis: Under the Leader's coordination, domain experts perform coarse-to-fine investigation across their respective subsystems. Agents use specialized debugging utilities (e.g., XPUTimer) and consult pre-registered diagnostic corpora (e.g., NVIDIA driver documentation, CUDA API traces, NCCL runtime logs). Note that each tool's APIs, usage methods, and return data types, along with document cross-references, are pre-configured within the tool and document libraries. The Leader aggregates partial findings and iteratively refines hypotheses, triggering additional queries or targeted measurements when ambiguity remains.

As a concrete example, Figure 10 shows how DLROVER-LM's Leader Agent performs chain-of-thought reasoning during the diagnosis of a representative "Process Hang" failure. The process begins with the leader distinguishing between user-space and kernel-space stalls by consulting the System Agent via `pyspi` stack tracing. Once the kernel-space hang is confirmed, the leader invokes both System and Network Agents to eliminate infrastructure causes through `dmesg` log and connectivity analyses. When these checks reveal normal GPU and network status, the leader hypothesizes a synchronization barrier fault and requests evidence from the CUDA

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

[Initial State] Anomaly Confirmed: Process Hang

[Reasoning r_1] Distinguish user-space vs. kernel-space hang. Check Python stack first.
[Action] Invoke System Agent: `pyspi` stack trace.
[Observation] Process state: In Kernel.

[Reasoning r_2] Kernel hang confirmed. Rule out infra failures (GPU, Network).
[Action] Invoke System Agent: Check `dmesg` for GPU errors. Invoke Network Agent: Check connectivity.
[Observation] GPU status: OK. Network status: OK.

[Reasoning r_3] Kernel hang likely at sync barrier.
[Action] Invoke CUDA Agent: Trace execution (`XPU-Timer`).
[Observation] Ranks 0, 2-7 blocked in `synchronize`; Rank 1 absent (stalled/suspended).

[Reasoning r_4] Root Cause: Rank 1 stalled, preventing collective sync completion (deadlock).
[Action] Report: Rank 1 Stall $\rightarrow$ Collective Block.

Fig. 10. Multi-agent collaborative reasoning for diagnosing a process hang.

Agent, which traces kernel execution using `XPU-Timer`. The collected trace indicates that ranks 0, 2–7 are blocked in a `synchronize()` call while rank 1 is absent, revealing a worker stall that prevents collective completion.

Step 3. Strategy generation and execution: Once the root cause is confirmed, a Decision Agent selects the most suitable recovery strategy, such as restarting a failed process, reallocating tasks, or isolating a degraded node, balancing recovery time, resource stability, and cost. Each decision, diagnostic interaction, and execution result is automatically logged and incorporated into a persistent knowledge base, which continuously improves future diagnostic accuracy. The logs are also made visible to operators for transparency, allowing them to audit the automated reasoning process and intervene when necessary.

3) **Flash Checkpointing:** Our operational analysis revealed that the majority of training interruptions (e.g., 75%) are caused by short-lived, transient faults, such as NCCL timeouts and network jitter, that rarely require full remote recovery. Yet, native checkpointing implementations tend to treat all faults equally, forcing every checkpoint to flush through slow remote storage. This not only wastes I/O bandwidth but also prolongs recovery cycles. To address this, we invent a flash checkpointing mechanism that (i) leverages local memory caching to sustain high-frequency, low-latency persistence, and (ii) integrates tightly with DLRover-LM's self-healing agents to adapt recovery behavior to detected fault types.

As shown in Figure 8, flash checkpointing involes a CPU-side cache module between accelerators and remote storage. During training, runtime states are streamed to a worker monitor, which triggers the cache module to retrieve checkpoint data from GPUs and temporarily store them in host memory via PCIe – avoiding long network transfers. These cached states are then asynchronously flushed to remote storage in the background, ensuring that checkpointing never blocks computation. This enables sub-second checkpointing operations (e.g., 0.2. seconds), even for multi-gigabyte model states.

Upon fault detection, DLRover-LM selects recovery strategies based on fault type. For transient failures (roughly 75% of observed cases) the system simply reloads the cached checkpoint locally, bypassing remote reads and resuming training within seconds. For severe node failures, checkpoints are reloaded from remote storage into the cache module on replacement nodes before GPU synchronization. This fault-aware reuse of cached checkpoints balances rapid recovery for soft faults with reliability guarantees for hard failures.

Summary. The self-healing design of DLRover-LM, integrating hierarchical health monitoring, multi-agent diagnosis, and flash checkpointing, achieves reliable, low-latency fault recovery at scale. The hierarchical monitoring and collaborative diagnosis components enable early fault detection and accurate classification, while flash checkpointing provides fault-adaptive recovery with minimal training interruption.

TABLE III
SPECIFICATIONS OF DIFFERENT AI ACCELERATORS.

Device	Peak FLOPS(T)	Memory(GB)	NVLink(GB/s)
A	989	140	900
B	989	80	900
C	312	80	600

TABLE IV
MODEL PARAMETER CONFIGURATION.

Model Name	Head Num	Expert Num	Hidden Size	Intermediate Size	Expert Size	Layer Num
Qwen2-57B	28	64	3584	18944	2560	28
Qwen3-235B	40	128	4096	12288	1536	94
Ling-Plus-290B	56	64	5376	12288	3072	88

V. EVALUATION

We have implemented DLRover-LM with ~30K lines of Python and ~5k lines of Go. DLRover-LM is open source and available publicly at: https://github.com/intelligent-machine-learning/dlrover. DLRover-LM has been deployed in AntGroup's production environment, operating stably since January 2024 on an HPC cluster with over 3,000 accelerators. In this section, we show that DLRover-LM achieves comparable training performance as state-of-the-art training frameworks (e.g., Megatron-LM), while substantially reducing trial-and-error tuning efforts, providing robust fault tolerance, and significantly improving stability and anomaly recovery times in real-world production environments.

A. Experimental Setup

Testbed. We evaluated DLRover-LM on a production HPC cluster at AntGroup. The cluster comprises 3,072 accelerators

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

TABLE V
STRONG-SCALING TRAINING PERFORMANCE FOR THE 290B MOE MODEL.

System	#Accelerators	Iteration Time(s)	Throughput (tokens/s)	Trial-and-error	
				Times	Time (min)
Megatron-LM	256	132.4	360.5k	6	60–90
	512	68.8	727.5k	7	90–130
	1024	35.1	1428.4k	10	150–220
	2048	18.7	2681.7k	12	240–360
	3072	13.3	3770.5k	16	400–600
DLRover-LM	256	131.0	372.1k	1	8–10
	512	69.0	727.0k	1	10–15
	1024	35.7	1402.6k	1	15–20
	2048	19.3	2596.8k	2	40–60
	3072	14.0	3582.7k	2	50–70

interconnected via high-speed InfiniBand networks. Unless noted otherwise, Device C (Table III) was used.

Workloads. We employed three models with different parameter scales: Qwen2-57B [38], Qwen3-235B [39], and our own Ling-Plus-290B [40]. Table IV provides detailed configurations for the MoE models used in our evaluation. To ensure fair comparisons, we consistently used the same batch size, and the sequence length was fixed at 4096.

Comparison Baselines. To comprehensively assess the performance of DLROVER-LM, we select two representative parallel training frameworks as comparison baselines: (1) Megatron-LM [19], a state-of-the-art open-source LLM training framework, which integrates multiple parallelism techniques. (2) PyTorch FSDP-V2 [41], the standard implementation of pure FSDP, serving as a direct baseline for our hybrid strategy and optimizations.

Evaluation Metrics. We evaluated DLROVER-LM across multiple dimensions: computational efficiency, end-to-end performance, usability, and stability. Our primary metric for computational efficiency is Model FLOPs Utilization (MFU), which measures the ratio of achieved FLOPs to theoretical hardware FLOPs. End-to-end performance and scalability are measured via iteration time and throughput (tokens/s). Usability is assessed by the estimated configuration tuning effort (attempts and time). System stability and fault tolerance are evaluated using metrics including Mean Time To Recovery (MTTR), diagnostic accuracy, and overall training goodput .

B. Performance Study

Scalability. Table V summarizes the strong-scaling training performance of Megatron-LM and DLROVER-LM on the 290B MoE model. We scaled the number of accelerators while keeping the global batch size fixed. Across all settings, the training tasks supported by Megatron-LM were optimized through multiple rounds of expert-driven 3D parallelism configurations to attain nearly optimal training performance. In contrast, DLROVER-LM– employing the FSDP+EP hybrid parallelism – achieves comparable performance (e.g., within 5% of the iteration time and throughput differences with 3072 accelerators) without a complex configuration. Meanwhile,

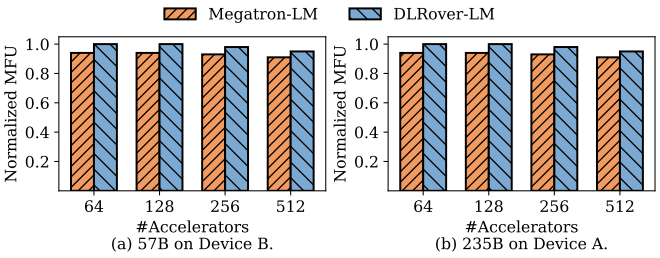


Fig. 11. Weak-scaling comparison (normalized MFU) of DLROVER-LM vs. Megatron-LM on different hardware.

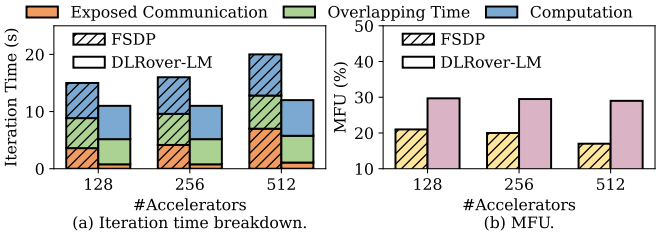


Fig. 12. Performance gain breakdown comparing DLROVER-LM against baseline FSDP for the 235B model on Device A.

DLROVER-LM sustains the model's convergence, as depicted in Figure 14(a) – the training loss curves between DLROVER-LM and Megatron-LM overlap to each other.

Crucially, DLROVER-LM drastically reduces the configuration overhead. As Table V indicates, through representative estimates based on our production experience, achieving optimal configuration with Megatron-LM typically requires 6 to 16 tuning attempts and hours of accelerator runtime (e.g., 6.7-10 hours at 3072 accelerators). In contrast, DLROVER-LM, due to its greatly simplified configuration space, typically requires only 1 to 2 attempts, converging to an efficient setup in tens of minutes. Moreover, the tuning "Time (min)" primarily reflects accelerator runtime and doesn't fully capture the extensive engineering analysis required between Megatron-LM runs, nor potential delays from resource contention in shared clusters. In contrast, DLROVER-LM's minimal tuning significantly alleviates these substantial hidden operational overheads.

Figure 11 presents the weak-scaling performance, where we scaled the global batch size proportionally with the number of accelerators (from 64 to 512) to maintain a constant workload per accelerator. In these experiments, DLROVER-LM demonstrates competitive throughput, achieving 1.05-1.06× that of Megatron-LM, indicating its efficiency at these scales. While performance scales reasonably well, a slight decline in DLROVER-LM's normalized MFU is observable as the cluster size increases (from 64 to 512 GPUs). This is expected in weak-scaling scenarios due to the increasing overhead of global communication operations (e.g., `AllGather` in FSDP), which typically increase with the number of participants, reducing the proportion of time spent on computation.

Performance Gain Breakdown. To understand the sources of DLROVER-LM's performance gains over the baseline FSDP-

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
MFU IMPROVEMENT BREAKDOWN WHEN TRAINING THE 290B MODEL.

Idx	Method	MFU (Δ MFU)
1	baseline	8.7%
2	(1) with expert parallelism	18.0% (9.3% ↑)
3	(2) with efficient operators	19.2% (1.2% ↑)
4	(3) with overlap optimization	23.0% (3.8% ↑)

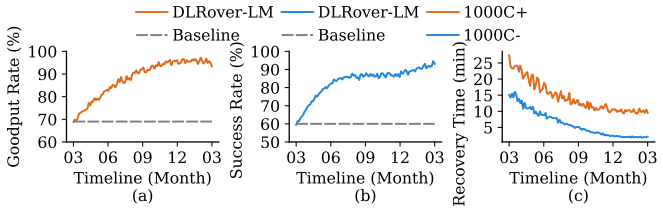


Fig. 13. Yearlong trends of (a) goodput rate, (b) fault-tolerance success rate, and (c) average anomaly recovery time using DLROVER-LM compared to the baseline.

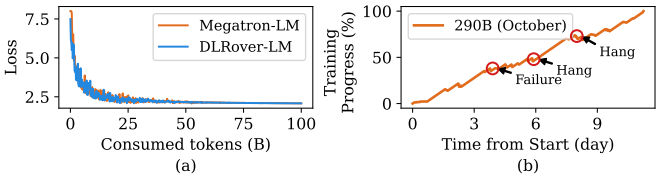


Fig. 14. (a) The training loss curve of DLROVER-LM compared with Megatron-LM. (b) The training progress of the 290B LLM, highlighting fault recovery events.

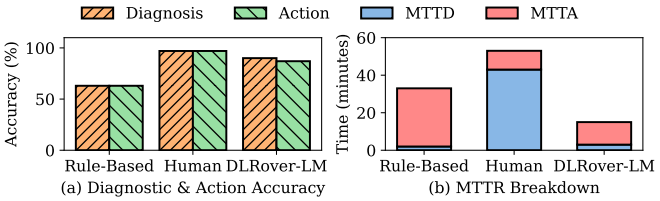


Fig. 15. Diagnosis system performance comparison (DLROVER-LM vs. Rule-Based vs. Human Expert) on 30 real-world cases.

V2, we present a detailed breakdown analysis in Figure 12. DLROVER-LM consistently achieves higher MFU, outperforming FSDP-V2 by up to 1.57x as shown in Figure 12(b). The iteration time breakdown, as shown in Figure 12(a), explains why: DLROVER-LM significantly reduces exposed communication time (i.e., communication not hidden by computation) compared to FSDP-V2. This reduction is primarily achieved through our hybrid FSDP+EP strategy combined with effective communication-computation overlapping .

Ablation Study. To access the effectiveness of each optimization component in DLROVER-LM, we conducted step-by-step ablation experiments. As depicted in Table VI, for a task executed on 3,072 accelerators, the baseline scheme based on FSDP-V2 attains only 8.7% MFU, primarily constrained by the cross-node communication overhead. By implementing a hybrid FSDP + EP parallel strategy (i.e., the FSDP-prioritized scheme), the MFU increases to 18.0% (+9.3%). The training speed is further enhanced by introducing optimized high-performance operators (+1.2%). Finally, with the computation-communication overlapping technique (+3.8%), the communication overhead is largely hidden, resulting in an MFU of 23.0%. Therefore, with all these optimizations, DLROVER-LM achieves a 1.64x relative performance improvement compared to the baseline.

C. Training Stability

Production Adoption and Evaluation. DLROVER-LM has been deployed in AntGroup's production cluster since January 2024. Figure 13 depicts the variations in goodput rate, fault-tolerance success rate, and average anomaly recovery time for multiple LLM training tasks in the production cluster from January 2024 to January 2025.

(1) *Improved goodput rate.* As illustrated in Figure 13(a), the goodput rate within the cluster experiences a notable increase due to DLROVER-LM's fault-tolerance capabilities (§IV-B). In comparison to the earlier (with a more basic stability system), the average goodput rate of LLM rises from 69% to 95%. (2) *Enhanced fault tolerance.* Figure 13(b) demonstrates that DLROVER-LM significantly boosts the fault-tolerance success rate, increasing it from the initial 60% to 93%, due to its accurate multi-agent anomaly diagnosis mechanism. (3) *Reduced anomaly mean repair time.* As shown in Figure 13(c), DLROVER-LM substantially shortens the anomaly-repair duration of LLM training tasks. *(i)* For tasks requiring fewer than 1,000 accelerators, the average anomaly-repair time is reduced from approximately 15 minutes to 2 minutes with DLROVER-LM. *(ii)* For tasks requiring more than 1,000 accelerators, the average anomaly-repair time is decreased from around 25 minutes to 10 minutes.

Production Task Tracking. We traced a two-week production task for the 290B model on 3,000+ accelerators, as depicted in Figure 14(b). During this run, DLROVER-LM automatically detected and rectified over 20 irregularities. For instance, it diagnosed a "Failure" as a GPU malfunction and triggered a node replacement. It also detected a "Hang" (network congestion) via XPUTimer [37] in 1 minute, avoiding a 30-minute timeout, and diagnosed another "Hang" as a subtle CUDA synchronization error. Overall, the system achieved 97% training uptime and 95% effective training time.

Pre-Check Mechanism Evaluation. For anomalies in a training task using 1024 accelerators, the earlier manual inspection approach requires approximately 2 hours. In contrast, DLROVER-LM's pre-check mechanism (§IV-B1) can identify the faulty node in approximately 5 minutes with its group detection method. This pre-check mechanism guarantees stability at the onset of task initiation and decreases the number of anomalies in the cluster by around 15%.

Multi-Agent Based Diagnosis. To evaluate the effectiveness and efficiency of our multi-agent system (§IV-B2), we

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

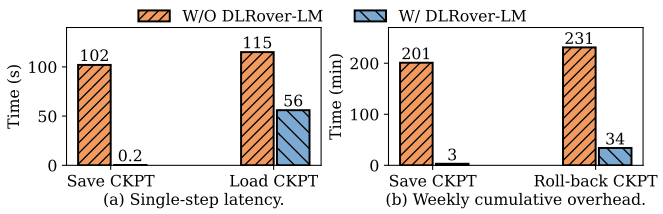


Fig. 16. (a) Time required for saving and loading checkpoints during the training of the 57B model. (b) The accumulated checkpointing time of the 57B model over a week.

benchmarked its performance against a traditional rule-based engine and human experts using 30 real-world anomaly cases. We measured diagnostic accuracy, action accuracy, and Mean Time To Recovery (MTTR), with results shown in Figure 15. DLROVER-LM achieves diagnostic and action accuracies (90% and 87%) comparable to human experts (97%) and significantly outperforms the rule-based engine (63%). This demonstrates that its collaborative reasoning overcomes the limitations of rigid rules, enabling correct identification of root causes even for complex and ambiguous faults where rule-based systems often fail (e.g., misdiagnosing kernel hangs as network issues). Regarding efficiency, DLROVER-LM drastically reduces the MTTR to just 15 minutes, automating the time-consuming manual diagnosis process required by human experts (average 53 minutes). While the rule-based engine offers fast initial diagnosis in simple cases, its low accuracy leads to frequent misdiagnoses, resulting in repeated failures and restarts, which ultimately inflate its overall MTTR to 33 minutes. Thus, DLROVER-LM uniquely combines near-expert accuracy with high automation efficiency.

The Effect of Flash Checkpointing. After applying the *flash checkpointing* technique (§IV-B3) to a task using 1,536 accelerators, the time overhead associated with checkpointing is significantly reduced, as depicted in Figure 16.

Specifically, prior to the implementation of *flash checkpointing*, the training process performed checkpointing every 250 steps (approximately 1.5 hours) and saved the training state to the Cloud Parallel File Storage (CPFS). Each checkpointing operation required blocking the training for approximately 2 minutes, as shown in Figure 16(a) (i.e., Save CKPT). After applying *flash checkpointing*, each checkpointing only requires blocking the training for 0.2 seconds – a 600× reduction in pause time – allowing checkpointing frequency to increase to every 10 steps without throughput loss. Figure 16(a) further shows that *flash checkpointing* reduces checkpoint load time (i.e., Load CKPT) by roughly 50% compared to the baseline. Although cached CPU-side loading eliminates I/O delays, the overall gain is limited by the high overhead of repeated layer-by-layer FSDP initialization – a bottleneck we identify as a future optimization target. Figure 16(b) reports cumulative overheads across a week of training: *flash checkpointing* reduces total checkpointing time from 201 minutes to 3 minutes. This 99% reduction in checkpointing overhead further translates to markedly improved recovery responsiveness – i.e., mean recovery time after a failure drops from 231 to 34 minutes (6.8× faster) due to a more frequent checkpointing schedule.

VI. Related Work

Distributed Parallelism. Scaling LLM training relies on techniques like data [24], [26], [42], tensor [19], [27], [43], pipeline [20], [44]–[46], and 3D [16] parallelism. Systems like ZeRO [47]–[49] reduce memory redundancy via state partitioning. While combining multiple modes (e.g., DeepSeek [50]–[53]) or using automatic parallelism compilers [28], [54]–[56] offers flexibility, they introduce significant complexity or search time. Further optimizations for MoE models involve intricate heterogeneous mappings (MoE Parallel Folding [57]) or customized communication strategies (MegaScale-MoE [58]). In contrast, our work focuses on a simple-yet-effective hybrid FSDP+EP strategy, enhanced by optimizations rather than complex search or mappings.

Communication Optimization. To address communication bottlenecks, techniques include traffic reduction (e.g., mixed-precision, gradient compression) [53], [59] and system-level optimizations. The latter includes heterogeneous memory management and communication-computation overlap [15], [47], [60]–[62]. While modern frameworks like TensorFlow [63], [64] and PyTorch [10], [42] integrate basic overlap, simply enabling overlap is insufficient. Our work provides a systematic scheduling optimization to maximize this overlap in a complex hybrid-parallel setup.

Fault Tolerance. Fault tolerance mechanisms are broadly reactive or proactive. Reactive methods, such as rollback recovery [65], checkpointing [66], and heartbeat detection [67], restore service post-failure but often incur significant recovery overhead. Proactive strategies try to anticipate failures via prediction and migration [68]–[72]. However, both approaches lack mechanisms for automated, high-accuracy root cause analysis of complex LLM failures. Our work addresses this gap with a multi-agent diagnostic framework inspired by D-Bot [36], tailored for cluster-scale LLM training failures, unlike D-Bot's system-level database focus.

VII. Conclusion

We have presented DLROVER-LM, a distributed training framework for efficient and reliable MoE-based model training in large-scale production environments with thousands of GPUs. DLROVER-LM integrates a hybrid FSDP+EP parallelism approach, with optimizations like communication-computation overlapping to hide communication latency and expert-assisted operator fusion for better efficiency. Additionally, DLROVER-LM includes an automated multi-agent self-healing system that detects faults and recovers from failures using collaborative efforts from domain-specific experts, improving training stability. We believe this work provides valuable insights and techniques for building LLM training systems, and the open-source release of DLROVER-LM will benefit the broader AI community.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," Proceedings of the VLDB Endowment, vol. 17, no. 5, pp. 1132–1145, 2024.

[2] H. Kim, B.-H. So, W.-S. Han, and H. Lee, "Natural language to sql: where are we today?" Proc. VLDB Endow., vol. 13, no. 10, p. 1737–1750, Jun. 2020. [Online]. Available: https://doi.org/10.14778/3401960.3401970

[3] H. Fu, C. Liu, B. Wu, F. Li, J. Tan, and J. Sun, "Catsql: Towards real world natural language to sql applications," Proc. VLDB Endow., vol. 16, no. 6, p. 1534–1547, Feb. 2023. [Online]. Available: https://doi.org/10.14778/3583140.3583165

[4] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, "The dawn of natural language to sql: Are we fully ready?" Proc. VLDB Endow., vol. 17, no. 11, p. 3318–3331, Jul. 2024. [Online]. Available: https://doi.org/10.14778/3681954.3682003

[5] T. Ren, Y. Fan, Z. He, R. Huang, J. Dai, C. Huang, Y. Jing, K. Zhang, Y. Yang, and X. S. Wang, "Purple: Making a large language model a better sql writer," in 2024 IEEE 40th International Conference on Data Engineering (ICDE). IEEE, 2024, pp. 15–28.

[6] T. Kano, S. Sakti, and S. Nakamura, "Transformer-based direct speech-to-speech translation with transcoder," in 2021 IEEE spoken language technology workshop (SLT). IEEE, 2021, pp. 958–965.

[7] J. Zhang and C. Zong, "Neural machine translation: Challenges, progress and future," Science China Technological Sciences, vol. 63, no. 10, pp. 2028–2050, 2020.

[8] "Introducing chatgpt," [Online], https://openai.com/index/chatgpt/.

[9] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale et al., "Llama 2: Open foundation and fine-tuned chat models," arXiv preprint arXiv:2307.09288, 2023.

[10] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 929–947. [Online]. Available: https://doi.org/10.1145/3620665.3640366

[11] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," arXiv preprint arXiv:2001.08361, 2020.

[12] T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray et al., "Scaling laws for autoregressive generative modeling," arXiv preprint arXiv:2010.14701, 2020.

[13] C. Liao, M. Sun, Z. Yang, J. Xie, K. Chen, B. Yuan, F. Wu, and Z. Wang, "Ratel: Optimizing holistic data movement to fine-tune 100b model on a consumer gpu," in 2025 IEEE 41st International Conference on Data Engineering (ICDE). IEEE, 2025, pp. 292–306.

[14] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," Artificial Intelligence Review, vol. 42, pp. 275–293, 2014.

[15] C. Chen, X. Li, Q. Zhu, J. Duan, P. Sun, X. Zhang, and C. Yang, "Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2024, pp. 178–191.

[16] 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.

[17] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo, Y. Wen, and T. Zhang, "Characterization of large language model development in the datacenter," in Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI'24. USA: USENIX Association, 2024.

[18] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania et al., "Pytorch distributed: Experiences on accelerating data parallel training," Proceedings of the VLDB Endowment, vol. 13, no. 12.

[19] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020. [Online]. Available: https://arxiv.org/abs/1909.08053

[20] 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.

[21] A. Kokolis, M. Kuchnik, J. Hoffman, A. Kumar, P. Malani, F. Ma, Z. DeVito, S. Sengupta, K. Saladi, and C.-J. Wu, "Revisiting reliability in large-scale machine learning research clusters," 2025. [Online]. Available: https://arxiv.org/abs/2410.21680

[22] Q. Wang, T. Lan, Y. Tang, B. Sang, Z. Huang, Y. Du, H. Zhang, J. Sha, H. Lu, Y. Zhou, K. Zhang, and M. Tang, "Dlrover-rm: Resource optimization for deep recommendation models training in the cloud," Proc. VLDB Endow., vol. 17, no. 12, p. 4130–4144, Aug. 2024. [Online]. Available: https://doi.org/10.14778/3685800.3685832

[23] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin et al., "Opt: Open pre-trained transformer language models," arXiv preprint arXiv:2205.01068, 2022.

[24] 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," Proceedings of the VLDB Endowment, vol. 16, no. 12, pp. 3848–3860, 2023.

[25] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," arXiv preprint arXiv:1806.03377, 2018.

[26] R. Zhang, T. Liu, W. Feng, A. Gu, S. Purandare, W. Liang, and F. Massa, "Simplefsdp: Simpler fully sharded data parallel with torch.compile," 2024. [Online]. Available: https://arxiv.org/abs/2411.00284

[27] 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.

[28] 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.

[29] P. Baptiste, C. Le Pape, and W. Nuijten, Constraint-based scheduling: applying constraint programming to scheduling problems. Springer Science & Business Media, 2001, vol. 39.

[30] "Google's operations research tools," [Online], https://github.com/google/or-tools.

[31] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in Advances in Neural Information Processing Systems (NeurIPS), 2022.

[32] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in International Conference on Learning Representations (ICLR), 2024.

[33] "Xla," [Online], https://github.com/openxla/xla.

[34] NVIDIA Corporation, NVIDIA Nsight Systems, 2025. [Online]. Available: https://developer.nvidia.com/nsight-systems

[35] P. Tillet and D. Cox, "Triton: an intermediate language and compiler for tiled neural network computations," in Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, 2019, pp. 10–19.

[36] X. Zhou, G. Li, Z. Sun, Z. Liu, W. Chen, J. Wu, J. Liu, R. Feng, and G. Zeng, "D-bot: Database diagnosis system using large language models," Proc. VLDB Endow., vol. 17, no. 10, p. 2514–2527, Jun. 2024. [Online]. Available: https://doi.org/10.14778/3675034.3675043

[37] W. Cui, J. Zhang, H. Zhao, C. Liu, W. Zhang, J. Sha, Q. Chen, B. He, and M. Guo, "Xputimer: Anomaly diagnostics for divergent llm training in gpu clusters of thousand-plus scale," arXiv preprint arXiv:2502.05413, 2025.

[38] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin,

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, and Z. Fan, "Qwen2 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2407.10671

[39] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, "Qwen3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388

[40] L. Team, "Every flop counts: Scaling a 300b mixture-of-experts ling llm without premium gpus," arXiv preprint arXiv:2503.05139, 2025.

[41] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "Pytorch fsdp: Experiences on scaling fully sharded data parallel," 2023. [Online]. Available: https://arxiv.org/abs/2304.11277

[42] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "Pytorch distributed: experiences on accelerating data parallel training," Proc. VLDB Endow., vol. 13, no. 12, p. 3005–3018, Aug. 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415530

[43] Y. Xu, H. Lee, D. Chen, B. Hechtman, Y. Huang, R. Joshi, M. Krikun, D. Lepikhin, A. Ly, M. Maggioni, R. Pang, N. Shazeer, S. Wang, T. Wang, Y. Wu, and Z. Chen, "Gspmd: General and scalable parallelization for ml computation graphs," 2021. [Online]. Available: https://arxiv.org/abs/2105.04663

[44] 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, 2019, pp. 1–15.

[45] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia et al., "Dapple: A pipelined data parallel approach for training large models," in Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2021, pp. 431–445.

[46] D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia, "Memory-efficient pipeline-parallel dnn training," in International Conference on Machine Learning. PMLR, 2021, pp. 7937–7947.

[47] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: breaking the gpu memory wall for extreme scale deep learning," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476205

[48] 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.

[49] 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.

[50] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi et al., "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," arXiv preprint arXiv:2501.12948, 2025.

[51] DeepSeek-AI, :, X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, H. Gao, K. Gao, W. Gao, R. Ge, K. Guan, D. Guo, J. Guo, G. Hao, Z. Hao, Y. He, W. Hu, P. Huang, E. Li, G. Li, J. Li, Y. Li, Y. K. Li, W. Liang, F. Lin, A. X. Liu, B. Liu, W. Liu, X. Liu, X. Liu, Y. Liu, H. Lu, S. Lu, F. Luo, S. Ma, X. Nie, T. Pei, Y. Piao, J. Qiu, H. Qu, T. Ren, Z. Ren, C. Ruan, Z. Sha, Z. Shao, J. Song, X. Su, J. Sun, Y. Sun, M. Tang, B. Wang, P. Wang, S. Wang, Y. Wang, Y. Wang, T. Wu, Y. Wu, X. Xie, Z. Xie, Z. Xie, Y. Xiong, H. Xu, R. X. Xu, Y. Xu, D. Yang, Y. You, S. Yu, X. Yu, B. Zhang, H. Zhang, L. Zhang, L. Zhang, M. Zhang, M. Zhang, W. Zhang, Y. Zhang, C. Zhao, Y. Zhao, S. Zhou, S. Zhou, Q. Zhu, and Y. Zou, "Deepseek llm: Scaling open-source language models with longtermism," 2024. [Online]. Available: https://arxiv.org/abs/2401.02954

[52] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[53] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang, R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan, "Deepseek-v3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2412.19437

[54] X. Miao, Y. Wang, Y. Jiang, C. Shi, X. Nie, H. Zhang, and B. Cui, "Galvatron: Efficient transformer training over multiple gpus using automatic parallelism," Proc. VLDB Endow., vol. 16, no. 3, p. 470–479, Nov. 2022. [Online]. Available: https://doi.org/10.14778/3570690.3570697

[55] L. Wang, L. Yang, Y. Yu, W. Wang, B. Li, X. Sun, J. He, and L. Zhang, "Morphling: Fast, near-optimal auto-configuration for cloud-native model serving," in Proceedings of the ACM Symposium on Cloud Computing, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 639–653. [Online]. Available: https://doi.org/10.1145/3472883.3486987

[56] Z. Lin, Y. Miao, Q. Zhang, F. Yang, Y. Zhu, C. Li, S. Maleki, X. Cao, N. Shang, Y. Yang, W. Xu, M. Yang, L. Zhang, and L. Zhou, "nnScaler: Constraint-Guided parallelization plan generation for deep learning training," in 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). Santa Clara, CA: USENIX Association, Jul. 2024, pp. 347–363. [Online]. Available:

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.

https://www.usenix.org/conference/osdi24/presentation/lin-zhiqi
[57] 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.
[58] 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.
[59] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. 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, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3505–3506. [Online]. Available: https://doi.org/10.1145/3394486.3406703
[60] A. Jangda, J. Huang, G. Liu, A. H. N. Sabet, S. Maleki, Y. Miao, M. Musuvathi, T. Mytkowicz, and O. Saarikivi, "Breaking the computation and communication abstraction barrier in distributed machine learning workloads," in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 402–416. [Online]. Available: https://doi.org/10.1145/3503222.3507778
[61] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang, S. Kumar, T. Guo, Y. Xu, and Z. Zhou, "Overlap communication with dependent computation via decomposition in large deep learning models," in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2022, p. 93–106. [Online]. Available: https://doi.org/10.1145/3567955.3567959
[62] L.-W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, C. Yao, Z. Jiang, H. Lin, X. Jin, and X. Liu, "Flux: Fast software-based communication overlap on gpus through kernel fusion," 2024. [Online]. Available: https://arxiv.org/abs/2406.06858
[63] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: a system for large-scale machine learning," in Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'16. USA: USENIX Association, 2016, p. 265–283.
[64] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv preprint arXiv:1603.04467, 2016.
[65] S. Haider, N. R. Ansari, M. Akbar, M. R. Perwez, and K. Ghori, "Fault tolerance in distributed paradigms," in In2011 International Conference on Computer Communication and Management, Proc. of CSIT, vol. 5, 2011.
[66] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in Proceedings of the Thirteenth EuroSys Conference, 2018, pp. 1–14.
[67] M. Van Steen, "Distributed systems principles and paradigms," Network, vol. 2, no. 28, p. 1, 2002.
[68] S. Athlur, N. Saran, M. Sivathanu, R. Ramjee, and N. Kwatra, "Varuna: scalable, low-cost training of massive deep learning models," in Proceedings of the Seventeenth European Conference on Computer Systems, 2022, pp. 472–487.
[69] J. Thorpe, P. Zhao, J. Eyolfson, Y. Qiao, Z. Jia, M. Zhang, R. Netravali, and G. H. Xu, "Bamboo: Making preemptible instances resilient for affordable training of large DNNs," in 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). Boston, MA: USENIX Association, Apr. 2023, pp. 497–513. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/thorpe
[70] Z. Mo, H. Xu, and C. Xu, "Heet: Accelerating elastic training in heterogeneous deep learning clusters," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2024, pp. 499–513.
[71] T. He, X. Li, Z. Wang, K. Qian, J. Xu, W. Yu, and J. Zhou, "Unicron: Economizing self-healing llm training at scale," arXiv preprint arXiv:2401.00134, 2023.
[72] Y. Xiao, L. Ju, Z. Zhou, S. Li, Z. Huan, D. Zhang, R. Jiang, L. Wang, X. Zhang, L. Liang et al., "Antdt: A self-adaptive distributed training framework for leader and straggler nodes," in 2024 IEEE 40th International Conference on Data Engineering (ICDE). IEEE, 2024, pp. 5238–5251.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on November 11,2026 at 03:19:08 UTC from IEEE Xplore. Restrictions apply.