Di-PS: System-Algorithm Co-Design for Asynchronous and Heterogeneous Cross-cluster LLM Training at Scale

Shengwei Li[*1], Qiaoling Chen[*2,3], Zhiquan Lai[1], Penglong Jiao[2], Wenwen Qu[2], Kun Cai[2], Jiaxing Li[2], Peng Sun[2], Xingcheng Zhang[2], Xiaoge Deng[1], Dongsheng Li[1], Kai Lu[1], Tianwei Zhang[3]
[1]*National Key Laboratory of Parallel and Distributed Computing, College of Computer Science and Technology, National University of Defense Technology*
[2] *Shanghai Artificial Intelligence Laboratory* [3] *Nanyang Technological University*

Abstract

Large language models (LLMs) have revolutionized artificial intelligence, exhibiting remarkable performance in various tasks. Training these models demands extensive computational resources, which are often economically and physically prohibitive. *Cross-cluster training* can balance infrastructure costs, alleviate physical and resource constraints, better match workload demands, and sustain higher efficiency through geo-distributed deployment. However, challenges arise from network variability, heterogeneous computational resources, and intrinsic training instability.

To address these issues, we present Di-PS, a novel framework for cross-cluster LLM training at scale. The core of Di-PS is the system-algorithm co-design of a parameter server paradigm, to achieve heterogeneous, asynchronous, and resilient training across decentralized clusters. We make several innovative contributions in Di-PS, including (i) an efficient parameter server design for cross-cluster communication of LLM parameters, (ii) a pseudo-gradient penalty strategy for convergence stability enhancement of asynchronous two-stage optimization, and (iii) a resilience mechanism for fault tolerance in cross-cluster training. Results from the controlled experimental setting demonstrate that Di-PS improves training efficiency by up to 4.67× over synchronous cross-cluster approaches while maintaining model quality, and achieving near-linear scalability in heterogeneous training resources. Di-PS has been deployed in the production environment, involving dynamic training scales with up to 9 clusters and more than 10,000 NPUs. At this scale, Di-PS enables successful cross-cluster training of a 100B-parameter LLM with only 6% overhead compared to single-cluster training, and effectively handles frequent failures and resource changes.

1 Introduction

Large language models (LLMs) are fundamentally reshaping the landscape of artificial intelligence. They exhibit unprecedented versatility and intelligence across a wide spectrum of tasks. This progress is primarily driven by the scaling of Transformer-based models such as GPT [10], LLaMA [28], Gemini [63], and DeepSeek [47]. Training LLMs requires vast computational resources over extended periods, which is critically dependent on large-scale clusters. For instance, LLaMA-3 was trained on 16,384 NVIDIA H100 [28], and this number increases to 32,000 for LLaMA-4 [2]. Recent studies suggest that LLM scaling has not yet reached its theoretical limits, demonstrating predictable performance gains with the increased model and dataset sizes [11, 39]. Consequently, the pursuit of ultra-large-scale training remains active, along with the escalating demand for computational resources.

Why cross-cluster training? While building or scaling a monolithic cluster for larger-scale LLM training can take advantage of high-bandwidth interconnects, consolidating existing clusters is often more feasible. Our key findings are summarized as follows:

- ***Cost efficiency***. Building multiple smaller clusters is more economical than constructing one large cluster. Intra-cluster training requires high-performance, low-oversubscription networks, typically achieved using topologies such as Clos or Multi-Rail. For example, a monolithic Clos network supporting 10,000 NPUs typically requires around 400 switches (based on 128-port switches) [31]. In contrast, partitioning the system into 10 independent clusters, each with 1,000 NPUs, reduces the switch count to 240, significantly lowering network infrastructure costs by 40%.
- ***Physical constraints***. The high power and cooling requirements of high-end NPUs limit how many can be deployed within a single datacenter. Smaller clusters improve power distribution and reliability, lower the risk of large-scale power failures [5, 20], and enhance fault isolation [29, 69].
- ***Scarcity of large clusters and modest workload demands***. High-end homogeneous NPU clusters are inherently scarce, as large-scale allocations of A100 or H100 GPUs are rarely attainable in practice [16, 18, 60]. At the same time, workload characterization reveals that over 98% of LLM jobs require fewer than 100 NPUs, since these jobs typically

*: Equal contribution.

Table 1: Peak computational power, number of failures, and mean time between failures (MTFB) of training clusters used in a 33-day production-level training of a 100B LLM.

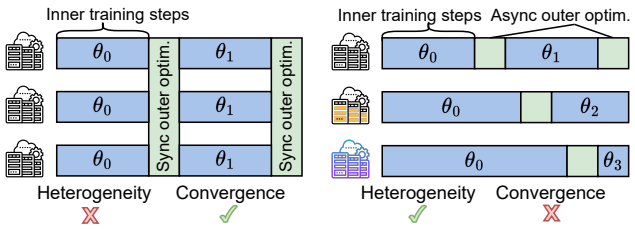
Cluster	Amount	#NPU	Total PFLOPS (FP16)	#Failures	MTBF (Days)
A	5	1024	378.9	3	55.0
B	1	896	331.5	1	33.0
C	1	1472	329.5	4	7.3
D	1	448	229.4	2	3.0
E	1	2176	479.2	31	0.9

involve evaluation, post-training, or debugging rather than massive-scale pretraining [33]. This imbalance between the scarcity of large homogeneous clusters and the modest scale of most LLM workloads indicates that building multiple smaller, potentially heterogeneous clusters better aligns with both hardware availability and workload demands, thereby avoiding resource underutilization while sustaining high throughput.

- ***Limited training performance***. Achieving high training performance at a cluster of extreme scales remains challenging. For instance, LLaMA-3 attains only 38–41% model FLOP utilization on NPUs [28], highlighting the difficulties in maintaining high utilization as the system scale increases. In contrast, DeepSeek-v3 attained higher utilizations, benefiting from fine-grained optimizations and partly from its smaller scale of 2,048 NPUs [47].

As a result, cross-cluster training is not just an attractive option, but often the only practical path forward. However, the significantly lower inter-cluster bandwidth [32], often hundreds of times less than intra-cluster bandwidth, prevents direct cross-cluster training. Existing approaches adopt a two-stage optimization [23] (Figure 1a): each cluster trains locally with an inner optimizer, while a low-frequency outer optimizer synchronizes models across clusters to reduce the cross-cluster communication overhead.

The state-of-the-art efforts. Existing cross-cluster LLM training adopts a decentralized architecture, where every cluster maintains an outer optimizer replica and synchronizes them with collective communications. However, decentralized training at scale exposes fundamental roadblocks in architecture: *(i) Network bottlenecks*. Inter-cluster bandwidth is up to 100× lower than intra-cluster, and highly variable across sites. Existing decentralized approaches suffer throughput collapse when bottleneck links dominate [32, 36, 51]. *(ii) Unstable convergence*. Heterogeneous clusters differ in NPU generations, interconnects, and performance. Asynchronous training reduces idle time but often diverges [48, 64]. *(iii) Limited resilience at scale*. Clusters show widely different failure characteristics and mean time between failures (MTBF), and current frameworks lack mechanisms to isolate faults and recover elastically [3, 21, 29, 38, 65].



(a) Training across homogeneous clusters and synchronous outer optimization.

(b) Training across heterogeneous clusters and asynchronous outer optimization.

Figure 1: Cross-cluster training with a two-stage optimization process [23]. Each cluster trains a model replica θ for multiple inner steps independently. Afterwards, clusters send local models to the outer optimizer, which performs an outer optimization to update the global model. The updated global model is then synchronized across clusters that participate in the outer optimization step.

Requirements and Design. As one of the largest model training providers with multiple heterogeneous clusters (Table 1), we decide to build an efficient, convergent, and resilient cross-cluster LLM training system. We find that designing a centralized coordination layer provides benefits: (i) it better exploits heterogeneous bandwidth than fully decentralized synchronization, (ii) it provides a global view of optimization states, which is critical for stabilizing asynchronous convergence, (iii) it enables failure localization, preventing instability from cascading across clusters. To this end, we introduce `Di-PS`, a new centralized parameter server (PS) to coordinate asynchronous training on decentralized clusters.

`Di-PS` first introduces an *efficient distributed PS* design for LLM, integrating a *leader-follower architecture* for scalable outer optimizer-state management, a *dual-workflow mechanism* to decouple operation orchestration from parameter exchange, *inter-cluster communication coordination* to improve bandwidth utilization, and *operation overlapping* to pipeline communication with computation. Together, these techniques enhance bandwidth efficiency, scalability, and reduce synchronization overhead in cross-cluster training. Besides, we integrate a *pseudo-gradient penalty strategy* on PS to enable robust asynchronous cross-cluster training to exploit available heterogeneous training resources. Both theoretical analysis and experimental results demonstrate that `Di-PS` achieves a convergence guarantee. Finally, we introduce a *resilience and fault-tolerance mechanism*, including dynamic management of cluster participation and departure and a self-recovering PS design. Our design not only improves scheduling flexibility but also enables scalable and reliable fault tolerance–an essential requirement for stable, efficient LLM training across multiple heterogeneous clusters.

Experimental results on three types of heterogeneous training clusters demonstrate that `Di-PS` achieves 1.27-4.67× training acceleration compared to synchronous two-stage opti-

mization approaches, while maintaining similar convergence performance. Compared to asynchronous cross-cluster training approaches, `Di-PS` achieves 1.00-1.60× training acceleration and exhibits better convergence.

Our contributions can be summarized as follows:

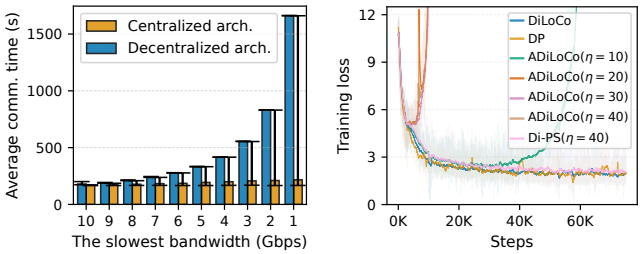
- **Centralized system for cross-cluster LLM training.** We propose a PS design that combines a leader-follower architecture, dual-workflow mechanism, communication coordination, and operation overlapping to improve scalability and reduce synchronization costs of cross-cluster LLM training.
- **System-algorithm co-design.** A pseudo-gradient penalty strategy stabilizes asynchronous two-stage optimization, enabling robust convergence across heterogeneous clusters.
- **Resilience at production scale.** A resilience and fault-tolerance design, including elastic cluster participation and PS self-recovery, to ensure reliable large-scale training.

We have deployed `Di-PS` across 9 heterogeneous production clusters with over 10,000 NPUs. It efficiently scales the training with stable convergence: the additional overhead for each training cluster incurred by `Di-PS` remains minimal, accounting for less than 6% of the total training time. These results demonstrate that cross-cluster LLM training is not only feasible, but practical at production scales.

2 Background and Challenge

Parallel LLM Training. The increased LLMs size necessitates parallel training, which splits the model across (up to hundreds of) devices using different parallelism such as tensor parallelism (TP) [57,73] and pipeline parallelism (PP) [52,54]. Cooperating with these parallelisms, existing parallel LLM training systems [13,44,46,53,72] train giant models using tens of thousands of devices [28, 38]. Such scales exceed the size of most existing clusters, training across multiple clusters [20] becomes increasingly important.

Cross-cluster LLM Training. Parallel LLM training [26] requires intensive communication among all devices. High model FLOPs utilization can be achieved in intra-cluster devices with low-latency connections. Compared to intra-cluster communication, inter-cluster communication is costly. To scale LLM training beyond a single cluster, DiLoCo [11,23] introduces a two-stage optimization process as shown in Figure 1a. Each cluster trains LLM locally with an inner optimizer, the inner training steps are identical to single cluster training, thus can employ existing intra-cluster training optimizations. Clusters perform the inner steps in parallel and only globally synchronize the model with a global outer optimizer at every H inner steps, to reduce the cross-cluster communication overhead. Formally, each cluster i update local parameters $\theta_t^{(i)}$ with learning rate γ_{in} and gradient $g_t^{(i)}$: $\theta_{t,h+1}^{(i)} = \theta_{t,h}^{(i)} - \gamma_{\text{in}} g_{t,h}^{(i)}$. After H inner steps, the outer optimizer aggregates pseudo-gradients $\Delta_t^{(i)} = \theta_t - \theta_{t,H}^{(i)}$ from all N clusters and updates the global model θ_t with learning rate γ_{out}: $\theta_{t+1} = \theta_t - \gamma_{\text{out}} \cdot \frac{1}{N} \sum_{i=1}^{N} \Delta_t^{(i)}$. Current implementations [22,35,36] of this two-stage optimization process use a decentralized architecture, where every cluster keeps an outer optimizer replica and synchronizes these outer optimizers with AllReduce communications.



(a) Average communication time for a 100B LLM on 4 clusters with network heterogeneity.

(b) The training loss on 4 clusters with different performance heterogeneity values (η).

Figure 2: Cross-cluster training challenges for heterogeneity.

Resilient LLM Training. Failures frequently occur in large-scale LLM training [28,33,38], which may lead to complete restarts on all devices, significantly wasting the training resource. Resilient training enables seamless scaling of computational resources during training to reduce resource waste. Recent approaches focus on scenarios of cloud spot instances along specific parallelisms [7,25,27,37] or intra-cluster training [3,29,38,65]. For cross-cluster training, current studies primarily address elastic job scheduling [17,58,69], meeting resource requirements from multiple jobs. Stable and resilient large-scale cross-cluster training remains largely unexplored.

2.1 Challenges of Cross-cluster LLM Training

Cross-cluster LLM training suffers from communication bottlenecks. Although the cross-cluster communication frequency can be reduced with the two-stage optimization process [23], each round requires synchronizing the complete LLM model. For a 100B LLM, every communication size will be approximately 400 GB. Besides, the heterogeneous computational performance of training clusters necessitates asynchronous distributed training. As shown in Figure 1b, training clusters perform the outer optimization independently, leading to distinct local models in clusters and parameter staleness in outer optimization. To deal with the heterogeneity in cross-cluster LLM training atop the asynchronous distributed training and two-stage optimization process, the following challenges still need to be addressed:

C1: Inefficient Cross-cluster Communications. Existing decentralized communication approaches struggle to accommodate network heterogeneity. We conduct an experiment of communicating a 100B parameter LLM across four clusters. Three clusters are equipped with 10 Gbps networks, while varying network speed ranging from 1 Gbps to 10 Gbps on the fourth cluster. Figure 2a demonstrates that performance degradation in the decentralized communication architecture

becomes more significant as network heterogeneity increases.

C2: Unstable Convergence of Asynchronous Training. Heterogeneity across clusters (including NPUs, networks, and memory) leads to significant variations in training performance. We pretrain a LLaMA3.2-1B model [28] across four emulation clusters under four asynchronous training scenarios defined by η, $\eta = x$ denotes that the training performance among the clusters varies uniformly by 0–x%. The number of inner steps is 64 ($H = 64$) in two-stage optimization methods. The results are shown in Figure 2b. Compared to the synchronous training methods DiLoCo and DP, naive asynchronous DiLoCo (ADiLoCo) trainings fail to converge the model, and a larger η exacerbates the convergence issues.

C3: Instability and Inconsistent Accessibility of Clusters. Large-scale cross-cluster training faces heightened unreliability. The failure rates vary across heterogeneous clusters. As reported in Table 1, we observe 31 failures in a newly established cluster during a 33-day production training, while the other eight clusters encountered fewer than 4 failures. And the decentralized training cluster availability is dynamic, we experience predictable 6 cluster changes due to resource accessibility. However, existing decentralized frameworks offer limited resilience, as frequent training cluster join/leave events impose substantial overhead.

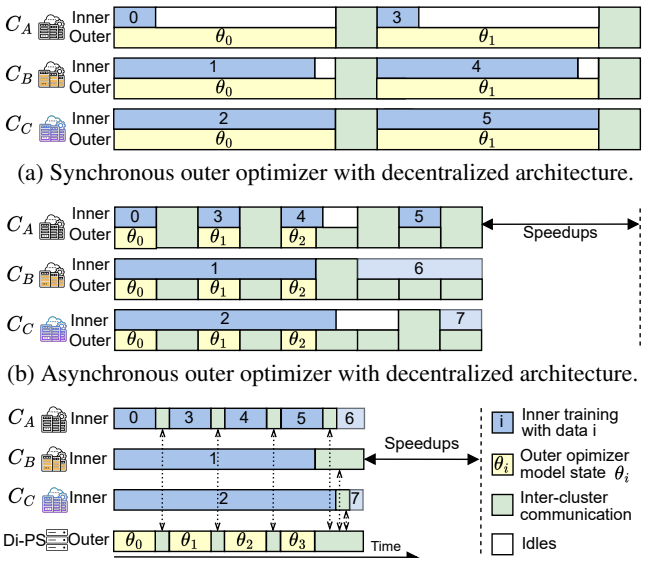
3 Observation and Requirement

To address the unique challenges of cross-cluster LLM training, we first examine the limitations of decentralized designs and the opportunities enabled by adopting a centralized parameter server (PS) architecture. We then distill the key requirements that a PS must satisfy to serve for efficient, convergent, and resilient cross-cluster LLM training.

3.1 Centralized PS for Cross-cluster Training

We summarize three key advantages of centralized PS in cross-cluster training, compared to decentralized designs:

Exploiting Cross-cluster Networks. Centralized PS better accommodates bandwidth heterogeneity across clusters than decentralized methods. As shown in Figure 3a, synchronous outer optimization causes faster clusters to wait for slower ones. Asynchronous training avoids this by allowing independent updates, but at the cost of more frequent communication. Current cross-cluster systems [22, 35, 36] often employ a fully decentralized architecture using AllReduce for parameter aggregation (Figure 3b). In AllReduce, communication is limited by the slowest inter-cluster link, leading to uniformly high overhead. In contrast, as shown in Figure 3c, the centralized PS employs point-to-point operations for outer optimization communication. Although some communications might remain limited by slower network links (e.g., cluster B C_B), others can benefit from faster network links and thus accelerate the overall process (e.g., C_A,C_C).



(a) Synchronous outer optimizer with decentralized architecture.

(b) Asynchronous outer optimizer with decentralized architecture.

(c) Asynchronous outer optimizer with centralized architecture.

Figure 3: Comparison of different outer optimizer and communication architectures on cross-cluster LLM training with heterogeneous clusters. The cluster B (C_B) has a slower inter-cluster bandwidth.

Complete Model Optimization History. Theoretical analysis of asynchronous training [59] highlights that anomalous gradients can impede the overall optimization stability. Detecting such outliers effectively requires access to the history of model updates, enabling the system to compare current gradients against historical statistics such as norms [15]. A centralized PS participates in every outer optimization and can maintain this historical record at minimal overhead. In contrast, as training clusters may dynamically join or leave, decentralized outer optimizers lack a repository, necessitating extensive additional communication for obtaining historical context. Thus, centralized PS enables low-cost outlier detection, supporting stable convergence under heterogeneity and asynchronism.

Localized Training Errors. The centralized PS is inherently more fault-tolerant than the decentralized manner for cross-cluster training. In decentralized training architectures, such as those relying on AllReduce collective communication [13, 36], fault tolerance becomes more complex, because an error or failure in one cluster must be synchronously detected and handled by all other clusters. This global coordination leads to poor fault isolation and high overhead during failure recovery. In contrast, centralized PS-based architecture localizes failure detection and recovery. The PS acts as the single point of coordination. When a failure occurs in one cluster, only the PS needs to be aware of and respond to the fault—there is no requirement for other clusters to synchronize their view of the system or halt their training progress. This isolation enables healthy clusters can continue training, reducing the error overhead.

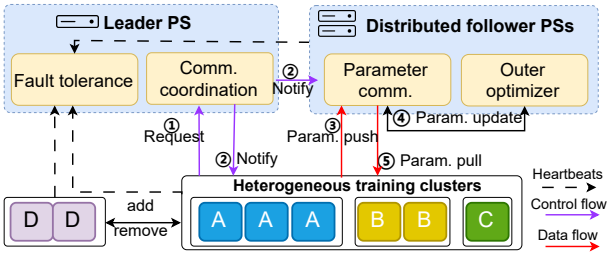


Figure 4: The leader-follower parameter server of Di-PS.

3.2 PS Design Requirement

Based on the observations of cross-cluster training with the centralized PS architecture, Di-PS is designed to meet three key requirements:

- **Scalable Efficiency.** The centralized PS avoids the slowest-link bottleneck of decentralized AllReduce by using point-to-point communication. However, cross-cluster LLM training still involves exchanging billions of parameters over heterogeneous links. The PS must therefore support highly efficient parameter exchange at scale, scheduling cross-cluster communication operations, and coordinating updates without becoming a bottleneck.
- **Convergence.** The centralized PS uniquely maintains the complete history of model updates, which is critical for detecting and filtering anomalous gradients in asynchronous training. To leverage this global visibility, the PS must stabilize optimization by penalizing stale updates and weighting contributions based on convergence trends appropriately, ensuring that two-stage asynchronous training preserves theoretical convergence guarantees.
- **Resilience.** Compared to decentralized architectures where failures propagate globally, the centralized PS localizes error handling, allowing healthy clusters to continue training. To fully realize this benefit in week-long training jobs across thousands of NPUs, the PS design must incorporate resilience: tolerating failures in both training clusters and PS instances, supporting elastic cluster membership, and maintaining model consistency with minimal training progress loss.

4 Di-PS Design

To meet the requirements in § 3.2, Di-PS introduces a parameter server (PS) tailored for cross-cluster LLM training. Its core design includes: (i) a leader-follower PS structure with dual-workflow mechanisms and communication coordination to achieve scalable efficiency, (ii) pseudo-gradient strategies and convergence analysis to stabilize asynchronous training, and (iii) resilience mechanisms that tolerate failures and enable elastic operation. Together, these components ensure that Di-PS delivers scalable, accurate, and fault-tolerant LLM training across geo-distributed clusters.

4.1 Efficient Parameter Server for LLM

Leader-follower PS. A key design of the parameter server (PS) for LLM training is the leader-follower PS architecture (Figure 4). The substantial sizes of LLMs make it impossible to hold the PS on a single device. For example, training a 100B parameter model requires 1600 GB of memory for model states and optimizer states, and at least 400 GB for buffering parameters from clusters, resulting in a minimum memory footprint of 2000 GB for the outer optimizer.

Consequently, we adopt distributed follower PSs to reduce the memory overhead and improve the communication performance. Each follower PS is deployed on a CPU server and manages several LLM model layers. The distributed follower PS design offers scalability, allowing us to incorporate additional follower PS to accommodate larger models and increased training cluster sizes. To orchestrate the operations between follower PSs and training clusters, a leader PS is introduced to serve as the central controller.

The workflow of the leader-follower PS is as follows:

1. Push Request: A training cluster requests to push parameters with its metadata to the leader PS. Metadata includes the cluster ID, an identifier to distinguish and track training clusters for coordination and fault handling.
2. Communication Coordination: The leader PS coordinates both the requesting cluster and the corresponding follower PS on how to perform the communication.
3. Parameter Push: The cluster starts sending all parameters to the distributed follower PSs, and notifies the leader PS when the transfer is complete.
4. Parameter Update: The leader PS instructs the follower PSs with the gradient penalty procedure (detailed in § 4.2) and processes the outer optimization with the received parameters.
5. Parameter Pull: Once the follower PSs complete the parameter updates, they notify the leader PS. The leader PS then informs all clusters involved in the current round to pull the latest parameters from the follower PSs.

Dual-workflow Mechanism. The control operations in the leader PS result in frequent signaling communication. We isolate these small message exchanges from model parameter communication to mitigate communication contention and prevent deadlocks. This separation is achieved through a *dual-workflow* design: the *control flow* handles operation orchestration in the leader PS, while a *data flow* manages communication between training clusters and follower PSs. We further use separate communication libraries to isolate the communications on the dual workflow. We evaluate the communication performance of gRPC [1] and ZeroMQ [4] across varying communication sizes on a 25 Gbps network. As shown in Figure 5, ZeroMQ exhibits performance advantages, owing to its streamlined data transmission and buffer management. Considering the data flow's higher sensitivity to transfer speed and message size is large and stable, the

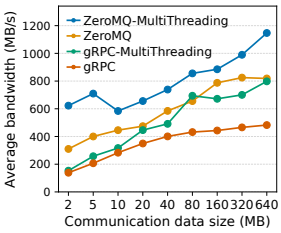


Figure 5: Communication performance of gRPC [1] and ZeroMQ [4].

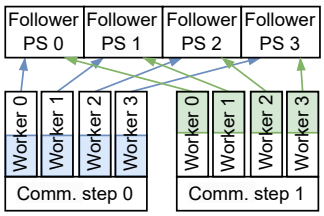


Figure 6: Communication schedule between training clusters and follower PSs.

multi-thread ZeroMQ is selected for data streaming. The message in control-flow is lightweight (<1 KB). To quantify the communication overhead of the leader PS, we measured the control-flow message rate during a cross-cluster training with 16 clusters. In the 3-day training, the message rate averaged 1.35 messages/s (peak 26), far below gRPC's capacity of handling over 10,000 sub-KB messages/s [9]. Therefore, gRPC is implemented for the control flow due to its flexibility in instruction of various data formats and its ability to prevent potential communication conflicts.

Communication Coordination in Leader PS. Efficient parameter communication between follower PSs and training clusters is essential for cross-cluster training. In training clusters, LLMs are typically trained using hybrid parallelism, including tensor parallelism (TP), pipeline parallelism (PP), and data parallelism (DP), with corresponding model partitioning. During the outer optimization process in training clusters, the training workers with TP and DP rank 0 firstly gather the model parameters in the TP communication group. Subsequently, these workers (with the number of PP sizes) request to communicate with the `Di-PS`, forming a many-to-many communication pattern.

To manage this complex process, the leader PS generates a communication schedule. As shown in Figure 6, this schedule is an ordered sequence of steps, where each step consists of worker–PS communication pairs executed in parallel. The objective of the schedule is to (i) maximize the cross-cluster link utilization and avoid congestion, and (ii) accommodate additional cluster request communication in asynchronous training. We adopt a simple greedy mapping strategy: starting from the first worker, we assign its earliest unsent layer to the least recently used follower PS, then proceed worker by worker until all layers are scheduled. This strategy maximizes the number of active worker–PS pairs without conflicts, achieving near-optimal concurrency in a single pass. Moreover, the greedy approach is naturally extensible. If a new cluster joins, its schedule can be generated independently without re-planning existing clusters. In contrast, optimal global scheduling requires solving a combinatorial assignment problem, incurring poor adaptability to dynamic arrivals. Details of our schedule strategy are provided in Appendix A.

Accommodate Asynchronism in Training. In asynchronous cross-cluster training, training clusters may request parameter push at any time. `Di-PS` supports accepting these requests most of the time, except during the parameter update or parameter pull phases. To mitigate delays caused by push requests arriving during these restricted phases, we introduce a grace time τ_{grace} before each outer optimizer update, allowing more clusters to join the current round. The selection of τ_{grace} needs to balance the system idle time and risk of missing late push requests. This tradeoff resembles the ski rental problem [66], a classic online decision model that captures the tradeoff between incurring recurring costs and paying a one-time upfront cost. Let λ denote the average cluster push arrival rate and C_m the request delay cost (of the parameter update and pull time). The expected total cost is $\tau_{\text{grace}} + C_m e^{-\lambda\tau_{\text{grace}}}$. This can be minimized at $\tau^*_{\text{grace}} = \frac{1}{\lambda}\ln(C_m\lambda)$. We estimate λ and C_m from runtime data to dynamically adjust τ_{grace}.

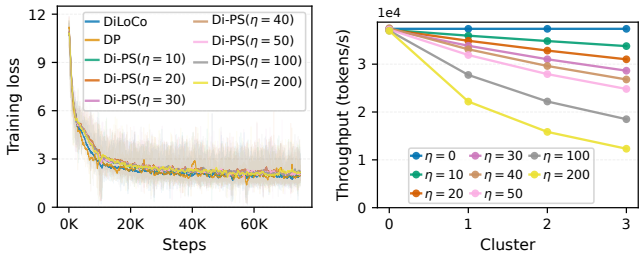
Operation Overlapping. For better network utilization, we use serialized data in the data flow. In practice, the serialization and deserialization of large model parameters can become a bottleneck in distributed follower PSs (e.g., 40% of time in optimizing a 100B model). `Di-PS` uses a Producer-Consumer model to deal with the serialization and deserialization operations. Specifically, as follower PSs receive parameters from training clusters in a pipeline manner, `Di-PS` stores them in memory pools and uses a dedicated consumer thread to asynchronously handle deserialization. This avoids blocking the data flow and is symmetrically applied to serialization and parameter sending. Besides, the communication in control flow and data flow can be overlapped with checkpointing, minimizing the extra overhead of follower PSs. On the training cluster side, intra-cluster overlapping techniques are fully leveraged to accelerate inner training steps, such as communication–computation overlap in DP [71], PP [54], and TP [12]. Cross-cluster and intra-cluster communications are naturally decoupled: the former runs over TCP/IP inter-cluster networks, while the latter uses dedicated training networks, avoiding bandwidth contention.

PS Deployment. `Di-PS` can be deployed on an arbitrary cluster with several CPU nodes. To minimize the communication overhead during the outer optimization phase, we adopt a cost model to select the optimal cluster for PS deployment. Consider N candidate clusters for PS deployment and M training clusters. We represent the inter-cluster bandwidth by an adjacency matrix $\mathbf{B} \in \mathbb{R}^{N\times M}$, where $\mathbf{B}_{ij}$ is the bandwidth between cluster i and training cluster j. The training performance of each training cluster is captured by a cost vector $\mathbf{p} \in \mathbb{R}^M$, where $\mathbf{p}_j$ denotes the training throughput of cluster j. Assuming that the communication frequency is proportional to the training throughput, the total communication demand on candidate PS cluster i can be modeled by $\mathbf{B}_{i,*}\mathbf{p}$, where $\mathbf{B}_{i,*}$ is the i-th row of $\mathbf{B}$. Thus, the optimal cluster for PS deployment is given by: $i^* = \arg\max_i(\mathbf{B}_{i,*}\mathbf{p})$. In practice, CPU servers are significantly more cost-effective than NPUs, and training clusters often have spare CPU capacity. Therefore,

Algorithm 1: Asynchronous outer optimizer

Input: Initial pretrained model θ_0, k training clusters, grace time τ_{grace}, total consumed tokens $tks_{\max}$

1 $tks_{\text{local}} \leftarrow 0$
2 $\theta \leftarrow$ `split_model_by_follower_PSs`(θ_0)
3 $G \leftarrow$ [`init_cluster`() for $i = 1, \ldots, k$]
4 $G_{\text{completed}} \leftarrow \emptyset$, $\tau_{\text{sync}} \leftarrow \infty$, $\Delta \leftarrow \emptyset$
5 **while** $tks_{local} < tks_{\max}$ **do**
6 $g \leftarrow$ `get_cluster`(G, τ_{sync})
7 **if** *g exists* **then**
8 $\tau_{\text{sync}} \leftarrow \tau_{\text{grace}}$
9 $\theta_g \leftarrow$ `recv_params`(g)
10 $g_t \leftarrow \theta - \theta_g$; `// Get the pseudo gradient.`
11 $\Delta \leftarrow \Delta \cup \{g_t\}$
12 $G_{\text{completed}} \leftarrow G_{\text{completed}} \cup \{g\}$
13 $tks_{\text{local}} \leftarrow tks_{\text{local}} + g$.local_consumed_tokens
14 **else**
15 $g_n \leftarrow$ `verify_pseudo_gradients`(Δ)
16 $\theta \leftarrow \text{Nesterov}(\theta, g_n)$
17 `send_params`$(\theta, G_{\text{completed}})$
18 $\tau_{\text{sync}} \leftarrow \infty$, $G_{\text{completed}} \leftarrow \emptyset$, $\Delta \leftarrow \emptyset$



(a) The training loss of asynchronous cross-cluster training with `Di-PS`.

(b) The detailed training performance distribution of different heterogeneity values η.

Dataset	Metric	DP	DiLoCo	Di-PS						
				η =10	η =20	η =30	η =40	η =50	η =100	η =200
BBH	acc	31.11	29.52	29.23	29.46	30.09	30.47	29.45	28.94	29.67
MMLU	acc	24.38	24.16	24.18	24.94	24.61	24.21	24.66	24.76	26.34
DROP	acc	31.77	27.88	31.45	32.75	31.02	31.22	31.53	31.42	29.94

(c) Model evaluation results.

Figure 7: `Di-PS` enables asynchronous cross-cluster training to converge as synchronous methods across various heterogeneous training cluster emulations. $\eta = x$ indicates that the computational performance among the clusters varies uniformly by 0–x%.

we treat training clusters as candidates for PS deployment.

4.2 Stable Asynchronous Training

Asynchronous cross-cluster training may compromise accuracy, as the outer optimization encounters instability when the pseudo-gradients from training clusters are low-quality or stale [15, 48]. Based on previous works [6, 59], and assuming the objective function is L-smooth and gradients satisfy the (M, σ^2)-bounded noise conditions, we can derive an upper bound on the error and proves the sub-linear convergence rate of asynchronous two-stage optimization as $O(\tau/T + \sigma/\sqrt{T})$, where τ denotes the asynchronous delay in the training system and σ represents the noise variance of the stochastic gradient. (The proof is presented in Appendix B.) Our proposed efficient PS design reduces the asynchronous delay τ. To improve the convergence of asynchronous training, inspired by EDiT [15], we implement a pseudo-gradient penalty strategy on the PS architecture to reduce σ of gradients caused by asynchronism. The gradient penalty procedure in `Di-PS` is follows:

1. **Distributed norm computation:** To get the complete view of pseudo-gradient, the leader PS aggregates norms of pseudo-gradient for each training cluster from follower PS in outer optimization step t. The total norm of training cluster j can be denoted as $G_t^j = \sum_{i=1}^{n} \|\Delta_t^{i,j}\|_2$, where n is the number of follower PSs and $\Delta_t^{i,j}$ denotes the pseudo-gradient computed in follower PS i.

2. **Outlier detection:** The leader PS uses an exponential moving average score vector to estimate convergence trends and find the outlier gradients. Specifically, we have $E_t = \frac{G_t - \mu_t}{\sigma_t}$, where μ_t and σ_t represent the exponentially weighted moving average mean and standard deviation of G_t, with the recurrence relation of $\mu_{t+1} = \alpha G_t + (1-\alpha)\mu_t$, $\sigma_{t+1} = \sqrt{(1-\alpha)(\sigma_t)^2 + \alpha(G_t - \mu_{t+1})^2}$. We maintain a stack $\mathbf{E}$ to record recent scores in the leader PS. When the score E_t^i of training cluster i exceeds $\beta \max(\mathbf{E})$, its gradient is flagged as abnormal and excluded from the parameter update across all follower PSs. The average factor α and scaling threshold β are hyperparameters, and set to 0.02 and 3 in our practice. Discussions of hyperparameter selections are provided in Appendix C. The updated set of participating training clusters is denoted by c. Leader PS then sends c back to follower PSs and pushes E_t^c into $\mathbf{E}$.

3. **Consumed-token based weighted averaging:** When multiple training clusters participate in this optimization step, their gradients can be averaged based on the corresponding consumed data tokens, which are tracked by the leader PS after each cluster pushes its parameters. The resulting pseudo-gradient at step t is $\delta_t^i = \frac{\sum_{j \in c} T_j \Delta_t^{i,j}}{\sum_{j \in c} T_j}$, where T_j is the number of consumed data tokens of cluster j. Then a gradient clipping ($g_n = \min(1, \frac{1}{\|\delta_t\|_2})\delta_t$) is applied, and the clipped pseudo-gradient g_n is used to update the outer optimizer in follower PSs.

To this end, we implement asynchronous outer optimization as shown in Algorithm 1, using Nesterov [61] as the outer optimizer in follower PSs and AdamW [50] as the inner optimizer. In each global step, we use `get_cluster(.)` to obtain a cluster that wants to perform outer optimization in the given time window τ_{sync}, while maintaining a short grace period τ_{grace} to allow other clusters to synchronize their local parameters (Lines 6–8). After calculating the pseudo-gradient

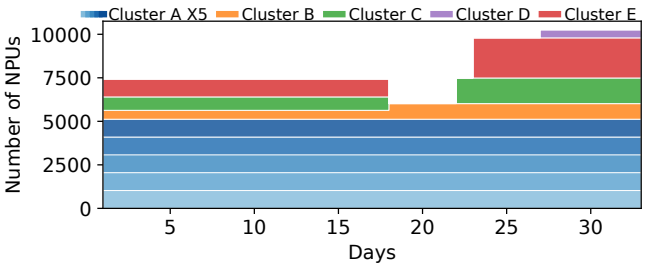


Figure 8: Available resource timeline of 9 training clusters during a 33-day training of a 100B LLM, with a notable achievement of scaling up to 10,122 NPUs.

Table 2: Categorized failures and their recovery overhead observed during the 33-day training of a 100B model.

Category	Reasons	Amount	Avg. Recovery Time (min)
Hardware	Network Interface	2	95.71
	Faulty NPUs	2	109.54
	HBM Overflow	5	88.21
	Storage Device	1	10.63
	Backplane	1	30.38
Software	Collective Failure	17	41.78
	Framework Issue	3	46.46
	User Code Bug	3	68.60
	Configuration Issue	2	92.73
	Management System	5	159.54
Di-PS	Leader PS Failure	1	44.43
	Follower PS Issue	3	3.05

with collected parameters and verifying it with the aforementioned penalty strategy, we can update global parameters with the outer optimizer (Lines 9–16). Next, the leader PS records the training progress and follower PSs send the updated parameters to the clusters that participate in this step (Lines 17–18).

To demonstrate robustness of Di-PS, we evaluate the pretraining of a LLaMA3.2-1B model on four simulated heterogeneous training clusters (as shown in Figure 7b) with up to 200% training performance disparity (η=200). Using an inner step of 64 for two-stage optimization, the training loss results in Figure 7a show that Di-PS can provide a stable convergence in asynchronous cross-cluster training. We further evaluate trained models on popular benchmarks [24, 30, 62], as shown in Figure 7c, results are comparable to synchronous DiLoCo across all heterogeneous scenarios, as the pseudo-gradient penalty strategy can avoid training failures of asynchronous DiLoCo in the same tasks (Figure 2b).

4.3 Resilience and Fault-tolerance Mechanism

Failures Analysis. In our production 100B LLM training that spans 33 days across 9 training clusters with up to 10,122 NPUs, we observe three trends: (i) Training resources across clusters are dynamic, with scaling events (both expansions and contractions) due to resource fluctuations (Figure 8). (ii) Clusters experience hardware and software failures that are recovered or alerted by the cluster management system, while

Table 3: The failures detect time and process restart time of components of Di-PS.

	Leader PS			Follower PS		
Model Size	1B	14B	100B	1B	14B	100B
Detect Time (min)	40.5	31.7	38.9	1.52	1.41	1.35
Restart Time (min)	0.21	0.22	0.21	0.18	2.56	1.14

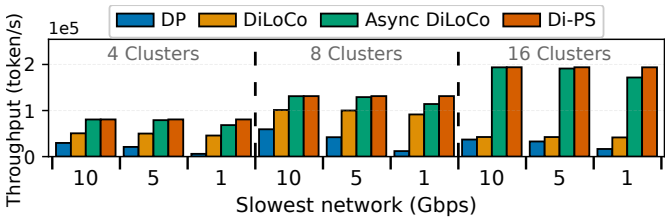
failures within the management system itself lead to longer downtime (Table 2). (iii) Failure occurrences are heterogeneous across clusters, with newly deployed clusters exhibiting higher failure frequency (Table 1), primarily due to limited burn-in time. Consequently, the cross-cluster training system needs to tolerate frequent cluster join and removal, while also incorporating resilience against failures within its own. While derived from a single long-running production job, these observations motivate system requirements that are applicable to other large-scale LLM trainings across multiple training clusters.

Training Cluster Resilience. To support the dynamic addition and removal of training clusters in Di-PS, live clusters periodically transmit heartbeat signals to the leader PS. These heartbeats allow the leader PS to integrate new clusters and remove unresponsive ones during training. When a new cluster joins, the leader PS instructs follower PSs to send the latest parameters for initialization. Conversely, if the leader PS fails to receive heartbeats from a cluster for three consecutive intervals, it automatically removes that cluster from the outer optimization process.

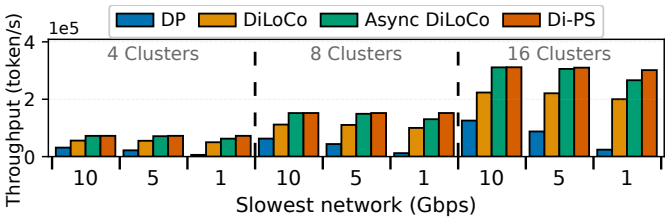
Failure Tolerance of Di-PS. Di-PS also encounters failures in large-scale training. We observe 4 failures in Di-PS during the 33-day training with 16 distributed follower PSs. To address failures in the Di-PS, the leader PS saves metadata (<100KB), and each follower PS asynchronously checkpoints its model state after every outer optimization step. The model state also supports model evaluation. To limit storage overhead, only a few recent snapshots are kept. Follower PSs periodically send heartbeats to the leader PS. The leader PS monitors these heartbeats and restarts failed follower PSs using the latest checkpoint.

If the leader PS fails, preventing heartbeats from training clusters and follower PSs, these components will persistently attempt to reconnect in a non-blocking manner until a connection is re-established. The cluster management system (e.g., Kubernetes) readily detects and restarts the leader PS. During this period, outer optimization steps are skipped while inner training steps continue, ensuring training throughput is unaffected.

Resilience Performance. To quantify the fault tolerance performance of Di-PS, we conduct fault-injection experiments on both leader and follower PSs with three model sizes, using 1 follower PS for the 1B/14B models and 16 for the 100B model. Table 3 reports the recovery time, broken down into fault detection and process restart. The leader PS detection



(a) Real clusters, LLaMA 3.2-1B.



(b) Emu-S clusters, LLaMA3.2-1B.

(c) Emu-L clusters, Qwen3-14B.

Dataset	Metric	LLaMA3.2-1B			Qwen3-14B		
		DP	DiLoCo	Di-PS	DP	DiLoCo	Di-PS
BBH	acc	31.24	31.13	31.35	69.44	68.41	72.61
MMLU	acc	27.04	26.02	26.69	77.08	73.37	76.10
DROP	acc	31.77	31.11	31.42	70.34	64.58	68.17

(d) Evaluation results on models trained with 16 clusters.

Figure 9: End-to-end training and model performance of different implementations on cross-cluster LLM training.

time is aligned with failures in training clusters (Table 2), while follower PS failures are detected faster due to more frequent heartbeat monitoring by the leader PS. Restarting follower PSs is slower due to model state reloading, which completes within minutes. The delay grows with the model size but is mitigated by partitioning states across distributed follower PSs.

5 Evaluation

We evaluate the effectiveness of Di-PS in both controlled experimental settings and large-scale production environments. The experimental evaluation utilizes small clusters with controllable network bandwidth and training performance to enable fair comparisons with baseline methods (§5.1–5.3). We also report the performance Di-PS in the production training of a 100B LLM involving up to nine training clusters and 10,112 NPUs (§5.4).

5.1 End-to-end Performance Comparison

Testbeds and Models. We evaluate the end-to-end cross-cluster training performance of Di-PS on three heterogeneous cluster configurations: (i) **16 real clusters**, including diverse GPU configurations—2×H800, 1×H800, 2×A100, 1×A100, 4×3090, 16×2080Ti, 16×2080, 8×2080Ti, and 8×2080. Among them, eight clusters use the 4×3090 configuration. These clusters exhibit heterogeneous training throughput, with performance disparities of up to 8.18×. (ii) **16 emulated small clusters (Emu-S)**, each equipped with a single 80GB H800 GPU. To simulate heterogeneity, we inject artificial training delays of up to 100% of a training iteration ($\eta = 100$), resulting in uniformly varying training speeds across clusters. (iii) **16 emulated large clusters (Emu-L)**, each equipped with 8×80GB H800 GPUs. Similarly, we introduce artificial training delays ($\eta = 100$) to emulate heterogeneous performance. We train two models: LLaMA3.2-1B [28], used in real and Emu-S clusters, and Qwen3-14B [67], used in Emu-L to evaluate performance and scalability on larger models. For all experiments, we report the aggregated training throughput across all clusters, as intra-cluster training performance under data parallelism remains consistent across baselines.

Baselines. We compare Di-PS with three representative baselines: (i) Data parallelism (DP), which synchronizes the model across all clusters in every iteration. (ii) Synchronous DiLoCo [23], which trains the model within each cluster for multiple inner steps and synchronously performs a global outer optimization to reduce the inter-cluster communication. (iii) Asynchronous DiLoCo (Async DiLoCo) on decentralized communication architecture [35, 36]. Similar to synchronous DiLoCo, but each cluster performs an asynchronous outer optimization whenever complete inner training steps. The pre-training hyperparameters of our experiments in each cluster are inner learning rate of 6e-5, batch size of 32K, inner step of 64, outer learning rate of 0.7, and outer momentum of 0.8, unless otherwise stated.

In experiments, one cluster is configured with the slowest inter-cluster bandwidth of 10, 5, or 1 Gbps, while other clusters use an inter-cluster bandwidth of 10 Gbps. The intra-cluster bandwidth is 100 Gbps in real clusters and 1600 Gbps in emulated clusters. Figure 9 compares the end-to-end training performance across different inter-cluster bandwidths and cluster numbers. Di-PS achieves 1.27–4.67× speedups over synchronous DiLoCo, as Di-PS adapts to heterogeneous computing resources. Async DiLoCo methods can leverage the training resources of clusters to achieve considerable training performance. However, it encounters convergence issues as shown in Figure 2b. Di-PS provides stable convergence (Figure 7a) and adapts to heterogeneous networks, accelerating end-to-end training by 1.00–1.60×. Specifically, Di-PS improves the outer optimization communication with higher acceleration as bandwidth disparity increases. This bandwidth gap can be even larger in distributed training clusters [32].

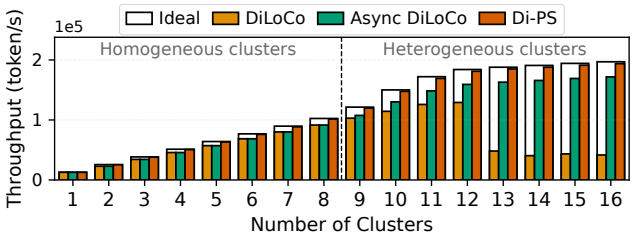


Figure 10: Weak scaling performance comparison and related ideal linear results.

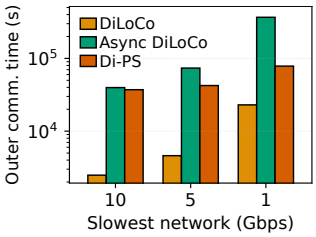


Figure 11: Accumulated outer communication overhead on heterogeneous inter-cluster networks and 16 training clusters.

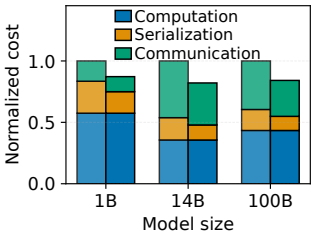


Figure 12: Effectiveness of optimizations in Di-PS. The right bar represents the performance after applying optimizations.

We further evaluate models trained with 16 clusters on benchmarks spanning diverse domains, including BBH [62] (reasoning), MMLU [30] (knowledge), and DROP [24] (comprehension). As shown in Figure 9d, models trained with Di-PS achieve performance comparable to DP and DiLoCo, confirming that our asynchronous outer optimization does not compromise model quality.

5.2 Scalability

We evaluate the scalability of Di-PS through weak scaling experiments on the real clusters in § 5.1, where the number of clusters is gradually increased to 16, and each cluster trains a LLaMA3.2-1B model. During the scaling, the first 8 clusters share the same configuration of 4×3090 GPUs to demonstrate scalability in a homogeneous setting, while the remaining clusters are added in descending order of training performance to illustrate scalability under heterogeneous conditions. The first cluster is connected with a 1 Gbps inter-cluster network, while all remaining clusters have 10 Gbps bandwidth.

Figure 10 compares the aggregated training performance across all clusters. In homogeneous settings, DiLoCo methods perform similarly, while Di-PS achieves 1.00–1.11× speedups by better utilizing heterogeneous inter-cluster bandwidth. In heterogeneous settings, Di-PS outperforms async DiLoCo with 1.11–1.13× speedups. The addition of a slower training cluster causes synchronous DiLoCo to suffer from straggler effects, which results in the fastest performance occurring with 12 clusters in synchronous DiLoCo. We also compare Di-PS against an ideal training performance in which each cluster trains independently without inter-cluster communication. Except for the single-cluster case, Di-PS achieves 98.3–98.8% of the ideal performance, demonstrating excellent scalability.

5.3 Ablation Study

Communication in Outer Optimization. We can compare the inter-cluster communication overhead alone to isolate the training performance benefits of Di-PS. Figure 11 shows the aggregated outer communication time from end-to-end experiments (§5.1) after training LLaMA3.2-1B model with 10B data tokens. Synchronous DiLoCo incurs fewer inter-cluster communications, as it waits for all clusters before proceeding an outer optimization. The overhead of each inter-cluster communication increases for both DiLoCo and Async DiLoCo as network heterogeneity grows. In contrast, Di-PS effectively mitigates this overhead by adapting to heterogeneous networks, achieving a 1.06–4.69× reduction in total inter-cluster communication time compared to Async DiLoCo.

Follower PS Optimizations. The key components of the follower PS's outer optimization loop include communication, communication data serialization/deserialization, and optimizer computation. To assess the effectiveness of the optimization techniques applied to the PS in Di-PS, we compared the overhead of the outer optimization process with and without our optimizations, training LLaMA-based LLMs with 1B, 14B, and 100B parameters, using 1, 1, and 16 follower PSs, respectively, over a 10 Gbps inter-cluster network.

Figure 12 presents the normalized operation costs for components of the outer optimization loop. As expected, the optimizer computation overhead remains constant. The communication scheduling from leader PS and multi-threading communication in follower PSs provide communication speedups of up to 1.39×. Additionally, overlapping operations within follower PSs improve data serialization performance by up to 1.48×. Overall, these optimizations in Di-PS result in a 1.21× acceleration of the outer optimization process.

5.4 Di-PS in Production Training

We deployed Di-PS for a production workload and trained a 100B LLaMA-based LLM (96 layers, hidden size 8192, intermediate size 36864), consuming a total of 2.3T tokens over 33 days. As previously mentioned, the training involves up to nine clusters with varying numbers of NPUs (Table 1), each of which is exclusively allocated to the workload. The training process dynamically scaled, reaching a peak of 10,112 NPUs (Figure 8). Figure 13 shows the topology of our training clusters, with intra-cluster topology details provided in Appendix D. To support the model's large size, we utilized 16 follower PSs, each deployed on a dedicated CPU server.

Convergency. To consistently present convergence performance in training with Di-PS, Figure 14 shows the training loss curves for all clusters. Over the training of 33 days,

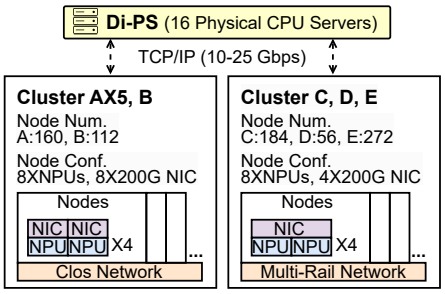


Figure 13: Topology of the training clusters in production training.

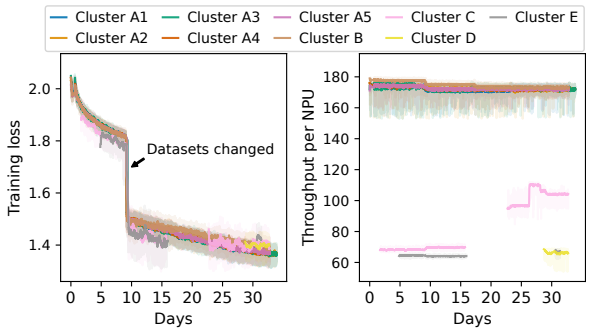


Figure 14: Training loss (left) and training performance (right) in production training.

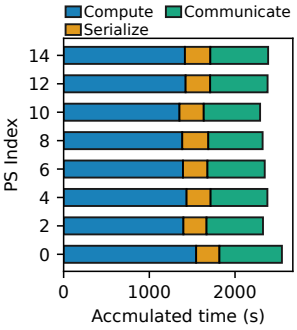


Figure 15: Operation breakdowns in follower PSs.

Table 4: Evaluation result comparison with recent LLMs.

Dataset Metric	BBH acc	MMLU acc	CMMLU acc	DROP acc	MBPP score	GSM8K acc	HellaSwag acc
Ours	**83.4**	81.4	83.5	80.2	72.0	84.5	**93.2**
LLaMA3.1-70B	81.6	79.3	68.8	79.6	66.2	83.6	79.9
Qwen2.5-72B	79.8	**85.0**	**89.5**	80.6	**72.6**	**88.3**	84.8
LLaMA3.1-405B	82.9	84.4	73.7	**86.0**	68.4	83.5	89.2

Table 5: Step time breakdown by clusters (seconds).

	Cluster A	Cluster B	Cluster C	Cluster D	Cluster E
Push	195	193	60	86	116
Update	110	110	110	110	110
Pull	135	134	67	80	121

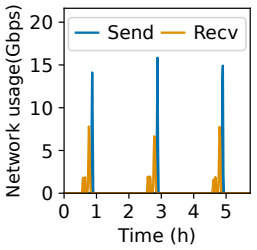


Figure 16: A six-hour network utilization trace of a follower PS.

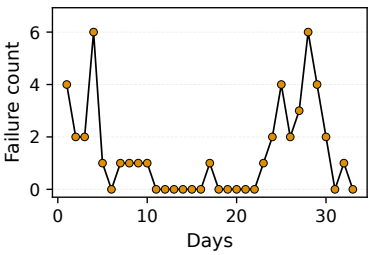


Figure 17: The temporal failure statistics over the production training.

Di-PS demonstrates stable convergence across all clusters in this large-scale production training scenario. Table 4 reports the evaluation results of our trained base model against recent LLaMA-based dense models [28, 55], with benchmarks of additional domains [8, 19, 42, 68]. Our model outperforms LLaMA3.1-70B and, despite having fewer parameters, surpasses LLaMA3.1-405B on most benchmarks. Its performance is comparable to Qwen2.5-72B, which is expected given that our training data is less recent. Overall, the evaluation results align well with our expectations.

Training Cluster Efficiency. On the training cluster side, we first show the throughput per NPU in Figure 14. Most NPUs achieve stable and consistent training efficiency, with failures being quickly recovered once they occur. The noticeable performance changes in Cluster C result from adjustments to intra-cluster parallel configurations. The average time breakdown for each cluster is presented in Table 5. As shown in the table, parameter communication (including push & pull) and update time constitute only a small fraction of the overall training process, accounting for about 6%. Notably, Clusters C, D, and E experienced significantly lower communication time compared to Clusters A and B, due to larger pipeline parallelism configurations. This allows more devices to interact with Di-PS simultaneously, improving communication efficiency.

Di-PS Efficiency. Diving into the performance of the Di-PS, we first present the accumulated running time breakdown over 24 hours for follower PSs, as shown in Figure 15. The optimization process on the CPU emerges as the dominant source of overhead. This bottleneck could potentially be alleviated by improving CPU operation implementations and by overlapping communication with computation to improve system efficiency. Figure 16 shows a network trace of a follower PS over a six-hour period. Follower PSs interact with training clusters approximately every two hours, aligned with training iteration schedules. Due to variations in training time across clusters and the grace time design described in § 4.1, follower PS receives parameters from clusters asynchronously and sends updated parameters in bursts, reaching a peak usage of 16.4 Gbps on a 25 Gbps network. Follower PSs are idle over 95% of the time with 9 training clusters, suggesting ample capacity for additional clusters and larger-scale training. While the centralized PS design may eventually face challenges at extreme scales (e.g., hundreds of clusters), this remains well beyond the scale of practical deployments today, where each cluster typically comprises thousands of NPUs.

Fault tolerance. To better characterize system robustness over time, we further analyze the temporal distribution of failures during the 33-day production run. On average, we observed 1.3 failure events per day, with the majority being transient and automatically recovered by the system. Figure 17 summarizes the per-day failure counts across the entire training period. The centralized PS of Di-PS plays a pivotal role in isolating faults and maintaining overall training progress. Failures in one training cluster and the joining or removal of training clusters, do not impact the training of other clusters.

6 Experience and Lessons

1. Building multiple small clusters can be more practical and feasible than a single large cluster. Intra-cluster training is highly bandwidth-sensitive, and sustaining high utilization typically requires premium topologies (e.g., low-oversubscription fabrics). The cost of such topologies grows superlinearly with the cluster size, making mega-clusters prohibitively expensive. By contrast, assembling multiple smaller geo-distributed clusters reduces cost and management overhead (e.g., power, cooling, and failure isolation) while still providing aggregate capacity comparable to a single large cluster—enabled by effective cross-cluster training.

2. Controlling heterogeneity is required to prevent wasted computation. Excessive performance disparity (e.g., over 100×) across clusters leads asynchronous optimizers to discard many stale updates. Very slow clusters may keep producing gradients that are never accepted, silently wasting resources, which was observed in early-stage small-scale experiments but did not occur in production training. This highlights the need for more sophisticated two-stage optimization strategies to balance contributions across clusters of varying speeds, ensuring that slower clusters can still meaningfully participate without compromising overall convergence. Without such mechanisms, adding highly imbalanced clusters yields diminishing returns.

3. Proactive error reporting improves recovery. Recovery is faster when faults are explicitly reported rather than inferred from secondary symptoms (e.g., throughput drops or loss spikes). For example, the configuration issue in Table 2 was detected through reduced training speed, while follower PS failures were quickly recovered thanks to proactive heartbeat signals. Such structured reporting shortens the detection-to-recovery time and improves end-to-end robustness. This suggests that training systems should treat explicit failure reporting as a important primitive rather than an auxiliary monitoring feature.

4. Data partitioning consistency. We find that ensuring consistent data partitioning across training clusters is crucial when the number of clusters is dynamic. In our setup, we pre-partition the dataset into a significantly larger number of chunks than the number of training clusters, ensuring that each cluster receives a balanced and representative data distribution within a chunk. By maintaining a high partition-to-cluster ratio, we ensure data consistency within each cluster, which is crucial for stable convergence during training. This highlights the need for principled, globally coordinated data partitioning to sustain reliable training at scale.

7 Related Works

Intra-cluster Parallel LLM Training. LLM training leverages multiple parallelism strategies to scale within a single cluster. Data and sharded data parallelism [14, 43, 56, 70] distribute training states across workers to balance memory and computation. Pipeline and tensor parallelism [40, 49, 57] further partition model layers or operators to improve utilization. Sequence parallelism [34, 45] extends support for extremely long sequences by sharding attention across devices. Recent systems [13, 38, 41] combine these strategies to provide efficient intra-cluster training.

Cross-cluster Training. Building on the two-stage optimization algorithm DiLoCo [23], OpenDiLoCo [36] further reduces inter-cluster communication size through FP16 AllReduce. Prime [35] introduces a hybrid DiLoCo-FSDP approach to lower memory overhead, while Streaming DiLoCo [22] overlaps inter-cluster communication with computation by synchronizing parameter subsets sequentially. These techniques are complementary to `Di-PS`, which does not compress inter-cluster communication. Gaia [32] reduces WAN traffic via its approximate synchronous parallel model, which is effective for canonical ML tasks.

Intra-cluster Resilience. Recent advances in intra-cluster fault tolerance introduce self-healing mechanisms to ensure high availability within a cluster. Oobleck [37] and ReCycle [27] leverage inherent computation redundancy in parallel LLM training to enable uninterrupted training with failures. Unicron [29] integrates in-band error detection and dynamic reconfiguration to minimize downtime across training jobs. Similarly, production systems such as MegaScale [38] adopt checkpoint-free recovery and proactive failure isolation.

8 Conclusion

In this work, we introduce `Di-PS`, a novel training system designed to train LLMs on multiple clusters. By leveraging a PS-based system-algorithm co-design, `Di-PS` efficiently reduces inter-cluster communication overhead, improves cross-cluster training convergence, and enables inter-cluster fault tolerance. Our system efficiently utilizes over 10,000 NPUs from 9 training clusters and enables the successful training of a 100B-parameter model, demonstrating a promising approach to large-scale LLM training.

Acknowledgments

We sincerely thank our shepherd, Hong Xu, and anonymous NSDI reviewers for their valuable comments. This work is sponsored in part by the National Natural Science Foundation of China under Grant No. 62025208 and No. 62421002; Shanghai Municipal Science and Technology Major Project; and the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative (including cash and in-kind contributions from industry partners). We also gratefully acknowledge Shanghai Artificial Intelligence Laboratory. The corresponding authors of this paper are Zhiquan Lai (zqlai@nudt.edu.cn) and Xingcheng Zhang.

References

[1] gRPC: A high performance, open source universal RPC framework, 2025. https://grpc.io/.

[2] The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, 2025. https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

[3] TorchElastic, 2025. https://pytorch.org/docs/stable/elastic/quickstart.html.

[4] ZeroMQ: An open-source universal messaging library, 2025. https://zeromq.org/.

[5] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 118–132, 2023.

[6] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *Algorithmic Learning Theory*, pages 111–132. PMLR, 2020.

[7] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 472–487, 2022.

[8] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[9] Marek Bolanowski, Kamil Żak, Andrzej Paszkiewicz, Maria Ganzha, Marcin Paprzycki, Piotr Sowiński, Ignacio Lacalle, and Carlos E Palau. Eficiency of rest and grpc realizing communication tasks in microservice-based ecosystems. In *New trends in intelligent software methodologies, tools and techniques*, pages 97–108. IOS Press, 2022.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco. *arXiv preprint arXiv:2503.09799*, 2025.

[12] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao 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*, pages 178–191, 2024.

[13] Qiaoling Chen, Diandian Gu, Guoteng Wang, Xun Chen, YingTong Xiong, Ting Huang, Qinghao Hu, Xin Jin, Yonggang Wen, Tianwei Zhang, et al. Internevo: Efficient long-sequence large language model training via hybrid parallelism and redundant sharding. *arXiv preprint arXiv:2401.09149*, 2024.

[14] Qiaoling Chen, Qinghao Hu, Guoteng Wang, Yingtong Xiong, Ting Huang, Xun Chen, Yang Gao, Hang Yan, Yonggang Wen, Tianwei Zhang, et al. Lins: Reducing communication overhead of zero for efficient llm training, 2024.

[15] Jialiang Cheng, Ning Gao, Yun Yue, Zhiling Ye, Jiadi Jiang, and Jian Sha. EDit: A local-SGD-based efficient distributed training method for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[16] Runxiang Cheng, Chris Cai, Selman Yilmaz, Rahul Mitra, Malay Bag, Mrinmoy Ghosh, and Tianyin Xu. Towards gpu memory efficiency for distributed training at scale. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pages 281–297, 2023.

[17] Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, et al. MAST: Global scheduling of ML training across Geo-Distributed datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 563–580, 2024.

[18] Tapan Chugh, Srikanth Kandula, Arvind Krishnamurthy, Ratul Mahajan, and Ishai Menache. Anticipatory resource allocation for ml training. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pages 410–426, 2023.

[19] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al.

Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[20] Haotian Dong, Jingyan Jiang, Rongwei Lu, Jiajun Luo, Jiajun Song, Bowen Li, Ying Shen, and Zhi Wang. Beyond a single ai cluster: A survey of decentralized llm training. *arXiv preprint arXiv:2503.11023*, 2025.

[21] Jianbo Dong, Kun Qian, Pengcheng Zhang, Zhilong Zheng, Liang Chen, Fei Feng, Yichi Xu, Yikai Zhu, Gang Lu, Xue Li, et al. Evolution of aegis: Fault diagnosis for AI model training service in production. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 865–881, 2025.

[22] Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, et al. Streaming diloco with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512*, 2025.

[23] Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

[24] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, 2019.

[25] Jiangfei Duan, Ziang Song, Xupeng Miao, Xiaoli Xi, Dahua Lin, Harry Xu, Minjia Zhang, and Zhihao Jia. Parcae: Proactive, Liveput-Optimized DNN training on preemptible instances. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1121–1139, 2024.

[26] Jiangfei Duan, Shuo Zhang, Zerui Wang, Lijuan Jiang, Wenwen Qu, Qinghao Hu, Guoteng Wang, Qizhen Weng, Hang Yan, Xingcheng Zhang, et al. Efficient training of large language models on distributed infrastructures: a survey. *arXiv preprint arXiv:2407.20018*, 2024.

[27] Swapnil Gandhi, Mark Zhao, Athinagoras Skiadopoulos, and Christos Kozyrakis. Recycle: Resilient training of large dnns using pipeline adaptation. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 211–228, 2024.

[28] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[29] Tao He, Xue Li, Zhibin Wang, Kun Qian, Jingbo Xu, Wenyuan Yu, and Jingren Zhou. Unicron: Economizing self-healing llm training at scale. *arXiv preprint arXiv:2401.00134*, 2023.

[30] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[31] Torsten Hoefler, Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Shigang Li, Marco Heddes, Jon Belk, Deepak Goel, Miguel Castro, and Steve Scott. Hammingmesh: A network topology for large-scale deep learning. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–18. IEEE, 2022.

[32] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. Gaia:Geo-Distributed machine learning approaching LAN speeds. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 629–647, 2017.

[33] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, et al. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 709–729, 2024.

[34] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

[35] Sami Jaghouar, Jack Min Ong, Manveer Basra, Fares Obeid, Jannik Straube, Michael Keiblinger, Elie Bakouch, Lucas Atkins, Maziyar Panahi, Charles Goddard, et al. Intellect-1 technical report. *arXiv preprint arXiv:2412.01152*, 2024.

[36] Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. Opendiloco: An open-source framework for globally distributed low-communication training. *arXiv preprint arXiv:2407.07852*, 2024.

[37] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. Oobleck: Resilient distributed

training of large models using pipeline templates. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 382–395, 2023.

[38] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

[39] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[40] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

[41] Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1466–1478, 2023.

[42] Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11260–11285, 2024.

[43] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, aug 2020.

[44] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-ai: A unified deep learning system for large-scale parallel training. In *Proceedings of the 52nd International Conference on Parallel Processing*, pages 766–775, 2023.

[45] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence parallelism: Long sequence training from system perspective. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2391–2404, 2023.

[46] Zhiqi Lin, Youshan Miao, Quanlu Zhang, Fan Yang, Yi Zhu, Cheng Li, Saeed Maleki, Xu Cao, Ning Shang, Yilei Yang, et al. nnScaler:Constraint-Guided parallelization plan generation for deep learning training. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 347–363, 2024.

[47] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[48] Bo Liu, Rachita Chhaparia, Arthur Douillard, Satyen Kale, Andrei A Rusu, Jiajun Shen, Arthur Szlam, and Marc'Aurelio Ranzato. Asynchronous local-sgd training for language modeling. *arXiv preprint arXiv:2401.09135*, 2024.

[49] Weijie Liu, Kai Lu, Zhiquan Lai, Shengwei Li, Keshi Ge, Dongsheng Li, and Xicheng Lu. Autopipe-h: A heterogeneity-aware data-paralleled pipeline approach on commodity gpu servers. *IEEE Transactions on Computers*, 2024.

[50] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[51] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. Heterogeneity-aware distributed machine learning training via partial reduce. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2262–2270, 2021.

[52] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[53] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan 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*, pages 1–15, 2021.

[54] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *The Twelfth International Conference on Learning Representations*, 2024.

[55] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang,

Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[56] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[57] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[58] Dharma Shukla, Muthian Sivathanu, Srinidhi Viswanatha, Bhargav Gulavani, Rimma Nehme, Amey Agrawal, Chen Chen, Nipun Kwatra, Ramachandran Ramjee, Pankaj Sharma, et al. Singularity: Planet-scale, preemptive and elastic scheduling of ai workloads. *arXiv preprint arXiv:2202.07848*, 2022.

[59] Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: Sgd with delayed gradients. *Journal of Machine Learning Research*, 21(237):1–36, 2020.

[60] Foteini Strati, Zhendong Zhang, George Manos, Ixeia Sánchez Périz, Qinghao Hu, Tiancheng Chen, Berk Buzcu, Song Han, Pamela Delgado, and Ana Klimovic. Sailor: Automating distributed training over dynamic, heterogeneous, and geo-distributed clusters. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pages 204–220, 2025.

[61] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[62] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

[63] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[64] Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. Metis: Fast automatic distributed training on heterogeneous {GPUs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 563–578, 2024.

[65] Marcel Wagenländer, Guo Li, Bo Zhao, Luo Mai, and Peter Pietzuch. Tenplex: Dynamic parallelism for deep learning using parallelizable tensor collections. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 195–210, 2024.

[66] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. GREYHOUND: Hunting Fail-Slows in Hybrid-Parallel training at scale. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pages 731–747, 2025.

[67] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[68] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.

[69] Xinchun Zhang, Aqsa Kashaf, Yihan Zou, Wei Zhang, Weibo Liao, Haoxiang Song, Jintao Ye, Yakun Li, Rui Shi, Yong Tian, et al. Reslake: Towards minimum job latency and balanced resource utilization in geo-distributed job scheduling. *Proceedings of the VLDB Endowment*, 17(12):3934–3946, 2024.

[70] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. Mics: near-linear scaling for training gigantic model on public cloud. *Proceedings of the VLDB Endowment*, 16(1):37–50, 2022.

[71] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12):3848–3860, 2023.

[72] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang,

Yuanzhong Xu, Danyang Zhuo, Eric 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)*, pages 559–578, 2022.

[73] Yonghao Zhuang, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, Hao Zhang, and Hexu Zhao. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems*, 5:526–540, 2023.

A Communication Scheduling Strategy

To specify the communication scheduling strategy in the leader PS (§ 4.1), we consider a model with N layers, each treated as an indivisible communication unit. The layers are evenly and contiguously partitioned across M workers in one training cluster, $\mathcal{M} = \{m_0, m_1, \ldots, m_{M-1}\}$, and across K follower PSs, $\mathcal{S} = \{s_0, s_1, \ldots, s_{K-1}\}$. Each NPU m_i holds an ordered set of layers $\mathcal{L}_i = \{l_0^i, l_1^i, \ldots\}$, and each layer l has a fixed destination $\text{dest}(l) \in \mathcal{S}$.

A communication schedule is an ordered sequence of communication steps. A communication step is defined as a set of transmissions executed in parallel under the no-conflict constraint: no two layers in the same step target the same follower PS. At step t, let $\mathcal{M}_t \subseteq \mathcal{M}$ be the set of workers with non-communicated layers, and D_t be the set of servers already assigned in this step. The step is constructed as

$$\mathcal{W}_t' = \left\{(m,l) \;\middle|\; m \in \mathcal{M}_t,\ l = \min \mathcal{L}_m,\ \text{dest}(l) \notin D_t\right\},$$

where $D_t = \{\text{dest}(l') \mid (m', l') \in \mathcal{W}_t'\}$. After scheduling $\mathcal{W}_t'$, we update $\mathcal{L}_m \leftarrow \mathcal{L}_m \setminus \{l \mid (m,l) \in \mathcal{W}_t'\}$ and repeat until all $\mathcal{L}_m = \varnothing$. The size of the communication step satisfies $|\mathcal{W}_t'| = \min(|\mathcal{M}_t|, K)$.

In the greedy mapping strategy, the leader PS iterates through workers that need to communicate with follower PSs in the order of PP rank, selecting the earliest unsent layer whose destination is not yet in D_t, while deferring the conflicting layers to subsequent steps. This one-pass procedure visits each layer exactly once with complexity $O(N)$, and maximizes concurrent non-conflicting transfers in each communication step. It achieves near-maximal utilization and supports dynamic integration of new clusters without re-planning existing schedules.

B Proof of Convergence Rate

The update steps of the asynchronous two-stage optimization algorithm can be expressed as follows. As the inner update on local model $\theta_t^{(i)}$ of cluster i with H local training steps is identical to the synchronous two-stage optimization, we have

$$\theta_{t,0}^{(i)} = \theta_t^{(i)}, \qquad \theta_{t,h+1}^{(i)} = \theta_{t,h}^{(i)} - \gamma_{\text{in}} \cdot g_{t,h}^{(i)},$$

where $g_{t,h}^{(i)}$ denotes the gradient at the h-th inner step with inner learning rate γ_{in}. The outer update then aggregates the local updates asynchronously and applies them with outer learning rate γ_{out} to the global model θ_t:

$$\theta_{t+1} = \theta_t - \gamma_{\text{out}} \cdot \Delta_{t-\tau},$$

where $\Delta_t = \theta_t - \theta_{t,H}^{(i)} = \gamma_{\text{in}} \sum_{h=0}^{H-1} g_{t,h}^{(i)}$ represents the pseudo-gradient accumulated over H inner steps, and τ denotes the staleness due to asynchronous training.

Following the analysis in [59], we define $v_t = \gamma_{\text{out}} \Delta_{t-\tau}$ for $t \geq \tau$ (and $v_t = 0$ otherwise), and introduce an error term $e_t = \sum_{j=1}^{\tau} \gamma_{\text{out}} \Delta_{t-j}$. Then the outer updates can be rewritten as

$$\theta_{t+1} = \theta_t - v_t, \qquad e_{t+1} = e_t + \gamma_{\text{out}} \cdot \Delta_t - v_t.$$

To facilitate the analysis, we further introduce a virtual sequence $\{\tilde{\theta}\}_{t \geq 0}$ defined as $\tilde{\theta}_t = \theta_t - e_t$. It then follows that

$$\begin{aligned}
\tilde{\theta}_{t+1} &= \theta_{t+1} - e_{t+1} \\
&= \theta_t - v_t - (e_t + \gamma_{\text{out}} \cdot \Delta_t - v_t) \\
&= \tilde{\theta}_t - \gamma_{\text{out}} \cdot \Delta_t.
\end{aligned}$$

Assuming the objective function is L-smooth and gradients satisfy the (M, σ^2)-bounded noise conditions (Assumptions 2 and 3 in [59]), a key observation is that the pseudo-gradient Δ, formed from H inner training steps, also satisfies a bounded noise property. Under the step-size condition $\gamma_{\text{in}}\gamma_{\text{out}} < \frac{1}{10LH(\tau+M)}$, the error-feedback framework in [59] (specifically, Lemmas 14, 20, and Theorem 16) can be extended to the two-stage setting, yielding the following convergence guarantee:

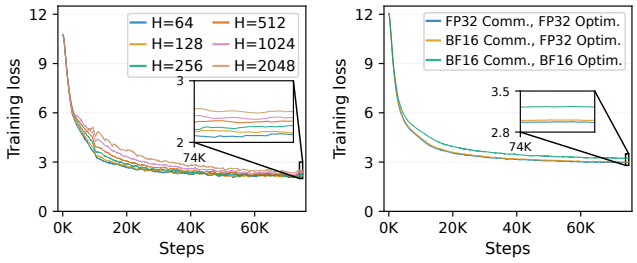
$$\mathbb{E}\left[\left\|\nabla f\left(\theta^{\text{out}}\right)\right\|^2\right] = O\left(\frac{\tau}{T} + \frac{\sigma}{\sqrt{T}}\right),$$

where θ^{out} is selected uniformly at random from the iterates $\{\theta_t\}_{t=0}^{T}$.

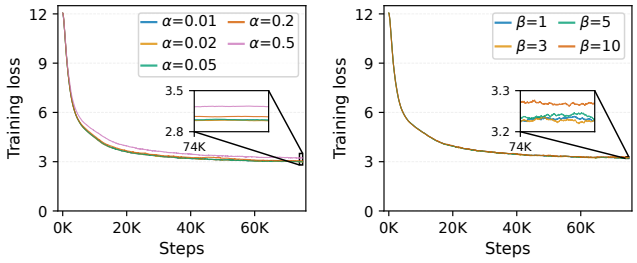
C Hyperparameter in Outer Optimization

To better understand the impact of hyperparameters in `Di-PS`, we conduct pretraining experiments on the LLaMA3.2-1B using four emulated training clusters with uniformly distributed performance disparities from 0 to 100% ($\eta = 100$).

Outer Optimization Intervals. We first study the effect of outer optimization intervals (i.e., the number of inner training steps). As shown in Figure 18a, varying the number of



(a) Inner steps (H). (b) Precisions.



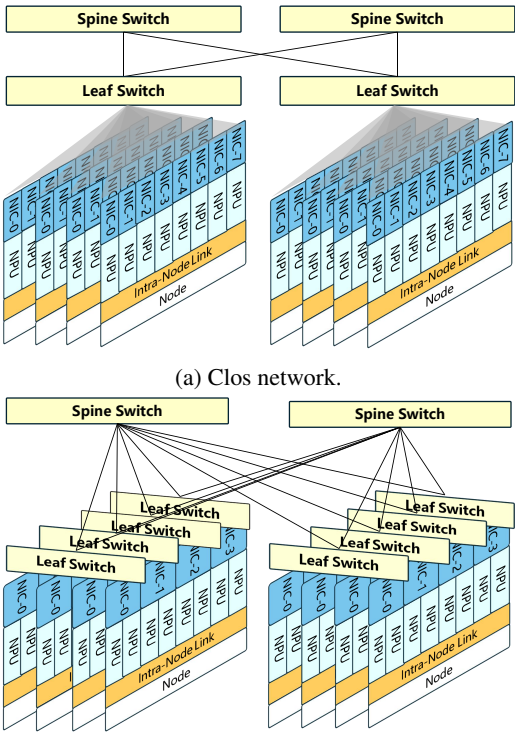
(c) Average factor (α). (d) Scaling threshold (β).

Figure 18: Sensitivity of hyperparameters in outer optimization.

inner training steps may impact convergence speed. The inner steps of 64 and 128 result in similar convergence performance, while other values tend to slow down convergence. In particular, larger inner steps lead to degraded convergence quality. Although increasing the number of inner steps improves overall cross-cluster training throughput, it can adversely affect convergence speed, presenting a trade-off that needs to be carefully balanced.

Precision Selection. To evaluate how precision affects convergence in the two-stage optimization of `Di-PS`, we conduct experiments with different precision configurations for inter-cluster communication of model parameters and the outer optimizer. We compare three configurations: (i) FP32 communication with FP32 outer optimizer; (ii) BF16 communication with FP32 outer optimizer; and (iii) BF16 communication with BF16 outer optimizer. Figure 18b shows the training loss curves for these settings. The results indicate that `Di-PS` converges reliably with FP32 outer optimizer. For efficiency, we choose BF16 for inter-cluster communication while keeping the outer optimizer in FP32, achieving a balance between communication cost and numerical stability.

Pseudo-gradient Penalty Hyperparameters. We provide additional results to evaluate the sensitivity of hyperparameters in the pseudo-gradient penalty, namely the averaging factor α and scaling threshold β. As shown in Figure 18c, we evaluate $\alpha \in 0.01, 0.02, 0.05, 0.2, 0.5$. Values below 0.05 achieve stable convergence. Larger α (e.g., 0.2 and 0.5) slow convergence slightly, while very small α (e.g., 0.01) increase noise. We therefore fix $\alpha = 0.02$ as the default. As shown in Figure 18d, we test $\beta \in 1, 3, 5, 10$. All settings maintain stable convergence. Higher values (e.g., $\beta = 10$) delay stabilization slightly, while very small values (e.g., $\beta = 1$) introduce unnecessary updates. We set $\beta = 3$ to default, which provides the best tradeoff. Overall, the pseudo-gradient penalty is robust to hyperparameter variations. The default values $\alpha = 0.02$ and $\beta = 3$ are well within the stable regions, and are consistently applied across other experiments in this paper.



(a) Clos network.

(b) Multi-rail network.

Figure 19: Intra-cluster network topology.

D Intra-cluster Topology

Communication overhead presents a significant challenge to scaling LLM training [33]. To address this bottleneck, Remote Direct Memory Access (RDMA) is utilized to facilitate high-speed, low-latency data transfer across training nodes. Unlike conventional TCP/IP networks, RDMA enables direct memory access between nodes without involving their operating systems, significantly reducing communication overhead. In this study, all NPUs within each of the 9 clusters are interconnected via an RDMA network utilizing RoCE-v2.

The intra-cluster network is configured with a 1:1 oversubscription ratio to ensure optimal data transmission efficiency. In clusters A and B, each training node is equipped with eight NPUs and eight network interface cards (NICs), each providing 200 Gbps of bandwidth. These servers are organized into racks connected to leaf switches, as illustrated in Figure 19a. The leaf switches, in turn, connect to spine switches, which provide inter-rack connectivity, forming a pod-based struc-

ture. In clusters C, D, and E, each node contains eight NPUs and four NICs, each offering 200 Gbps of bandwidth. Within each rail, NPUs that share the same index across different servers are interconnected via the same leaf switch, as depicted in Figure 19b. This configuration enhances collective communication performance. However, the multi-rail network design necessitates connecting NPUs to distant switches, requiring costly and power-intensive optical transceivers, which increases both power consumption and heat dissipation [5].