(Operational Systems Track) `Di-PS`: System-Algorithm Co-Design for Asynchronous and Heterogeneous Cross-cluster LLM Training at Scale

Abstract

Large language models (LLMs) have revolutionized artificial intelligence, exhibiting remarkable performance in different applications. Training these models demands extensive computational resources, which are often economically and physically prohibitive. *Cross-cluster training* can reduce the infrastructure costs, better match the task requirements of LLM development, and alleviate physical constraints through geo-distributed deployment. However, challenges arise from heterogeneous computational resources, network variability, and the 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 new master-slave parameter server paradigm, to achieve heterogeneous, asynchronous, and resilient training across decentralized clusters. We further make several innovative contributions in `Di-PS`, including (i) a dual-workflow mechanism for optimized cross-cluster communication and orchestration, (ii) a pseudo-gradient penalty strategy for convergence stability enhancement of asynchronous two-stage optimization, and (iii) a two-level resilience mechanism for inter-cluster fault tolerance guarantee. Results from the controlled experimental setting demonstrate that `Di-PS` improves training efficiency by up to 2.93× over synchronous cross-cluster approaches while maintaining convergence, along with 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 cross-cluster training of a 100B-parameter LLM with only 6% overhead compared to single-cluster training, and effectively handles frequent training 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, including intelligent assistants, code generation, and scientific research [20, 78]. This rapid progress is primarily driven by the scaling of Transformer-based models such as GPT [9], LLaMA [24], Gemini [69], and DeepSeek [49]. Training LLMs requires vast computational resources over extended periods, which is critically dependent on large-scale compute clusters. For instance, Meta's LLaMA-3 was trained on NVIDIA H100 GPUs [24], 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 [37]. Consequently, the pursuit of ultra-large-scale training remains active, along with the escalating demand for computational resources. However, such large scales of dedicated resources are either unavailable or economically unjustifiable for a broader range of researchers.

Benefits of Cross-cluster Training. Orchestrating training across multiple independent clusters offers an effective approach to scaling out LLM training without requiring substantial infrastructure overhauls, compared to constructing a single large-scale AI training cluster. This computing paradigm, known as *cross-cluster training*, exhibits several advantages in terms of cost efficiency, resource utilization, physical constraints, and training performance. (i) *Infrastructure cost*. Compared to setting up one large cluster interconnecting over 10,000 NPUs, building multiple smaller clusters is more economical. For example, a monolithic Clos network supporting 10,000 NPUs typically requires around 400 switches (based on 128-port switches), comprising 160 leaf switches, 160 spine switches, and 80 super-spine switches [27]. 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%. (ii) *LLM task requirements*. Most LLM development tasks do not necessitate all the resources of a massive cluster. While LLM pretraining jobs may involve thousands or tens of thousands of NPUs, more than 98% of LLM workloads—such as evaluation, post-training, and debugging—typically require no more than 100 NPUs [30]. (iii) *Physical constraints*. Deploying

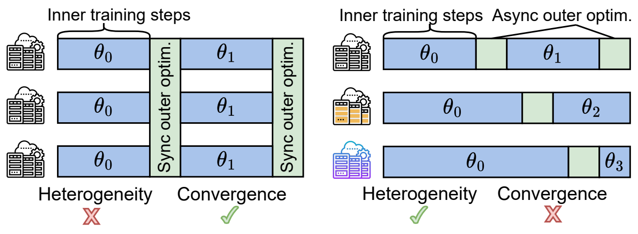
Table 1: Peak computational power, number of failures, and mean time between failures (MTFB) of training clusters used in a 33-day production-level training of a 100B LLM.

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

smaller clusters alleviates physical constraints by enabling geo-distributed deployment. The decentralized deployment eases demands on power, space, cooling, and infrastructure expansion, supporting more flexible deployment strategies. Smaller clusters also improve power distribution and reliability, lower the risk of large-scale power failures [5, 16], enhance fault isolation, and prevent localized failures from disrupting global training tasks [26,75]. (iv) *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 [24], 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 [49].

Challenges of Cross-cluster Training. Driven by these advantages, it becomes imperative to aggregate the capabilities of multiple clusters into a cohesive training framework. However, the significantly lower inter-cluster bandwidth [29], often hundreds of times less than intra-cluster bandwidth, prevents direct cross-cluster training. Inspired by federated learning (FL) [57,61], existing cross-cluster LLM training approaches use the Distributed Low-Communication (DiLoCo) algorithm [18]. It introduces a two-stage optimization process (Figure 1a), where each cluster trains a model in parallel with an inner optimizer and synchronizes models across clusters with the outer optimizer at a low frequency to reduce the cross-cluster communication overhead. While the centralized parameter server (PS) architecture in FL [62, 63] can support the distributed training of moderately sized models, it faces significant capacity limitations when applied to large-scale LLMs. Therefore, existing cross-cluster LLM training systems [17, 33, 34] adopt a decentralized architecture, where every cluster maintains an outer optimizer replica and synchronizes them with AllReduce communications.

The decentralized cross-cluster training architecture faces critical limitations regarding heterogeneity and stability. (i) *Heterogeneous computational power.* Variations in training scales and NPU types across clusters result in considerable disparities in training performance. For instance, Meta built multiple clusters with NVIDIA H100 and A100 GPUs [42],



(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 [18]. 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.

and DeepSeek manages different clusters with PCIe A100 [6] and H800 GPUs [49]. Table 1 shows the resource heterogeneity among different production clusters of our institute. Synchronous training methods force faster clusters to stay idle and wait for slower ones, causing underutilization of computational resources [70]. Asynchronous training algorithms [50] can mitigate idle time while suffering from unstable convergence. (ii) *Heterogeneous network.* The inter-cluster network bandwidth varies significantly between different clusters. For instance, the bandwidth between close clusters is up to 12× of the bandwidth between distant clusters in AWS [29]. This variability results in a significant drop in the communication efficiency of current decentralized approaches, as the overall throughput is dictated by the weakest connection [54]. (iii) *Instability of large-scale training.* Fault-tolerance mechanisms in existing LLM training frameworks are primarily designed for intra-cluster environments [3,26,36,71]. Large-scale cross-cluster training is inherently less reliable or resilient, with failure rates differing markedly between clusters. As reported in Table 1, we observe up to a 66× difference in the mean time between failures (MTBF) across the training clusters.

Our Contributions. To address these challenges, we introduce `Di-PS`, a novel cross-cluster LLM training framework. The key design of `Di-PS` is a new centralized master-slave PS architecture to coordinate asynchronous training on multiple decentralized clusters. Specifically, `Di-PS` manages the outer optimizer states in the two-stage optimization process through a set of CPU-based PSs, comprising *a master PS* and scalable, distributed *slave PSs*. This improves the network utilization, supports scalability for models of varying sizes, and mitigates communication performance degradation associated with heterogeneous networks. Additionally, we propose a *dual-workflow mechanism* for orchestrating the training operations, including interactions between clusters and master

PS, clusters and slave PSs, and master PS and slave PSs.

We further introduce some novel techniques to tackle the training stability and fault tolerance issues. In particular, we design a *pseudo-gradient penalty strategy* on the PS architecture 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. We further introduce a *two-level resilience mechanism*, including failure isolation, 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 two types of heterogeneous training clusters demonstrate that Di-PS achieves 1.23-2.93× training acceleration compared to synchronous two-stage optimization approaches, while maintaining similar convergence performance. Compared to asynchronous cross-cluster training approaches, Di-PS achieves 1.08-1.16× training acceleration and exhibits better convergence. We further deploy Di-PS in 9 heterogeneous production-level training clusters, integrating over 10,000 NPUs. It efficiently scales the training with stable convergence: the additional overhead for each training cluster incurred by communication with the PS remains minimal, accounting for less than 6% of the total training time.

Our contributions can be summarized as follows:

- We propose Di-PS, the first centralized cross-cluster LLM training system solution tailored for decentralized clusters. By employing a master-slave PS architecture and a dual-workflow mechanism, our system efficiently coordinates large-scale LLM training operations between training clusters, the master PS, and slave PSs.
- The system-algorithm co-design of Di-PS bridges the gap between system performance and algorithmic robustness effectively. The distributed PS design and pseudo-gradient penalty strategy enable stable asynchronous training across heterogeneous clusters while mitigating the communication bottlenecks associated with decentralized approaches.
- We enable resilient cross-cluster LLM training with a two-level resilience mechanism in the PS architecture, ensuring stable and scalable training in the presence of hardware or network failures.

2 Background

2.1 LLM Training

Parallel LLM Training. LLMs are iteratively trained on datasets with trillions of tokens. Each training iteration typically involves a forward pass (FP), a backward pass (BP), and model optimization. Scaling laws [28, 37] demonstrate that increasing both the model and data sizes improves the model performance. The increased model size necessitates parallel training, which splits the model and training computation across (up to hundreds of) devices using different types of parallelism techniques, such as tensor parallelism (TP) [64, 79] and pipeline parallelism (PP) [31, 55, 58]. The increased data size requires data parallel training, which partitions data across the paralleled model training replicas. Cooperating with these parallelisms and optimizations of memory usage [13, 39, 60] and communication overhead [11, 72], existing parallel LLM training systems [12, 44, 48, 56, 77] train giant models using tens of thousands of devices [24, 36]. Such scales exceed the size of most existing clusters. As researchers keep pursuing higher model performance while scaling laws last, training across multiple clusters [16] becomes increasingly important.

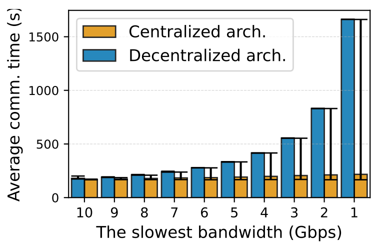
Cross-cluster LLM Training. Parallel LLM training [20] requires intensive communication among all devices. High model FLOPs utilization can be achieved in intra-cluster devices with low-latency connections. Compared to intra-cluster communication, inter-cluster communication is costly. To scale LLM training beyond a single cluster, DiLoCo [10, 18] introduces a two-stage optimization process, as shown in Figure 1a. It leverages two levels of optimizers: a global outer optimizer and inner optimizers within each cluster. 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 an outer optimizer at every H inner steps, to reduce the cross-cluster communication overhead. The outer optimizer holds a global model and updates it with pseudo-gradients, which are calculated with parameters aggregated from training clusters. Current implementations [17, 33, 34] 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.

Resilient LLM Training. Failures frequently occur in large-scale LLM training [24, 30, 36], 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 [8, 19, 23, 35] or intra-cluster training [3, 26, 36, 71]. For cross-cluster training, current studies primarily address elastic job scheduling [15, 65, 75], meeting resource requirements from multiple jobs. Stable and resilient large-scale cross-cluster training remains largely unexplored.

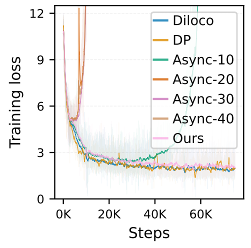
2.2 Challenges of Cross-cluster LLM Training

Cross-cluster LLM training introduces the following challenges that need to be addressed.

Heterogeneous Networks and Devices across Clusters.



(a) The average communication time for a 100B LLM across four clusters.



(b) The training loss with four clusters.

Figure 2: Cross-cluster training challenges for heterogeneity.

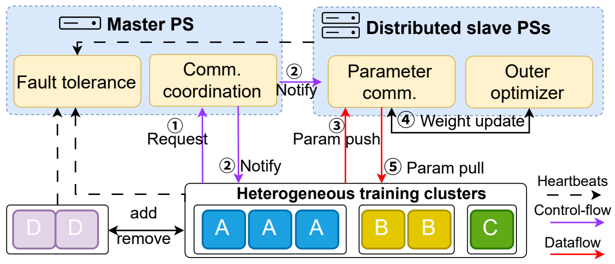


Figure 3: Overview of `Di-PS`.

LLM training over multiple clusters suffers from communication bottlenecks due to low bandwidth. Although the cross-cluster communication frequency can be reduced with the two-stage optimization process [18], 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 larger parameter staleness in outer optimization. To deal with the heterogeneity of networks and computational performance in cross-cluster LLM training atop the asynchronous distributed training and two-stage optimization process, two issues still need to be addressed:

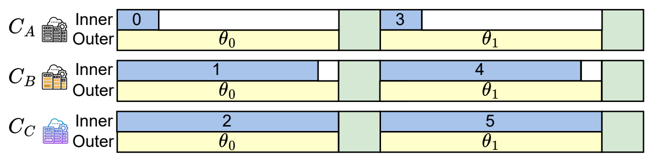
The first issue is *inefficient communications*. Existing decentralized communication approaches struggle to accommodate network heterogeneity. We conduct an experiment of communicating a 100-billion parameter LLM across four clusters. Three clusters are equipped with 10 Gbps networks, while varying 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. Although the communication related to the slower cluster remains a bottleneck in both centralized and decentralized architectures, a centralized architecture can improve overall efficiency by decoupling synchronization processes from the slow inter-cluster links, thereby enhancing communication speed and scalability.

The second issue is *unstable convergence*. We pretrain a LLaMA3.2-1B model [24] across four clusters under four asynchronous training scenarios, where the emulation computational performance gaps between clusters ranged from 10% to 40%. The number of inner steps is 64 ($H = 64$) in two-stage optimization methods. The results are shown in Figure 2b, where the Async-x means the training performance among the clusters varies uniformly by 0–x%. Compared to the synchronous training methods DiLoCo and DP, naive asynchronous trainings fail to converge the model. Furthermore, larger performance gaps exacerbate the convergence issues. To address this, we advocate for optimized asynchronous algorithms coupled with asynchronous communication designs, which together aim to maximize parallel training efficiency while maintaining acceptable convergence behavior.

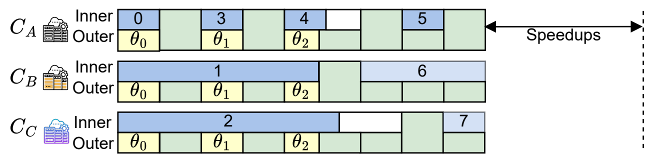
Training Instability. Large-scale cross-cluster training faces heightened unreliability. The failure rates vary across heterogeneous clusters. We observe 31 failures in one cluster during a 33-day productive training, while the other eight clusters encounter fewer than 4 failures. Additionally, existing decentralized frameworks offer limited resilience; frequent training cluster join/leave events impose substantial overhead. Moreover, the outer optimization itself requires fault-tolerance as well. Highly resilient designs built on a centralized framework have shown promise in mitigating these issues by absorbing node failures and reducing retraining costs, thus ensuring more robust multi-cluster training.

3 `Di-PS` Architecture

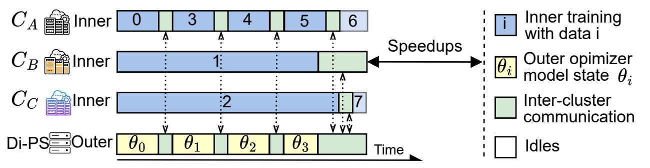
To address the challenges in §2.2, we propose `Di-PS`, an efficient and robust distributed framework for cross-cluster LLM training. Figure 3 shows the overview of `Di-PS`. Its key design is a master-slave PS architecture. The substantial sizes of LLMs make it impossible to hold the PS on a single device. 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 partition the LLM across distributed PSs to reduce the memory overhead and improve the communication performance. Each PS is deployed on a CPU server and manages several LLM model layers. To manage these distributed PSs, we introduce a master-slave PS architecture, where the master PS coordinates communication between the slave PSs and training clusters via a control flow. Meanwhile, the slave PSs handle data flows, enabling efficient communication of LLM parameters with geo-distributed clusters. The distributed slave PS design in the master-slave architecture also offers scalability, allowing us to incorporate additional slave PS to accommodate larger models and increased training cluster sizes.



(a) Synchronous outer optimizer with decentralized architecture.



(b) Asynchronous outer optimizer with decentralized architecture.

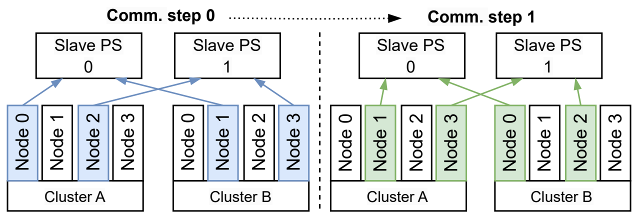

C_A Inner
C_B Inner
C_C Inner
Di-PS Outer
0 3 4 5 6 1 2 7
θ_0 θ_1 θ_2 θ_3
Speedups
Time
i Inner training with data i
θ_i Outer opimizer model state θ_i
Inter-cluster communication
Idles

(c) Asynchronous outer optimizer with centralized architecture.

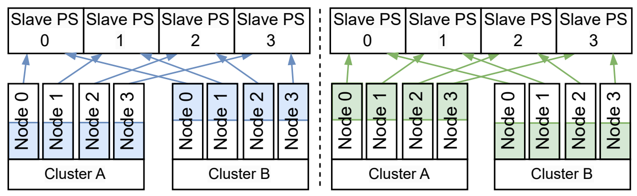
Figure 4: Comparison of different outer optimizer and communication architectures on cross-cluster LLM training with heterogeneous clusters. The cluster B (C_B) has a slower inter-cluster bandwidth.

3.1 Master-slave PS Design

Exploiting Cross-cluster Networks. Compared to the decentralized outer optimization methods, `Di-PS` can alleviate the issue of bandwidth heterogeneity across clusters. In the two-stage optimization process, each inner cluster trains the model for H inner steps and then performs an outer optimization. As shown in Figure 4a, using a synchronous outer optimization method on clusters with distinct performance leads to idle waiting for faster clusters. Asynchronous training can reduce this idle time by allowing clusters to update parameters independently with the outer optimizer. However, asynchronous outer optimization increases communication frequency. Specifically, assuming an LLM training process involves a total of T iterations and N clusters, synchronous training requires $\frac{T}{NH}$ communication rounds, while asynchronous training may require up to $\frac{T}{H}$ rounds. Current cross-cluster systems [17, 33, 34] often employ a fully decentralized architecture using AllReduce for parameter aggregation (Figure 4b). The communication performance of AllReduce is bottlenecked by the slowest network link due to its inherently decentralized nature, resulting in uniformly high communication overhead. The lack of a coordinator results in operation contention and idle inner training. In contrast, as shown in Figure 4c, the centralized slave PSs in `Di-PS` employ point-to-point operations for outer optimization communication, and the master PS can aggregate nearby inter-cluster communications. Although some communications might remain



(a) The number of training workers requesting outer communication in clusters exceeds slave PSs.



(b) The number of slave PSs exceeds or equals training workers requesting outer communication in clusters.

Figure 5: Communications scheduling between training clusters and slave PSs.

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).

Communication Coordination with Master PS. In training clusters, LLMs are typically trained based on hybrid parallelism encompassing tensor parallelism (TP), pipeline parallelism (PP), and data parallelism (DP). The model states are correspondingly partitioned. 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. Next, these workers (with the number of PP size) send model parameters to slave PSs and receive updated parameters in return. They then partition parameters according to TP size, scatter parameters to each TP rank, and broadcast parameters to all DP workers. In this procedure, communication between training workers and slave PSs constitutes a many-to-many operation. Multiple clusters may simultaneously request outer optimization, and within each cluster, multiple training workers in a cluster concurrently attempt to communicate with the PS. To manage this complex process, the master PS schedules communication operations with the granularity of a model layer. As shown in Figure 5, it can stagger the communication order both within each cluster and across the participating training clusters, according to the number of training workers attempting communication and the number of slave PSs. The communication is divided into multiple steps, each step enables different clusters to connect with distinct slave PSs, avoiding contention for the same PS and improving overall communication efficiency.

`Di-PS` can be deployed in CPU servers on an arbitrary cluster. Given that the cost of CPU servers is significantly lower than that of NPU servers, we can deploy the PS on a training cluster composed of CPU servers. To minimize the communication overhead during the outer optimization phase, we adopt a cost model to select the optimal cluster for PS deployment. Let the inter-cluster bandwidth be represented by an adjacency matrix $\mathbf{B} \in \mathbb{R}^{N\times N}$, where $\mathbf{B}_{ij}$ denotes the bandwidth between the CPU servers in cluster i and the NPU servers in cluster j. The training performance of each cluster is captured by a cost vector $\mathbf{p} \in \mathbb{R}^{N}$, 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 cluster i (as a PS candidate) can be modeled by $\mathbf{B}_i^\top \mathbf{p}$, where $\mathbf{B}_i$ is the i-th column of $\mathbf{B}$. Thus, the optimal cluster for PS deployment is given by: $i^* = \arg\max_i(\mathbf{B}_i^\top \mathbf{p})$.

3.2 Dual-workflow Mechanism

The master PS serves as the central controller, with one of its key responsibilities being the orchestration of communication between training clusters and slave PSs. To mitigate communication contention and prevent deadlocks arising from frequent small message exchanges, we isolate control signaling from model parameter communication. This separation is achieved through a dual-workflow design: a *control flow* handles operation orchestration, while a *data flow* manages communication between training clusters and slave PSs. The control flow for asynchronous outer optimization operates as follows:

1. **Parameter Push**: A training cluster initiates a request with its metadata to the master PS to push parameters. The master PS determines the appropriate slave PSs who are not currently receiving parameters and instructs both the requesting cluster and the corresponding slave PS to initiate the transfer on data flow.
2. **Push Completion**: Once the cluster has finished sending all parameters to the distributed slave PSs, it notifies the master PS of the transfer completion using a two-stage commit protocol. The master PS then records the cluster ID and the number of tokens consumed during this round.
3. **Time Window Initiation**: Upon receiving the completion signal, the master PS adds the cluster to the group of clusters ready for parameter updates. A short time window is then initiated to allow other training clusters to join the current update round.
4. **Time Window Completion**: Additional clusters can push their parameters in the same manner until a grace time is reached. All clusters that complete their parameter transfers within this time window will participate in the current round of parameter updates.
5. **Parameter Update**: The master PS instructs the slave PSs with the gradient penalty procedure to update the parameters using the received data from all participating training clusters.

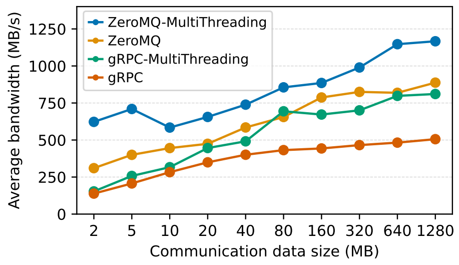


Figure 6: Communication performance comparison between gRPC [1] and ZeroMQ [4].

6. **Parameter Pull**: Once the slave PSs complete the parameter updates, they notify the master PS. The master PS then informs all clusters involved in the current round to pull the latest parameters from the slave PSs on data flow.

We use separate communication protocols to isolate the communications on the control flow and data flow. To choose proper protocols, we evaluate the communication performance of gRPC [1] and ZeroMQ [4] across varying communication sizes on a 25 Gbps network. As shown in Figure 6, ZeroMQ exhibits an average performance advantage of 1.74× over gRPC. Considering the data flow's higher sensitivity to transfer speed and message size is large and stable, ZeroMQ is selected for data streaming. Conversely, gRPC is implemented for the control flow due to its flexibility in managing complex instruction scheduling between slave PSs, master PS, and training clusters, and its ability to prevent potential communication conflicts. Furthermore, each slave PS manages multiple LLM layers. We thereby use multi-threading, with each thread handling one LLM layer, to enhance network utilization. Figure 6 shows that multi-threaded ZeroMQ further improves communication performance by 1.41×.

Operation Overlapping. For better network utilization, we use serialized data in the data flow. In practice, the serialization and deserialization operations for parameters can become the bottleneck in distributed slave PSs (e.g., it takes about 40% of time in optimizing a 100B model). To address this, `Di-PS` uses a Producer-Consumer model to deal with the serialization and deserialization operations. Specifically, the LLM parameters are transferred in a pipeline manner. `Di-PS` leverages memory pools to store parameters, and a dedicated consumer thread handles de-serialization asynchronously, ensuring that the data flow will not be blocked. This strategy is symmetrically applied to the serialization process. Besides, the communication in control flow and data flow can be overlapped with checkpointing, minimizing the extra overhead of PS in training clusters. In addition, each cluster can leverage the existing operation overlapping strategies to accelerate inner model training steps.

4 Stability and Fault Tolerance Enhancement

4.1 Pseudo-gradient Penalty Strategy

Asynchronous training may compromise accuracy, as global parameters in the outer optimizer might encounter instability, such as loss spikes, when the pseudo-gradients from training clusters are low-quality or stale [14, 50]. To guarantee the accuracy of asynchronous local SGD training, inspired by EDiT [14], we design a pseudo-gradient penalty strategy on the PS architecture to address the training instability caused by asynchronism. We detail the gradient penalty procedure in `Di-PS` as follows:

1) **Distributed norm computation:** At the outer optimization step t, each slave PS receives parameters from the training clusters and calculates the pseudo-gradients Δ_t. It then computes the L2 norm of the pseudo-gradient for every group and sends these values to the master PS. The master PS aggregates these norms to derive the total norm for each training cluster. 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 slave PSs and $\Delta_t^{i,j}$ denotes the pseudo-gradient from slave PS i.

2) **Outlier detection:** The master PS then uses an exponential moving average score vector, E_t, to estimate convergence trends of each participating cluster. 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 $\mu_{t+1} = \alpha G_t + (1-\alpha)\mu_t$, $\sigma_{t+1} = \sqrt{(1-\alpha)(\sigma_t)^2 + \alpha(G_t - \mu_{t+1})^2}$. The average factor α is set to 0.02 in our practice. We maintain a matrix $\mathbf{E}$ for recording score vector E_t by stacking its values into $\mathbf{E}$. 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 slave PSs. In our experience, the scaling threshold β can be set to 3. The updated set of participating training clusters is denoted by c. Master PS then sends c back to slave PSs and updates $\mathbf{E} = \mathbf{E} \oplus E_t^c$.

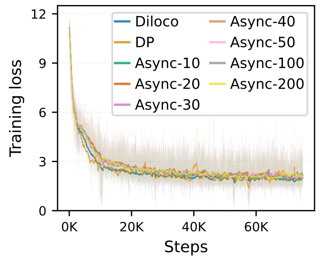
3) **Consumed-token based weighted averaging:** When multiple training clusters participate in this optimization step, their gradients are averaged based on the corresponding consumed data tokens. The resulting pseudo-gradient in the slave PS at step t is $\delta_t^i = \frac{\sum_{j \in c} T_j \Delta_t^{i,j}}{\sum_{j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

Previous works [7, 66] have established convergence guarantees for asynchronous SGD algorithms under the assumptions of smooth functions and bounded noise. Stich et al. [66] proposed an error feedback framework for theoretical analysis, reformulating asynchronous SGD as an error-compensated SGD algorithm. This framework derives an upper bound on the error and proves the sub-linear convergence rate $\mathcal{O}(\tau/T + \sigma/\sqrt{T})$ through a one-step progress analysis of the objective function, where τ denotes the asynchronous delay in the training system and σ represents the noise variance of the stochastic gradient. As demonstrated by the `Di-PS` algorithm, when these assumptions hold, the proposed efficient centralized communication architecture reduces the asynchronous delay τ, while the gradient penalty strategy diminishes the gradient variance σ through outlier detection. Collectively, these mechanisms enhance convergence in asynchronous settings.

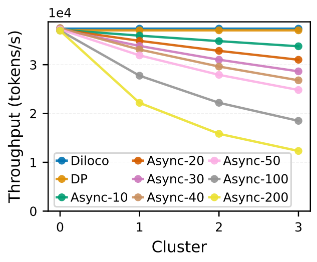
Algorithm 1 Asynchronous outer optimizer

```
Input: Initial pretrained model $\theta^0$
Input: $k$ training clusters
Input: Grace period $\tau_{grace}$
Input: Total consumed tokens $tks_{max}$
 1: $tks_{local} \leftarrow 0$
 2: $\theta \leftarrow$ split_model_by_slave_PSs($\theta^0$)
 3: $G \leftarrow$ [init_cluster() for $i = 1, \ldots, k$]
 4: $G_{completed} \leftarrow \varnothing$
 5: $\tau_{sync} \leftarrow \infty$
 6: $\Delta \leftarrow \varnothing$
 7: while $tks_{local} < tks_{max}$ do
 8:     $g \leftarrow$ get_cluster($G, \tau_{sync}$)
 9:     if $g$ exists then
10:         $\tau_{sync} \leftarrow \tau_{grace}$
11:         $\theta^g \leftarrow$ recv_params($g$)
12:         $g_t \leftarrow \theta - \theta^g$              ▷ Get the pseudo gradient.
13:         $\Delta \leftarrow \Delta \cup \{g_t\}$
14:         $G_{completed} \leftarrow G_{completed} \cup \{g\}$
15:         $tks_{local} \leftarrow tks_{local} + g$.local_consumed_tokens
16:     else
17:         $g_n \leftarrow$ verify_pseudo_gradients($\Delta$)
18:         $\theta \leftarrow$ Nesterov($\theta, g_n$)
19:         send_params($\theta, G_{completed}$)
20:         $\tau_{sync} \leftarrow \infty$
21:         $G_{completed} \leftarrow \varnothing$
22:         $\Delta \leftarrow \varnothing$
23:     end if
24: end while
```

To this end, we present our asynchronous outer optimization design in Algorithm 1. We use the Nesterov optimizer [67] as the outer optimizer in slave PSs, and AdamW [53] is employed 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}. When a cluster finishes its local training and requests the outer optimization, the master PS will maintain a time window with length τ_{grace} for other clusters that also wish to synchronize their local parameters (Lines 8–10). This allows clusters with varying training performance to communicate with the outer optimizer whenever it is ready. Upon



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



(b) The detailed training performance distribution of different settings.

Figure 7: `Di-PS` can converge the asynchronous cross-cluster training as synchronous methods (DP and DiLoCo) across various heterogeneous training cluster emulations. The Async-*x* indicates that the computational performance among the clusters varies uniformly by 0–*x*%.

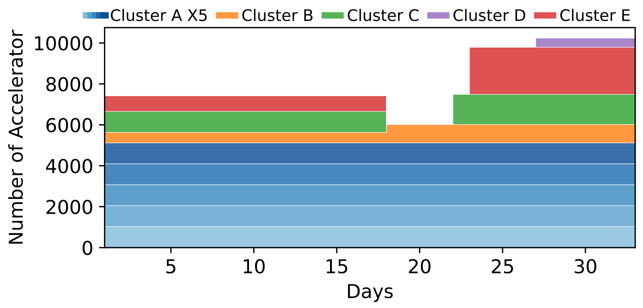


Figure 8: Available resource timeline of 9 training clusters during a 33-day training of a 100B LLM, with a notable achievement of scaling up to 10,122 NPUs.

receiving local parameters from the clusters and calculating the pseudo-gradient, the outer optimizer verifies the received pseudo-gradient with the aforementioned penalty strategy and updates its global parameters within this time window (Lines 11–18). Subsequently, the outer optimizer records the training progress and sends the updated parameters to the clusters that participate in this optimization process (Lines 19–22).

To demonstrate that `Di-PS` is consistently robust with synchronized training frameworks, 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. 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, performing comparably to synchronous methods across all heterogeneous scenarios, as the pseudo-gradient penalty strategy can avoid training failures in the same tasks (Figure 2b).

4.2 Two-level Resilience Mechanism

Cross-cluster LLM training at scale is a computationally intensive process and involves many NPUs. We conduct a productive 100B LLM training that spans 33 days across 9 training clusters and utilizes up to 10,122 NPUs. We observe two phenomena: (i) training clusters encounter failures frequently, as shown in Table 1, with each cluster meeting more or less failures; however, the job schedules in each cluster can detect and recover from these failures. (ii) The availability of training resources across clusters is dynamic, as illustrated in Figure 8, with multiple instances of training scaling events (both expansions and contractions) occurring due to resource fluctuations. Each failure within training clusters and each training scale change leads to a cluster removal or join operation in `Di-PS`.

Cluster-level Resilience. The centralized PS architecture of `Di-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 [12, 34], 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, our centralized PS-based architecture localizes failure detection and recovery. The master PS acts as the single point of coordination, maintaining training metadata and communication state across all participating clusters. When a failure occurs in one cluster, only the master 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 simplifies error handling and enables faster recovery, and healthy clusters can continue training.

To realize the dynamic addition and removal of training clusters without causing failures in `Di-PS`, live training clusters are required to transmit heartbeat signals to the master PS periodically. These heartbeats enable the integration of new clusters and the removal of unresponsive clusters during model training. Upon receiving a heartbeat from a newly joined cluster, the master PS instructs slave PSs to transmit the latest model parameters for initialization. Conversely, if the master PS fails to receive heartbeats from a cluster for a predefined number of consecutive intervals, it automatically removes that cluster from the training process and excludes it from the outer optimization process.

The number of training clusters thereby is dynamic during cross-cluster training. Coupled with the diverse data resources inherent in LLM training datasets, this necessitates a strategy for consistent training. To this end, 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.

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 slave PSs. For communication failures, we ensure model consistency between slave PSs and training clusters through version control. Each cluster independently saves complete model states and training progress, reflecting its parallelization strategy and local update status. After each outer optimization update, each slave PS serializes its model state and maintains multiple up-to-date snapshots for model evaluation to facilitate the joining of new training clusters.

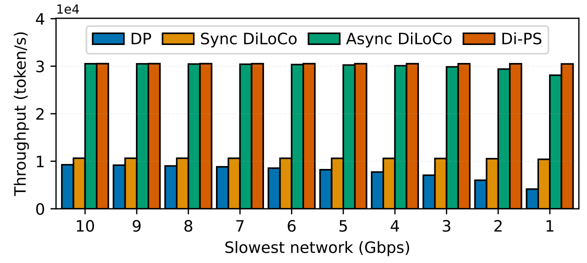
To address failures in the distributed architecture of `Di-PS`, the master PS and slave PSs operate under a master-slave architecture, and slave PSs periodically send heartbeats to the master PS. The master PS monitors these heartbeats and restarts failed slave PSs using the latest snapshot. If the master PS fails, preventing heartbeats from reaching training clusters and slave PSs, these components will persistently attempt to reconnect in a non-blocking manner until a connection is re-established. As the master PS stores no training data, cluster job schedulers, like Kubernetes, can readily detect and restart it. Specifically, when a failure occurs in `Di-PS` during parameter update or data transfer, all participating clusters skip the current outer optimization step and proceed to the next inner training steps.

5 Evaluation

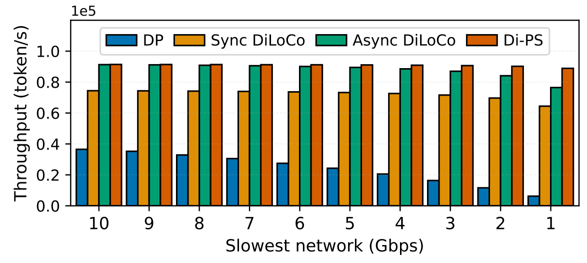
We evaluate the effectiveness of `Di-PS` in both controlled experimental settings and large-scale production environments. The experimental evaluation utilizes small clusters with controllable network bandwidth and training performance to enable fair comparisons with baseline methods (§5.1–5.3). We also report the performance `Di-PS` in the productive 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. We evaluate the end-to-end cross-cluster training performance of `Di-PS` on two types of experimental heterogeneous cluster configurations: (i) Four real clusters, each with the following NPU configurations—1 A100, 4 RTX 3090, 8 RTX 2080 Ti, and 8 RTX 2080 GPUs. The different number of NPUs across clusters ensures a similar total memory capacity, enabling deployment of the same LLM. These real clusters exhibit heterogeneous training performance, with disparities of up to 5.23×. (ii) Four emulated clusters, each equipped with an 80GB H800 GPU. To simulate heterogeneity, we introduce extra training delays (0, 16, 33, and 50% of an iteration) to each training iteration, creating uniformly varying training speeds across clusters.



(a) Four real clusters of 1 A100 GPU, 4 RTX 3090 GPUs, 8 RTX 2080ti GPUs and 8 RTX 2080 GPUs.



(b) Four emulation clusters of training performance varies uniformly by 0–50%.

Figure 9: End-to-end training performance of different outer optimizers implementations on cross-cluster LLM training with two types of heterogeneous clusters.

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 [18], 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 on decentralized communication architecture [33, 34]. Similar to Synchronous DiLoCo, but each cluster performs an asynchronous outer optimization whenever complete inner training steps.

We train a LLaMA3.2-1B model [24] in both configurations and compare the aggregated training throughput of all clusters, as the intra-cluster training performance is data parallelism and is consistent across all baselines. The pretraining 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 both configurations, three clusters have an intra-cluster bandwidth of 10 Gbps, while the bandwidth of the remaining cluster is varied from 1 Gbps to 10 Gbps. Figure 9 compares the end-to-end training performance across different inter-cluster bandwidths of the remaining cluster. Compared to global DP, the synchronous DiLoCo accelerates training 1.15–2.51× in real clusters and 2.03–10.35× in emulated clusters, benefiting from reduced inter-cluster communication frequency via two-stage optimization. `Di-PS` further achieves

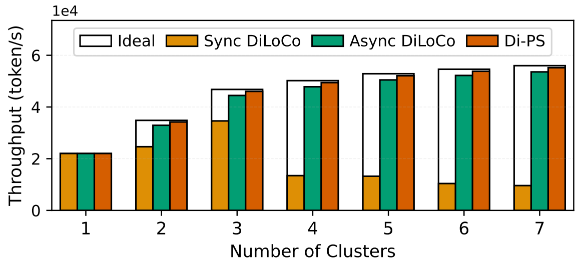


Figure 10: Weak scaling performance comparison and related ideal linear results.

1.23–1.38× speedups over synchronous DiLoCo in emulation clusters, as Di-PS adapts to heterogeneous computing resources. In the real cluster configuration with more severe performance disparity, Di-PS achieves 2.87–2.93× speedups over synchronous DiLoCo. Asynchronous DiLoCo methods can leverage the training resources of clusters to achieve considerable training performance. However, it encounters the training failure issues as shown in Figure 2b. Di-PS provides stable convergence (Figure 7a) and adapts to heterogeneous networks, accelerating end-to-end training by 1.00–1.16×. 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 [29].

5.2 Scalability

We can evaluate the scalability of Di-PS through weak scaling experiments, where the number of clusters is gradually increased, and each cluster trains a LLaMA3.2-1B model. Clusters are added sequentially in descending order of training performance. The cluster configurations are as follows: two A100 GPUs, four RTX 3090 GPUs, one A100 GPU, eight RTX 2080 Ti GPUs, eight RTX 2080 GPUs, four RTX 2080 Ti GPUs, and four RTX 2080 GPUs. The first cluster is connected with a 5 Gbps inter-cluster network, while all remaining clusters have 10 Gbps bandwidth.

Figure 10 shows the comparison of aggregated training performance across all training clusters. Di-PS achieves 1.00–1.03× speedups over the asynchronous DiLoCo approach, as the network bandwidth heterogeneity is relatively low in this experiment. In comparison to the synchronous DiLoCo approach, Di-PS achieves speedups ranging from 1.00 to 5.74×. With a single training cluster, outer optimization becomes unnecessary, leading to equivalent performance across all methods. The addition of a slower training cluster causes synchronous DiLoCo to suffer from straggler effects, which results in the fastest performance occurring with only three 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 98.2–98.6% of the ideal performance, demonstrating excellent scalability.

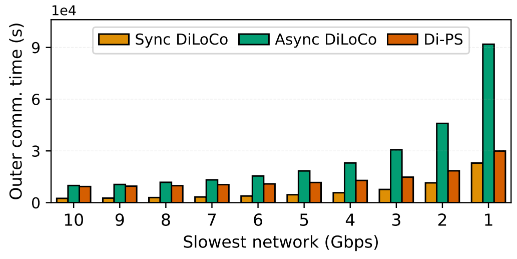
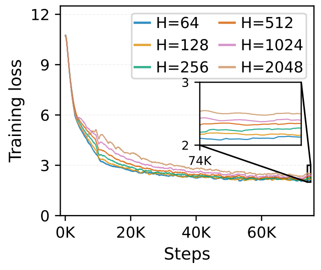


Figure 11: Accumulated outer communication overhead on heterogeneous inter-cluster networks.



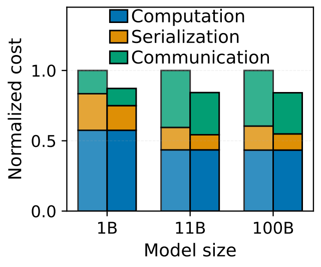


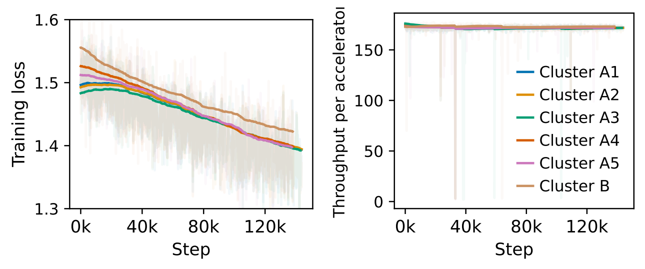
Figure 12: Training loss of Di-PS in different inner steps (H) on four emulation clusters of training performance varies uniformly by 0–50%.

Figure 13: Effectiveness of optimizations in Di-PS. In each model size, the right bar represents the performance after applying optimizations.

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 over asynchronous DiLoCo. Figure 11 shows the aggregated outer communication time from end-to-end experiments (§5.1) with training 2.4B 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 synchronous and asynchronous DiLoCo as network heterogeneity grows. In contrast, Di-PS effectively mitigates this overhead by adapting to heterogeneous networks, achieving a 1.06–3.07× reduction in total inter-cluster communication time compared to asynchronous DiLoCo.

Outer Optimization Intervals. To better understand the convergence behavior of asynchronous two-stage optimization methods, we conduct pretraining experiments on the LLaMA3.2-1B model using four emulated training clusters with uniformly distributed performance disparities ranging from 0 to 50%. As shown in Figure 12, varying the number of inner training steps may impact convergence speed. The inner



(a) Training loss. (b) Training performance.

Figure 14: Training loss and training performance in training clusters that participate all the time.

Table 2: 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

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.

Slave PS Optimizations. We also conduct an ablation study to assess the effectiveness of the optimization techniques applied to the PS in Di-PS. The key components of the slave PS's outer optimization loop include communication, communication data serialization/deserialization, and optimizer computation. We compared the overhead of the outer optimization process with and without our optimizations, training LLaMA-based LLMs with 1B, 11B, and 100B parameters, using 1, 8, and 16 slave PSs, respectively, over a 10 Gbps inter-cluster network.

Figure 13 presents the normalized operation costs for each component of the outer optimization loop. As expected, the optimizer computation overhead remains constant. The communication scheduling from master PS and multi-threading communication in slave PSs provide communication speedups of up to 1.41×. Additionally, overlapping operations within slave PSs improve data serialization performance by up to 1.49×. Overall, these optimizations in Di-PS result in a 1.16× acceleration of the outer optimization process.

5.4 Di-PS in Productive Training.

We deployed Di-PS for a productive LLM training and trained a 100B LLaMA-based LLM with 96 model layers, hidden size of 8192, and intermediate size of 36864, consuming a total of 2.3T tokens. As previously mentioned, the training involves

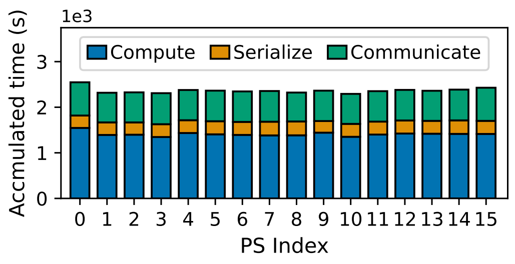
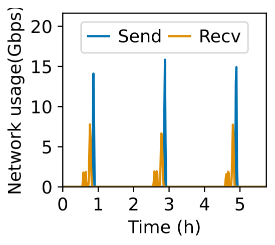


Figure 15: Accumulated operation time breakdowns in slave PSs through a day.



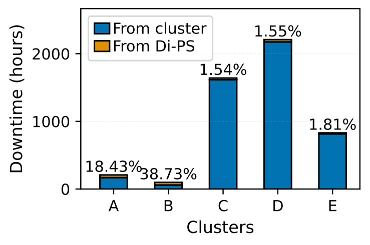


Figure 16: A six-hour network utilization trace of a slave PS.

Figure 17: The total downtime of NPUs resulted from training cluster and Di-PS.

up to nine clusters, each with varying numbers of NPUs (Table 1). The training process spanned 33 days and dynamically scaled, reaching a peak of 10,112 NPUs (Figure 8). To support the model's large size, we utilized 16 slave PSs, each deployed on a dedicated CPU server. We evaluated the performance of Di-PS across convergence behavior, training efficiency, and fault tolerance.

Convergence. To consistently present convergence performance in training with Di-PS, Figure 14a shows the training loss curves for clusters that are active throughout the entire training process (Cluster A×5 and Cluster B). Over the 140K training steps, Di-PS demonstrates stable convergence across all clusters in this large-scale production training scenario.

Training Efficiency. We evaluate training efficiency from both the perspective of the training clusters and the Di-PS framework. On the training cluster side, we first show the throughput per NPU in Figure 14b. Most NPUs achieve stable and consistent training efficiency, with only a few experiencing occasional failures, from which they recover swiftly.

The average time breakdown for each cluster is presented in Table 2. 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 6-8 experienced significantly lower communication time compared to Clusters 0-5, due to larger pipeline parallelism configurations. This allows more devices to interact with Di-PS simultaneously, improving communication efficiency.

Diving into the performance of the Di-PS, we first present the accumulated time breakdown over a 24-hour period for slave 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. Importantly, slave PSs remain idle for over 95% of the time, indicating that the current `Di-PS` configuration can support more training clusters and significantly larger training scales.

Figure 16 shows a network trace of a slave PS over a six-hour period. Slave 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, slave 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.

Fault tolerance. The centralized PS architecture 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. Figure 17 presents the total NPU downtime for each training cluster, detailing the sources of downtime and the proportion attributable to `Di-PS`. The NPU downtime after a failure caused by `Di-PS` lies in the parameter pull during the cluster restart training. Clusters A and B achieve low downtime, with fewer failures and quick cluster training recovery. Other clusters suffer from hardware issues, wasting considerable resources in intra-cluster error handling; however, `Di-PS` effectively accommodates these failures and restores the latest outer training state with minimal overhead.

6 Related Works

Intra-cluster Parallel LLM Training. Parallel LLM training employs various parallelism strategies to distribute computational workload and reduce memory footprint. Data parallelism [43] distributes the dataset across workers for concurrent processing. Sharded data parallelism [13, 60, 76] further shards model states across workers, reducing memory overhead and synchronizing states through collective communication. Pipeline parallelism [21, 31, 46, 51, 52] partitions the model into sequential sub-modules, distributing them across a worker group. Batch data is divided into microbatches and executed in a pipeline fashion to improve device utilization and minimize communication overhead. Tensor parallelism [38, 39, 41, 64] partitions operators across multiple workers to parallelize matrix multiplications, employing communication protocols to ensure result accuracy. To train LLMs on extremely long sequences, sequence parallelism [22, 25, 32, 45] shards the data at the sequence dimension across multiple devices in the attention operation to reduce memory bottlenecks. Combining these parallel training methods, parallel training systems [12, 36, 56] provide efficient intra-cluster training.

Cross-cluster Training. Building on the two-stage optimization algorithm DiLoCo [18], cross-cluster training systems extend intra-cluster training with a decentralized outer optimizer, thereby reducing the frequency of inter-cluster synchronization. OpenDiLoCo [34] further reduces inter-cluster communication size through FP16 AllReduce. Prime [33] introduces a hybrid DiLoCo-FSDP approach to lower memory overhead, while Streaming DiLoCo [17] 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 and applies overlapping only on the server side. Another line of research focuses on compressing inter-cluster communication to approximate intra-cluster conditions. StellaTrain [47] targets training on laboratory-scale clusters, whereas FusionLLM [68] addresses training across geo-distributed, sparsely available GPUs.

Federated Learning for LLM. FL for LLM concentrates on model fine-tuning, proposing strategies such as compression [59, 73], secure aggregation [40], and adaptive sampling [74]. While FL is typically designed for edge devices, the LLM size is often limited. Photon [62] extends FL to LLM pretraining, improving robustness to data heterogeneity by utilizing small client batch sizes coupled with exceptionally high learning rates. But its nondistributed PS faces potential scalability bottlenecks.

Intra-cluster Resilience. Recent advances in intra-cluster fault tolerance introduce self-healing mechanisms to ensure high availability within a single cluster. Oobleck [35] and ReCycle [23] leverage inherent computation redundancy in parallel LLM training to enable uninterrupted training during failures. Unicron [26] integrates in-band error detection and dynamic reconfiguration to minimize downtime across concurrent training jobs. Similarly, production systems such as MegaScale [36] adopt checkpoint-free recovery and proactive failure isolation. `Di-PS` addresses fault tolerance at the inter-cluster level, making it orthogonal to these approaches.

7 Conclusion

In this work, we introduce `Di-PS`, a novel training system designed to address the heterogeneity, stability, and reliability challenges faced when training 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, enabling the successful training of a 100B-parameter model. With these advancements, `Di-PS` demonstrates a promising approach to large-scale LLM training, offering both enhanced scalability and efficiency while maintaining reliability across heterogeneous clusters.

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] Wei An, Xiao Bi, Guanting Chen, Shanhuang Chen, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Wenjun Gao, Kang Guan, Jianzhong Guo, Yongqiang Guo, Zhe Fu, Ying He, Panpan Huang, Jiashi Li, Wenfeng Liang, Xiaodong Liu, Xin Liu, Yiyuan Liu, Yuxuan Liu, Shanghao Lu, Xuan Lu, Xiaotao Nie, Tian Pei, Junjie Qiu, Hui Qu, Zehui Ren, Zhangli Sha, Xuecheng Su, Xiaowen Sun, Yixuan Tan, Minghui Tang, Shiyu Wang, Yaohui Wang, Yongji Wang, Ziwei Xie, Yiliang Xiong, Yanhong Xu, Shengfeng Ye, Shuiping Yu, Yukun Zha, Liyue Zhang, Haowei Zhang, Mingchuan Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, and Yuheng Zou. Fire-flyer ai-hpc: A cost-effective software-hardware co-design for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '24. IEEE Press, 2024.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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. *arXiv preprint arXiv:2412.07210*, 2024.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021.

[22] Jiarui Fang and Shangchun Zhao. Usp: A unified sequence parallelism approach for long context generative ai. *arXiv preprint arXiv:2405.07719*, 2024.

[23] 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.

[24] 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.

[25] Diandian Gu, Peng Sun, Qinghao Hu, Ting Huang, Xun Chen, Yingtong Xiong, Guoteng Wang, Qiaoling Chen, Shangchun Zhao, Jiarui Fang, et al. Loongtrain: Efficient training of long-sequence llms with head-context parallelism. *arXiv preprint arXiv:2406.18485*, 2024.

[26] 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.

[27] 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.

[28] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[29] 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.

[30] 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.

[31] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Can Karakus, Rahul Huilgol, Fei Wu, Anirudh Subramanian, Cade Daniel, Derya Cavdar, Teng Xu, Haohan Chen, Arash Rahnama, and Luis Quintela. Amazon sagemaker model parallelism: A general and flexible framework for large model training. *arXiv preprint arXiv:2111.05972*, 2021.

[39] 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.

[40] Weirui Kuang, Bingchen Qian, Zitao Li, Daoyuan Chen, Dawei Gao, Xuchen Pan, Yuexiang Xie, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope-llm: A comprehensive package for fine-tuning large language models in federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5260–5271, 2024.

[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] Kevin Lee, Adi Gangidi, and Mathew Oldham. Building Meta's GenAI Infrastructure, 2024. https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/.

[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] Shigang Li and Torsten Hoefler. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.

[47] Hwijoon Lim, Juncheol Ye, Sangeetha Abdu Jyothi, and Dongsu Han. Accelerating model training in multi-cluster environments with consumer-grade gpus. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 707–720, 2024.

[48] 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.

[49] 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.

[50] 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.

[51] Weijie Liu, Zhiquan Lai, Shengwei Li, Yabo Duan, Keshi Ge, and Dongsheng Li. Autopipe: A fast pipeline parallelism approach with balanced partitioning and micro-batch slicing. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 301–312. IEEE, 2022.

[52] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2023.

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

[54] 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.

[55] 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.

[56] 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.

[57] Jose Javier Gonzalez Ortiz, Jonathan Frankle, Mike Rabbat, Ari Morcos, and Nicolas Ballas. Trade-offs of local sgd at scale: An empirical study. *arXiv preprint arXiv:2110.08133*, 2021.

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

[59] Zhen Qin, Daoyuan Chen, Bingchen Qian, Bolin Ding, Yaliang Li, and Shuiguang Deng. Federated full-parameter tuning of billion-sized language models with communication cost under 18 kilobytes. *arXiv preprint arXiv:2312.06353*, 2023.

[60] 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.

[61] Sashank J Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[62] Lorenzo Sani, Alex Iacob, Zeyu Cao, Royson Lee, Bill Marino, Yan Gao, Dongqi Cai, Zexi Li, Wanru Zhao, Xinchi Qiu, et al. Photon: Federated llm pre-training. *arXiv preprint arXiv:2411.02908*, 2024.

[63] Lorenzo Sani, Alex Iacob, Zeyu Cao, Bill Marino, Yan Gao, Tomas Paulik, Wanru Zhao, William F Shen, Preslav Aleksandrov, Xinchi Qiu, et al. The future of large language model pre-training is federated. *arXiv preprint arXiv:2405.10853*, 2024.

[64] 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.

[65] 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.

[66] Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication. *arXiv preprint arXiv:1909.05350*, 2019.

[67] 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.

[68] Zhenheng Tang, Xueze Kang, Yiming Yin, Xinglin Pan, Yuxin Wang, Xin He, Qiang Wang, Rongfei Zeng, Kaiyong Zhao, Shaohuai Shi, et al. Fusionllm: A decentralized llm training system on geo-distributed gpus with adaptive compression. *arXiv preprint arXiv:2410.12707*, 2024.

[69] 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.

[70] 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.

[71] 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.

[72] Haiquan Wang, Chaoyi Ruan, Jia He, Jiaqi Ruan, Chengjie Tang, Xiaosong Ma, and Cheng Li. Hiding

communication cost in distributed llm training via micro-batch co-execution. *arXiv preprint arXiv:2411.15871*, 2024.

[73] Huiwen Wu, Xiaohan Li, Deyi Zhang, Xiaogang Xu, Jiafei Wu, Puning Zhao, and Zhe Liu. Cg-fedllm: How to compress gradients in federated fune-tuning for large language models. *arXiv preprint arXiv:2405.13746*, 2024.

[74] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. Fwdllm: Efficient fedllm using forward gradient. *arXiv preprint arXiv:2308.13894*, 2023.

[75] 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.

[76] 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.

[77] 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.

[78] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.

[79] 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.