DREAMDDP: ACCELERATING LOW-BANDWIDTH GEO-DISTRIBUTED LLM TRAINING WITH LAYER-WISE PARTIAL SYNCHRONIZATION

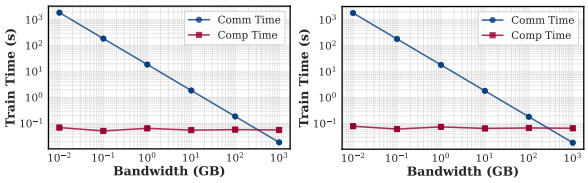
Zhenheng Tang [*1] **Zichen Tang** [*1] **Junlin Huang** [2] **Xinglin Pan** [2] **Rudan Yan** [2] **Yuxin Wang** [2]
Amelie Chi Zhou [3] **Shaohuai Shi** [4] **Xiaowen Chu** [2] **Bo Li** [1]

ABSTRACT

Scaling up training large language models (LLMs) in computing and data perspectives motivates distributed training across different geo-distributed data centers. Communication in geo-distributed data parallel training (DDP) with stochastic gradient descent (S-SGD) is the main bottleneck in low-bandwidth environments. Recent studies have successfully applied Local SGD to mitigate the communication overhead and geo-distributedly pre-train LLMs. However, we identify that the strict model synchronization mechanism in Local SGD prevents overlapping communication and computation, which makes the system lose opportunities to overlap communication and computation. To overcome this limitation, we expand the design space of local SGD by layer-wisely decoupling model synchronization. In each iteration, only partial layers are synchronized instead of the entire model after a specific number of iterations. Leveraging this methodology, we introduce DreamDDP, a training framework to accelerate low-bandwidth distributed training with three key innovations: (1) partial local SGD with theoretical assurances of convergence rates comparable to S-SGD; (2) overlapping parameter synchronization with computation without extra GPU memory occupation; (3) identifying and exploiting three properties to schedule communication and computation based on fine-grained layer-wise profiling to reduce training time. Empirical evaluations conducted on 32 GPUs using prominent deep learning models, including ResNet-18, ResNet-50, GPT-2, and Llama-2, demonstrate that DreamDDP enhances the convergence properties of Local SGD (and Adam) and achieves speedups ranging from $1.49\times$ to $3.91\times$ over leading baseline methods.

1 INTRODUCTION

Modern deep learning (DL) models, such as large language models (LLMs) like GPT-3 (Radford et al., 2018), Gemini (Reid, 2024), LLaMA (Touvron et al., 2023a), are extremely large. Training them is computationally expensive, prompting proposals to scale training across geographically distributed data centers (Ryabinin et al., 2023a; Tang et al., 2024b; Ryabinin & Gusev, 2020; Jaghouar et al., 2024b; TANG et al., 2026). Moreover, data exhaustion is a critical training issue (Parekh), as high-quality public data may be exhausted by 2026 (Villalobos et al., 2022). Using private data (e.g., medical (Thirunavukarasu et al., 2023) or financial (Wu et al., 2023)) raises privacy concerns. Federated learning (FL) has emerged as a promising solution (Ye et al., 2024a), as it allows multiple parties to collaboratively train a model without sharing private data, thereby protecting data privacy and security.



(a) Llama-2 (b) GPT-2

Figure 1. Comparison of communication and computation times under varying bandwidths for GPT-2 and Llama-2.

However, as shown in Figure 1, the low communication bandwidth of 10 Mbps $\sim$ 1 Gbps between different data centers and multiple parties makes communication the main bottleneck of the system (Ryabinin et al., 2023a; Diskin et al., 2021; Ryabinin & Gusev, 2020; Wang et al., 2023b; Yuan et al., 2022; Tang et al., 2024b). For the synchronous stochastic gradient descent (S-SGD) (Bottou et al., 2016) widely used in distributed training, in each training iter-

[*]Equal contribution [1]The Hong Kong University of Science and Technology [2]The Hong Kong University of Science and Technology (Guangzhou) [3]Hong Kong Baptist University [4]Harbin Institute of Technology. Correspondence to: Shaohuai Shi <shaohuais@hit.edu.cn>, Xiaowen Chu <xwchu@hkust-gz.edu.cn>, Bo Li <bli@ust.hk>.

Proceedings of the 9th MLSys Conference, Bellevue, WA, USA, 2026. Copyright 2026 by the author(s).

ation, gradients are required to be synchronized between different workers, of which the communication has significant time costs. To this end, existing works that accelerate Geo-distributed training mainly focus on scheduling different data centers to conduct Data Parallel (DP) or Pipeline Parallel (PP) (Ryabinin et al., 2023a; Yuan et al., 2022), or compressing gradients to reduce communication overheads (Wang et al., 2023b; 2022b; Tang et al., 2024b; 2023). However, the large compression ratio significantly influences the convergence rate, and the effect of communication reduction in the scheduling is limited.

To further mitigate communication overhead, *Local SGD (LSGD)* (Stich, 2019; Woodworth et al., 2020; Guan et al., 2023) performs H local updates before synchronizing model parameters (or pseudo-gradients), reducing communication by a factor of H compared to S-SGD. Its practicality for training LLMs has been demonstrated in decentralized LLM training (e.g., the 10B-parameter INTELLECT-1) (Jaghouar et al., 2024a), and LSGD has emerged as a paradigm for geo-distributed training (Jaghouar et al., 2024b;a; Ye et al., 2024a; Douillard et al., 2024; Xu et al., 2024; Qin et al., 2024; Zhuang et al., 2023; Kuang et al., 2024; Su et al., 2024; Sani et al., 2024; Sun et al., 2024; Cheng et al., 2024; TANG et al., 2026). However, existing LSGD frameworks rely on **full synchronization**—exchanging all model parameters after every H iterations—which precludes overlapping communication with backpropagation (BP), a common technique to improve throughput (Zhang et al., 2017; Peng et al., 2019; Shi et al., 2021). For instance, WFBP (Zhang et al., 2017) overlaps layer-wise gradient communication with subsequent BP, and ASC-WFBP (Shi et al., 2021) further enhances overlap via simultaneous communications and scheduling. Therefore, accelerating LSGD in geo-distributed settings requires: (i) relaxing full synchronization to enable communication–computation overlap like (Zhang et al., 2017; Shi et al., 2021), (ii) designing an effective overlap schedule that minimizes wall-clock time while preserving convergence, and (iii) avoiding additional GPU memory overhead given the scale of modern LLMs (Ren et al., 2021; Rajbhandari et al., 2019; Liu et al., 2025).

In this work, we design a novel training framework named DreamDDP to significantly accelerate DDP and improve LSGD. Firstly, we proposed **partial synchronization**, which synchronizes different parts of the model in each iteration, instead of the entire model after H iterations like the loose synchronization in large-scale network systems (Albrecht & Tuttle, 2006). Thus, it offers opportunities to overlap the communication and computation of LSGD. We theoretically prove that the convergence property of partial synchronization-based local SGD (PLSGD) can be guaranteed to be the same as LSGD. Secondly, DreamDDP divides neural networks into different fine-grained L layers and implements *in-place partial parameter synchronization* by launching layer synchronization after its BP, thus introducing no extra GPU memory occupation. Thirdly, we build an overall time cost model of the FP, BP, and communication process with respect to L layers and H iterations of the partial synchronization and formulate it as an optimization problem. We identify three properties including *optimal hiding*, *delayed CO assignment*, *at-least-one assignment* that can be leveraged to design a depth-first search (DFS) with significantly pruned search space to solve this optimization problem. Lastly, DreamDDP inserts more communication messages (model parameters) to fill the bubble time as long as the extra communication can be hidden by the computation, thus accelerating the training convergence.

Our contributions are summarized as follows.

- We propose partial synchronization based local SGD (PLSGD), which relaxes strong synchronization by layer-wisely decoupling parameters, enabling communication and computation overlap to improve LSGD's training efficiency.
- We mathematically formulate the time complexity of PLSGD and its minimization as an objective function. DreamDDP leverages three identified properties in a DFS algorithm with a pruned search space to optimize PLSGD's throughput.
- We theoretically prove DreamDDP has the same convergence rate as S-SGD and empirically show similar convergence speeds when training GPT-2, Llama-2, ResNet-18, and ResNet-50.
- We implement DreamDDP on PyTorch Distributed (Li et al., 2020; Ansel, 2024) and conduct experiments on two 32-GPU clusters. Results show DreamDDP achieves a $1.49 - 3.91\times$ iteration time improvement over SOTA algorithms like LSGD and ASC-WFBP.

2 PRELIMINARIES

Distributed Synchronous SGD. Give a model parameterized with $w \in \mathbb{R}^d$, where d is the model size, training is to minimize the objective function $F(w) \triangleq \mathbb{E}_{x\sim\mathcal{D}} f(w;x)$, with sampling data x from the dataset $\mathcal{D}$. We consider K different workers that conduct distributed training with S-SGD (Bottou et al., 2016; Shi et al., 2021). In each iteration, workers select different data samples from the original dataset $\mathcal{D}$. Then, workers conduct feedforward (FP) and back-propagation (BP) to calculate gradients $\nabla f(w_r^k;\xi_r^k)$. Next, workers communicate and average these gradients to obtain $\frac{1}{K}\sum_{k=1}^{K}\nabla f(w_r^k;\xi_r^k)$. The model is updated as $w_{r+1} = w_r - \eta_r \frac{1}{K}\sum_{k=1}^{K}\nabla f(w_r^k;\xi_r^k)$.

Communicating and averaging gradients $\nabla f(w_r^k;\xi_r^k)$ requires extra time, which limits the scalability of distributed training, especially for low-bandwidth environments (Ye

et al., 2024c;b; Yuan et al., 2022; Ye et al., 2024b). Note that in this work we do not limit distributed training with collective communication (Fei et al., 2021b; Li et al., 2020; Thakur et al., 2005) or parameter-server (PS) based architectures (Li et al., 2014; Jiang et al., 2020), which do not influence our scheduling.

Local SGD. In LSGD, each k-th worker calculates the local gradient $\nabla f(w_r^k; \xi_r^k)$ and directly uses it to update the local model. After H training iterations, workers communicate and average their model parameters as $\frac{1}{K}\sum_{k=1}^{K} w_r^k$. This process can be formulated as follows.

$$w_{r+1}^k = \begin{cases} w_r^k - \eta_r \nabla f(w_r^k; \xi_r^k), & \text{if } r+1\%H \neq 0 \\ \frac{1}{K}\sum_{k=1}^{K}(w_r^k - \eta_r \nabla f(w_r^k; \xi_r^k)), & \text{if } r+1\%H = 0 \end{cases}. \tag{1}$$

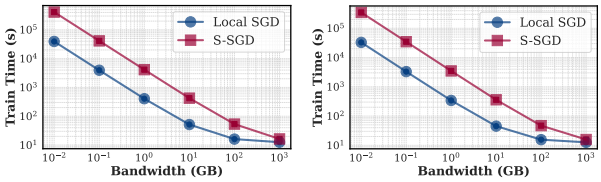
Different from S-SGD, workers with LSGD have different model parameters during local training but synchronize them after H iterations.

Time Complexity. Considering the forward and backward time is t_{FP} and t_{BP}, and the communication time of the whole model is t_{COMM}, the S-SGD and LSGD require whole training time:

$$T_{SSGD} = R \times (t_{FP} + t_{BP} + t_{COMM}), \tag{2}$$

$$T_{LSGD} = R \times (t_{FP} + t_{BP}) + R/H \times t_{COMM}. \tag{3}$$

Thus, the larger H means more time reduction. Compared with normal S-SGD, LSGD reduces the communication cost by up to H times. Figure 2 shows the reduced communication time of LSGD than S-SGD significantly when the communication bandwidth is low. However, large H may cause slow convergence and worse final model performance.



(a) Llama-2 (b) GPT-2

Figure 2. Dissecting training time of GPT-2 and LLaMA-2 with Local SGD and S-SGD under different bandwidth.

2.1 Challenges of Improving LSGD

Challenges 1: The hard synchronization. As shown in Fig. 3(c), LSGD suffers from a hard synchronization because it schedules non-overlapping backpropagation (BP) and communication, leaving either the GPU or the network link idle and causing severe efficiency limitation (Li et al., 2020; Wang & Joshi, 2019). This inefficiency is linked to managing model divergence, defined as $\Gamma_r = \frac{1}{K}\sum_{k=1}^{K}||\bar{w}_r - w_r^k||^2$, which represents the approximation error of LSGD to S-SGD (Wang & Joshi, 2019; Mishchenko et al., 2022). While more frequent synchronization (a smaller H) reduces this divergence and improves convergence speed (Stich, 2019; Woodworth et al., 2020), it also magnifies the idle time caused by the hard synchronization.

Challenge 2: Overlapping Parameter Synchronization and Computation. Overlapping communication and computation in LSGD requires synchronizing model parameters, unlike S-SGD which synchronizes gradients (Zhang et al., 2017; Shi et al., 2021). To avoid increasing GPU memory costs for large models like LLMs (Ren et al., 2021; Rajbhandari et al., 2019), parameters should be synchronized in-place. Besides, the in-place synchronization modifies model parameters during the forward and backward passes, which can corrupt feature and gradient computations.

Challenge 3: Scheduling Communication and Computation. Scheduling the overlap is straightforward in S-SGD methods like WFBP (Zhang et al., 2017; Shi et al., 2021), where a layer's gradient communication can start immediately after its BP within a single iteration (Shi et al., 2020). In contrast, LSGD's scheduling is more complex. Because parameters are synchronized only every H iterations, one must consider how to assign the communication of L layers across these H iterations. This creates an enormous search space of H^L, making it difficult to find an optimal schedule.

3 DreamDDP Design

DreamDDP is designed to address above challenges to improve the training efficiency of LSGD. The overall design of DreamDDP is shown in Fig. 4. We first propose a partial synchronization method for LSGD, which synchronizes different model parts in different iterations to create opportunities for overlapping communication and computation (Section 3.1). DreamDDP implements overlapping by analyzing dependencies, ensuring each layer's in-place synchronization only occurs after its own BP and overlaps with the BP of subsequent layers (Section 3.2). Then, to schedule the overlapping between BP and synchronization, we formulate this as an optimization problem and solve it with a profiler-informed DFS algorithm that leverages specific properties like optimal hiding to efficiently prune the large search space (Section 3.3). Finally, DreamDDP identifies and fills any remaining idle communication bubbles with extra parameter synchronizations to accelerate training convergence (Section 3.4).

3.1 Partial Synchronization

Original LSGD uses *full synchronization* (Fig. 3(a)), where its strict synchronization prevents overlapping communication during the H training iterations, unlike methods such as WFBP (Fig. 3(b)). To better hide communication, we

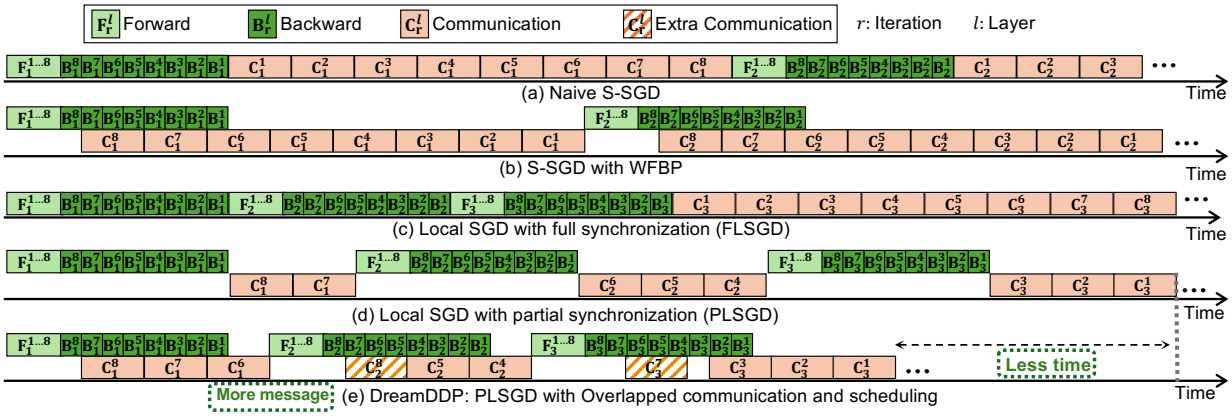


Figure 3. Timelines of different distributed training algorithms.

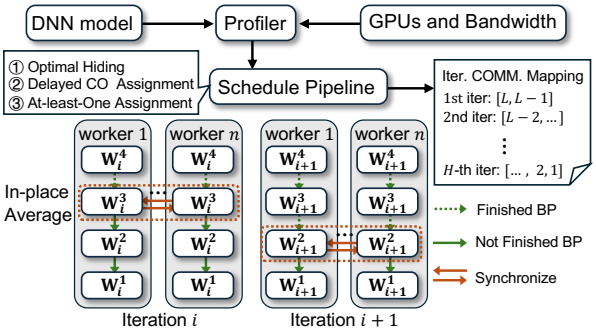


Figure 4. An overview of how DreamDDP schedules and launches the overlapping of backward propagation and parameter synchronization. In different iterations, DreamDDP synchronizes model parameters according to the scheduled iteration-layer mappings.

propose partial synchronization LSGD (PLSGD). We split the model into segments w_l based on layer indexes $\mathcal{L}$ and assign them to different local training iterations for communication, as shown in Fig. 3(d). This creates more overlap opportunities compared to Fig. 3(c).

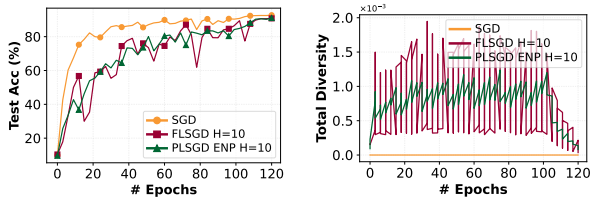
To implement this overlap, during automatic differentiation, we instantly launch an in-place asynchronous parameter average as soon as a layer's BP completes. As shown in the timeline (Fig. 3(e)), workers only wait for all communications to finish after the entire BP pass is done, before starting the next SGD step. We now formally define this process and provide an example of evenly assigning layer synchronization in PLSGD.

Definition 1. (**Partial Synchronization.**) Assuming that layer indexes $\mathcal{L} = \{1, ..., L\}$ of the model are sequentially split to H disjoint subsets $\mathcal{L}_{1:H} = \{\mathcal{L}_1, ..., \mathcal{L}_H\}$. Given the synchronization period H, the updating form of LSGD with partial synchronization is

$$w_{r+1}^{k,l} = \begin{cases} w_r^{k,l} - \eta_r \nabla^l f(w_r^k; \xi_r^k), & \text{if } r + 1\%H \neq H_l \\ \frac{1}{K}\sum_{k=1}^{K}(w_r^{k,l} - \eta_r \nabla^l f(w_r^k; \xi_r^k)), & \text{if } r + 1\%H = H_l \end{cases}, \quad (4)$$

in which $l \in \mathcal{L}$, and $H_l = \mathcal{I}(l)$ is the index function that indicates which subset of $\mathcal{L}_{1:H}$ the l-th layer belongs to. Note that the gradient is calculated w.r.t. the whole model w_r^k. $\nabla^l f(w_r^k; \xi_r^k)$ represents the gradient of the l-th layer.

Example 1. (**Equal-Number Partition.**) Given $H = L/2$, we partition $\mathcal{L}$ into H sets: $\mathcal{L}_1 = \{1, 2\}, \mathcal{L}_2 = \{3, 4\}, ..., \mathcal{L}_H = \{L-1, L\}$, denoted as $\mathcal{L}_{1:H}$. Definition 1 implies different averaging periods for each set. As with full synchronization, all parameters are synchronized once every H iterations. However, while full synchronization clears the accumulated divergence $\Gamma_r = \frac{1}{K}\sum_{k=1}^{K}||\bar{w}_r - w_r^k||^2$ every H iterations (Stich, 2019; Mishchenko et al., 2022), PLSGD clears the layer-wise divergence Γ_r^l on the same schedule.



(a) Test accuracy (b) Model divergence

Figure 5. Training ResNet-18 on 32 workers.

Reduced Divergence Amplification. In PLSGD, some model divergence Γ_r always exists, contrasting with full synchronization, which fully clears divergence every H steps (et al, 2021; Stich, 2019). We empirically find PLSGD has less overall divergence Γ_r and better convergence as shown in Fig. 5[1]. Full synchronization divergence is periodically accumulated and cleared, while PLSGD maintains a lower and more stable divergence. This benefit arises from how divergence propagates. Considering an intermediate feature $z^{k,l}(x) = f^{k,l}(w^{k,1:l}, x)$, divergence in weights

[1]Note that the S-SGD converges faster because it synchronizes gradients at each iteration. However, it will converge more slowly when switching x-axis to wall-clock time because of much more communication time, as shown later in experiments.

$w^{k,1:l}$ causes divergence in features $z^{k,l}$. Since features are dependent, this divergence amplifies layer-by-layer. In full synchronization, this amplification creates large divergence spikes before each reset (Fig. 5(b)). In contrast, PLSGD frequently clears divergence in some layers, mitigating this amplification effect.

We theoretically prove that PLSGD has the same convergence rate as the S-SGD (Bottou et al., 2016).

Theorem 1. *Let f be β-smooth and μ-strongly convex, and $\mathbb{E}_{\xi_r^k}||\nabla f(w_r^k;\xi_r^k) - \nabla f(w_r^k)||^2 \leq \sigma^2$, $\mathbb{E}_{\xi_r^k}||\nabla f(w_r^k;\xi_r^k)||^2 \leq G^2$, for $r = 0, 1, ..., R-1$, where $\{w_r^k\}_{r=0}^R$ for $k \in [K]$ are generated according to (4) with $H_l \leq H, l = 1, 2, ..., L$ and for stepsizes $\eta_r = \frac{4}{\mu(a+r)}$ with shift parameter $a > \max\{16\kappa, H\}$, for $\kappa = \frac{L}{\mu}$. Then*

$$\mathbb{E}f(\hat{w}_R) - f^* \leq \frac{\mu a^3}{2S_R}||w_0 - w^*||^2 + \frac{4R(R+2a)}{\mu K S_R}\sigma^2 + \frac{256R}{\mu^2 S_R}G^2H^2\beta \tag{5}$$

where $\hat{w}_R = \frac{1}{KS_R}\sum_{k=1}^K\sum_{r=0}^{R-1} p_r w_r^k$, for $p_r = (a+r)^2$ and $S_R = \sum_{r=0}^{R-1} p_r \geq \frac{1}{3}R^3$.

Corollary 2. *Let $\hat{w}_R$ be defined as in Theorem 1, for parameter $a = \max\{16\kappa, H\}$. Then*

$$Ef(_R) - f^* = \mathcal{O}\Big(\frac{\kappa H^2}{\mu R^2} + \frac{\kappa^3 + H^3}{\mu R^3}\Big)G^2 + \mathcal{O}\Big(\frac{1}{\mu K R} + \frac{\kappa + H}{\mu K R^2}\Big)\sigma^2.$$

Remark. Theorem 1 and Corollary 2 show that LSGD with partial synchronization has the same convergence rate as S-SGD (Bottou et al., 2016; Woodworth et al., 2020) as $\mathcal{O}(1/R)$. Small H can help accelerate the convergence of other terms containing $\mathcal{O}(1/R^2)$ or $\mathcal{O}(1/R^3)$, which are not dominant in the convergence bound with respect to the training iterations R. The detailed proof is in Appendix A.9.

Difference with standard asynchronous SGD. Unlike standard asynchronous SGD analyses that bound the impact of *gradient staleness* from staled updates, our analysis focuses on *model divergence* (Γ_r). Operating within the Local SGD paradigm, DreamDDP performs in-place layer-wise parameter synchronization rather than asynchronous gradient updates. Thus, our theoretical novelty lies in proving that layer-wisely decoupled local updates can frequently clear partial divergence, maintaining a convergence rate comparable to S-SGD without suffering from gradient staleness.

Wall-clock Time Complexity Analysis. Following definitions in Section 2, we analyze and compare the time complexity of one-iteration overlapped LSGD and partial synchronization. With Definition 1, the new training time of LSGD with partial synchronization is

$$T_{P-LSGD} = R \times t_{FP} + \frac{R}{H}\sum_{h=1}^{H}(t_{BP}^{\mathcal{L}_{1:h-1}} + t_{BP}^{h_0} + \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h})), \tag{6}$$

in which, h_0 represents the first layer in set $\mathcal{L}_h$, $t_{BP}^{\mathcal{L}_{1:h-1}} = \sum_{i=1}^{h-1}(\sum_{l\in\mathcal{L}_i} t_{BP}^l)$ is the backward time of layers in $\mathcal{L}_1, ..., \mathcal{L}_h$, $t_{BP}^{\mathcal{L}_{h:H}} = \sum_{i=h}^{H}(\sum_{l\in\mathcal{L}_i} t_{BP}^l)$ is the BP time of layers in $\mathcal{L}_{h+1}, ..., \mathcal{L}_H$, $t_{COMM}^{\mathcal{L}_i} = \tau_{COMM}^{h_{|\mathcal{L}_i|-1}} + t_{COMM}^{h_{|\mathcal{L}_i|-1}} - \tau_{COMM}^{h_0}$ is the communication time of layers in $\mathcal{L}_i$, where $\tau_{COMM}^{h_j} = \max(t_{BP}^{h_{0:j}}, \tau_{COMM}^{h_{j-1}} + t_{COMM}^{h_{j-1}})$ defines the timestamp for the communication time of layer h_j. The expression $t_{BP}^{h_{0:j}}$ indicates the completion timestamp for the backward time of layer h_j, and $\tau_{COMM}^{h_{j-1}} + t_{COMM}^{h_{j-1}}$ marks the completion timestamp for the communication time of layer h_{i-1}. The term $t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}$ comes from that the communication of the first layer in the set $\mathcal{L}_h$ must wait for the BP finishing to avoid simultaneous parameter accessing.

3.2 Overlapping Communication and Computation

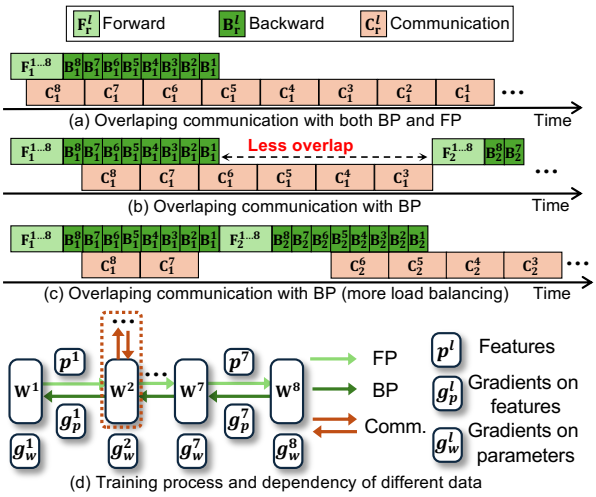


Figure 6. The dependency and overlap opportunity between communication and computation of LSGD.

Now, we analyze dependency of different operations in PLSGD and overlap them. Figure 6 (d) shows the phases of LSGD, where workers synchronize parameters W instead of gradients g_W in S-SGD. Guided by the principle of introducing no extra GPU memory, we must use in-place parameter averaging, forcing us to carefully schedule communication around the BP process. Since both FP and BP computations depend on W_l, in-place parameter synchronization cannot overlap with a layer's own FP computation, as this would lead to incorrect results. Thus, we can overlap the BP with the in-place synchronization.

However, naive overlapping BP with synchronization may lead to suboptimal scheduling as shown in Figure 6 (b), which simply communicates all layers in the first local iteration. We must determine how to schedule layer synchronization across H iterations to minimize the total training time. A more balanced strategy (Figure 6 (c)) is often better. In

the next section, we formally define the overall training time as an optimization problem and identify three properties to drastically reduce the search space for its solution.

3.3 Scheduling and Optimization

With partial synchronization, the intuitive way to reduce the training time is to minimize the training time T_{P-LSGD} by adjusting the partition of layer index sets $\mathcal{L}_{1:H}$. According to the time cost of T_{P-LSGD} (Eq. 6), the time t_{FP}, t_{BP} and t_{COMM} are fixed, and different partitions of layer indexes in to $\mathcal{L}_{1:H}$ will influence $\sum_{h=1}^{H} \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h})$. If we can adjust the $\mathcal{L}_{1:H}$ well, $t_{COMM}^{\mathcal{L}_h}$ might be fully overlapped. The scheduling problem is highly complex; we approximate its solution by modeling each layer partition as an interval. Note that the communication of the last layer cannot be overlapped, because the communication must wait for the backward. Then, we propose the following objective function P_H^L as the goal of DreamDDP.

$$\min_{\mathcal{L}_{1:H}^{L:1}} P_H^L = \sum_{h=1}^{H} (t_{BP}^{\mathcal{L}_{1:h-1}} + t_{BP}^{h_0} + \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h})), \quad (7)$$

which is a combinatorial optimization problem, and $\mathcal{L}_{1:H}^{L:1}$ represents the possible partitions $\mathcal{L}_{1:H}$ of layers $\{L, ..., 1\}$. The brute force solution of this problem requires time complexity of $O(C_{H+L}^{L}) = \frac{(H+L)!}{L!H!}$.

Searching Process and Sub-structure Problem. To efficiently solve this problem, we identify the optimal substructure of problem 7. Now, we solve this problem with an efficient searching algorithm with reduced complexity as $\mathcal{O}(2^{\min(L-H,H)})$. The search space is significantly reduced by the optimal substructure of this problem. The searching process is to iteratively decide whether to assign l-th layer to the h-th set, $\mathcal{L}_h$. From this perspective, given assigned layers $\{L, ..., l+1\}$ in disjoint sets $\{\mathcal{L}_1, ..., \mathcal{L}_{h-1}, \mathcal{L}_h\}$, we define the sub-problem as follows,

$$\min_{\mathcal{L}_{h:H}^{l:1}} S_h^l = \sum_{i=h}^{H} (t_{BP}^{\mathcal{L}_{1:i-1}} + t_{BP}^{i_0} + \max(t_{BP}^{\mathcal{L}_{i:H}} - t_{BP}^{i_0}, t_{COMM}^{\mathcal{L}_i})),$$

in which the i iterates from h to H, H and L are fixed in S_h^l, optimizing on $\mathcal{L}_{h:H}^{l}$ means to split layers $\{1, ..., l\}$ to sets $\{\mathcal{L}_h, \mathcal{L}_{h+1}, ..., \mathcal{L}_H\}$. Note that l follows descending order as the backward process is from output to input, and h follows the ascending order as the local training follows time order in a synchronization period $r+1, ..., t+H$. When, $h = 1$ and $l = L$ in S_h^l, S_1^L becomes the original problem 7, i.e. $P_H^L = S_1^L$.

Definition 2. (**Assignment** A^l.) We call assigning a layer l into $\mathcal{L}_h$ as replacing the set $\mathcal{L}_h$ by $\mathcal{L}_h \cup \{l\}$ in original sets $\mathcal{L}_{1:H}$ and we have $A^l = h$.

Optimizing P_H^L can be seen as an iterative searching process from optimizing S_1^L to S_H^1. In each iteration of optimizing S_h^l, we have $\{L, ..., l+1\}$ that have been assigned in disjoint sets $\{\mathcal{L}_1, ..., \mathcal{L}_{h-1}, \mathcal{L}_h\}$. Then, based on whether to assign the layer l to set $\mathcal{L}_h$, each S_h^l has the substructure as:

$$\mathrm{S}_h^l = \begin{cases} S_h^{l-1} + \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h \cup \{l\}}) \\ \quad - \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h}) & \text{if } A^l = h \\ S_{h+1}^l + t_{BP}^{\mathcal{L}_{1:h-1}} + t_{BP}^{h_0} \\ \quad + \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h}), & \text{if } A^l \neq h \end{cases}, \quad (8)$$

in which, for simplicity, we do not expand terms $t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0} = \sum_{i=h}^{H} (\sum_{l \in \mathcal{L}_i} t_{BP}^{l}) - t_{BP}^{h_0}$ and $t_{COMM}^{\mathcal{L}_h} = \sum_{l \in \mathcal{L}_h} t_{COMM}^{l}$ that follows definition in. The first term means that we assign l to $\mathcal{L}_h$, thus the total time cost S_h^{l-1} re-estimates the time cost of the set $\mathcal{L}_h$ for the h-th training round. Note that the $t_{BP}^{\mathcal{L}_{h:H}} = \sum_{i=h}^{H} (\sum_{l \in \mathcal{L}_i} t_{BP}^{l})$ actually is fixed when scheduling S_h^l no matter whether $l \in \mathcal{L}_h$, because all left layers $\{\mathcal{L}_h, ..., \mathcal{L}_H\}$ will conduct BP operations at h-th iteration, which is fixed for S_h^l. We write the optimal solution of $\min_{\mathcal{L}_{h:H}^{l:1}} S_h^l$ as $S_{h,\star}^l$.

The optimal $\mathcal{L}_{h:H}^{l:1}$ of Equation (7) should minimize the communication time as much as possible, because the total BP time is fixed as $t_{BP}^{\mathcal{L}}$ for each iteration h from 1 to H. Now, we give following definition and properties which are used to analyze and reduce the search space.

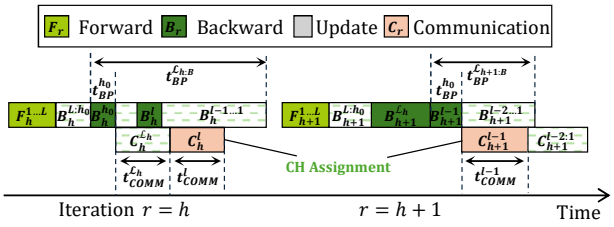
Definition 3. (**Communication-Hide Assignment.**) Given assigned layers $\{L, ..., l+1\}$ in disjoint sets $\{\mathcal{L}_1, ..., \mathcal{L}_h\}$, and the optimization goal S_h^l, we call the assignment (Definition 2) $A^l = h$ is Communication-Hide (CH) if it satisfies

$$t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0} \geq t_{COMM}^{\mathcal{L}_h \cup \{l\}}. \quad (9)$$

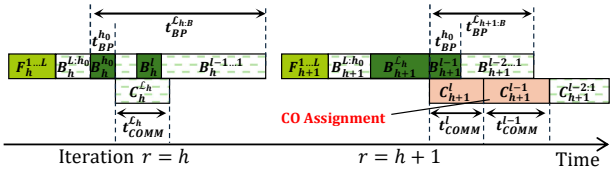
Otherwise, we call the assignment $A^l = h$ is communication-overflowed (CO).

Property 1. (**Optimal Hiding.**) As shown in Fig. 7(a), considering the case that $A^l = h$ is CH for S_h^l, then $\max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h \cup \{l\}}) = \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}, t_{COMM}^{\mathcal{L}_h}) = t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0}$, we have $S_{h,\star}^{l-1} \leq S_{h+1,\star}^{l} + t_{BP}^{\mathcal{L}}$. This property indicates that once the synchronization of the l-th layer at the h-th iteration fully hides its communication time, there is no need for further synchronization in the subsequent $H - h$ iterations, as shown in Fig. 7(b). This prevents the occupation of overlapping opportunities for the other $l - 1$ layers {l-1, ..., 1}. Thus, based on the optimal hiding (property 1), DreamDDP assigns $A^l = h$ if it is fully hidden by computation (line 20 in Algorithm 1), which significantly reduces the search space.

Property 2. (**Delayed CO Assignment.**) As shown in Fig. 8(a), considering the case that communication of set $\mathcal{L}_h$ cannot be hidden by the BP computation, i.e. $t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0} \leq t_{COMM}^{\mathcal{L}_h}$. In this situation, the assignment $A^l = h$ is also CO, because of $t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0} \leq t_{COMM}^{\mathcal{L}_h} \leq t_{COMM}^{\mathcal{L}_h \cup \{l\}}$ and Definition 3. Now, $A^l = h$ will lead to extra communication time as t_{COMM}^{l}, which is completely not hidden and



(a) At iteration r, $S_h^l \rightarrow S_h^{l-1}$ $(A^l = h)$.

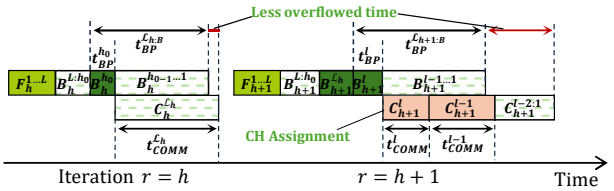


(b) At iteration r, $S_h^l \rightarrow S_{h+1}^l (A^l \neq h)$.

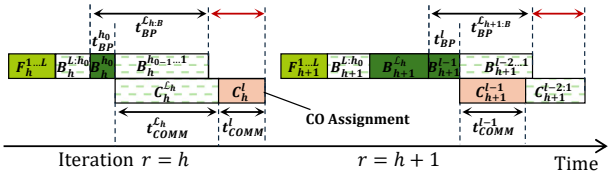
Figure 7. Comparing whether assigning $A^l = h$ when A_h^l is CH at iteration r. The time of $F_h^{1...L}$, $B_h^{L...l}$, $B_h^{l-1...1}$ are shrunk in Figure for clear illustration. (a) assigning $A^l = h$ can fully hide the communication time; (b) delaying l-th layer may result in sub-optimal overlapping.

increases the total training time (in Fig. 8(b)). However, if we delay communicating layer l at iteration $h + 1$, the communication cost might be fully or partly hidden by the computation as shown in Fig. 8(a).

Note that the delayed CO assignment (Property 2) may not be optimal. However, this heuristic insight can help us significantly reduce the search space. For each h, if $t_{BP}^{\mathcal{L}_h:H} - t_{BP}^{h_0} \leq t_{COMM}^{\mathcal{L}_h}$, the l-th layer will be delayed for future assignment (Line 23 in Algorithm 1).



(a) At iteration r, $S_h^l \rightarrow S_{h+1}^l (A^l \neq h)$.



(b) At iteration r, $S_h^l \rightarrow S_h^{l-1}$ $(A^l = h)$.

Figure 8. Comparing whether assigning $A^l = h$ when the t_{COMM}^l is completely not hidden at iteration r. (a) Delaying synchronizing l-th layer to $h + 1$ iteration may help overlapping; (b) Assigning $A^l = h$ results in completely non-overlapping.

Property 3. **(At-Least-One Assignment (ALO).)** If $\mathcal{L}_h$ is $\emptyset$ when scheduling l-th layer at iteration h, assigning $A^l = h$ is optimal. We provide the proof with discussing all cases as follows. The first case is that $t_{COMM}^l \leq t_{BP}^{\mathcal{L}_h:H} - t_{BP}^{h_0}$, which satisfies property 1. In the other case $t_{COMM}^l > t_{BP}^{\mathcal{L}_h:H} - t_{BP}^{h_0}$, assigning $A^l = h$ helps overlap t_{COMM}^l with part of computation. Otherwise, delaying $A^l = h + 1$ still causes the same overflowed communication time and narrows the overlap opportunity of communicating subsequent layers. In light of the ALO (property 3), when deciding l-th layer and h-th iteration, if $\mathcal{L}_h$ is $\emptyset$, DreamDDP instantly assigns $A^l = h$ (Line 17 in Algorithm 1).

With properties 1, 2 and 3, the scheduling algorithm 1 conducts depth-first search (DFS) with reduced search space to find the solution.

Complexity Analysis of Algorithm 1. We consider the worst case for the algorithm complexity, where the DFS search (Line 25 in Algorithm 1) frequently happens during the scheduling. Each DFS search leads to two branches that need to be saved into the solution set. When $\mathcal{L}_h$ is $\emptyset$, the

Algorithm 1 Assigning Communication of DreamDDP

Input: Layer set $\mathcal{L}$, synchronization frequency H, backward times $\{t_{BP}^l | l \in \mathcal{L}\}$, communication times $\{t_{COMM}^l | l \in \mathcal{L}\}$, initial assignment set $\Psi = \{\mathcal{L}_h = \emptyset | h \in \{1, ..., H\}\}$, solutions set Ω.
Output: Final trained model w_R.

1: SOLVESUBPROBLEM(Ψ, L, L, 1, H) ▷ Scheduling
2: $\Psi^\star = \text{argmin}_{\Psi \in \Omega} P_H^L$; ▷ Find one optimal solution
3: **for** $\mathcal{L}_h \in \Psi^\star$ **do**
4: $l_h^- = \min_l t_{COMM}^{L...l}$ $s.t.$ $l \notin \mathcal{L}_h$ and Eq. 10.
5: $\mathcal{L}_h = \mathcal{L}_h \cup \{L, ..., l\}$ ▷ Add more communication
6: Return $\Psi^\star$.
7:
8: **procedure** SOLVESUBPROBLEM(Ψ, l, L, h, H)
9: **if** $l == 0$ **then**
10: Add Ψ in Ω. ▷ Get one solution
11: Return
12: **if** $h == H$ **then**
13: **for** $i = l, ..., 1$ **do**
14: Replace $\mathcal{L}_h \in \Psi$ as $\mathcal{L}_h = \mathcal{L}_h \cup \{i\}$
15: Add Ψ in Ω. ▷ Get one solution
16: Return
17: **if** $\mathcal{L}_h$ is $\emptyset$ **then** ▷ At-least-one Assignment
18: Replace $\mathcal{L}_h \in \Psi$ as $\mathcal{L}_h = \mathcal{L}_h \cup \{l\}$
19: SOLVESUBPROBLEM(Ψ, $l - 1$, L, h, H)
20: **else if** $t_{BP}^{\mathcal{L}_h:H} - t_{BP}^{h_0} \geq t_{COMM}^{\mathcal{L}_h \cup \{l\}}$ **then** ▷ Optimal Hiding
21: Replace $\mathcal{L}_h \in \Psi$ as $\mathcal{L}_h = \mathcal{L}_h \cup \{l\}$
22: SOLVESUBPROBLEM(Ψ, $l - 1$, L, h, H)
23: **else if** $t_{BP}^{\mathcal{L}_h:H} - t_{BP}^{h_0} < t_{COMM}^{\mathcal{L}_h}$ **then** ▷ Delayed COA
24: SOLVESUBPROBLEM(Ψ, l, L, $h + 1$, H)
25: **else** ▷ Search
26: $\Psi_- = \text{DeepCopy}(\Psi)$
27: Replace $\mathcal{L}_h \in \Psi_-$ as $\mathcal{L}_h = \mathcal{L}_h \cup \{l\}$ ▷ Assign $A^l = h$
28: SOLVESUBPROBLEM(Ψ_+, $l - 1$, L, h, H)
29: $\Psi_+ = \text{DeepCopy}(\Psi)$
30: SOLVESUBPROBLEM(Ψ_+, l, L, $h - 1$, H)

l-th layer will be added into $\mathcal{L}_h$ without searching, which does not generate more solutions. Thus, the size of $|\Omega|$ is less than 2^{L-H}. Considering the branch $A^l = h$ (Line 28 in Algorithm 1), the next layer $l-1$ must be delayed to assign to iteration $h+1$ because $t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0} < t_{COMM}^{\mathcal{L}_h \cup \{l\}}$. Thus, the $|\Omega|$ is less than 2^H. In summary, the size of the solution set Ω is upper bounded as $\mathcal{O}(2^{\min(L-H,H)})$. Normally, the H in LSGD cannot be set too large, of which the value ranges in $5 \sim 10$ (Patel et al., 2024; Wang & Joshi, 2019), which makes the time complexity $\mathcal{O}(2^{\min(L-H,H)})$ of scheduling algorithm acceptable.

Profiling. Before training, DreamDDP launches the profiler to collect the FP, BP and communication time of different layers $l \in \{L, ..., 1\}$ as t_{FP}^l, t_{BP}^l and t_{COMM}^l respectively. The profiled BP and communication time will be used for scheduling in Algorithm 1.

3.4 Filling the Bubble Time

As illustated in Section 3.1 and Theorem 1, the model divergence $\Gamma_r = \frac{1}{K}\sum_{k=1}^{K} ||\bar{w}_r - w_r^k||^2$ is a main influence factor of the convergence rate. Higher communication frequency (lower H) leads to less model divergence Γ_r during training, thus accelerating convergence (et al, 2021; Stich, 2019). Orthogonal to increasing synchronization frequency H, which means to synchronize the whole model more frequently, another way to reduce the model divergence is to assign different layers with different communication frequencies.

After scheduling the overlapping communication, we find that there may exist some idle communication time of the late layers during the late local training stages (larger local r of in one synchronization period) as shown in Figure 9. This implies that it is possible to synchronize these layers more frequently to reduce the model divergence, while not increasing the communication time.

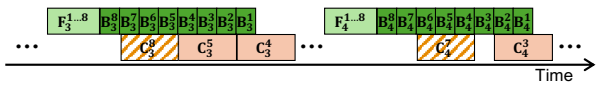


Figure 9. The idle communication time of during the late local training stages that can be used to synchronize the late layers more frequently.

It is known that the early layers of deep learning models converge faster than the late layers (Chen et al., 2023; Raghu et al., 2017), including the transformer (He et al., 2021). This is because when low-level representations stabilize, later layers can train on more reliable intermediate features from early layers (Chen et al., 2023; Raghu et al., 2017). In PLSGD, the idle communication windows align with late layers (Fig. 9). Using the idle time to more frequent synchronization of late layers is therefore more effective for reducing model divergence than prioritizing early layers.

Formally, DreamDDP communicates more layers $\mathcal{L}_h^- = \{L, ..., l\}$ for communication iteration h (Line 5 in Algorithm 1) to accelerate convergence, while ensuring the communication time is not increased as follows.

$$\max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0},\ t_{COMM}^{\mathcal{L}_h \cup \{L,...,l\}}) \leq \max(t_{BP}^{\mathcal{L}_{h:H}} - t_{BP}^{h_0},\ t_{COMM}^{\mathcal{L}_h}) \quad (10)$$

in which $\{L, ..., l\}$ disjoints with $\mathcal{L}_h$.

3.5 Robustness and Fault Tolerance

In large-scale geo-distributed training, system resilience against node failures is critical. While DreamDDP decouples model synchronization layer-wisely across different iterations, it is designed to strictly ensure model consistency for fault tolerance. Specifically, the DFS-based scheduler guarantees that every layer is synchronized at least once within a bounded synchronization period H.

Consistent Checkpointing and Rollback: At the end of every synchronization period H, all layer-wise divergences are cleared at least once in H iterations, resulting in a globally consistent model state across all workers. This in-place synchronization allows the system to capture standard, fault-tolerant model checkpoints at these deterministic boundaries without incurring any additional GPU memory overhead. In the event of a node failure, the training can safely rollback to the last consistent snapshot at period H, ensuring robust recovery without data corruption or stale parameter states.

4 Experimental Studies

4.1 Experimental Settings

Models and datasets. We conduct distributed training on deep models including ResNet-{18,50}, GPT-2 small (Radford et al., 2019), small Llama-2 (Touvron et al., 2023b) of 175M parameters. We also conduct experiments with Qwen2.5-7B-Instruct (Qwen et al., 2025) and Qwen1.5-MoE-A2.7B (Team, 2024), both of which employ LoRA (Hu et al., 2021) for parameter-efficient fine-tuning due to hardware memory constraints. ResNet-18 is trained on CIFAR-10 and ResNet-50 is trained on CIFAR-100 with SGD momentum. All LLMs are trained on WikiText-2 (Merity et al., 2017) with Adam (Kingma & Ba, 2015).

Simulating Geo-distributed Training Clusters. ResNet models are trained on one cluster with 8 nodes, each of which has 4 NVIDIA 2080Ti (totally 32 GPUs) and connected with 1GB/s Ethernet. LLMs are trained on the second cluster with 4 nodes, each of which has 8 NVIDIA A6000 (totally 32 GPUs) and connected with 20GB/s Ethernet. The 1GB/s and 20GB/s bandwidth represent diverse network conditions across different geographic locations.

Baselines. We compare our algorithms with the naive S-

SGD, ASC-WFBP (Shi et al., 2021) which is the state-of-the-art SGD overlapping algorithm as an improvement of WFBP based S-SGD (Zhang et al., 2017), and LSGD with full synchronization (FLSGD) (Patel et al., 2024; Zhang et al., 2023). As the ablation study, we conduct experiments of the basic versions of DreamDDP, including LSGD with partial synchronization (PLSGD-ENP) of Equal-Number Partition (Example 1). Note that the optimizer for LLMs is Adam, following common practice for LLMs (Radford et al., 2019; Touvron et al., 2023b). The computation and communication dependency of stochastic Adam and SGD are the same, as well as the local Adam and LSGD. Thus, we also call the stochastic Adam and local Adam as S-SGD and LSGD to highlight the different modes of parallelization instead of the specific optimizers for simplicity. Unless otherwise stated, the local update period H is set to 10.

Adaptations for Large LLMs and MoE. For Qwen2.5-7B-Instruct and Qwen1.5-MoE-A2.7B, we introduce specific system adaptations. We employ LoRA for both models and communicate only trainable LoRA matrices for all baselines and DreamDDP. While MoE training conventionally employs Expert Parallelism (EP) (Fedus et al., 2022; Lepikhin et al., 2020), this paradigm is fundamentally compatible with DreamDDP. Adapting DreamDDP to EP simply requires restricting the `all_reduce` synchronization of each expert to the specific subset of GPUs hosting its replicas. In this paper, we adopt a pure DP approach where all experts are replicated locally on each worker and synchronized globally via `all_reduce`. Due to hardware constraints, we only test the average iteration time for these large-scale LLMs, omitting convergence evaluations.

4.2 Profiling and Theoretical Time Reduction

We profiled and averaged BP and communication time for the whole model over 1000 iterations. The profiling is launched before training for scheduling. Llama-2 and GPT-2 are trained with 32 A6000 (20GB/s Bandwidth), while ResNet-{18,50} are trained with 32 2080Ti (1GB/s Bandwidth) clusters. Table 1 shows the time cost of one round of Local SGD (with $H = 10$). The time after overlapping shows the estimated time costs when overlapping all communication with the computation. The results show that the ideal time reduction can achieve around 32.6% $\sim$ 45.0%.

Table 1. Comparison of profiled COMM and BP time of different models and estimated LSGD time with 10 local updating steps.

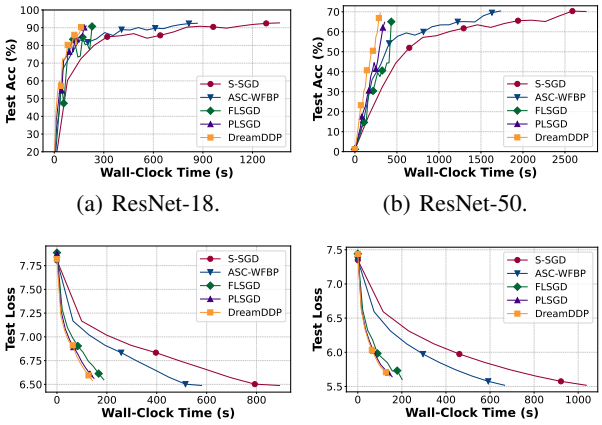
Model	Comm. Time (s)	Comp. Time (s)	Total Time (s) (No Overlap)	Time (s) After Overlapping (Ideal)
Llama-2	0.9369	0.552	1.489	0.9369 (-37.1%)
GPT-2	0.9140	0.648	1.562	0.9140 (-41.5%)
ResNet18	0.8036	0.388	1.192	0.8036 (-32.6%)
ResNet50	1.6431	1.347	2.990	1.6431 (-45.0%)
Qwen-2.5-7B	2.18431	21.1920	24.0351	22.6422 (-5.8%)
Qwen1.5-MoE	20.4317	51.0923	71.5240	53.6112 (-25.0%)

4.3 Wall-clock Iteration Time

Table 2 presents the wall-clock iteration time for each algorithm across different models and worker counts. DreamDDP consistently achieves the lowest iteration time across all models and worker configurations, demonstrating its superior efficiency in hiding extra communication. Specifically, compared to SOTA DDP training method ASC-WFBP, DreamDDP achieves a speedup of $1.73 \sim 5.22\times$. Similarly, compared to FLSGD, DreamDDP achieves a speedup of $1.16 \sim 1.5\times$. This performance improvement can be even more pronounced in low-bandwidth environments, which are common in geo-distributed scenarios.

Table 2. Average iteration wall-clock time (in seconds) of 1000 running iterations. S_1 and S_2 represent the speedup of DreamDDP over ASC-WFBP and FLSGD.

Model	ResNet-18		ResNet-50		GPT-2	
# of workers	8	32	8	32	8	32
S-SGD/Adam	1.31	2.40	3.08	4.52	2.63	8.67
ASC-WFBP	0.99	1.75	2.33	3.39	1.72	5.38
FLSGD/Adam	0.66	0.57	1.59	0.94	0.80	2.08
PLSGD/Adam-ENP	0.58	0.46	1.37	0.69	0.70	1.99
DreamDDP	**0.57**	**0.38**	**1.33**	**0.65**	**0.66**	**1.60**
S_1	1.73×	4.61×	1.75×	5.22×	2.61×	3.36×
S_2	1.16×	1.50×	1.20×	1.45×	1.21×	1.23×
Model	Llama-2		Qwen2.5-7B (LoRA)		Qwen1.5-MoE	
# of workers	8	32	8	32	8	32
S-SGD/Adam	2.80	8.75	12.93	28.55	17.43	25.54
ASC-WFBP	1.83	5.36	9.20	18.93	12.49	17.37
FLSGD/Adam	0.86	1.96	4.53	6.88	6.29	7.13
PLSGD/Adam-ENP	0.77	1.89	3.71	6.37	5.17	5.53
DreamDDP	**0.72**	**1.55**	**3.65**	**6.07**	**5.08**	**5.35**
S_1	2.54×	3.46×	2.52×	3.12×	2.45×	3.24×
S_2	1.20×	1.26×	1.24×	1.13×	1.23×	1.33×



(a) ResNet-18. (b) ResNet-50.

(c) GPT-2. (d) Llama-2.

Figure 10. Convergence w.r.t. wall-clock time of training with different algorithms on 32 workers.

4.4 Wall-clock Convergence

Fig. 10 illustrates the convergence of different algorithms w.r.t. wall-clock time and Table 3 presents wall-clock time to reach target performance. Both visualizations show that the wall-clock convergence of all LSGD methods is significantly

faster than that of SGD. The periodic model synchronization scheme effectively reduces the high communication cost associated with the frequent synchronization in S-SGD and ASC-WFBP. Specifically, DreamDDP achieves a maximum speedup of $3.91\times$ over ASC-WFBP.

Additionally, DreamDDP achieves a maximum speedup of $1.56\times$ over FLSGD, due to the advanced scheduling algorithm that eliminates the extra communication overhead. Furthermore, compared to PLSGD with a similar overlap strategy, DreamDDP demonstrates improved convergence speed, as displayed in Fig. 10 and Table 3. This improvement shows that DreamDDP's supplementary communication mechanism maximally exploits overlapping opportunities without incurring extra training time.

Different Synchronization Frequency. We compare different DreamDDP synchronization frequencies H with FLSGD with $H = 10$ and S-SGD in Fig. 11 and Fig. 12. The results on both 8 and 32 workers show that the larger H makes convergence w.r.t. epochs slower for DreamDDP, which aligns with the intuition. Nevertheless, the supplementary communication in DreamDDP helps accelerate the convergence, achieving comparable convergence speed with FLSGD with higher frequencies. As shown in Fig. 12, increasing the number of workers does not affect DreamDDP's performance gain, demonstrating its strong scalability.

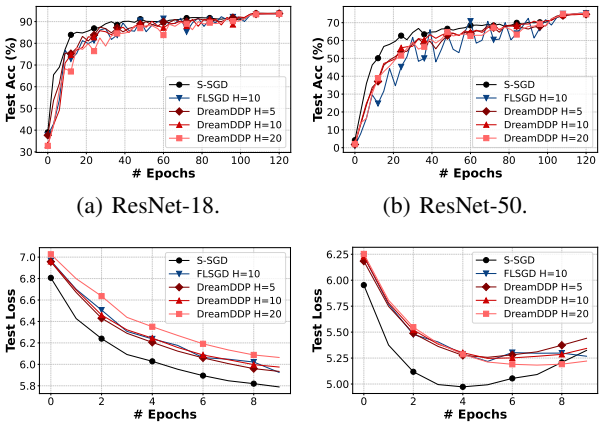
Table 3. Wall-clock time to target model performance of different training methods. Different training methods require different wall-clock time to achieve the target performance. For DL models, test accuracy is used. While for LLMs, training loss is used. S_1 and S_2 represent the speedup of DreamDDP over ASC-WFBP and FLSGD. We use the **bold font** to indicate the best results.

Model	ResNet-18		ResNet-50		GPT-2		Llama-2	
# of workers	8	32	8	32	8	32	8	32
S-SGD/Adam	1863.78	1282.08	6353.16	3546.24	394.15	167.29	342.57	247.55
ASC-WFBP	1608.00	1420.26	5384.58	3192.24	361.19	148.32	319.68	156.57
FLSGD/Adam	687.96	651.06	2141.64	1351.98	294.42	114.38	213.15	127.10
PLSGD/Adam-ENP	603.36	653.04	1900.14	1261.08	270.68	94.093	176.32	114.32
DreamDDP	**573.10**	**515.58**	**1375.68**	**1067.76**	**247.97**	**88.63**	**166.28**	**105.07**
S_1	2.80×	2.75×	3.91×	2.99×	1.46×	1.67×	1.92×	1.49×
S_2	1.20×	1.26×	1.56×	1.27×	1.19×	1.29×	1.28×	1.21×

4.5 Scheduling Performance

Different number of layers. To evaluate whether DreamDDP effectively maximizes communication–computation overlap, we compare its non-overlapped communication time with that of the brute-force optimal solution in Fig. 13. As the time complexity of the brute force is too large ($\frac{(H+L)!}{L!H!}$), we report scheduling results for up to 30 layers with $H = 5$. Fig. 13(a) shows that with varying number of layers, during a single training cycle (5 iterations), DreamDDP performs equivalently to the brute force optimal solution in most cases, with only slightly longer extra communication times in a few scenarios.

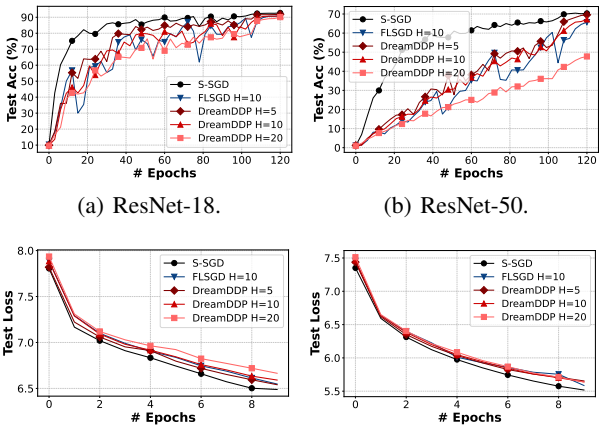
Different bandwidth. As shown in Fig. 13(b), DreamDDP achieves performance comparable to the brute-force optimum across bandwidths, indicating its efficiency to find near-optimal schedules with much lower search cost.



(a) ResNet-18. (b) ResNet-50.

(c) GPT-2. (d) Llama-2.

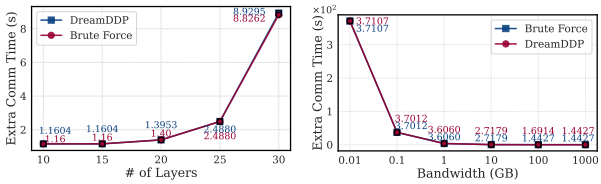
Figure 11. Scheduling performance with varying H on 8 workers.



(a) ResNet-18. (b) ResNet-50.

(c) GPT-2. (d) Llama-2.

Figure 12. Scheduling performance with varying H on 32 workers.



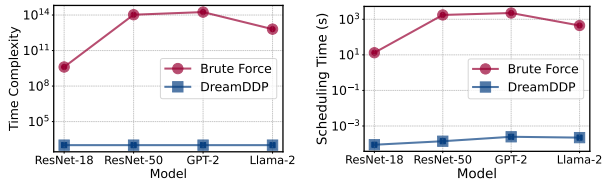
(a) different # of layers (b) different bandwidths

Figure 13. Comparison of extra communication time not overlapped with computation between.

4.6 Profiling and Searching Overhead

We compare the theoretical and realistic profiled running time of the scheduling algorithms, including DreamDDP and brute force searching on scheduling different mod-

els. Figure 14(a) shows the theoretical time complexity of scheduling four models with $H = 5$. As these models have too many layers, the time complexity of scheduling them with the brute force is infeasible. Thus, we only choose 30 layers for these four models to compare the realistic running time of them. Figure 14(b) shows that DreamDDP's scheduling method significantly reduces the searching complexity by several orders of magnitude, compared with brute force method. DreamDDP reduces the search space, effectively eliminating the need to explore all possible layer partitions.



(a) Theoretical Time Complexity (b) Real Scheduling Time

Figure 14. Estimated and wall-clock time of scheduling.

5 Related Works

Asynchronous Training vs. Partial Synchronization. In real-world scenarios, building massive clusters of homogeneous GPUs connected via high-bandwidth interconnects can often be infeasible (Luo et al., 2020; Barham et al., 2022; Ryabinin et al., 2023b; Tang et al., 2024a). To tackle the computational and communication heterogeneity, several works explore the asynchronous training paradigm (Dean et al., 2012; Barham et al., 2022; Feyzmahdavian & Johansson, 2023), which relaxes strict global synchronization barriers and allows workers to update the global model independently. In contrast, DreamDDP synchronously operates within the LSGD paradigm (Stich, 2019) and employs layer-wise partial parameter synchronization to overcome low bandwidth in geo-distributed training scenarios.

Overlapping Communication and Computation. In distributed training with S-SGD, the gradient of each layer can be communicated instantly after its BP and overlapped with BP of subsequent layers, thus saving total training time (Zhang et al., 2017). Overlapping communication with feed-forward computing (Peng et al., 2019) helps further reduce total training time. Some observe that merging small tensors for communication (Shi et al., 2020) can help improve communication efficiency. ASC-WFBP (Shi et al., 2021) further improves overlapping by simultaneous communications and scheduling. Streaming DiLoCo (Douillard et al., 2025b) explores a similar partial parameter synchronization by communicating subsets of layers in a fixed schedule. DreamDDP proposes a more fine-grained and adaptive solution with the DFS-based optimal scheduler that dynamically profiles the layer-wise computation to discover idle communication bubbles.

LSGD. LSGD (Stich, 2019; Woodworth et al., 2020; Zhang et al., 2023; Sadiev et al., 2022) reduces the communication time by periodically averaging model weights instead of gradients. The synchronization frequency strongly influences the convergence and communication time. Thus, some works vary synchronization periods to tradeoff convergence and communication (Stich, 2019; Zhang et al., 2023; Wang & Joshi, 2019). The strong synchronization severely restricts the potential improvements of LSGD, preventing it from communication overlapping. Some works in FL (Lee et al., 2023) propose layer-wise model synchronization without overlapping. Recently, some concurrent works also propose the partial local SGD to reduce communication time in Local SGD (Douillard et al., 2025a). Different from them, we identify three properties and propose better scheduling policy to optimize the overlapping.

Reducing Communication Costs. To reduce the communication time, some works compress the communicated gradients (Fei et al., 2021a; Tang et al., 2020) in S-SGD or model parameters (Guan et al., 2023; Wang et al., 2023a; Tang et al., 2022a; Sarfi et al., 2025) in LSGD. (Guan et al., 2023; Wang et al., 2023a) proposed compressing local results of Federated Learning (Tang et al., 2022b;a), in which LSGD is used. Orthogonal to this direction, our method aims to modify LSGD to improve system efficiency.

Optimizations in Orthogonal Parallel Paradigms. Recent literature has extensively explored communication efficiency across other parallel dimensions, including Pipeline Parallelism (PP) (Shoeybi et al., 2020; Huang et al., 2019; Narayanan et al., 2019; Ramasinghe et al., 2025), Tensor Parallelism (TP) (Zheng et al., 2022; Wang et al., 2022a), and Expert Parallelism (EP) (He et al., 2022; Rajbhandari et al., 2022). Because DreamDDP specifically optimizes the communication *schedule* for DP by overlapping communication with computation, our approach is fundamentally orthogonal. Consequently, DreamDDP is highly complementary to these methods and can be seamlessly integrated with them to unlock compounded efficiency gains.

6 Conclusion

To overcome the high communication costs of S-SGD and the strict synchronization of LSGD in geo-distributed training, we propose partial synchronization. Based on this, we design DreamDDP to schedule communication overlapped with backpropagation, further supplementing communication by maximally exploiting idle time. Experimental results show DreamDDP accelerates both wall-clock iteration time and convergence speed compared to ASC-WFBP and LSGD on models including ResNets and LLMs.

ACKNOWLEDGEMENTS

This work was partially supported by National Natural Science Foundation of China under Grant No. 62272122, Hong Kong CRF grants under Grant No. C7004-22G and C6015-23G, an NSFC grant 62432008, RGC RIF grant R6021-20, an RGC TRS grant T43-513/23N-2, RGC CRF grants C7004-22G, C1029-22G and C6015-23G, NSFC/RGC grant CRS_HKUST601/24 and RGC GRF grants 16207922, 16207423 and 16203824.

This work was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62302123 and the Shenzhen Science and Technology Program under Grant No. KJZD20240903104103005, Grant No. KJZD20230923114213027 and Grant No. KJZD20230923115113026.

REFERENCES

Albrecht, J. and Tuttle, C. Loose synchronization for Large-Scale networked systems. In *2006 USENIX Annual Technical Conference (USENIX ATC 06)*, 2006.

Ansel, J. e. a. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *ASPLOS*, pp. 929–947, 2024.

Barham, P., Chowdhery, A., Dean, J., Ghemawat, S., Hand, S., Hurt, D., Isard, M., Lim, H., Pang, R., Roy, S., et al. Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems*, 4:430–449, 2022.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2016.

Chen, Y., Yuille, A., and Zhou, Z. Which layer is learning faster? a systematic exploration of layer-wise convergence rate for deep neural networks. In *ICLR*, 2023.

Cheng, J., Gao, N., Yue, Y., Ye, Z., Jiang, J., and Sha, J. Edit: A local-sgd-based efficient distributed training method for large language models. *arXiv preprint arXiv:2412.07210*, 2024.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M. a., Senior, A., Tucker, P., Yang, K., Le, Q., and Ng, A. Large scale distributed deep networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

Diskin, M., Bukhtiyarov, A., Ryabinin, M., Saulnier, L., Sinitsin, A., Popov, D., Pyrkin, D. V., Kashirin, M., Borzunov, A., Villanova del Moral, A., et al. Distributed deep learning in open collaborations. In *NeurIPS*, 2021.

Douillard, A., Feng, Q., Rusu, A. A., Chhaparia, R., Donchev, Y., Kuncoro, A., Ranzato, M., Szlam, A., and Shen, J. Diloco: Distributed low-communication training of language models, 2024. URL https://arxiv.org/abs/2311.08105.

Douillard, A., Donchev, Y., Rush, J. K., Kale, S., Charles, Z., Teston, G., Garrett, Z., Shen, J., McIlroy, R., Lacey, D., Rame, A., Szlam, A., Ranzato, M., and Barham, P. R. Streaming diloco with overlapping communication. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=yYk3zK0X6Q.

Douillard, A., Donchev, Y., Rush, K., Kale, S., Charles, Z., Garrett, Z., Teston, G., Lacey, D., McIlroy, R., Shen, J., Ramé, A., Szlam, A., Ranzato, M., and Barham, P. Streaming diloco with overlapping communication: Towards a distributed free lunch, 2025b. URL https://arxiv.org/abs/2501.18512.

et al, P. K. Advances and open problems in federated learning, 2021.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1), January 2022. ISSN 1532-4435.

Fei, J., Ho, C.-Y., Sahu, A. N., Canini, M., and Sapio, A. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *ACM SIGCOMM*, SIGCOMM '21, pp. 676–691, 2021a.

Fei, J., Ho, C.-Y., Sahu, A. N., Canini, M., and Sapio, A. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *SIGCOMM*, pp. 676–691, 2021b.

Feyzmahdavian, H. R. and Johansson, M. Asynchronous iterations in optimization: New sequence results and sharper algorithmic guarantees. *Journal of Machine Learning Research*, 24(158):1–75, 2023.

Guan, Y., Liu, X., Ren, T., and Niu, J. Enabling communication-efficient federated learning via distributed compressed sensing. In *INFOCOM*, pp. 1–10, 2023.

He, C., Li, S., Soltanolkotabi, M., and Avestimehr, S. Pipetransformer: Automated elastic pipelining for distributed training of large-scale models. In *ICML*, 2021.

He, J., Zhai, J., Antunes, T., Wang, H., Luo, F., Shi, S., and Li, Q. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 120–134, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *NeurIPS*, 32, 2019.

Jaghouar, S., Ong, J. M., Basra, M., Obeid, F., Straube, J., Keiblinger, M., Bakouch, E., Atkins, L., Panahi, M., Goddard, C., Ryabinin, M., and Hagemann, J. Intellect-1 technical report, 2024a. URL https://arxiv.org/abs/2412.01152.

Jaghouar, S., Ong, J. M., and Hagemann, J. Opendiloco: An open-source framework for globally distributed low-communication training, 2024b. URL https://arxiv.org/abs/2407.07852.

Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., and Guo, C. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *OSDI*, 2020.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kuang, W., Qian, B., Li, Z., Chen, D., Gao, D., Pan, X., Xie, Y., Li, Y., Ding, B., and Zhou, J. 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*, KDD '24, pp. 5260–5271, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901.

Lee, S., Zhang, T., and Avestimehr, A. S. Layer-wise adaptive model aggregation for scalable federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8491–8499, 2023.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. In *NeurIPS*, pp. 19–27, 2014.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., and Chintala, S. Pytorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, aug 2020.

Liu, X., Tang, Z., Dong, P., Li, Z., Li, B., Hu, X., and Chu, X. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *arxiv preprint arXiv:2502.00299*, 2025.

Luo, Q., He, J., Zhuo, Y., and Qian, X. Prague: High-performance heterogeneity-aware asynchronous decentralized training. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 401–416, 2020.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *ICLR*, 2017.

Mishchenko, K., Malinovsky, G., Stich, S., and Richtárik, P. Proxskip: Yes! local gradient steps provably lead to communication acceleration! finally! In *ICML*, pp. 15750–15769, 2022.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: Generalized pipeline parallelism for dnn training. In *SOSP*, 2019.

Parekh, M. Ai: Our data exhaust. https://medium.com/@mparekh/ai-our-data-exhaust-e08ba5885af8.

Patel, K. K., Glasgow, M., Zindari, A., Wang, L., Stich, S. U., Cheng, Z., Joshi, N., and Srebro, N. The limits and potentials of local sgd for distributed heterogeneous learning with intermittent communication. In *COLT*, volume 247, pp. 4115–4157, 2024.

Peng, Y., Zhu, Y., Chen, Y., Bao, Y., Yi, B., Lan, C., Wu, C., and Guo, C. A generic communication scheduler for distributed dnn training acceleration. In *SOSP*, pp. 16–29, 2019.

Qin, Z., Chen, D., Qian, B., Ding, B., Li, Y., and Deng, S. Federated full-parameter tuning of billion-sized language models with communication cost under 18 kilobytes. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=cit0hg4sEz.

Qwen, :, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R.,

Li, T., Tang, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_understanding_paper. pdf*, 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. Svcca: singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NeurIPS*, 2017.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, 2019. URL https://api.semanticscholar.org/CorpusID:269617042.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

Ramasinghe, S., Ajanthan, T., Avraham, G., Zuo, Y., and Long, A. Protocol models: Scaling decentralized training with communication-efficient model parallelism, 2025. URL https://arxiv.org/abs/2506.01260.

Reid, M. e. a. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Ryabinin, M. and Gusev, A. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *NeurIPS*, volume 33, pp. 3659–3672. Curran Associates, Inc., 2020.

Ryabinin, M., Dettmers, T., Diskin, M., and Borzunov, A. Swarm parallelism: training large models can be surprisingly communication-efficient. In *ICML*, 2023a.

Ryabinin, M., Dettmers, T., Diskin, M., and Borzunov, A. Swarm parallelism: Training large models can be surprisingly communication-efficient, 2023b. URL https://arxiv.org/abs/2301.11913.

Sadiev, A., Kovalev, D., and Richtárik, P. Communication acceleration of local gradient methods via an accelerated primal-dual algorithm with an inexact prox. *NeurIPS*, 35: 21777–21791, 2022.

Sani, L., Iacob, A., Cao, Z., Lee, R., Marino, B., Gao, Y., Cai, D., Li, Z., Zhao, W., Qiu, X., et al. Photon: Federated llm pre-training. *arXiv preprint arXiv:2411.02908*, 2024.

Sarfi, A., Thérien, B., Lidin, J., and Belilovsky, E. Communication efficient llm pre-training with sparseloco, 2025. URL https://arxiv.org/abs/2508.15706.

Shi, S., Wang, Q., Chu, X., Li, B., Qin, Y., Liu, R., and Zhao, X. Communication-efficient distributed deep learning with merged gradient sparsification on gpus. In *IEEE INFOCOM*, 2020.

Shi, S., Chu, X., and Li, B. Exploiting simultaneous communications to accelerate data parallel distributed deep learning. In *IEEE INFOCOM*, pp. 1–10, 2021.

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

Stich, S. U. Local SGD converges fast and communicates little. In *ICLR*, 2019.

Su, N., Hu, C., Li, B., and Li, B. Titanic: Towards production federated learning with large language models. In *IEEE INFOCOM*, 2024.

Sun, W., Qin, Z., Sun, W., Li, S., Li, D., Shen, X., Qiao, Y., and Zhong, Y. Co2: Efficient distributed training with full communication-computation overlap, 2024. URL https://arxiv.org/abs/2401.16265.

Tang, Z., Shi, S., Chu, X., Wang, W., and Li, B. Communication-efficient distributed deep learning: A comprehensive survey. *arXiv preprint arXiv:2003.06307*, 2020.

Tang, Z., Shi, S., Li, B., and Chu, X. Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication. *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–13, 2022a. doi: 10.1109/TPDS.2022.3230938.

Tang, Z., Zhang, Y., Shi, S., He, X., Han, B., and Chu, X. Virtual homogeneity learning: Defending against data

heterogeneity in federated learning. In *ICML*, volume 162, pp. 21111–21132, 2022b.

Tang, Z., Wang, Y., He, X., Zhang, L., Pan, X., Wang, Q., Zeng, R., Zhao, K., Shi, S., He, B., et al. Fusionai: Decentralized training and deploying llms with massive consumer-level gpus. In *IJCAI Symposium on LLMs*, 2023.

Tang, Z., Huang, J., Yan, R., Wang, Y., Tang, Z., Shi, S., Zhou, A. C., and Chu, X. Bandwidth-aware and overlap-weighted compression for communication-efficient federated learning. In *Proceedings of the 53rd International Conference on Parallel Processing*, pp. 866–875, 2024a.

Tang, Z., Kang, X., Yin, Y., Pan, X., Wang, Y., He, X., Wang, Q., Zeng, R., Zhao, K., Shi, S., Zhou, A. C., Li, B., He, B., and Chu, X. Fusionllm: A decentralized llm training system on geo-distributed gpus with adaptive compression, 2024b. URL https://arxiv.org/abs/2410.12707.

TANG, Z., Tang, Z., Pan, G., Liu, B., He, X., Lai, K., Chu, X., and Li, B. Ghost in the cloud: Your geo-distributed large language models training is easily manipulated. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FwnmQnVc7g.

Team, Q. Qwen1.5-moe: Matching 7b model performance with 1/3 activated parameters. https://qwen.ai/blog?id=qwen-moe, March 2024.

Thakur, R., Rabenseifner, R., and Gropp, W. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, feb 2005. ISSN 1094-3420. doi: 10.1177/1094342005051521.

Thirunavukarasu, A. J., Ting, D. S. J., Elangovan, K., Gutierrez, L., Tan, T. F., and Ting, D. S. W. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. *ArXiv*, 2023a.

Touvron, H. et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Villalobos, P., Sevilla, J., Heim, L., Besiroglu, T., Hobbhahn, M., and Ho, A. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *arXiv e-prints*, pp. arXiv–2211, 2022.

Wang, B., Xu, Q., Bian, Z., and You, Y. Tesseract: Parallelize the tensor parallelism efficiently. In *Proceedings of the 51st International Conference on Parallel Processing*, pp. 1–11, 2022a.

Wang, J. and Joshi, G. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. In *MLsys*, 2019.

Wang, J., Yuan, B., Rimanic, L., He, Y., Dao, T., Chen, B., Ré, C., and Zhang, C. Fine-tuning language models over slow networks using activation quantization with guarantees. In *NeurIPS*, 2022b.

Wang, J., Liang, B., Dong, M., Boudreau, G., and Afana, A. Online distributed optimization with efficient communication via temporal similarity. In *INFOCOM*, pp. 1–10, 2023a.

Wang, J., Lu, Y., Yuan, B., Chen, B., Liang, P., De Sa, C., Re, C., and Zhang, C. CocktailSGD: Fine-tuning foundation models over 500Mbps networks. In *ICML*. PMLR, 2023b.

Woodworth, B. E., Patel, K. K., and Srebro, N. Minibatch vs local sgd for heterogeneous distributed learning. *NeurIPS*, 33:6281–6292, 2020.

Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., and Mann, G. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

Xu, M., Cai, D., Wu, Y., Li, X., and Wang, S. FwdLLM: Efficient federated finetuning of large language models with perturbed inferences. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 579–596, 2024. URL https://www.usenix.org/conference/atc24/presentation/xu-mengwei.

Ye, R., Wang, W., Chai, J., Li, D., Li, Z., Xu, Y., Du, Y., Wang, Y., and Chen, S. Openfedllm: Training large language models on decentralized private data via federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 6137–6147, 2024a.

Ye, S., Du, J., Zeng, L., Ou, W., Chu, X., Lu, Y., and Chen, X. Galaxy: A resource-efficient collaborative edge ai system for in-situ transformer inference. In *INFOCOM*, 2024b.

Ye, S., Zeng, L., Chu, X., Xing, G., and Chen, X. Asteroid: Resource-efficient hybrid pipeline parallelism for collaborative dnn training on heterogeneous edge devices. In *ACM MobiCom*, pp. 312–326, 2024c.

Yuan, B., He, Y., Davis, J., Zhang, T., Dao, T., Chen, B., Liang, P. S., Re, C., and Zhang, C. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 35: 25464–25477, 2022.

Zhang, H., Zheng, Z., Xu, S., Dai, W., Ho, Q., Liang, X., Hu, Z., Wei, J., Xie, P., and Xing, E. P. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *USENIX ATC*, pp. 181–193, Santa Clara, CA, July 2017.

Zhang, H., Wu, T., Cheng, S., and Liu, J. Aperiodic local sgd: Beyond local sgd. In *ICPP*, 2023.

Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E. P., 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)*, pp. 559–578, 2022.

Zhuang, W., Chen, C., and Lyu, L. When foundation model meets federated learning: Motivations, challenges, and future directions. *arXiv preprint arXiv:2306.15546*, 2023.

A ARTIFACT APPENDIX

A.1 Abstract

This appendix provides the necessary materials—including the source code for DreamDDP and baselines, minimum software and hardware requirements, and detailed configurations—to reproduce the speed and performance results presented in this paper. Due to hardware limitations, we do not support full convergence reproduction (which requires massive GPU hours). Instead, the artifacts focus on verifying the core system-level improvements of DreamDDP. Specifically, the artifacts include 2 parts: 1) **Iteration Time Estimation**: utilizing the profiler to measure average computation and communication overheads, thereby estimating the iteration time and scheduling efficiency for all methods (corresponding to Table 2); 2) **End-to-End Speed Verification**: executing the provided scripts to measure the real end-to-end iteration throughput and speedup factors for DreamDDP and all baselines (reflecting the speedup ratios in Table 3). As the original 32-GPU clusters require strict institutional access approvals and cannot be shared publicly, we have provisioned an alternative testbed for artifact evaluation. Specifically, we provide access to a distributed environment consisting of **two NVIDIA RTX 4090 GPUs located on two distinct physical machines** (We will use the computing resources provided by ChameleonCloud if our application is successful). Consequently, this artifact evaluation focuses exclusively on reproducing the system-level speedups (Table 2 in the paper). We omit the full convergence reproduction due to the prohibitive computational cost and extensive GPU hours required. The complete source code is available in our GitHub repository: `https://github.com/trl730109/DreamDDP`.

A.2 Artifact check-list (meta-information)

- **Algorithm:** DreamDDP, naive SGD (S-SGD), Wait-free Based SGD (ASC-WFBP), LSGD (FLSGD), LSGD with partial synchronization (PLSGD)
- **Program:** Python, PyTorch
- **Compilation:** N/A (Python interpreted), CUDA (via PyTorch)
- **Data set:** WikiText-2, CIFAR-10 and CIFAR-100
- **Hardware:** 2 NVIDIA RTX 4090 GPUs (distributed across 2 physical machines). We will use the computing resources provided by ChameleonCloud if our application is successful.
- **Metrics:** Iteration Time, E2E Speedup Factor, Communication-Computation Overlap
- **Output:** Log files, JSON statistics, PyTorch Profiler Traces
- **Experiments:** Profiling results, Real Iteration Time measurement, Overlap visualization
- **How much disk space required (approximately)?:** ∼50 GB (datasets and logs)
- **How much time is needed to prepare workflow (approximately)?:** ∼30 minutes (Environment setup and data prep)
- **How much time is needed to complete experiments (approximately)?:** ∼1 hour to run the whole training pipeline consisting of profiling and training.
- **Publicly available?:** Yes (GitHub)
- **Code licenses (if publicly available)?:** MIT License
- **Workflow framework used?:** Bash scripts

A.3 Description

A.3.1 How delivered

The source code, scheduling algorithms, and execution scripts are publicly available in our GitHub repository: `https://github.com/trl730109/DreamDDP`. Please follow the instructions in the `README.md` file to set up the software environment, execute the quick start pipelines, and customize the training configurations.

A.3.2 Hardware dependencies

For artifact evaluation, we provide a cross-node cluster consisting of two physical machines, each equipped with a single NVIDIA RTX 4090 GPU. This physical separation ensures a genuine network bottleneck. Linux Traffic Control (`tc`) can be further utilized to strictly constrain the inter-node network bandwidth (e.g., 1Gbps or 10Gbps) to reproduce the specific bandwidth conditions in the paper (Refer to the ./set_mul_bandwidth.sh and the ./train_exps/launch_transformer_A6000.sh).

A.3.3 Software dependencies

- OS: Ubuntu 20.04 or later
- Python: 3.8+
- Framework: PyTorch 2.0+ (compiled with CUDA support)
- Libraries: `transformers`, `datasets`, `mpi4py`, `tensorboard`, `torch-tb-profiler`

A.3.4 Datasets and Models

We use the below datasets in our paper:

- **CIFAR-10 / CIFAR-100**: Automatically downloaded via `torchvision.datasets`.
- **WikiText-2**: Automatically downloaded via HuggingFace `datasets` library for LLM (GPT-2, Llama-2, Qwen2.5-7B with LoRA) experiments.

After downloading the datasets, you should revise the path in the config file, located at *./train_exps/env_configs/A6000.sh*

A.4 Installation

Clone the repository and install the required dependencies (assuming access to the provided 4090 environment):

```
git clone https://github.com/trl730109/DreamDDP.git
cd DreamDDP
pip install -r requirements.txt
```

A.5 Setup

A.5.1 SSH Password-Free Login

To enable multi-node distributed training, SSH password-free login must be configured first. The project provides `ssh_conf.sh` to automate this. Before running it, edit the following variables in the script:

- `HOST`: The target host IP address (or the jump server IP if all nodes are reached through it)
- `PORTS`: SSH ports for each node, space-separated, e.g., `(31972 31521)`
- `USER`: SSH username
- `EMAIL`: Email used as a label when generating the SSH key

Then run:

```
bash ssh_conf.sh
```

The script will: (1) generate `~/.ssh/id_rsa` if it does not exist; (2) use `ssh-copy-id` to copy the public key to all ports in `PORTS`, enabling password-free login.

A.5.2 Node Configuration in the Training Script

Configure the IP addresses and ports of training nodes in `train_exps/transformer_pipeline.sh`. The main variables are:

- `hosts`: Array of node IPs, e.g., `('10.244.3.188' '10.244.4.109')`
- `ports`: Array of SSH ports for each node, aligned with `hosts`, e.g., `(22 22)`
- `master_port`: Starting port for the distributed training master process (e.g., 3188)

Example: for a 2-node cluster with node 1 at `10.244.3.188:22` and node 2 at `10.244.4.109:22`, use:

```
hosts=('10.244.3.188' '10.244.4.109')
ports=(22 22)
master_port=3188
```

After these settings are in place, run `transformer_pipeline.sh` to start multi-node distributed training.

A.6 Experiment workflow

The workflow focuses on reproducing the system-level speedup and profiling traces. We provide an integrated script `transformer_pipeline.sh` that handles all phases. You can run with the following command:

```
bash ./train_exps/transformer_pipeline.sh
```

We disengage the 3 steps in our DreamDDP training pipelines below:

1. Profiling: This step runs S-SGD and LocalSGD to collect layer-wise backward and communication times. In `transformer_pipeline.sh`, this is handled by:

```
echo "========== Starting Profile ==========="
profile=True
enable_wandb=false

for dnn in "${dnn_list[@]}"; do
    ...
    # Profile transformer_sgd (full-precision)
    alg='transformer_sgd'
    source train_exps/launch_transformer_A6000.sh
    master_port=$((master_port + 1))

    # Profile transformer_localsgd (low-precision)
    alg='transformer_localsgd'
    source train_exps/launch_transformer_A6000.sh
    master_port=$((master_port + 1))
done
```

2. Scheduling: The DFS scheduling algorithm then uses the profiled data to generate the optimal iteration-layer mapping and saves it to `dreamddp_scheduling.json`. The path to the scheduling results is `./time/.../dreamddp_scheduling.json`. The estimated speedup of DreamDDP over the two baselines, pipe_sgd (**s1** in Table 2) and LocalSGD (**s2** in Table 2) is in `./time/.../scheduling_results.json`.

In `transformer_pipeline.sh`:

```
echo "= Step 2: Scheduling Generation Starts ="
for dnn in "${dnn_list[@]}"; do
    echo "--- Scheduling Generation: ${dnn} ---"
    python3 Scheduling/dreamddp_scheduling.py \
        ${time_base}/${dnn}/${worker_path}/${bandwidth} \
        --H 10 --bp_multiplier ${BP_MULTIPLIER} \
        --comm_multiplier ${COMM_MULTIPLIER}
done
```

3. End-to-End Speed Evaluation: Execute the main script across the two provided nodes to measure the wall-clock iteration time for DreamDDP and the baseline methods. The script automatically computes the resulting speedup. *Important:* To accurately capture end-to-end latency, ensure that either the `profiler_trace` (`torch.profiler`) or `cpu_clock` (`time.time()`) flag is set to `True`. In `transformer_pipeline.sh`:

```
for dnn in "${dnn_list[@]}"; do
    ...
    profiler_trace=False
    cpu_clock=True
    ...
    # Train transformer_pipe_sgd
    alg='transformer_pipe_sgd'
    source train_exps/launch_transformer_A6000.sh
    master_port=$((master_port + 1))

    # Train transformer_localsgd
    alg='transformer_localsgd'
    source train_exps/launch_transformer_A6000.sh
    master_port=$((master_port + 1))

    # Train transformer_dream_ddp
    alg='transformer_dream_ddp'
    source train_exps/launch_transformer_A6000.sh
    master_port=$((master_port + 1))

    # Calculate speedup of dreamddp
    $PY train_exps/speedup_stats.py \
        --time_stamp "$time_stamp" \
        --dnn "$dnn" \
        --nworkers "$nworkers" \
        --bandwidth "$bandwidth"
done
```

A.7 Evaluation and expected result

Following the workflow, evaluators can reproduce the results in Table 2 in our original paper:

- **Estimated average iteration wall-clock time (in seconds) of 1000 running iterations**: We originally profiled the BP and Comm time to estimate the iteration time for DreamDDP and baselines (**SSGD (sgd)**, **ASC-WFBP (pipe-sgd)**, **PLSGD (localsgd)**). The estimation formulas are provided in *./Scheduling/cal_alg_time.py*. Please specially note that Qwen2.5-7B with ASC-WFBP is not supported yet (SSGD, PLSGD, DreamDDP are supported). After scheduling, the estimated speedup of DreamDDP over **ASC-WFBP (s1)** and **PLSGD (s2)** is stored in `./time/.../scheduling_results.json`.
- **End-to-end wallclock times (in seconds)**: We provide the end-to-end speedup of DreamDDP over baselines as supplementary materials to Table 2 to show the effectiveness of DreamDDP. The results are stored in *./results/dnn-time_stamp/speedup.txt*.

A.8 Experiment customization

Users can easily customize the experiments by modifying the following arguments in *./train_exps/transformer_pipeline.sh*:

- `-dnn`: Choose from `resnet18`, `resnet50`, `gpt2`, `llama2-175M`, `Qwen2.5-7B-Instruct`.
- `-nsteps_localsgd` (H): Modify the synchronization period (e.g., 5, 10, 20) to observe how it affects iteration time.
- `-bandwidth`: Adjust the bandwidth constraint argument to simulate varying network topologies.
- `-epoch`: Adjust the bandwidth constraint argument to simulate varying network topologies.

Please specially note that Qwen2.5-7B-Instruct with ASC-WFBP is not supported yet (only SSGD, PLSGD, DreamDDP allowed). We will later add support for LLM LoRA training.

A.9 Notes

Because we are using 2 GPUs instead of the 32 GPUs mentioned in the full paper evaluation, the absolute iteration times will naturally differ from the paper's exact numbers. However, the *relative speedup* (DreamDDP vs. baselines) and the *overlap phenomenon* in the profiler traces will be clearly reproducible and will exhibit the exact same trends discussed in the paper.

APPENDIX

In this appendix, we present the detailed proof of Theorem 1.

Lemma 3. ***Convergence of averaged weights.*** *Let $\{w_r\}_{r\geq 0}$ and $\{\bar{w}_r\}_{r\geq 0}$ for $k \in [K]$ be defined as in Equation (4), and let f be β-smooth and μ-strongly convex and $\eta_r \leq \frac{1}{4\beta}$. Then*

$$\mathbb{E}||\bar{w}_{r+1} - w^*||^2 \leq (1-\mu\eta_r)\mathbb{E}||\bar{w}_r - w^*||^2 + \eta_r^2\mathbb{E}||g_r - \bar{g}_r||^2 - \frac{1}{2}\eta_r\mathbb{E}(f(\bar{w}_r) - f^*) + 2\eta_r\frac{\beta}{K}\sum_{k=1}^{K}\mathbb{E}||\bar{w}_r - w_r^k||^2$$

Proof of Lemma 3. Using the update Eq. 4, we have

$$\begin{aligned}||\bar{w}_{r+1} - w^*||^2 =& ||\bar{w}_r - \eta_r g_r - w^* - \eta_r\bar{g}_r + \eta_r\bar{g}_r||^2 \\ =& ||\bar{w}_r - \eta_r\bar{g}_r - w^*||^2 + \eta_r^2||g_r - \bar{g}_r||^2 \\ &+ 2\eta_r\langle\bar{w}_r - w^* - \eta_r\bar{g}_r, g_r - \bar{g}_r\rangle.\end{aligned} \tag{11}$$

By L-smoothness and μ-strong convexity of f, we can derive the upper bound of the first term as

$$\begin{aligned}||\bar{w}_r - w^* - \eta_r\bar{g}_r||^2 \leq& (1-\mu\eta_r)||\bar{w}_r - w^*||^2 \\ &- \frac{1}{2}\eta_r(f(\bar{w}_r) - f^*) + \frac{2\eta_r L}{M}\sum_{k=1}^{K}||\bar{w}_r - w_r^k||^2.\end{aligned} \tag{12}$$

Taking (12) back into (11), by taking expectation we have

$$\begin{aligned}\mathbb{E}\|\bar{w}_{r+1} - w^*\|^2 \leq& (1-\mu\eta_r)\mathbb{E}\|\bar{w}_r - w^*\|^2 + \eta_r^2\mathbb{E}\|g_r - \bar{g}_r\|^2 \\ &- \tfrac{1}{2}\eta_r\mathbb{E}(f(\bar{w}_r) - f^*) + \tfrac{2L\eta_r}{M}\sum_{k=1}^{K}\mathbb{E}\|\bar{w}_r - w_r^k\|^2\end{aligned} \tag{13}$$

□

Lemma 4. ***Bounded model divergence.*** *Given the bounded synchronization frequency $H_l \leq H$ for $l \in [L]$ and sequence of decreasing positive learning rates $\{\eta_r\}_{r\geq 0}$ satisfying $\eta_r \leq 2\eta_{r+H_l}$ for all $r \geq 0$, then we have*

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}||\bar{w}_r - w_r^k||^2 \leq 4H^2\eta_r^2G^2. \tag{14}$$

Proof of Lemma 4. As w_r^k is the concatenation of $w_r^{k,l}$ with respect to l, we have $||\bar{w}_r - w_r^k||^2 = \sum_{l=1}^{L}||\bar{w}_r^l - w_r^{k,l}||^2$, $\sum_{l=1}^{L}||\nabla^l f(w_h^k;\xi_h^k)||^2 = ||\nabla f(w_h^k;\xi_h^k)||^2$. Using $\mathbb{E}||X - \mathbb{E}X||^2 = \mathbb{E}||X||^2 - ||\mathbb{E}X||^2$ and $||\sum_{i=1}^{H_l}a_i||^2 \leq H_l\sum_{i=1}^{H_l}||a_i||^2$, we have

$$\begin{aligned}\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left\|\bar{w}_r - w_r^k\right\|^2 &= \frac{1}{K}\sum_{k=1}^{K}\sum_{l=1}^{L}\mathbb{E}\left\|w_r^{k,l} - w_{r_0}^{k,l} - (\bar{w}_r^l - w_{r_0}^{k,l})\right\|^2 \\ &\leq \frac{1}{K}\sum_{k=1}^{K}\sum_{l=1}^{L}H_l\eta_{r_0}^2\sum_{h=r_0}^{r-1}\mathbb{E}\left\|\nabla^l f(w_h^k;\xi_h^k)\right\|^2 \\ &\leq \frac{1}{K}\sum_{k=1}^{K}H\eta_{r_0}^2\sum_{h=r_0}^{r-1}\mathbb{E}\left\|\nabla f(w_h^k;\xi_h^k)\right\|^2 \\ &\leq 4H^2\eta_r^2G^2.\end{aligned}$$

□

Lemma 5. ***Martingale convergence.*** *Let $\{a_r\}_{r\geq 0}, a_r \geq 0, \{e_r\}_{r\geq 0}, e_r \geq 0$ be sequences satisfying*

$$a_{r+1} \leq (1-\mu\eta_r)a_r - \eta_r e_r A + \eta_r^2 B + \eta_r^3 C, \tag{15}$$

for $\eta_r = \frac{4}{\mu(a+r)}$ and constants $A > 0, B, C \geq 0, \mu > 0, a > 1$. Then

$$\frac{A}{S_R}\sum_{r=1}^{R-1}p_r e_r \leq \frac{\mu a^3}{4S_R}a_0 + \frac{2R(R+2a)}{\mu S_R}B + \frac{16R}{\mu^2 S_R}C, \tag{16}$$

for $p_r = (a+r)^2$ and $S_R \triangleq \sum_{r=0}^{R-1}p_r = \frac{R}{6}(2R^2 + 6aR - 3R + 6a^2 - 6a + 1) \geq \frac{1}{3}R^3$.

Proof of Theorem 1. Based on Lemma 3 and Lemma 4 we get

$$\begin{aligned}\mathbb{E}||\bar{w}_{r+1} - w^*||^2 \leq& (1-\mu\eta_r)\mathbb{E}||\bar{w}_r - w^*||^2 + \frac{\sigma^2}{K}\eta_r^2 \\ &- \frac{1}{2}\eta_r\mathbb{E}(f(\bar{w}_r) - f^*) + 8G^2H^2\beta\eta_r^3.\end{aligned} \tag{17}$$

By Lemma 5, taking $a_r = \mathbb{E}||\bar{w}_r - w^*||^2, e_r = \mathbb{E}(f(\bar{w}_r) - f^*), A = \frac{1}{2}, B = \frac{\sigma^2}{K}, C = 8G^2H^2\beta$, we have

$$\begin{aligned}\frac{1}{2S_R}\sum_{r=1}^{R}p_r\mathbb{E}(f(\bar{w}_r) - f^*) \leq& \frac{\mu a^3}{4S_R}\mathbb{E}||\bar{w}_0 - w^*||^2 \\ &+ \frac{2R\sigma^2(R+2a)}{\mu K S_R} + \frac{128G^2H^2R}{\mu^2 S_R}.\end{aligned} \tag{18}$$

By the convexity of f, we have

$$\mathbb{E}f(\hat{w}_R) - f^* \leq \frac{1}{S_R}\sum_{r=1}^{R}\mathbb{E}(f(\bar{w}_r) - f^*) \tag{19}$$

By Eq. 18, 19 we get the result

$$\begin{aligned}\mathbb{E}f(\hat{w}_R) - f^* \leq& \frac{\mu a^3}{2S_R}||w_0 - w^*||^2 \\ &+ \frac{4R(R+2a)}{\mu K S_R}\sigma^2 + \frac{256R}{\mu^2 S_R}G^2H^2\beta,\end{aligned} \tag{20}$$

which completes the proof of Theorem 1. □