# ParaBlock: Communication-Computation Parallel Block Coordinate Federated Learning for Large Language Models

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Federated learning (FL) has been extensively studied as a privacy-preserving training paradigm. Recently, federated block coordinate descent scheme has become a popular option in training large-scale models, as it allows clients to train only a subset of the model locally instead of the entire model. However, in the era of large language models (LLMs), even a single block can contain a significant number of parameters, posing substantial communication latency, particularly for resource-constrained clients. To address this challenge in federated training/fine-tuning LLMs, we propose ParaBlock, a novel approach that establishes two parallel threads for communication and computation to enhance communication efficiency. We theoretically prove that the proposed ParaBlock achieves the same convergence rate as the standard federated block coordinate descent methods. Empirical evaluations on fine-tuning LLMs on general instruction following and mathematical reasoning confirm that ParaBlock not only maintains strong performance but also significantly improves communication efficiency.

## 1 Introduction

Federated learning (FL) (McMahan et al., 2017) is known as a promising privacy-preserving solution, as it keeps data on local clients without exposing sensitive information. A typical FL framework consists of a central server and multiple local clients operating in a single-threaded design. In this setup, clients repeatedly train local models, transmit the updates to the server for aggregation, and wait for the updated model from the server to continue training in the next round. While standard FedAvg type frameworks (McMahan et al., 2017; Karimireddy et al., 2020; Li et al., 2020; Wang & Ji, 2023; Wang et al., 2024b) have demonstrated success across various applications (Koutsoubis et al., 2024; Beutel et al., 2020), federated block coordinate descent schemes (Liu et al., 2019; Wu et al., 2021; Liu et al., 2024; Wang et al., 2024a) have recently gained increasing attention. Such schemes integrate local block coordinate descent (BCD) update strategies into FL, allowing clients to train only a subset of the model (commonly a block) locally instead of the entire model. Although federated BCD still operates in a single-thread manner, where clients need to wait for the model transmission to complete before the next computation step, such communication time is often negligible when models are smaller and the block size is limited.

However, with the recent trend of billion-scale large language models (LLMs), such as GPT-3 (Brown et al., 2020), Llama 3 (AI@Meta, 2024) and Gemma 2 (Team et al., 2024)), even a single layer can encompass tens to hundreds of millions of parameters. This massive scale significantly increases the communication time between the server and clients. While such communication time was considered negligible in traditional federated block coordinate methods, it has now become a notable factor when fine-tuning LLMs on edge clients. This raises a critical bottleneck for fine-tuning LLMs with federated block coordinate descent methods: for clients with limited network bandwidth or those requiring long-distance transmission, the communication latency significantly impacts the overall efficiency of FL deployment. This inefficiency of such a single-thread approach hinders the scalability of federated block coordinate descent methods for LLMs, underscoring the need for solutions to reduce communication latency for clients.

To address the aforementioned challenge, we propose ParaBlock, a federated communication-computation **Para**llel **Block** coordinate descent method that provides a simple yet effective way to improve the communication efficiency of BCD updates for fine-tuning large-scale models on resource-constrained clients. The key contributions of this paper are summarized as follows:

- We propose ParaBlock, a novel method designed to address communication latency during the fine-tuning of LLMs using federated block coordinate descent. ParaBlock replaces the traditional single-threaded design with two parallel threads for communication and computation, effectively reducing communication delays and improving efficiency.

- We rigorously show that the proposed ParaBlock achieves a convergence rate of $\mathcal{O}(1/\sqrt{T})$ for non-convex objectives. This rate is consistent with that of standard federated block coordinate descent methods, ensuring that the improvements of ParaBlock in communication efficiency do not compromise its convergence performance.

- We conduct extensive experiments across various models and tasks to empirically validate the effectiveness of ParaBlock. Specifically, we evaluate the downstream performance of ParaBlock on general instruction following and mathematical reasoning tasks. Our results demonstrate that ParaBlock significantly reduces wall-clock runtime while maintaining performance on par with standard federated block coordinate descent baselines, highlighting its ability to enhance efficiency without compromising utility.

## 2 Related Works

**BCD for FL** Block Coordinate Descent (BCD) has been extensively studied for optimization problems (Tseng, 2001; Nesterov, 2012; Beck & Tetruashvili, 2013), particularly due to its efficiency in solving large-scale optimization tasks. Recent advancements in BCD variants, such as those proposed by Luo et al. (2024); Pan et al. (2024), further highlight its adaptability and effectiveness, especially in the context of large language models. Federated BCD schemes (Liu et al., 2019; 2024; Wu et al., 2021) enhances communication efficiency by assigning each client the responsibility for a partition of the model in a collaborative training framework. FedCyBGD (Wang et al., 2024a) focuses on fine-tuning large language models by cyclically elects one client at a time to train its assigned block of the model, enabling efficient fine-tuning with limited computational resources.

**Communication-efficiency for FL** Various methods have been proposed to improve communication efficiency in federated learning, focusing on techniques such as quantization and update compression (Reisizadeh et al., 2020; Haddadpour et al., 2019; Jin et al., 2020; Jhunjhunwala et al., 2021; Wang et al., 2022; Li et al., 2024), model pruning (Li et al., 2021; Jiang et al., 2022; Isik et al., 2022), and model distillation (Wu et al., 2022). Several existing works have demonstrated to enhance communication efficiency specifically in the context of federated fine-tuning of LLMs. For example, many applications of PEFT in FL help reduce communication costs (Liu et al., 2019; 2024). Additionally, FedMKT (Fan et al., 2024) leverages a mutual knowledge transfer framework, where clients and the server exchange knowledge datasets instead of model updates. Similarly, CG-FedLLM (Wu et al., 2024) introduces an encoder on clients to compress gradient features, with a corresponding decoder on the server to reconstruct the transmitted features.

**FL for LLMs** In the era of LLMs, FL has evolved to develop more complex fine-tuning and deployment of large language models (Zhang et al., 2024; Ye et al., 2024). Several existing approaches explore the use of Parameter Efficient Fine-Tuning (PEFT) techniques for federated fine-tuning of LLMs. These include leveraging low-rank adapters (LoRA) in methods such as FedIT (Zhang et al., 2024), the OpenFedLLM framework (Ye et al., 2024), HetLoRA (Cho et al., 2024), SLoRA (Babakniya et al., 2023), FLoRA (Wang et al., 2024d), FlexLoRA (Bai et al., 2024) as well as employing adapters and bias-tuning in FedPEFT (Sun et al., 2022). In addition to the widely adopted PEFT methods in FL, recent work, such as FedCyBGD (Wang et al., 2024a), also highlights the potential of BCD in improving federated fine-tuning performance while managing resource limitations.

## 3 Preliminaries and Motivations

We start with a standard federated learning objective with $N$ total workers:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} f(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\xi_i\sim\mathcal{D}_i}[f_i(\boldsymbol{\theta};\xi_i)], \tag{1}$$

where $\boldsymbol{\theta}\in\mathbb{R}^d$ indicates the model parameter with $d$ dimensions, $f_i(\boldsymbol{\theta})$ represents the loss function associated with client $i$, and $\mathcal{D}_i$ denotes the local data distribution for client $i$. FedAvg (McMahan et al., 2017) is widely used for solving (1). In $t$-th global round, each participating client $i$ performs local training using standard SGD optimizers. The server periodically collects and aggregates the local models or updates to obtain a new global model.

**BCD and Block-wise update for FL** BCD usually has the same optimization objective as (stochastic) gradient descent, it iteratively optimizes over a small subset of parameters, while keeping the remaining parameters fixed. This approach makes BCD practical for large-scale model training. Federated BCD is then applied in FL by substituting the SGD optimizer to BCD in FedAvg. Consider a block partitioning of the model parameters $\boldsymbol{\theta}$ into $B$ blocks, i.e., $\boldsymbol{\theta} = \big[[\boldsymbol{\theta}]_1, [\boldsymbol{\theta}]_2, \ldots, [\boldsymbol{\theta}]_B\big]$. In federated BCD schemes, the goal is to approximately optimize the objective in (1) by updating only one block during local training, while keeping the remaining blocks unchanged across clients. For example, if block $b_t$ is assigned for training at global round $t$, the block-wise optimization for $[\boldsymbol{\theta}]_{b_t}$ is defined as:

$$\underset{[\boldsymbol{\theta}]_{b_t}}{\operatorname{argmin}}\, f(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} f_i([\boldsymbol{\theta}]_1, \ldots, [\boldsymbol{\theta}]_{b_t}, \ldots, [\boldsymbol{\theta}]_B),$$

where $[\boldsymbol{\theta}]_{b_t}\in\mathbb{R}^{d_{b_t}}$ and $d_{b_t}$ denotes the dimension for block $b_t$ and there is $\sum_{b=1}^{B} d_b = d$. We summarize the key component of the local BCD update for federated BCD in Algorithm 1. Specifically, during global round $t$, client $i$ approximates to optimize local objective $f_i$ with the stochastic gradient of $f_i$:

$$\nabla f_i(\boldsymbol{\theta};\xi) = \left[\frac{\partial f_i}{\partial[\boldsymbol{\theta}]_1}, \ldots, \frac{\partial f_i}{\partial[\boldsymbol{\theta}]_{b_t}}, \ldots, \frac{\partial f_i}{\partial[\boldsymbol{\theta}]_B}\right]^T, \tag{2}$$

where the partial stochastic gradient for block $b_t$ is denoted as $[\boldsymbol{g}^i]_{b_t} = [\nabla f_i(\boldsymbol{\theta};\xi)]_{b_t}$. The local model training could be summarized in Line 4 in Algorithm 1. After completing $K$ local steps of training, the client calculates the difference in local updates as shown in Line 6. The client then sends $\boldsymbol{\Delta}^i$ to the server to update the global model, and the server sends back the global model to clients for next round of local training.

**Algorithm 1** `LocalBlockTraining`

**Input:** local learning rate $\eta_l$, global model $\boldsymbol{\theta}$ with $B$ blocks, number of local steps $K$, assigned block $b$

```
1: Initialize model $\boldsymbol{\theta}^i = \boldsymbol{\theta}$ on client $i$
2: for $k = 0$ to $K-1$ do
3:     Compute local partial stochastic gradient $[\boldsymbol{g}_k^i]_b = [\nabla f_i(\boldsymbol{\theta}_k^i;\xi)]_b$
4:     Local update: $\boldsymbol{\theta}_{k+1}^i \leftarrow \boldsymbol{\theta}_k^i$,   $[\boldsymbol{\theta}_{k+1}^i]_b \leftarrow [\boldsymbol{\theta}_k^i]_b - \eta_l[\boldsymbol{g}_k^i]_b$
5: end for
6: Client gets $\boldsymbol{\Delta}^i = [\boldsymbol{\theta}_K^i]_b - [\boldsymbol{\theta}^i]_b$
```

**Output:** $\boldsymbol{\Delta}^i$

**The communication bottleneck for federated BCD** While federated BCD has been applied to large-scale FL training, it still faces subsequent communication inefficiency in the era of LLMs. With the significantly larger model size and the accompanied large communication time, the current single-thread design in the federated BCD method (communication-computation-communication, as shown in Figure 1), has led to increased overall runtime due to the outstanding communication latency. This prolonged communication time forces clients to wait extensively before receiving the latest global information and proceeding to the next round of local computation. Such communication latency poses a critical bottleneck during the deployment of federated BCD, motivating us to explore new methods for improving communication efficiency and the scalability of federated BCD. Drawing inspiration from distributed training paradigms, if clients can change from the single thread design to two parallel threads that overlap the local computation with the model communication, then the communication latency can be ideally eliminated.

## 4 ParaBlock: A new federated fine-tuning method

To achieve this communication computation overlapping in federated BCD methods, we introduce ParaBlock, a **Parallel Block** coordinate descent method designed for fine-tuning LLMs in federated learning. ParaBlock aims to parallelize the communication thread and computation thread on local clients (as shown in Figure 1), enabling clients to perform local updates while concurrently communicating with the server. This design ensures that clients no longer need to wait for communication to finish, significantly reducing clients' idle time and accelerating the federated fine-tuning process. We summarize the proposed ParaBlock in Algorithm 2. In a nutshell, the communication thread and computation thread are proceed *in parallel*. Specifically, for each client $i$, the computation thread fine-tunes block $b_t$ using the `LocalBlockTraining` summarized in Algorithm 1. During this computation, except for the first round when $t = 0$, client $i$ proceeds the communication thread to synchronize with the server regarding the model update variables from the previous round (*one round behind*). This involves sending $\boldsymbol{\Delta}_{t-1}^i$ to the server and receiving the aggregated $\boldsymbol{\Delta}_{t-1}$ from the server. Once both threads are completed, client $i$ immediately updates its local model for the next round of training, as illustrated in Line 9. Note that the new local model $\boldsymbol{\theta}_{t+1}^i$ inherits most of the parameters from the current model $\boldsymbol{\theta}_t^i$ but incorporates two key update steps:

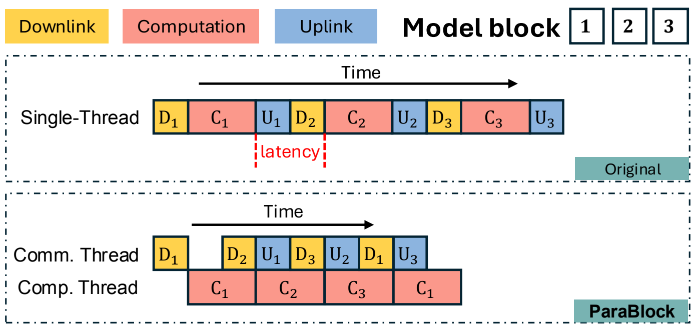


Figure 1: Comparison between the original federated BCD and the proposed ParaBlock. The original BCD's single-thread approach leads to higher runtime due to communication latency, while ParaBlock improves efficiency by overlapping communication and computation.

1. The latest fine-tuned block $b_t$ of the local model is updated with $\boldsymbol{\Delta}_t^i$, ensuring that $[\boldsymbol{\theta}_{t+1}^i]_{b_t}$ is updated with the latest local information before starting the next computation thread.

2. The client uses the received global variable $\boldsymbol{\Delta}_{t-1}$ to correct the previous update on the local block $b_{t-1}$ by applying a correction $[\boldsymbol{\theta}_{t+1}^i]_{b_{t-1}} \leftarrow [\boldsymbol{\theta}_t^i]_{b_{t-1}} + \eta(\boldsymbol{\Delta}_{t-1} - \boldsymbol{\Delta}_{t-1}^i)$. This is because the previous update on block $b_{t-1}$ only used the local update $\boldsymbol{\Delta}_{t-1}^i$ (since the global update $\boldsymbol{\Delta}_{t-1}$ is one round behind and has not arrived yet at that time), and now we want it to be consistent with the global model by replacing the local update with the global one.

After two blocks are updated in $\boldsymbol{\theta}_{t+1}^i$, client $i$ begins the new communication and computation thread for global round $t+1$. Therefore, unlike standard federated BCD methods and most traditional single-thread FL paradigms, the two-thread parallel design of ParaBlock allows client to continue local fine-tuning while conducting communication. Although the global information is updated to the local model with one-round delay, the impact caused by this delay can be effectively mitigated by correction of ParaBlock.

Moreover, while clients update the local model, the server simultaneously updates the global model using the aggregated $\boldsymbol{\Delta}_{t-1}$ (Line 10 in Algorithm 2). Compared with standard federated BCD methods, here the global model $\boldsymbol{\theta}_t$ at round $t$ exhibits one-round staleness because the most recently trained block $b_t$ has not yet been incorporated into the global model. Due to this staleness, an extra communication and aggregation step is required at the end of the fine-tuning (Line 12-14) to ensure that the final computation results are aggregated and updated to the global model, then the server obtains the final model $\boldsymbol{\theta}_{T+1}$.

In the following, we will mathematically demonstrate how the correction in Line 9 addresses the inconsistency in local models and we provide an analysis of the relationship between the global and local models.

**Recursive derivation of local and global models** We begin with round $t-1$, where client $i$ continues fine-tuning using the local model $\boldsymbol{\theta}_t^i$, where $[\boldsymbol{\theta}_t^i]_{b_{t-1}} \leftarrow [\boldsymbol{\theta}_{t-1}^i]_{b_{t-1}} + \eta\boldsymbol{\Delta}_{t-1}^i$. At this stage, the model parameters of block $b_{t-1}$ differ across all clients, as they have not yet finished the synchronizing the latest updates with

**Algorithm 2** ParaBlock

**Input:** local learning rate $\eta_l$, global learning rate $\eta$, number of block partition $B$

```
1:  Initialize global model $\boldsymbol{\theta}_0$ and generate a block partition $b = 1, 2, \ldots, B$
2:  for $t = 0$ to $T - 1$ do
3:      Each client $i$ runs a compute thread and a communication thread in parallel:
4:      Compute thread:
        $\boldsymbol{\Delta}_t^i \leftarrow$ LocalBlockTraining $(\boldsymbol{\theta}_t^i, \eta_l, K, b_t)$
5:      Communication thread:
6:      if $t > 0$ then
7:          Client $i$ send $\boldsymbol{\Delta}_{t-1}^i$ to the server and wait for the aggregated $\boldsymbol{\Delta}_{t-1}$
8:      end if
9:      // When both threads finishes, client $i$ process:
        updating local model: $\boldsymbol{\theta}_{t+1}^i \leftarrow \boldsymbol{\theta}_t^i$, $[\boldsymbol{\theta}_{t+1}^i]_{b_t} \leftarrow [\boldsymbol{\theta}_t^i]_{b_t} + \eta\boldsymbol{\Delta}_t^i$
        if $t > 0$ then
            $[\boldsymbol{\theta}_{t+1}^i]_{b_{t-1}} \leftarrow [\boldsymbol{\theta}_t^i]_{b_{t-1}} + \eta\boldsymbol{\Delta}_{t-1} - \eta\boldsymbol{\Delta}_{t-1}^i$
10:     Server maintains the global model
        $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1}, \quad [\boldsymbol{\theta}_t]_{b_{t-1}} = [\boldsymbol{\theta}_{t-1}]_{b_{t-1}} + \eta\boldsymbol{\Delta}_{t-1}$
11: end for
12: Client $i$ send $\boldsymbol{\Delta}_T^i$ to the server and wait for the aggregated $\boldsymbol{\Delta}_T$
13: Server updates the global model
    $\boldsymbol{\theta}_{T+1} \leftarrow \boldsymbol{\theta}_T, \quad [\boldsymbol{\theta}_{T+1}]_{b_T} = [\boldsymbol{\theta}_T]_{b_T} + \eta\boldsymbol{\Delta}_T$
14: Output $\boldsymbol{\theta}_{T+1}$
```

the server. Then by global round $t$, as the synchronization between the server and clients regarding block $b_{t-1}$ is completed, clients can correct block $b_{t-1}$ in $\boldsymbol{\theta}_{t+1}^i$ using $\boldsymbol{\Delta}_{t-1} - \boldsymbol{\Delta}_{t-1}^i$:

$$\begin{aligned}[\boldsymbol{\theta}_{t+1}^i]_{b_{t-1}} &= [\boldsymbol{\theta}_t^i]_{b_{t-1}} + \eta(\boldsymbol{\Delta}_{t-1} - \boldsymbol{\Delta}_{t-1}^i) = [\boldsymbol{\theta}_{t-1}^i]_{b_{t-1}} + \eta\boldsymbol{\Delta}_{t-1} \\ &= \ldots = [\boldsymbol{\theta}_0^i]_{b_{t-1}} + \eta\sum_{s=0}^{t-1}\mathbb{I}_s(b_{t-1})\boldsymbol{\Delta}_s = [\boldsymbol{\theta}_0]_{b_{t-1}} + \eta\sum_{s=0}^{t-1}\mathbb{I}_s(b_{t-1})\boldsymbol{\Delta}_s,\end{aligned} \tag{3}$$

where $\mathbb{I}_s(b_{t-1})$ is an indicator function that equals to 1 if block $b_{t-1}$ is assigned for round $s$, and 0 otherwise. This recursive derivation shows that the local model for block $b_{t-1}$ can be expressed as the initial global model $\boldsymbol{\theta}_0$ combined with all relevant global updates. Thus, the one-round delayed $\boldsymbol{\Delta}_{t-1}$ corrects the inconsistent parameters in block $b_{t-1}$ of local models. This ensures that the inconsistent parameters in block $b_{t-1}$ are effectively corrected after one global round, making sure that most of the parameters across local models align in a consistent direction for subsequent local training/fine-tuning.

Similarly, for the global model $\boldsymbol{\theta}_t$, there is

$$[\boldsymbol{\theta}_t]_{b_{t-1}} = [\boldsymbol{\theta}_{t-1}]_{b_{t-1}} + \eta\boldsymbol{\Delta}_{t-1} = \ldots = [\boldsymbol{\theta}_0]_{b_{t-1}} + \eta\sum_{s=0}^{t-1}\mathbb{I}_s(b_{t-1})\boldsymbol{\Delta}_s. \tag{4}$$

This indicates that the global model's block $b_{t-1}$ can also be recursively expressed as the initial global model with relevant updates. Based upon the result in Eq. (3) and Eq. (4), the block $b_{t-1}$ of local model $\boldsymbol{\theta}_{t+1}^i$ exactly matches the block $b_{t-1}$ in global model $\boldsymbol{\theta}_t$. This demonstrated the effectiveness of the model correction mechanism in ParaBlock.

## 5 Theoretical Analysis

In this section, we delve into the convergence guarantee of the proposed ParaBlock algorithm. We begin by outlining the key assumptions necessary for the analysis. Following this, we present the convergence rate and provide a detailed discussion of the results.

**Assumption 5.1** (Smoothness)**.** The local objective function $f_i(\boldsymbol{\theta})$ is $L$-smooth, i.e., $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^d$,

$$\|\nabla f_i(\boldsymbol{\theta}_1) - \nabla f_i(\boldsymbol{\theta}_2)\| \leq L\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|.$$

**Assumption 5.2** (Bounded Variance)**.** The stochastic gradient computed on local client is unbiased and has a bounded local variance, i.e., for all $\boldsymbol{\theta}$ and $i \in [N]$, we have $\mathbb{E}\big[\|\nabla f_i(\boldsymbol{\theta}; \xi) - \nabla f_i(\boldsymbol{\theta})\|^2\big] \leq \sigma^2$, and the loss functions has a global variance bound, $\frac{1}{N}\sum_{i=1}^{N} \|\nabla f_i(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta})\|^2 \leq \sigma_g^2$.

Assumption 5.1 and 5.2 are common assumptions in analyzing federated non-convex optimization methods (Li et al., 2019; Yang et al., 2021; Reddi et al., 2021; Wang et al., 2022; Wang & Ji, 2023; Wang et al., 2024c). The global variance upper bound of $\sigma_g^2$ in Assumption 5.2 measures the data heterogeneity across clients, where $\sigma_g^2 = 0$ indicates i.i.d. data distribution across clients.

In the following, we present the theoretical convergence analysis of ParaBlock. For clarity and fair comparison with existing analyses of FedBCD methods, we conduct the analysis under the local SGD optimizer. Extensions to local adaptive optimizers, along with *additional discussions on the connection between general and block-wise properties* are provided in Appendix B.

**Theorem 5.3.** *Under Assumptions 5.1–5.2, let $T$ represent the total number of global rounds, $K$ be the number of local SGD training steps and $N$ be the number of the clients. If the learning rate $\eta$ and $\eta_l$ satisfy $\eta_l \leq \frac{1}{22KL}$ and $\eta\eta_l \leq \frac{1}{4KL}$ , then the global iterates $\{\boldsymbol{\theta}_t\}_{t=1}^T$ of Algorithm 2 satisfy*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[\|\nabla_{b_t} f(\boldsymbol{\theta}_t)\|^2] \leq \frac{8\mathcal{F}}{\eta\eta_l TK} + 40\eta_l^2 L^2 K(\sigma^2 + 6K\sigma_g^2)$$
$$+ \left(8\eta^2\eta_l L^2 K + \frac{\eta L}{2}\right)\frac{\eta_l}{N}\sigma^2 + 64\eta^2\eta_l^2 L^2 K[\sigma^2 + 10\eta_l^2 L^2 K(\sigma^2 + K\sigma_g^2)], \tag{5}$$

*where $\mathcal{F} = f(\boldsymbol{\theta}_1) - f_*$ and $f_* = \min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) > -\infty$.*

**Corollary 5.4.** *If we choose the global learning rate $\eta = \Theta(\sqrt{KN})$ and $\eta_l = \Theta\big(\frac{1}{\sqrt{T}K}\big)$ in Theorem 5.3, then for sufficiently large $T$, the global iterates $\{\boldsymbol{\theta}_t\}_{t=1}^T$ of Algorithm 2 satisfy*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[\|\nabla_{b_t} f(\boldsymbol{\theta}_t)\|^2] \leq \mathcal{O}\bigg(\frac{\mathcal{F} + \sigma^2}{\sqrt{TKN}} + \frac{N\sigma^2 + \sigma_g^2}{T}\bigg). \tag{6}$$

*Remark* 5.5. Compared with the existing federated BCD methods such as FedBCD (Liu et al., 2019; Wu et al., 2021) and FedBCGD (Liu et al., 2024), our proposed method obtains the same $\mathcal{O}(1/\sqrt{T})$ convergence rate under general non-convex settings. This demonstrates that the one round staleness introduced by the communication-computation parallel thread in ParaBlock does not compromise its overall convergence guarantee.

## 6 Experiments

In general, the proposed method leverages local block updates combined with a communication-computation parallel scheme during fine-tuning. This section presents a series of experiments to evaluate both the performance and efficiency of our approach. First, we assess the performance of ParaBlock and compare it against several existing federated fine-tuning algorithms. Next, we analyze the time efficiency of ParaBlock under varying network conditions and computational demands. Finally, we perform ablation studies to gain deeper insights into the key factors influencing the effectiveness of ParaBlock.

### 6.1 Experimental settings

We summarize some crucial implementation details in the following, and we leave some additional results and experiment details to Appendix A.

**Datasets, models, and evaluations** We utilize the Alpaca-GPT4 dataset (Peng et al., 2023) for general instruction following tasks, and we use 50000 data samples sampling from MathInstruct dataset (Yue et al., 2023) for mathematical reasoning task. We fine-tune two models, Llama 3-8B (Dubey et al., 2024; AI@Meta, 2024) and an lightweight Llama 3.2-3B (AI@Meta, 2024) designed for on-device uses. To assess the performance when fine-tuning on instruction following task, we utilize MT-Bench (Zheng et al., 2023) with GPT-4o as a judge model. For evaluating the performance on mathematical reasoning, we employ the widely used OpenLLM Leaderboard (Beeching et al., 2023) as the evaluation benchmark and we report the evaluation score on GSM8K (Cobbe et al., 2021) to show the math problem solving capability.

Table 1: Fine tune Llama 3-8B and Llama 3.2-3B (AI@Meta, 2024) on Alpaca-GPT4 dataset (Peng et al., 2023) and MathInstruct dataset (Yue et al., 2023). In our results, we highlight the best score in **bold** and the second-best score with an <u>underline</u>.

| Method | Alpaca-GPT4 | | | | Math Instruct | | | |
|---|---|---|---|---|---|---|---|---|
| | Llama 3-8B | | Llama 3.2-3B | | Llama 3-8B | | Llama 3.2-3B | |
| | MT-Bench↑ | RT(m) ↓ | MT-Bench↑ | RT(m) ↓ | GSM8K↑ | RT(m) ↓ | GSM8K↑ | RT(m) ↓ |
| Base | 4.72 | - | 4.18 | - | 51.55 | - | 27.98 | - |
| Full FT | **5.33** | **16.0** | <u>4.36</u> | <u>12.5</u> | <u>55.80</u> | **15.5** | **32.22** | <u>12.3</u> |
| FedIT | 5.11 | 34.5 | 4.31 | 23.8 | 54.60 | 23.1 | 29.87 | 15.4 |
| FFA-LoRA | 5.08 | 30.2 | 4.25 | 23.3 | 54.59 | 21.2 | 30.55 | 14.9 |
| FLoRA | 4.95 | 65.4 | 4.23 | 34.0 | 52.54 | 64.9 | 28.51 | 29.3 |
| FedCyBGD | 4.87 | 59.9 | 4.29 | 57.8 | 51.40 | 63.2 | 27.60 | 53.3 |
| FedBCD | <u>5.14</u> | 30.2 | 4.33 | 19.1 | 54.74 | 24.9 | <u>31.84</u> | 17.3 |
| ParaBlock | <u>5.14</u> | <u>21.1</u> | **4.40** | **11.9** | **55.88** | <u>15.8</u> | 31.77 | **10.1** |

**Baselines** We compare the ParaBlock with several federated fine-tuning baselines including 1) Full model fine-tuning (Full FT), 2) FedIT (Zhang et al., 2024), which is one of the most commonly used federated fine-tuning methods that integrates LoRA (Hu et al., 2021) into standard FedAvg (McMahan et al., 2017) method, 3) FFA-LoRA (Sun et al., 2024), which fixes the LoRA matrix $\mathbf{B}$ and fine-tunes the LoRA matrix $\mathbf{A}$ to reduce server aggregation bias, 4) FLoRA (Wang et al., 2024d), a recent method for federated fine-tuning with LoRA, 5) FedCyBGD (Wang et al., 2024a), which employs the cyclic update for block coordinate federated fine-tuning, and 6) FedBCD, a standard federated BCD scheme that directly integrates local BCD updates into the original FedAvg aggreagation schemes. We provide the GPU consumption of all baselines in Table 8 in Appendix A.

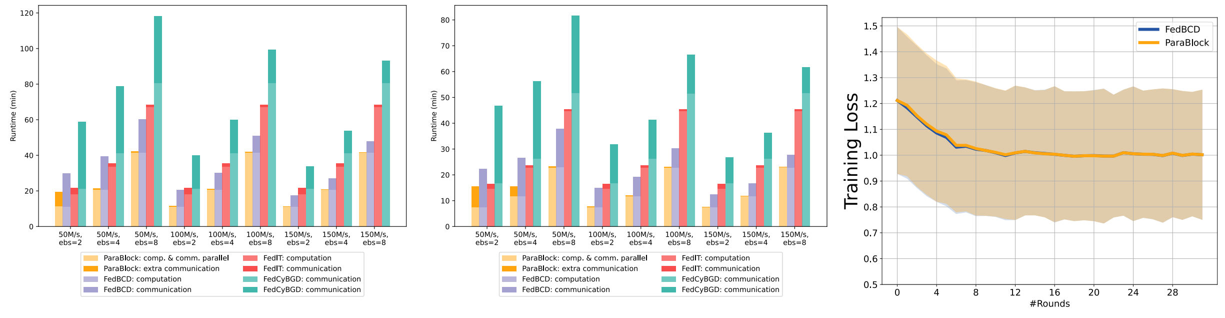
**Implementation details** We implement federated fine-tuning of LLMs by setting up an FL framework with 10 clients, each assigned a local dataset. Notably, we perform heterogeneous data partitioning for both datasets. For the Alpaca-GPT4 dataset, we adopt a text clustering method similar to the one used in Lin et al. (2021) to get a cluster label. For the MathInstruct dataset, we use the "source" from the original dataset as a label and follow traditional data partitioning using a Dirichlet distribution as described in Wang et al. (2020b;a). Specifically, we adopt Dirichlet(0.1) for the Alpaca-GPT4 dataset and Dirichlet(0.6) for the MathInstruct dataset. Regarding LoRA-related methods, the LoRA rank is set to 32 for FedIT, FFA-LoRA and FLoRA. For FedCyBGD, the model is partitioned into 10 blocks, with each block assigned to a corresponding client, and clients perform fine-tuning sequentially. For FedBCD and the proposed ParaBlock, the model is partitioned into 16 blocks for Llama 3-8B and 14 blocks for Llama 3.2-3B. The block partition is based on the default layer of the language model. During each global round, the server randomly selects one block for fine-tuning. We conduct 32 global rounds for fine-tuning Llama 3-8B and 28 rounds for fine-tuning Llama 3.2-3B for all BCD-based and LoRA-based baselines, and we conduct 3 global rounds for Full FT. We adopt AdamW as the local optimizer, i.e., conducting local block training via AdamW, as it is the default optimizer for most of LLMs training and fine-tuning. The default effective batch size in our experiment is set to 4. All experiments are conducted on NVIDIA A100 GPUs.

## 6.2 Main results

**Results on general instruction following task** We begin by evaluating the performance of the proposed ParaBlock on the general instruction-following dataset, Alpaca GPT-4 (Peng et al., 2023), for language model fine-tuning. We report the MT-bench score for evaluation. As shown in the middle main columns of Table 1, ParaBlock achieves lower score than Full FT but consistently outperforms most LoRA-based and BCD-based PEFT methods across two models. For the Llama 3-8B model, ParaBlock achieves an MT-bench score of 5.14, which is the same score as the FedBCD, and ParaBlock shows substantial improvement over FedCyBGD, another federated block-coordinate method. Additionally, ParaBlock significantly outperforms LoRA-based FL methods such as FedIT, FFA-LoRA and FLoRA. A key highlight of ParaBlock is its *superior runtime efficiency*, as it requires outstanding less runtime compared to other baselines, particularly LoRA-based FL methods, while achieving notable performance improvements. Similarly, when fine-tuning the lightweight Llama 3.2-3B model, ParaBlock achieves the best performance among all baselines. Its ability to surpass competing methods in both performance and time efficiency highlights the effectiveness of ParaBlock in fine-tuning LLMs for general instruction-following tasks.

**Results on mathematical reasoning task** We evaluate ParaBlock on mathematical reasoning tasks using the MathInstruct (Yue et al., 2023) dataset, with GSM8K (Cobbe et al., 2021) as the benchmark for evaluation. As shown in the right main columns of Table 1, ParaBlock outperforms all baseline methods with achieves the second-best runtime on the 8B model, and outperforms most baselines with the less runtime on the 3B model. For 8B model, ParaBlock demonstrate improvements over FedBCD, while for 3B model, it shows slighly less accurate than FedBCD. This indicates that while the communication thread of ParaBlock exists one round behind, it does not compromise overall performance. ParaBlock also consistently exhibits strong performance compared to other baselines, with a notable 11.7% improvement over FedCyBGD when fine-tuning the lightweight 3B model. Furthermore, ParaBlock keeps its advantage of runtime saving among all baselines, highlighting its ability to reduce communication latency for federated BCD schemes and achieve communication efficiency in fine-tuning LLMs.

**Time efficiency** We conduct a detailed comparison of the runtime and emphasize the time efficiency benefits achieved by leveraging communication-computation parallelism, as illustrated in Figure 2. To evaluate the necessity and benefits of this scheme, we measure wall-clock time across different network bandwidth conditions (50M/s, 100M/s, and 150M/s) and varying effective batch sizes (2, 4, and 8) on each client. Among the LoRA-based federated fine-tuning methods, FFA-LoRA requires slightly less runtime than FedIT, while FLoRA incurs significantly more runtime than FedIT. Due to space limitations, we include only the detailed runtime comparison results for FedIT here, with a comprehensive discussion of all methods provided in Appendix A.



(a) Llama 3-8B (b) Llama 3.2-3B (c) Training loss comparison

Figure 2: Time efficiency: wall-clock runtime for various network communication bandwidths and effective batch sizes.

Figure 2(a) illustrates the runtime when fine-tuning Llama 3-8B model using Alpaca GPT-4 dataset under various conditions. In scenarios with low network bandwidth and minimal client-side computation, such as a bandwidth of 50M/s and an effective batch size of 2, the communication time cannot be fully overlapped by client computation. This results a significantly extra communication overhead for ParaBlock, represented by the dark yellow portion of the histogram in Figure 2(a). As the effective batch size grows, requiring more client-side computation, a greater portion of the computation time can be overlapped with communication. We notice that for faster communication networks (100M/s and 150M/s), the computation time for ParaBlock does not introduce extra communication during fine-tuning. The negligible dark yellow portion in Figure 2(a) represents the final communication process, as described in Line 12 of Algorithm 2.

Figure 2(a) also demonstrates that ParaBlock significantly reduces runtime compared to other baselines. Notably, we emphasize the time savings achieved over block-coordinate baselines, including vanilla FedBCD and FedCyBGD. The overall runtime reduction can exceed 30% for network bandwidths of 50M/s and 100M/s with an effective batch size (ebs) of 2, with efficiency gains over FedCyBGD being particularly pronounced. It is important to note that the higher runtime costs for FedCyBGD can be attributed to several factors. First, FedCyBGD employs a cyclic update approach that selects only one client at a time, inherently prolonging the fine-tuning process. Additionally, FedCyBGD introduces extra communication overhead, as all clients must periodically synchronize to receive model updates. Thus FedCyBGD's design results in increased latency and contributes to the overall longer runtime observed in our evaluations.

The runtime savings over the LoRA-based FedIT are also notable. Figure 2(a) indicates that ParaBlock achieves comparable, and slightly better, time efficiency than FedIT under conditions of low computational cost and high communication overhead (e.g., 50M/s network and ebs=2). This is primarily due to the additional communication time associated with ParaBlock. Moreover, under most other settings, ParaBlock consistently reduces runtime by approximately 40% compared to FedIT. This improvement is attributed to the communication-computation parallelism of ParaBlock and the computational efficiency of the block coordinate method.

Figure 2(b) illustrates the fine-tuning runtime of the Llama 3.2-3B model under conditions similar to those previously described. In addition to highlighting the superior time efficiency of ParaBlock, we observe that parameter communication constitutes a larger proportion of the total runtime during the fine-tuning of the Llama 3.2-3B model. This is reflected in the increased size of the dark yellow portion in Figure 2(b), which represents a greater share of the overall runtime. These underscore the critical need to reduce communication latency for clients. By mitigating the impact of increased communication overhead, ParaBlock achieves even greater time savings. These results further demonstrate the effectiveness of ParaBlock in enhancing fine-tuning efficiency by minimizing the influence of communication delays.

We verify the convergence of the proposed ParaBlock through the training loss as shown in Figure 2(c). From an optimization perspective, ParaBlock and FedBCD exhibit very similar convergence behavior, with ParaBlock showing only a marginally slower convergence than FedBCD in the initial stages, while this gap diminishes significantly in later rounds. Note that the primary distinction between FedBCD and ParaBlock lies in the global model of ParaBlock, which introduces one-step staleness. Nevertheless, the results from Figure 2(c) indicate that this staleness has a minimal impact on the fine-tuning convergence of ParaBlock.

## 6.3 Ablation studies

We analyze several aspects of the proposed ParaBlock, including: 1) How many blocks should ParaBlock use to balance utility and time efficiency? 2) Is there any block scheduling strategy can achieve better fine-tuning results? 3) How does the data distribution among clients impact the fine-tuning performance? We further study staleness beyond one round, extend the analysis beyond LLaMA architectures, and evaluate under cross-silo partial participation. Additional results are provided in Appendix A.

Table 2: Ablation for the block assignment the number of layers.

| Models | MT-B↑ | RT(m)↓ | GSM8K↑ | RT(m)↓ |
|---|---|---|---|---|
| Partial layer | 5.12 | **20.3** | 53.22 | **15.0** |
| 1 layer | 5.08 | <u>20.4</u> | 53.90 | <u>15.1</u> |
| 2 layers | **5.14** | 21.1 | **55.88** | 15.8 |
| 4 layers | <u>5.13</u> | 22.7 | <u>55.04</u> | 19.4 |

**Ablation for block partition** In our experiments, as previously mentioned, we partition blocks based on the default layers in language models. To investigate the impact of block partitioning, we explore different configurations by varying the number of layers within each block in the proposed ParaBlock approach. As shown in Table 2, we evaluate configurations where each block contains partial, 1, 2, or 4 layers during fine-tuning of the Llama 3-8B model on both general and math tasks. The results in Table 2 indicate that the runtime gets longer as there are more layers assigned to one block. Note that the increase in runtime encompasses the growth of both computation and communication runtime, and we notice that assigning 2 layers per block yields slightly better performance compared to other configurations in both tasks.

**Ablation for block scheduling** We investigate how various block partition would impact on the overall performance. We compare the default random scheduling which randomly select two layers based on the default layers in Llama 3-8B, the sequentially, reverse sequentially layer scheduling and a gradient-guided scheduling based on the original gradient. As shown in Table 3, we found that Random scheduling can achieve best result in math reasoning but with a little bit worse than gradient-based in general instruction tuning. However, the gradient-based approach incurs extra computational cost, as it requires a full-model backward pass prior to fine-tuning in order to determine the scheduling.

Table 3: Ablation for block scheduling

| Models | MT-B↑ | GSM8K↑ |
|---|---|---|
| Random | 5.14 | **55.88** |
| Seq. | 5.03 | 54.21 |
| Rev. seq. | 5.06 | 54.59 |
| Gradient based | **5.19** | 53.60 |

**Ablation for heterogeneous distribution on clients** We compare the i.i.d. and non-i.i.d. data partitions when fine-tuning on the general instruction dataset Alpaca-GPT4 (Peng et al., 2023) and math reasoning dataset Math Instruct (Yue et al., 2023), as shown in Table 4. We further conduct experiments under an extreme non-i.i.d. setting with Dirichlet(0.01) applied to both datasets. Our observations indicate that i.i.d. data sampling consistently results in higher evaluation scores for both general instruction tuning and mathematical reasoning tasks.

Table 4: The results for fine-tuning LLaMA 3-8B on two datasets, considering i.i.d., non-i.i.d. and extreme non-i.i.d. partitions.

| Data distribution | MT-B↑ | GSM8K↑ |
|---|---|---|
| i.i.d. | **5.21** | **56.14** |
| Non-i.i.d. | 5.14 | 55.88 |
| Extreme non-i.i.d. | 5.13 | 54.66 |

**Ablation for the number of staleness rounds** The one-round staleness in the original ParaBlock can be readily extended to multi-round staleness settings. For example, if we assume that all clients in the network experience lower communication bandwidth, leading to communication-computation parallelism with two rounds of staleness, we obtain the following results as shown in Table. It shows a slight performance degradation when all clients incur two-round staleness; however, the performance remains comparable to the base model score.

**Experiments beyond Llama model architectures** We conduct mathematical reasoning experiments using the same MathInstruct dataset as in the main experiments with the Qwen-2.5-1.5B-Instruct model (Team, 2024), and evaluate the fine-tuned models on GSM8K. The results in Table 6 demonstrate that ParaBlock consistently outperforms other baselines in terms of accuracy, while also achieving the second-best runtime, highlighting the effectiveness of our design.

Table 5: Ablation for the number of staleness rounds.

| Method | GSM8K↑ | RT(m)↓ |
|---|---|---|
| Base | 51.55 | - |
| one-round staleness | **55.88** | **15.8** |
| two-round staleness | 54.74 | 16.1 |

Table 6: Fine tune Qwen-2.5-1.5B-Instruct model on MathInstruct dataset. We highlight the best score in **bold** and the second-best score with an <u>underline</u>.

| Method | GSM8K↑ | RT(m)↓ |
|---|---|---|
| Base | 54.28 | – |
| Full FT | 60.58 | **9.3** |
| FedIT | <u>62.32</u> | 19.76 |
| FFA-LoRA | 61.56 | 17.12 |
| FLoRA | 57.77 | 52.42 |
| FedCyBGD | 54.51 | 42.01 |
| FedBCD | 61.79 | 18.48 |
| ParaBlock | **62.70** | <u>10.56</u> |

Table 7: Ablation for the number of staleness rounds.

| Method | MT-B ↑ | RT(m)↓ | GSM8K↑ | RT(m)↓ |
|---|---|---|---|---|
| FedIT | 5.06 | 49.31 | 53.75 | 44.15 |
| FedBCD | 5.08 | 55.36 | 53.60 | 52.16 |
| ParaBlock | **5.10** | **28.45** | **53.90** | **28.44** |

**Extension to cross-silo partial participation settings** ParaBlock is fully compatible with partial participation, and we have conducted experiments demonstrating its effectiveness under such settings. Specifically, we consider a cross-silo setup where 20% of 50 clients are randomly selected in each round, and we fine-tune the Llama-3 8B model on Alpaca-GPT4 and Math Instruct dataset. We compare this cross-silo ParaBlock with FedBCD and FedIT. As shown in Table 7, ParaBlock outperforms FedIT and FedBCD while retaining its computation time-saving advantage.

## 7 Conclusion

In this paper, we propose ParaBlock, a communication-computation parallel block coordinate methods for federated BCD to enhance communication efficiency. To better support this two thread parallel method, we adjust the local model initialization and global model update based on the FL-BCD schemes. We theoretically show the convergence rate for the proposed ParaBlock under general non-convex settings, which indicates our design does not sacrifice the convergence of the standard FL-BCD. We perform extensive experiments on diverse models and tasks to empirically assess the effectiveness of ParaBlock. The results show that ParaBlock significantly improves communication efficiency while achieving performance comparable to standard FL-BCD baselines, highlighting its effectiveness in enhancing efficiency in FL deployments.

## References

AI@Meta. Llama 3 model card. 2024. URL `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

Sara Babakniya, Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Qingfeng Liu, Kee-Bong Song, Mostafa El-Khamy, and Salman Avestimehr. Slora: Federated parameter efficient fine-tuning of language models. *arXiv preprint arXiv:2308.06522*, 2023.

Jiamu Bai, Daoyuan Chen, Bingchen Qian, Liuyi Yao, and Yaliang Li. Federated fine-tuning of large language models under heterogeneous language tasks and client resources. *arXiv e-prints*, pp. arXiv–2402, 2024.

Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard, 2023.

Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Yae Jee Cho, Luyang Liu, Zheng Xu, Aldi Fahrezi, and Gauri Joshi. Heterogeneous low-rank approximation for federated fine-tuning of on-device foundation models. *arXiv preprint arXiv:2401.06432*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Tao Fan, Guoqiang Ma, Yan Kang, Hanlin Gu, Yuanfeng Song, Lixin Fan, Kai Chen, and Qiang Yang. Fedmkt: Federated mutual knowledge transfer for large and small language models. *arXiv preprint arXiv:2406.02224*, 2024.

Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck Cadambe. Local SGD with periodic averaging: Tighter analysis and adaptive synchronization. *Advances in Neural Information Processing Systems*, 32, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Berivan Isik, Francesco Pase, Deniz Gunduz, Tsachy Weissman, and Michele Zorzi. Sparse random networks for communication-efficient federated learning. *arXiv preprint arXiv:2209.15328*, 2022.

Divyansh Jhunjhunwala, Advait Gadhikar, Gauri Joshi, and Yonina C Eldar. Adaptive quantization of model updates for communication-efficient federated learning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3110–3114. IEEE, 2021.

Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):10374–10386, 2022.

Richeng Jin, Yufan Huang, Xiaofan He, Huaiyu Dai, and Tianfu Wu. Stochastic-sign SGD for federated learning with theoretical guarantees. *arXiv preprint arXiv:2002.10940*, 2020.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pp. 5132–5143. PMLR, 2020.

Nikolas Koutsoubis, Yasin Yilmaz, Ravi P Ramachandran, Matthew Schabath, and Ghulam Rasool. Privacy preserving federated learning in medical imaging with uncertainty estimation. *arXiv preprint arXiv:2406.12815*, 2024.

Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pp. 42–55, 2021.

Shiwei Li, Wenchao Xu, Haozhao Wang, Xing Tang, Yining Qi, Shijie Xu, Weihong Luo, Yuhua Li, Xiuqiang He, and Ruixuan Li. Fedbat: Communication-efficient federated learning via learnable binarization. *arXiv preprint arXiv:2408.03215*, 2024.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

Xiang Li, Wenhao Yang, Shusen Wang, and Zhihua Zhang. Communication-efficient local decentralized SGD methods. *arXiv preprint arXiv:1910.09126*, 2019.

Bill Yuchen Lin, Chaoyang He, Zihang Zeng, Hulin Wang, Yufen Huang, Christophe Dupuy, Rahul Gupta, Mahdi Soltanolkotabi, Xiang Ren, and Salman Avestimehr. Fednlp: Benchmarking federated learning methods for natural language processing tasks. *arXiv preprint arXiv:2104.08815*, 2021.

Junkang Liu, Fanhua Shang, Yuanyuan Liu, Hongying Liu, Yuangang Li, and YunXiang Gong. Fedbcgd: Communication-efficient accelerated block coordinate gradient descent for federated learning. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 2955–2963, 2024.

Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. A communication efficient collaborative learning framework for distributed features. *arXiv preprint arXiv:1912.11187*, 2019.

Qijun Luo, Hengxu Yu, and Xiao Li. Badam: A memory efficient full parameter training method for large language models. *arXiv preprint arXiv:2404.02827*, 2024.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *arXiv preprint arXiv:2403.17919*, 2024.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

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

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031. PMLR, 2020.

Guangyu Sun, Umar Khalid, Matias Mendieta, Taojiannan Yang, Pu Wang, Minwoo Lee, and Chen Chen. Conquering the communication constraints to enable large pre-trained models in federated learning. *arXiv preprint arXiv:2210.01708*, 2022.

Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. Improving lora in privacy-preserving federated learning. *arXiv preprint arXiv:2403.12313*, 2024.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109:475–494, 2001.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020a. URL `https://openreview.net/forum?id=BkluqlSFDS`.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*, 2020b.

Lin Wang, Zhichao Wang, and Xiaoying Tang. Save it all: Enabling full parameter tuning for federated large language models via cycle black gradient descent. *arXiv preprint arXiv:2406.11187*, 2024a.

Shiqiang Wang and Mingyue Ji. A lightweight method for tackling unknown participation probabilities in federated averaging. *arXiv preprint arXiv:2306.03401*, 2023.

Yujia Wang, Lu Lin, and Jinghui Chen. Communication-efficient adaptive federated learning. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 22802–22838. PMLR, 2022.

Yujia Wang, Yuanpu Cao, Jingcheng Wu, Ruoyu Chen, and Jinghui Chen. Tackling the data heterogeneity in asynchronous federated learning with cached update calibration. In *The Twelfth International Conference on Learning Representations*, 2024b. URL `https://openreview.net/forum?id=4aywmeb97I`.

Yujia Wang, Shiqiang Wang, Songtao Lu, and Jinghui Chen. Fadas: Towards federated adaptive asynchronous optimization. *arXiv preprint arXiv:2407.18365*, 2024c.

Ziyao Wang, Zheyu Shen, Yexiao He, Guoheng Sun, Hongyi Wang, Lingjuan Lyu, and Ang Li. Flora: Federated fine-tuning large language models with heterogeneous low-rank adaptations. *arXiv preprint arXiv:2409.05976*, 2024d.

Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Communication-efficient federated learning via knowledge distillation. *Nature communications*, 13(1):2032, 2022.

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

Ruiyuan Wu, Anna Scaglione, Hoi-To Wai, Nurullah Karakoc, Kari Hreinsson, and Wing-Kin Ma. Federated block coordinate descent scheme for learning global and personalized models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10355–10362, 2021.

Haibo Yang, Minghong Fang, and Jia Liu. Achieving linear speedup with partial worker participation in non-IID federated learning. In *International Conference on Learning Representations*, 2021.

Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. 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*, pp. 6137–6147, 2024.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Tong Yu, Guoyin Wang, and Yiran Chen. Towards building the federatedgpt: Federated instruction tuning. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6915–6919. IEEE, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.