# Communication-Efficient Federated Deep Learning via Dynamic Averaging

MSc Thesis


**Author:**

Michail Theologitis

**Advisor:**

Assoc. Prof. Vasilis Samoladas

**Committee:**

Assoc. Prof. Vasilis Samoladas
Prof. Antonios Deligiannakis
Asst. Prof. Nikos Giatrakos



School of Electrical and Computer Engineering
Technical University of Crete


Greece, 2025

# Abstract

The ever-growing volume and decentralized nature of data have led to the extensive use of distributed deep learning (DDL) and Federated Learning (FL), both of which struggle with the high cost of transmitting large models. State-of-the-art techniques typically prescribe rigid communication intervals in arbitrary and non-principled ways. To make matters worse, modern language and vision models are rapidly increasing in size. These limitations call for a more principled, adaptive approach to synchronization. To address this, we propose Federated Dynamic Averaging (FDA), a communication-efficient strategy that dynamically triggers synchronization based on real-time training dynamics by monitoring model variance. Our experiments with well-established vision models and tasks show that FDA significantly reduces communication costs while maintaining robust performance across diverse heterogeneity settings. Building on these insights, we also introduce the FDA-Opt family of algorithms—a unified generalization of both FDA and the widely used FedOpt—designed to work out of the box without any calibration. Our experiments focus on fine-tuning pre-trained Language Models (LMs) to downstream NLP tasks and demonstrate that FDA-Opt consistently outperforms FedOpt, even when configured with hyper-parameters optimized for the latter. These results establish FDA-Opt as a practical, drop-in replacement for FedOpt in modern FL libraries and systems.

# Acknowledgements

First, I want to thank my advisor, Prof. Samoladas, for guiding and supporting me these past years. He has always been by my side, open to discussing any concerns or ideas I might have. The lab is across from his office, and—from day one—he has made me feel super comfortable simply knocking on his door if I wanted to talk. No matter how busy he is, he has always found at least a little time for me. This has been a constant source of inspiration for me. Most importantly, through our countless discussions, his way of thinking—especially his approach to complex problems—has shaped the way I think.

I also want to thank Prof. Deligiannakis for the support and the many opportunities he has given me. I have always felt that I could go to him with any problem, academic or otherwise, and that he would be there to help. Furthermore, through his constructive criticism, he has taught me how to write research papers that actually convince the readers.

Moreover, I'm super grateful I had the opportunity to attend Prof. Liavas' world-class courses on "Optimization". What I learned in his two classes has equipped me with all the tools I need to follow any ML literature I want—no matter how math-heavy.

Finally, I want to thank the ECE School as a whole, since everyone has been very supportive of me. Throughout the years, I've faced many hurdles—some due to my own mistakes and some not—and I can't remember a single time someone turned me away. Both professors and staff have always gone out of their way to help me. I feel very blessed to have been part of this environment.

# Contents

# 1. Introduction

In today's data-driven landscape, leveraging data effectively has become both an opportunity and a challenge. Sensitive data, such as patient records in healthcare or personal information from mobile applications, is often siloed due to privacy concerns [91] and stringent regulations [77]. While necessary for protecting user rights, these restrictions hinder the potential of machine learning solutions, as they prevent data from being centralized for training. Such constraints pose significant challenges to the application of Deep Learning (DL) techniques, particularly in scenarios where scalability and efficiency are paramount.

To tackle these scalability concerns, Distributed Deep Learning (DDL) has emerged as an alternative to traditional centralized training approaches [96, 10], offering efficient learning over large-scale data across multiple worker-nodes, enhancing the speed of training DL models and paving the way for more scalable and resilient DL applications [75, 50, 36, 95, 13]. This distributed training approach typically follows an iterative process involving local computation followed by a phase of model synchronization. For example, the bulk synchronous parallel (BSP) approach [76], performs a single step of local training and then averages the local model updates to form a global model [96]; this is repeated until convergence.

A significant challenge inherent in the traditional techniques, especially in federated settings where worker interconnections are slow, is the communication bottleneck, which severely restricts system scalability [71, 83]. More specifically, the communication bottleneck arises from the frequent exchange (synchronization) of model parameters—often in the range of billions—across distributed workers. The synchronization process entails substantial data volume transfer and generally dominates the overall training time, leading to a low computation-to-communication ratio [61, 19]. Addressing this challenge to expedite DDL algorithms has been a focal point of research for many years; speeding-up SGD is arguably among the most impactful and transformative problems in machine learning [81].

Among the most communication-sensitive and practically significant applications of DDL is Federated Learning (FL)—a paradigm specifically designed to train models across decentralized data sources without requiring raw data exchange [31]. To better understand the importance of communication-efficiency in FL, it is crucial to examine how this learning approach operates in practice. In a typical FL setting, a central server orchestrates the training process across multiple data owners. Training proceeds in iterative rounds (Figure 1.1), each consisting of four key steps: broadcast, local training, collection, and aggregation. The server first broadcasts a global model to participating clients, who then train the model locally using their private data. After training, clients send their updates back to the server, who aggregates them into a new global model. These FL training rounds are repeated thousands of times until the model converges. Despite its conceptual simplicity, a critical open question remains: *When should the server collect model updates? When should a round end?*

This question lies at the heart of communication-efficient federated and distributed optimization [79]. Since communication is expensive, the frequency and timing of model collection must be carefully chosen to strike a balance between convergence speed and resource consumption. Over the years, several strategies have emerged to address this

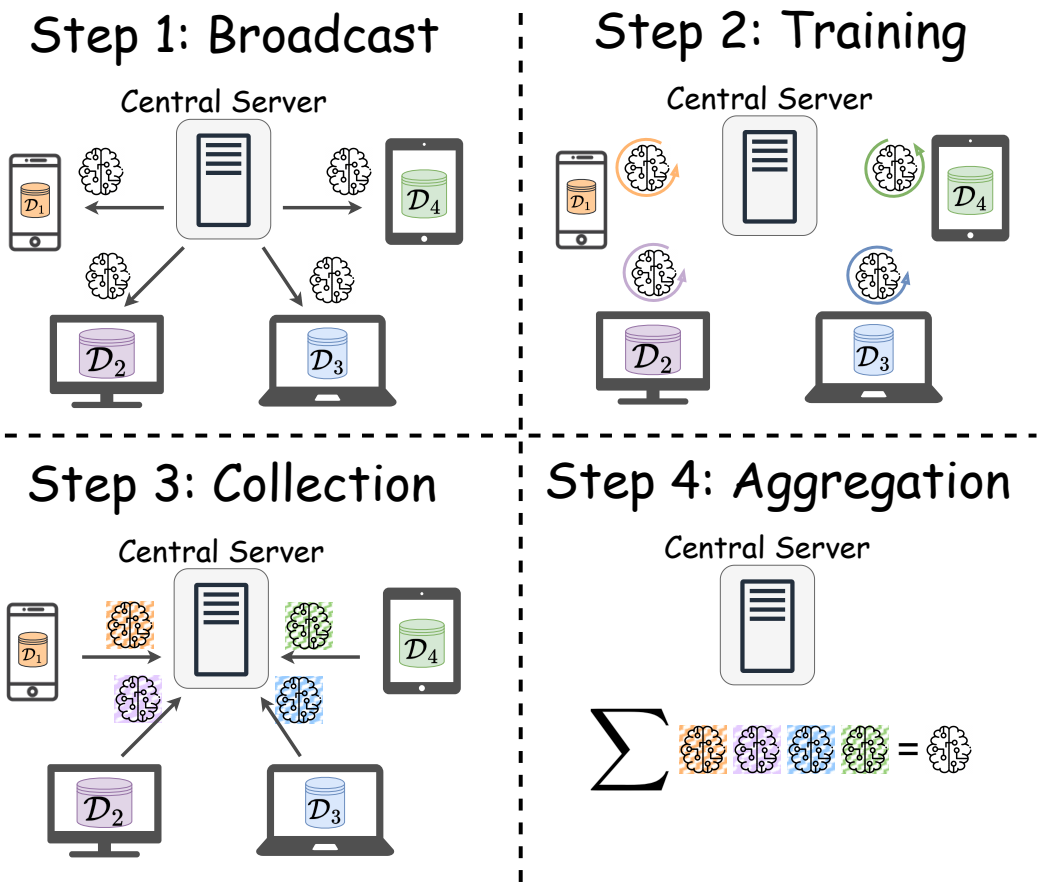


Figure 1.1: A Federated Learning Round

trade-off, each offering a different answer to how often synchronization should occur. In what follows, we overview prominent solutions to this problem.

The most direct approach is Local-SGD, which reduces the frequency of model collection by allowing workers to perform $\tau$ local update steps before aggregation [93, 23]. Although Local-SGD is effective in reducing communication while maintaining comparable model quality [81], selecting an optimal value for $\tau$ remains a major challenge. Only a handful of studies offer theoretical insights into its impact on convergence [81, 66, 93]. Building upon both theoretical and empirical findings from the Local-SGD literature, researchers have proposed the FedOpt family of algorithms [58], which retains the same collection schedule but replaces simple averaging with adaptive server-side optimization techniques.

However, these methods have two significant limitations. First, they assume the training process is static, with configurations predefined throughout training, without taking into account real-time training dynamics. Second, the choice of the number of local steps is arbitrary and lacks justification. Lower values may improve convergence but skyrocket communication overhead, while higher values reduce communication cost but risk unpredictable training behavior or, in many cases, non-convergence [93].

To address these limitations, more sophisticated communication strategies introduce varying sequences of local update steps $\{\tau_0, ..., \tau_R\}$, instead of a fixed $\tau$. In [80], in order to minimize convergence error with respect to wall-time, the authors proposed a decreasing sequence of local update steps. Conversely, the focus in [23] was on reducing the number of communication rounds for a fixed number of model updates and an increasing sequence emerged. These contrasting approaches underscore the multifaceted nature of commu-

nication strategies in distributed deep learning, highlighting not only the absence of a one-size-fits-all solution but also the growing need for dynamic, context-aware strategies that can continuously adapt to the specific intricacies of the learning task.

**Contributions.** To address this problem, we first introduce the Federated Dynamic Averaging (FDA) [72] algorithm, presented in Chapter 3. Then, in Chapter 4, we propose the FDA-Opt [73] family of algorithms, which generalizes both FDA and FedOpt. The main contributions of this thesis are summarized as follows:

- We propose FDA, an algorithm that dynamically decides to synchronize local workers when *model variance across workers* exceeds a threshold. This strategy drastically reduces communication, while preserving cohesive progress towards the shared training objective.
- We propose two variants of FDA, which differ in the amount of information preserved in the local states that are transmitted by each worker and aggregated for subsequent estimation of model variance. These two variants, termed SketchFDA and LinearFDA, offer a different balance between communication efficiency and approximation accuracy.
- We evaluate and compare FDA with other DDL algorithms through a comprehensive suite of experiments with diverse datasets, models, and tasks. Our experiments demonstrate that FDA outperforms traditional and contemporary FL algorithms by 1-2 orders of magnitude in communication savings, while maintaining equivalent model performance. Furthermore, it effectively balances the competing demands of communication and computation, providing greatly improved trade-offs.
- We demonstrate FDA's robustness in various challenging Non-IID settings, common in real-world Federated Learning applications. While state-of-the-art methods typically require substantially more resources to converge under Non-IID conditions, FDA maintains consistent and comparable performance across both IID and Non-IID settings.
- We propose the FDA-Opt family of algorithms, a unified generalization of both FDA and FedOpt. In doing so, we introduce a dynamic scheme for the variance threshold, removing the need for any manual configuration, and completely alleviate the original synchronization bottleneck.
- We show that FDA-Opt outperforms FedOpt in communication-efficiency, achieving improvements of at least 2× while operating under conditions optimized for FedOpt (the competitor). Specifically, we manually identify the optimal hyper-parameter configurations for each FedOpt algorithm, and then apply the same configurations to our proposed FDA-Opt counterparts. Remarkably, even with these "unfair" and "rigged" hyper-paremeters, FDA-Opt achieves 2× greater communication-efficiency for the same model performance. This has the important implication that hyper-parameters proven effective for FedOpt in the literature can be directly leveraged to configure our proposed algorithms.
- We show that all FDA-Opt algorithms converge to 5×–10× lower training loss than their FedOpt counterparts within the same number of rounds.

**Thesis Outline.** This thesis is structured as follows. Chapter 2 provides the necessary background on federated optimization. Chapter 3 presents the first main contribution of the thesis: the FDA algorithm. Chapter 4 builds on this by proposing the FDA-Opt family of algorithms, which addresses the key limitations of both FDA and FedOpt. Finally, Chapter 5 concludes the thesis and outlines directions for future work.

# 2. Preliminaries

## 2.1 Descend Methods

Consider a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$. We will refer to $f$ as the *objective function*. We want to solve the unconstrained optimization problem:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}} \;\; f(\boldsymbol{w})$$

The algorithms described in this section produce a minimizing sequence $\boldsymbol{w}_t$, $t = 1, \ldots,$ where

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + m_t \boldsymbol{\Delta w}_t$$

and $m_t > 0$ (except when $\boldsymbol{w}_t$ is optimal). Here the concatenated symbols $\boldsymbol{\Delta}$ and $\boldsymbol{w}$ that form $\boldsymbol{\Delta w}_t$ are to be read as a single entity, a vector in $\mathbb{R}^d$ called the *step* or *search direction*, and $t = 1, \ldots,$ denotes the iteration number. The scalar $m_t$ is called the *step size* or *step length* at iteration $t$. We will drop the superscripts and use the lighter notation $\boldsymbol{w} \leftarrow \boldsymbol{w} + m\boldsymbol{\Delta w}$, in place of $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + m_t \boldsymbol{\Delta w}_t$.

The outline of a general descent method is as follows: It alternates between two steps: determining a descend direction $\boldsymbol{\Delta w}$, and the selection of a step size $m$.

**Algorithm 1** General descent method [8]

**given** A starting point $\boldsymbol{w} \in$ **dom** $f$
**repeat**
1: Determine a descent direction $\boldsymbol{\Delta w}$
2: *Line search.* Choose a step size $m > 0$
3: *Update.* $\boldsymbol{w} \leftarrow \boldsymbol{w} + m\boldsymbol{\Delta w}$
**until** stopping criterion is satisfied

The second step is called the *line search* since selection of the step size $t$ determines where along the line (more accurately, *ray*) $\{\boldsymbol{w} + m\boldsymbol{\Delta w} \mid m \in \mathbb{R}_+\}$ the next iterate will be. There are many different line search methods, each with its own advantages and specific use cases [8].

### 2.1.1 Gradient Descend

A natural choice for the search direction is the negative gradient $\boldsymbol{\Delta w} = -\nabla f(\boldsymbol{w})$. The resulting algorithm is called *Gradient algorithm* or *Gradient Descend method* [8].

In the context of machine learning, particularly when training deep neural networks, it's common to use a fixed *step size* (*learning rate*) denoted as $\eta$ for the gradient descent method. This is primarily due to computational inefficiency, as performing a line search at each step can be computationally expensive, especially when dealing with large datasets and complex models. The update rule becomes:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f(\boldsymbol{w})$$

### 2.1.2 SGD

*Stochastic Gradient Descent* (SGD) is a variation of the gradient descent method that computes the gradient using a single randomly chosen example from the dataset at each step, rather than the entire dataset. This makes SGD faster and more scalable to large datasets compared to the standard gradient descent method.

In the context of training a machine learning model, we typically have a dataset of examples and a loss function that measures how well the model fits each example. The goal is to find the model parameters (denoted by $\boldsymbol{w}$) that minimize the average loss over all examples in the dataset. This is equivalent to minimizing the function

$$f(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{w})$$

where $n$ is the number of examples in the dataset, and $f_i(\boldsymbol{w})$ is the loss for the $i$-th example. More specifically, $f_i(\boldsymbol{w})$ is the loss $l(x_i, y_i; \boldsymbol{w})$ on example $(x_i, y_i)$ with parameters $\boldsymbol{w}$.

For the sake of succinctness and ease of understanding, we will use the notation $f_i(\boldsymbol{w})$ to denote the loss for the $i$-th example throughout this thesis. This notation will allow us to express complex mathematical concepts more clearly and concisely.

The update rule for SGD is similar to that of gradient descent, but with the gradient $\nabla f(w)$ replaced by a stochastic estimate of the gradient

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_i(\boldsymbol{w})$$

where $i$ is chosen uniformly at random from $\{1, \ldots, n\}$ at each step [8].

Despite its simplicity, SGD has proven to be highly effective for training a wide range of machine learning models, including deep neural networks. However, one of the challenges with SGD is the selection of the learning rate $\eta$. If $\eta$ is too large, SGD may fail to converge; if $\eta$ is too small, the convergence may be too slow. Various strategies have been proposed to adaptively adjust the learning rate during the course of optimization, leading to several variants of SGD.

### 2.1.3 Mini-batch SGD

One popular variant of SGD is *Mini-Batch Gradient Descent*, which computes in parallel the gradient using a small batch of examples at each step, rather than a single example. This can lead to a more stable and accurate estimate of the gradient, while still being much faster than computing the gradient over the entire dataset.

The update rule for Mini-Batch Gradient Descent is

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_B(\boldsymbol{w})$$

where $B$ is a mini-batch of examples chosen at random at each step. The function $f_B(\boldsymbol{w})$ is the average loss over the examples in the mini-batch, and its gradient is computed as

$$\nabla f_B(\boldsymbol{w}) = \frac{1}{|B|} \sum_{i \in B} \nabla f_i(\boldsymbol{w}) \tag{2.1}$$

where $|B|$ is the number of examples in the mini-batch.

Mini-batch gradient descent strikes a balance between the computational efficiency of stochastic gradient descent and the stability and accuracy of full-batch gradient descent[1].

[1]Full-batch gradient descent refers to the variant of gradient descent where the gradient is computed over the entire dataset at each step. This method provides the most accurate estimate of the gradient, but it is computationally expensive and less scalable.

By averaging the gradients over a mini-batch of examples, it reduces the variance of the gradient estimates, which can lead to more stable and consistent progress towards a minimum. This can be particularly beneficial in the presence of noisy data or non-smooth optimization landscapes, where the gradient can vary significantly from one example to another.

Moreover, Mini-Batch Gradient Descent is highly amenable to parallelization, as the gradients for different examples in the mini-batch can be computed simultaneously. This makes it a popular choice for training deep neural networks on modern hardware accelerators such as GPUs, which can perform many computations in parallel.

However, similar to SGD, the choice of the learning rate and the mini-batch size can significantly affect the performance of Mini-Batch Gradient Descent. A learning rate that is too large can cause the method to diverge, while a learning rate that is too small can result in slow convergence. Similarly, a mini-batch size that is too large can lead to less frequent updates, which might slow down the convergence. On the other hand, a mini-batch size that is too small can result in noisy gradient estimates. Therefore, these hyperparameters often need to be carefully tuned for each specific problem.

In the machine learning community, the term Stochastic Gradient Descent (SGD) often implicitly encompasses Mini-Batch Gradient Descent. While the original SGD algorithm computes the gradient using a single example at each step, it's common in practice to use a mini-batch of examples for computational efficiency and stability. This mini-batch variant is technically a different algorithm, but it's often simply referred to as SGD, without explicitly mentioning the use of mini-batches. The transition from single-sample SGD to its mini-batch counterpart is straightforward, which further blurs the distinction between the two in practical applications. Therefore, when interpreting machine learning literature or using machine learning software, it's important to understand that SGD might actually be implemented as Mini-Batch Gradient Descent. This understanding simplifies the discourse, as we can refer to both single-sample and mini-batch variants under the umbrella term of SGD, without the need for constant differentiation.

### 2.1.4 SGDM

*Stochastic Gradient Descent with Momentum* (SGDM) [68] improves upon vanilla SGD by incorporating a moving average of past gradients to accelerate convergence and reduce oscillations, particularly in regions with pathological curvature.

At each iteration, SGDM maintains a velocity vector $v_t$ that accumulates a fraction of the previous velocity and the current gradient. This results in updates that persist in consistent directions and dampen oscillations in noisy or high-curvature areas.

The update rule is given by:

$$\begin{aligned} v_t &\leftarrow \mu \cdot v_{t-1} + \eta \cdot \nabla f_i(\boldsymbol{w}_{t-1}) \\ \boldsymbol{w}_t &\leftarrow \boldsymbol{w}_{t-1} - v_t \end{aligned}$$

where $\mu \in [0, 1)$ is the momentum coefficient and $\eta$ is the learning rate. The gradient $\nabla f_i(\boldsymbol{w}_{t-1})$ is typically computed using a mini-batch.

Compared to plain SGD, SGDM is more effective at navigating ravines and escaping shallow local minima. It is widely used in deep learning and forms the basis of several other optimizers.

### 2.1.5 Adam

*Adaptive Moment Estimation* (Adam), introduced in 2014 [34], combines the benefits of Momentum [21] and RMSProp[2]. From Momentum, it incorporates exponentially weighted

[2]RMSProp was introduced in a Coursera lecture and not formally published.

**Algorithm 2** SGD with Momentum (SGDM) [68]

**Require:** $\eta$: Learning rate, $\mu \in [0, 1)$: Momentum coefficient
**Require:** $f(\boldsymbol{w})$: Stochastic objective function
**Require:** $\boldsymbol{w}_0$: Initial parameters
$v_0 \leftarrow 0$ ▷ Initialize velocity
$t \leftarrow 0$
**repeat**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_w f_t(\boldsymbol{w}_{t-1})$ ▷ Compute stochastic gradient
$v_t \leftarrow \mu \cdot v_{t-1} + \eta \cdot g_t$ ▷ Update velocity
$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - v_t$ ▷ Update parameters
**until** $\boldsymbol{w}_t$ has converged
**return** $\boldsymbol{w}_t$

averages of past gradients to smooth updates. From RMSProp, it borrows per-parameter scaling using an exponential average of squared gradients, helping navigate ill-conditioned loss surfaces.

Adam maintains two moving averages: one for gradients ($m_t$) and one for squared gradients ($v_t$), and applies bias correction to both. Parameters are updated by scaling the step size $\eta$ with the ratio $\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$, where $\epsilon$ prevents division by zero. All operations are element-wise.

A key feature of Adam is its adaptive learning rate: parameters with consistently large gradients receive smaller updates, and vice versa. This makes it well-suited for problems with sparse gradients or noisy objectives, as in NLP and computer vision.

Adam is widely used due to its robustness, fast convergence, and minimal need for hyperparameter tuning. Default values ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) generally perform well, with the learning rate $\eta$ being the most sensitive parameter.

In summary, Adam is a practical and effective optimizer, and remains a standard choice in deep learning pipelines.

**Algorithm 3** Adam [34]

**Require:** $\eta$: Step size (learning rate)
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\boldsymbol{w})$: Stochastic objective function with parameters $\boldsymbol{w}$
**Require:** $\boldsymbol{w}_0$: Initial parameter vector
$m_0 \leftarrow 0$ ▷ Initialize 1st moment vector
$v_0 \leftarrow 0$ ▷ Initialize 2nd moment vector
$t \leftarrow 0$ ▷ Initialize step
**repeat**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_w f_t(\boldsymbol{w}_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at step $t$
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second raw moment estimate
$\hat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate
$\hat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate
$\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \eta \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ ▷ Update parameters
**until** $\boldsymbol{w}_t$ has converged
**return** $\boldsymbol{w}_t$ ▷ Resulting parameters

### 2.1.6 AdamW

AdamW (Adam with decoupled weight decay), proposed in [48], is a modification of the Adam optimizer that improves generalization by decoupling the weight decay term from the gradient-based parameter updates.

Standard Adam applies $\ell_2$ regularization by adding the weight decay term directly to the gradient, which has been shown to cause undesirable interactions with the adaptive moment estimates. AdamW corrects this by applying weight decay *after* the Adam update, resulting in clearer separation between optimization and regularization.

The update rule is as follows:

**Algorithm 4** AdamW [48]

**Require:** $\eta$: Learning rate
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Decay rates
**Require:** $\lambda$: Weight decay coefficient
**Require:** $f$: Objective
**Require:** $\boldsymbol{w}_0$: Initial parameters
  $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
  **repeat**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_w f_t(\boldsymbol{w}_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
    $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) - \eta \cdot \lambda \cdot \boldsymbol{w}_{t-1}$
  **until** $\boldsymbol{w}_t$ converges
  **return** $\boldsymbol{w}_t$

AdamW retains the adaptive learning rate of Adam but separates regularization by directly shrinking the weights after each update. This decoupling has been shown to improve generalization and training stability in deep networks. Default hyperparameters remain similar to Adam: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, with $\lambda$ typically set between $10^{-5}$ and $10^{-2}$ depending on the application.

## 2.2 Federated Learning

Federated Learning (FL) is a distributed learning paradigm designed for scenarios where data is generated and stored across multiple decentralized sources—such as smartphones, hospitals, or edge devices—and cannot be shared due to privacy, regulatory, or communication constraints. Rather than aggregating data at a central location, FL enables collaborative model training by bringing computation to the data. In this section, we formally define the FL setup, present the foundational training process, and introduce the FedOpt family of optimization algorithms that generalize the classical FedAvg method. This context establishes the basis for our proposed dynamic training framework, Federated Dynamic Averaging (FDA in Chapter 3), and its generalization, FDA-Opt, which will be developed in Chapter 4.

### 2.2.1 Federated Learning Basics

Consider a scenario where we aim to train deep neural networks on data distributed across multiple devices or organizations—such as smartphones, hospitals, or sensors—that cannot

be exchanged due to constraints like privacy or regulatory restrictions. Federated Learning (FL) addresses this challenge by enabling a set of clients, $\mathcal{K}$, to collaboratively train a deep learning model without sharing their data. Each client $k \in \mathcal{K}$ retains its private local dataset, $\mathcal{D}_k$.

Instead of centralizing data on a powerful cluster for training, FL reverses the traditional approach: training and computation are brought to the data, that is, to the clients. Each client trains a model locally and shares updates with a central server, which consolidates them into a common global model, $\boldsymbol{w} \in \mathbb{R}^d$. The collective goal is to minimize the overall training loss across all clients, which can be expressed as the following distributed optimization problem [88]:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}} \;\; F(\boldsymbol{w}) \triangleq \frac{1}{|\mathcal{K}|} \sum_{k\in\mathcal{K}} F_k(\boldsymbol{w}) \tag{2.2}$$

Here, $F_k(\boldsymbol{w}) \triangleq \mathbb{E}_{\zeta_k\sim\mathcal{D}_k}\left[\ell(\boldsymbol{w};\zeta_k)\right]$ is the local objective function for client $k$, and $\ell(\boldsymbol{w};\zeta_k)$ denotes the loss for a data sample $\zeta_k$ given the model $\boldsymbol{w}$.

### 2.2.2 FedOpt: Generalized Federated Averaging

The most widely used algorithm for solving (2.2) is Federated Averaging (FedAvg) [51]. This method divides the FL training process into iterative rounds. At the beginning of round $t$, the server broadcasts the current global model $\boldsymbol{w}_t$ to a selected cohort of participating clients $\mathcal{S}_t \subseteq \mathcal{K}$. Each client $k \in \mathcal{S}_t$ initializes its local model as $\boldsymbol{w}_{t,0}^{(k)} = \boldsymbol{w}_t$, and performs $\tau$ local updates of Stochastic Gradient Descent (SGD) using its private dataset $\mathcal{D}_k$. After training, the client sends its updated model $\boldsymbol{w}_{t,\tau}^{(k)}$ back to the server. The server then aggregates the updates—typically via averaging[3]—to form the new global model, $\boldsymbol{w}_{t+1}$.

**Algorithm 5** FedOpt [58]

**Input**: Initial $\boldsymbol{w}_0$; ClientOpt, ServerOpt; Total rounds $T$; Local training steps $\tau$

**for each** round $t = 0, \ldots, T-1$ **do**
  Sample a subset $\mathcal{S}_t \subseteq \mathcal{K}$ of clients
  Set $\boldsymbol{w}_{t,0}^{(k)} = \boldsymbol{w}_t$ for all $k \in \mathcal{S}_t$
  **for each** client $k \in \mathcal{S}_t$ **in parallel do**
    **for each** local step $i = 1, ..., \tau$ **do**
      Compute a gradient estimate $\boldsymbol{g}_{t,i-1}^{(k)}$ of $\nabla F_k(\boldsymbol{w}_{t,i-1}^{(k)})$
      $\boldsymbol{w}_{t,i}^{(k)} = \text{ClientOpt}(\boldsymbol{w}_{t,i-1}^{(k)}, \boldsymbol{g}_{t,i-1}^{(k)})$
    $\boldsymbol{\Delta}_{t,\tau}^{(k)} = \boldsymbol{w}_{t,\tau}^{(k)} - \boldsymbol{w}_{t,0}^{(k)}$ ▷ Local model change
  $\boldsymbol{g}_{t,\tau} = -\frac{1}{|\mathcal{S}_t|}\sum_{k\in\mathcal{S}_t} \boldsymbol{\Delta}_{t,\tau}^{(k)}$ ▷ "Pseudo"-gradient
  $\boldsymbol{w}_{t+1} = \text{ServerOpt}(\boldsymbol{w}_t, \boldsymbol{g}_{t,\tau})$
**return** $\boldsymbol{w}_T$

An indirect yet highly effective way to mitigate the communication burden of FL is to accelerate convergence which led to efforts to incorporate accelerated optimization techniques—such as Adam [34]—into the FL setting. However, FL clients typically lack

[3]To be precise, the aggregation is a weighted average based on the number of samples $|\mathcal{D}_k|$ owned by each client; however, to simplify notation, and because the extension to a weighted average is straightforward, we will use averaging throughout this chapter.

Table 2.1: Overview of the FedOpt and FDA-Opt families of algorithms. Depending on ClientOpt and ServerOpt a different name is derived for the FL algorithm. FDA-Opt is our proposed algorithmic family, which will be introduced in Section 4.3.

| **FedOpt** | **FDA-Opt** (ours) | **ClientOpt** | **ServerOpt** |
|---|---|---|---|
| FedAvg [51] | FDA-SGD [73] | SGD | SGD (lr = 1.0) |
| FedAvgM [28] | FDA-SGDM [73] | SGD | SGDM [68] |
| FedAdam [58] | FDA-Adam [73] | SGD | Adam [34] |
| FedAdamW [89] | FDA-AdamW [73] | SGD | AdamW [48] |
| FedAdaGrad [58] | FDA-AdaGrad [73] | SGD | AdaGrad [18] |

enough data for robust statistical coverage and are often stateless, making adaptive optimization at the client level impractical. Instead, the key idea is to shift adaptive optimization to the server.

Specifically, clients still perform $\tau$ local update steps of SGD, but instead of sending back the final model weights themselves, each client computes its local *model change* (or *drift*)

$$\boldsymbol{\Delta}_{t,\tau}^{(k)} = \boldsymbol{w}_{t,\tau}^{(k)} - \boldsymbol{w}_{t,0}^{(k)},$$

and sends this to the server. The server then averages the updates across clients to obtain the average update direction

$$\boldsymbol{g}_{t,\tau} = -\frac{1}{|\mathcal{S}_t|} \sum_{k \in \mathcal{S}_t} \boldsymbol{\Delta}_{t,\tau}^{(k)},$$

which we refer to as the "pseudo-gradient". The term "pseudo" is used because even though $\boldsymbol{g}_{t,\tau}$ is not a traditional gradient—it was not derived from a loss function via backpropagation—it can still be treated in the same way [58]. The server then applies a server-side optimizer, ServerOpt, to update the global model using this "pseudo"-gradient. Importantly, since $\boldsymbol{g}_{t,\tau}$ aggregates updates from multiple clients, it offers sufficient and trustworthy statistics, and the server can easily maintain the optimizer state across rounds. In effect, this addresses the two challenges that make adaptive methods impractical on clients: unreliable statistics, and statelessness. This approach underpins the FedOpt family of algorithms [58] (see Algorithm 5). Table 2.1 summarizes the FedOpt family.

# 3. Federated Dynamic Averaging

## 3.1 Motivation

Our work in this chapter addresses critical efficiency challenges in DDL, particularly in communication-constrained environments, such as the ones encountered in Federated Learning (FL) applications [31]. We introduce Federated Dynamic Averaging (FDA), a novel, adaptive distributed deep learning strategy that massively improves communication efficiency over previous work.

FDA utilizes a novel 2-action, conditional synchronization protocol, designed to avoid the need to decide or guess the proper values of local update steps, or to synchronize after each training step, but rather only performs the costly synchronization process *when needed*. Our FDA algorithm dynamically triggers synchronization based on the value of *model variance* across worker-nodes. In a nutshell, the costly synchronization step is only triggered if the local models have diverged significantly, which implies that the global model may no longer be accurate.

As Figure 3.1 demonstrates, at the start, workers enter the local training step with the same global model (Figure 3.1.A). Then, local training commences and each distributed worker-node computes its local state, which encapsulates helpful information for estimating the model variance (Figure 3.1.B). This is followed by the transmission (Figure 3.1.C) of these small-size local states, an operation that is bandwidth- and time-efficient because of their small size. During transmission, the local states are aggregated and their average is made available to all workers—an operation known as AllReduce. This operation does not require (or prohibit) the use of a central node. Based on the aggregated state, the workers can estimate (Figure 3.1.D) whether the variance of the local models may have exceeded a threshold. If this is not the case, the costly synchronization step (Figure 3.1.E) is avoided and local training continues. What is important is how to properly pick these local states computed at, and then transmitted by, the local workers. To address this problem, we propose two variants of our FDA algorithm. Our contributions can be summarized as follows:

- We propose FDA, an algorithm that dynamically decides to synchronize local workers when *model variance across workers* exceeds a threshold. This strategy drastically reduces communication, while preserving cohesive progress towards the shared training objective.
- We propose two variants of FDA, which differ in the amount of information preserved in the local states that are transmitted by each worker and aggregated for subsequent estimation of model variance. These two variants, termed SketchFDA and LinearFDA, offer a different balance between communication efficiency and approximation accuracy.
- We evaluate and compare FDA with other DDL algorithms through a comprehensive suite of experiments with diverse datasets, models, and tasks. Our experiments demonstrate that FDA outperforms traditional and contemporary FL algorithms

by 1-2 orders of magnitude in communication savings, while maintaining equivalent model performance. Furthermore, it effectively balances the competing demands of communication and computation, providing greatly improved trade-offs.

- We demonstrate FDA's robustness in various challenging Non-IID settings, common in real-world Federated Learning applications. While state-of-the-art methods typically require substantially more resources to converge under Non-IID conditions, FDA maintains consistent and comparable performance across both IID and Non-IID settings.

**Outline.** The remainder of this chapter is organized as follows: Section 3.2 reviews related work. Section 3.3 introduces our DDL technique, Federated Dynamic Averaging (FDA), and its two variants. Section 3.4 details the experimental setup, and discusses the insights and conclusions drawn from our empirical investigation. Lastly, Section 3.5 contains concluding remarks.

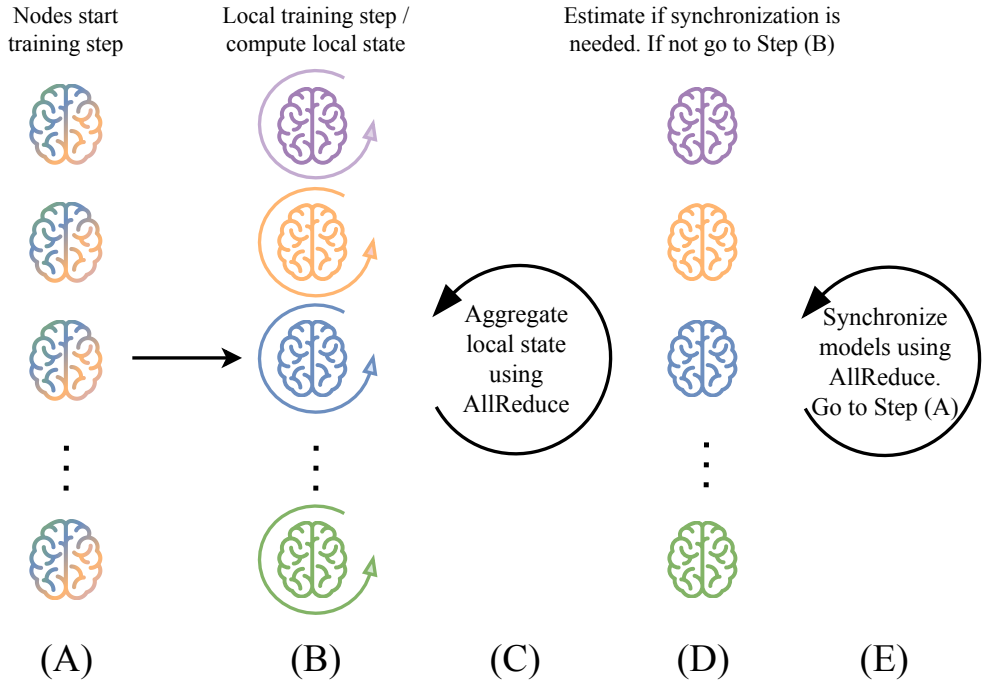


Figure 3.1: FDA. The local training step is followed by the computation of a local state by all worker-nodes. Then, the (small in size) local states are aggregated. Based on the aggregated result, all workers estimate if synchronization is required. In most cases, the expensive synchronization step of the models is avoided and local training continues

## 3.2 Related Work

**Problem formulation**. Consider distributed training of deep neural networks over multiple workers [14, 42]. In this setting, each worker represents a data owner (equivalently, a local model owner) and has access to its own set of training data $\mathcal{D}_k$. Workers can utilize any available hardware they possess (e.g., GPUs, CPUs) to perform learning steps. The collective goal is to find a common model $\boldsymbol{w} \in \mathbb{R}^d$ by minimizing the overall training loss. This scenario can be effectively modeled as a distributed optimization problem, formulated

as follows:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}} \;\; F(\boldsymbol{w}) \triangleq \frac{1}{K}\sum_{k=1}^{K} F_k(\boldsymbol{w}) \tag{3.1}$$

where $K$ is the number of workers and $F_k(\boldsymbol{w}) \triangleq \mathbb{E}_{\zeta_k\sim\mathcal{D}_k}\left[\ell(\boldsymbol{w};\zeta_k)\right]$ is the local objective function for worker $k$. Function $\ell(\boldsymbol{w};\zeta_k)$ represents the *loss* for data sample $\zeta_k$ given model $\boldsymbol{w}$.

**Solution direction.** As noted in the seminal work [31], research in FL should focus primarily on synchronous solutions. This allows different lines of research (e.g., compression, privacy, etc.) to be developed independently and then combined seamlessly. Our work, along with most communication-efficient FL strategies, adheres to this synchronous paradigm. However, such approaches may be less effective in environments where each communication operation incurs significant overhead regardless of the size of the data being transmitted (e.g., high-latency). In these scenarios, asynchronous mechanisms become necessary, though they typically fall outside the primary focus of contemporary FL research. That said, FDA can be modified to work asynchronously (as explained in Section 3.3.3).

**Communication efficient Local-SGD.** The work in [42] decomposes each round into two phases. In the first phase, each worker runs Local-SGD with $\tau = I_1$, while the second phase runs $I_2$ steps with $\tau = 1$; [42] proposes to exponentially decay $I_1$ every $M$ rounds. In the heterogeneous setting, the work in [55], by analysing the convergence rate, proposes an increasing sequence of local update steps for strongly-convex local objectives and fixed local update steps for other types of local objectives. The study in [92] dynamically increases batch sizes to reduce communication rounds, maintaining the same convergence rate as SSP-SGD. However, the large-batch approach leads to poor generalization [27], a challenge addressed by the post-local SGD method [45], which divides training into two phases: BSP-SGD followed by Local-SGD with a fixed number of steps. In the Lazily Aggregated Algorithm (LAG) [9], a different approach was taken, using only new gradients from some selected workers and reusing the outdated gradients from the rest, which essentially skips communication rounds.

Federated Averaging (FedAvg) [51] is another representative of communication efficient Local-SGD algorithms, which is a pivotal method in Federated Learning (FL) [31]. In the FL setting with edge computing systems, the work in [82] tries to find the optimal synchronization period $\tau$ subject to local computation and aggregation constraints. Recently [52], in the FL setting with the assumption of strongly-convex objectives, by analysing the balance between fast convergence and higher-round completion rate, a decaying local update step scheme emerged.

Unlike previous approaches that rely on predetermined synchronization schedules (fixed, decaying, or otherwise), our work introduces a dynamic synchronization strategy. FDA adapts continuously during the training process, basing synchronization decisions on a real-time metric: the model variance across workers.

**Accelerating convergence.** An indirect, yet highly effective way to mitigate the communication burden in DDL, is to speed up convergence. Consequently, recent works have built upon communication efficient Local-SGD methods by deploying accelerated versions of SGD to the distributed setting. Specifically, FedAdam [57] extends Adam [34] and FedAvgM [28] extends SGD with momentum (SGD-M) [69]. Recently, Mime [32] provides a framework to adapt arbitrary centralized optimization algorithms to the FL setting. However, these methods still suffer from the model divergence problem, particularly in heterogeneous settings. When solving equation (3.1), the disparity between each worker's optimal solution $\boldsymbol{w}_k^*$ for their objective $F_k$, and the global optimum $\boldsymbol{w}^*$ for $F$, can poten-

tially cause worker models to diverge (drift) towards their disparate minima [33, 57, 86]. The result is slow and unstable convergence with significant communication overhead. To address this problem, the SCAFFOLD algorithm [33] used control-variates (in the same spirit to SVRG), with significant speed-up. FedProx [41] re-parameterized FedAvg [51] by adding $L^2$ regularization in the workers' objectives to be near the global model. Lastly, FedDyn [2] improved upon these ideas with a dynamic regularizer making sure that if local models converge to a consensus, this consensus point aligns with the stationary point of the global objective function.

While these approaches primarily focus on enhancing the optimization process and typically employ fixed synchronization intervals (e.g., every local epoch), our work addresses a complementary aspect: determining the optimal timing for synchronization. FDA's dynamic synchronization strategy is orthogonal to these optimization techniques and can be integrated with them by simply adjusting the synchronization decision.

**Compression.** To reduce communication overhead in DDL, significant efforts have been directed towards minimizing message sizes. Key strategies include sparsification, where only crucial components of information are transmitted, as explored in [3], and quantization techniques, which involve transmitting only quantized gradients, as detailed in [62]. These techniques can be combined with Local-SGD methods to enhance communication-efficiency further. An example is Qsparse-local-SGD [6], which integrates aggressive sparsification and quantization with Local-SGD, achieving substantial communication savings. Crucially, FDA is fully compatible with any technique that reduces the cost of synchronization (e.g. model compression). Our approach simply adjusts the timing of the synchronization decision without altering the data being synchronized. This ensures that any compression technique effective in traditional methods (BSP, Local-SGD, etc.) will be equally effective when deployed with FDA. Therefore, the communication savings demonstrated in the relevant literature [84] can be safely expected to carry over to our approach as well.

Additionally, sketching emerges as another fundamental tool in large-scale machine learning. It effectively compresses high-dimensional problems into lower dimensions to save runtime and memory, typically utilizing hash-based probabilistic data structures. For instance, [65] use Count Sketches to compress auxiliary variables in optimization algorithms, significantly freeing up memory. Similarly, FetchSGD [59] employs Count Sketches to compress model updates and leverages their linearity for efficient merging. In contrast to these applications, our approach utilizes sketches not for compression but to estimate local state information, and based on this to decide whether a synchronization is required—an orthogonal application to traditional use cases. A comprehensive survey of compression techniques in DDL can be found in [84].

## 3.3 FDA: Federated Dynamic Averaging

We now present our algorithms, based on our notion of Federated Dynamic Averaging (FDA). Our algorithms deviate from prior work in these two key ways:

1. The decision on when to synchronize.
2. The actual synchronization process.

To the best of our knowledge, this is the first Distributed Deep Learning algorithm that dynamically decides when to synchronize based on the current collective state of the training progress—whether it is advancing well or poorly.

**Notation.** At each time step $t$, each worker $k$ independently maintains its own vector of

model parameters[1], denoted as $\boldsymbol{w}_t^{(k)} \in \mathbb{R}^d$. Let $\boldsymbol{w}_t$ represent the $K \times d$ tensor of all local model vectors, and $\overline{\boldsymbol{w}}_t$ be the average model vector (this notation applies to all vector quantities):

$$\boldsymbol{w}_t = \left[\boldsymbol{w}_t^{(1)}, \ldots, \boldsymbol{w}_t^{(K)}\right] \quad , \quad \overline{\boldsymbol{w}}_t = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{w}_t^{(k)}$$

Furthermore, let $\textsc{Optimize}(\boldsymbol{w}, B)$ be the updated model [22] computed by some optimization algorithm (e.g., SGD, Adam) using the model $\boldsymbol{w}$, and the batch $B$ of training data. It incorporates the learning rate, loss function and relevant gradients. During step $t$, each worker $k$ first applies the update:

$$\boldsymbol{w}_t^{(k)} = \textsc{Optimize}(\boldsymbol{w}_{t-1}^{(k)}, B_t^{(k)})$$

Moreover, operation $\textsc{AllReduce}(\boldsymbol{w}_t^{(k)})$ computes and returns the average model vector [40]:

$$\overline{\boldsymbol{w}}_t = \textsc{AllReduce}(\boldsymbol{w}_t^{(k)})$$

Workers *synchronize* by executing $\textsc{AllReduce}(\boldsymbol{w}_t^{(k)})$, thereby setting $\boldsymbol{w}_t^{(k)} := \overline{\boldsymbol{w}}_t$. If synchronization is not performed at step $t$, each worker continues training with its locally updated model. A comprehensive list of the notation used throughout this section is provided in Table 3.1.

**Model Variance and FDA.** The *model variance* quantifies the dispersion or spread of worker models around the average model:

$$\mathrm{Var}\left(\boldsymbol{w}_t\right) = \frac{1}{K}\sum_{k=1}^{K} \left\| \boldsymbol{w}_t^{(k)} - \overline{\boldsymbol{w}}_t \right\|_2^2 \tag{3.2}$$

This measure provides insight into how closely aligned the workers' models are at any given time. High variance indicates that the models are widely spread out, essentially drifting apart, leading to a lack of cohesion in the aggregated model. Conversely, a moderate or low variance suggests that the workers' models are closely aligned, working collectively towards the shared objective.

The FDA algorithm (Algorithm 6) is based on the premise that, as long as the variance is below a threshold $\Theta$, synchronization is not needed. Thus, we introduce the *Round Invariant* (RI):

$$\mathrm{Var}\left(\boldsymbol{w}_t\right) \leq \Theta \tag{3.3}$$

To preserve the RI, our FDA algorithm maintains (Lines 4-6 of Algorithm 6) at each worker $k$ a local (low-dimensional) state-vector $\mathbf{A}_t^{(k)}$, which is computed based on $\boldsymbol{w}_t^{(k)}$. These state vectors are vital for the subsequent estimation of the model variance, and underpin the two variants of the FDA algorithm (provided in Sections 3.3.1 and 3.3.2, respectively). Our estimation techniques begin by performing $\textsc{AllReduce}$ on the states $\mathbf{A}_t^{(k)}$, consolidating them into the average state $\overline{\mathbf{A}}_t$ (Line 7). Importantly, this communication step requires significantly less bandwidth and resources than transmitting the full models $\boldsymbol{w}_t^{(k)}$.

For each FDA variant, we also define a (different) function $H(\overline{\mathbf{A}}_t)$ that overestimates the variance, i.e., it ensures that as long as $H(\overline{\mathbf{A}}_t) \leq \Theta$ then the variance is bounded by $\Theta$. This guarantee is probabilistic for the Sketch-based variant of FDA, and deterministic

[1]The terms "model" and "model parameters" are used interchangeably, as is common in the literature.

Table 3.1: Notation

| Symbol | Meaning |
|---|---|
| $\langle \cdot\,, \cdot \rangle$ | Dot product |
| $t$ | Time step index |
| $K$ | Number of workers |
| $d$ | Model dimension |
| $\mathcal{D}_k$ | Training data of worker $k$ |
| $B_t^{(k)}$ | A batch sampled from $\mathcal{D}_k$ |
| $\boldsymbol{w}_t^{(k)} \in \mathbb{R}^d$ | Model of worker $k$ |
| $\boldsymbol{w}_t = [\boldsymbol{w}_t^{(1)}, \ldots, \boldsymbol{w}_t^{(K)}]$ | Tensor of local models |
| $\overline{\boldsymbol{w}}_t = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{w}_t^{(k)}$ | Average model (global model) |
| $\overline{\boldsymbol{w}}_{t_0}$ | Model after most recent sync. |
| $\overline{\boldsymbol{w}}_{t_{-1}}$ | Model after 2nd most recent sync. |
| $\boldsymbol{\Delta}_t^{(k)} = \boldsymbol{w}_t^{(k)} - \overline{\boldsymbol{w}}_{t_0}$ | Local model drift |
| $\overline{\boldsymbol{\Delta}}_t = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\Delta}_t^{(k)}$ | Average model drift (global drift) |
| $\mathrm{Var}\,(\boldsymbol{w}_t)$ | Model variance |
| $\Theta$ | Model variance threshold |
| $\mathbf{A}_t^{(k)}$ | State of worker $k$ |
| $\overline{\mathbf{A}}_t = \frac{1}{K}\sum_{k=1}^{K} \mathbf{A}_t^{(k)}$ | Average state |
| $H(\cdot)$ | Function for variance estimation |
| $\mathrm{sk}(\cdot) : \mathbb{R}^d \to \mathbb{R}^{l \times m}$ | AMS sketch operator (§3.3.1) |
| $\mathcal{M}_2(\cdot) : \mathbb{R}^{l \times m} \to \mathbb{R}$ | $L^2$ norm squared estimate (§3.3.1) |
| $\epsilon$ | Error of sketch estimate (§3.3.1) |
| $(1-\delta)$ | Confidence of approximation (§3.3.1) |
| $l = \mathcal{O}(\log 1/\delta)$ | #Rows of sketch matrix (§3.3.1) |
| $m = \mathcal{O}(1/\epsilon^2)$ | #Columns of sketch matrix (§3.3.1) |
| $\xi = \frac{\overline{\boldsymbol{w}}_{t_0} - \overline{\boldsymbol{w}}_{t_{-1}}}{\left\|\overline{\boldsymbol{w}}_{t_0} - \overline{\boldsymbol{w}}_{t_{-1}}\right\|_2}$ | Heuristic vec. for LinearFDA (§3.3.2) |

for its Linear counterpart. Consequently, if $H(\overline{\mathbf{A}}_t) > \Theta$ then synchronization is performed (Lines 8-9) — the RI invariant cannot be guaranteed. After synchronization, the model variance is zero.

**Efficiently Monitoring the RI.** Estimating model variance efficiently is at the heart of FDA. To this end, we first introduce the *local model drift*, $\boldsymbol{\Delta}_t^{(k)}$, and *average drift*, $\overline{\boldsymbol{\Delta}}_t$, defined as follows:

$$\boldsymbol{\Delta}_t^{(k)} = \boldsymbol{w}_t^{(k)} - \overline{\boldsymbol{w}}_{t_0} \quad , \quad \overline{\boldsymbol{\Delta}}_t = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\Delta}_t^{(k)}$$

Here, $\overline{\boldsymbol{w}}_{t_0}$ denotes the model vector after the most recent synchronization. Subsequently, the model variance can be written as:

$$\mathrm{Var}\left(\boldsymbol{w}_t\right) = \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\mathbf{u}_t^{(k)}\right\|_2^2\right) - \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2 \tag{3.4}$$

*Proof.* Adding an offset $(-\overline{\boldsymbol{w}}_{t_0})$ to each $\boldsymbol{w}_t^{(k)}$ does not alter the variance, therefore:

$$\begin{aligned}
\mathrm{Var}\left(\boldsymbol{w}_t\right) = \mathrm{Var}\left(\boldsymbol{w}_t - \overline{\boldsymbol{w}}_{t_0}\right) = \mathrm{Var}\left(\boldsymbol{\Delta}_t\right) &= \frac{1}{K}\sum_{k=1}^{K} \left\|\boldsymbol{\Delta}_t^{(k)} - \overline{\boldsymbol{\Delta}}_t\right\|_2^2 \\
&= \frac{1}{K}\sum_{k=1}^{K} \left(\left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - 2\left\langle \boldsymbol{\Delta}_t^{(k)}, \overline{\boldsymbol{\Delta}}_t \right\rangle + \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2\right) \\
&= \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\mathbf{u}_t^{(k)}\right\|_2^2\right) - 2\left(\frac{1}{K}\sum_{k=1}^{K} \left\langle \boldsymbol{\Delta}_t^{(k)}, \overline{\boldsymbol{\Delta}}_t \right\rangle\right) + \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2\right) \\
&= \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\mathbf{u}_t^{(k)}\right\|_2^2\right) - 2\left\langle \left(\frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\Delta}_t^{(k)}\right), \overline{\boldsymbol{\Delta}}_t \right\rangle + \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2 \\
&= \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\mathbf{u}_t^{(k)}\right\|_2^2\right) - 2\left\langle \overline{\boldsymbol{\Delta}}_t, \overline{\boldsymbol{\Delta}}_t \right\rangle + \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2 \\
&= \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\mathbf{u}_t^{(k)}\right\|_2^2\right) - 2\left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2 + \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2 \\
&= \left(\frac{1}{K}\sum_{k=1}^{K} \left\|\mathbf{u}_t^{(k)}\right\|_2^2\right) - \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2
\end{aligned}$$

□

Conceptually, following Eq equation (3.4), to precisely monitor the variance, we need to calculate two quantities: (1) $\frac{1}{K}\sum_{k=1}^{K} \|\mathbf{u}_t^{(k)}\|_2^2$, and (2) $\|\overline{\boldsymbol{\Delta}}_t\|_2^2$. The first quantity requires an AllReduce operation on the squared norm of the worker drifts, which involves minimal overhead since these values are scalar. In contrast, the second quantity necessitates an AllReduce operation on the worker drifts themselves, which are of model dimension, thus incurring a high communication cost. In fact, this operation is equivalent to synchronization, which is exactly what we aim to avoid in the first place. Thus, it becomes evident that communication-efficient model variance estimation hinges on estimating $\|\overline{\boldsymbol{\Delta}}_t\|_2^2$ efficiently.

Upcoming sections will detail two techniques for communication efficient variance estimation (which primarily involves estimating $\|\overline{\boldsymbol{\Delta}}_t\|_2^2$): SketchFDA and LinearFDA. To

**Algorithm 6** Federated Dynamic Averaging - FDA

**Require:** $K$: The number of workers indexed by $k$
**Require:** $\Theta$: The model variance threshold
**Require:** $b$: The local mini-batch size

Initialize $\boldsymbol{w}_0^{(k)} = \overline{\boldsymbol{w}}_0 \in \mathbb{R}^d$
**for** each step $t = 1, 2, \ldots$ **do**
  **for** each worker $k = 1, \ldots, K$ **in parallel do**
    $B_t^{(k)} \leftarrow$ (sample a batch of size $b$ from $\mathcal{D}_k$)
    $\boldsymbol{w}_t^{(k)} \leftarrow \textsc{Optimize}(\boldsymbol{w}_{t-1}^{(k)}, B_t^{(k)})$
    Update $\mathbf{A}_t^{(k)}$
    $\overline{\mathbf{A}}_t \leftarrow \textsc{AllReduce}(\mathbf{A}_t^{(k)})$
    **if** $H(\overline{\mathbf{A}}_t) > \Theta$ **then**
      $\boldsymbol{w}_t^{(k)} \leftarrow \textsc{AllReduce}(\boldsymbol{w}_t^{(k)})$ ▷ In-place

present them uniformly, we introduce the *local state* $\mathbf{A}_t^{(k)}$, a tensor which contains: (1) the scalar value $\|\mathbf{u}_t^{(k)}\|_2^2$ for precisely calculating the first quantity, and (2) a low-dimensional summary of $\boldsymbol{\Delta}_t^{(k)}$, different for each technique, for estimating the second quantity. For each technique we define an estimation function $H(\cdot)$ that calculates the current variance estimate from *average state* $\overline{\mathbf{A}}_t = \frac{1}{K}\sum_{k=1}^{K} \mathbf{A}_t^{(k)}$ (obtained via AllReduce).

### 3.3.1 SketchFDA: Sketch-based Estimation

An optimal estimator for $\left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2$ can be obtained through the utilization and properties of AMS sketches, as detailed in [12]. An AMS sketch of a vector $\mathbf{v} \in \mathbb{R}^d$ is an $l \times m$ real matrix:

$$\mathrm{sk}(\mathbf{v}) = \begin{bmatrix}\psi_1 & \psi_2 & \ldots & \psi_l\end{bmatrix}^\top \in \mathbb{R}^{l\times m} \;\;,\;\; l \cdot m \ll d$$

An estimate for squared-norm $\|\mathbf{v}\|_2^2$ is provided by the formula

$$\mathcal{M}_2\left(\mathrm{sk}(\mathbf{v})\right) = \mathrm{median}\left\{\|\psi_i\|_2^2 \;,\; i = 1, \ldots, l\right\}$$

The quality of estimation depends on the size of the sketch. For chosen $\epsilon, \delta > 0$, where sketch dimensions are given by $l = \mathcal{O}\left(\log 1/\delta\right)$ and $m = \mathcal{O}\left(1/\epsilon^2\right)$, we have the following probabilistic guarantee: with confidence at least $1 - \delta$,

$$\mathcal{M}_2(\mathrm{sk}(\mathbf{v})) \in (1 \pm \epsilon)\, \|\mathbf{v}\|_2^2$$

Notably, observe that the accuracy ($\epsilon$) and confidence ($1 - \delta$) only depend on the size of the sketch and not on the dimensionality of vector $\mathbf{v}$.

Two crucial properties of the AMS sketch are that (a) it is a linear transformation, i.e., for $\alpha_1, \alpha_2 \in \mathbb{R}$ and $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^d$,

$$\mathrm{sk}(\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2) = \alpha_1\,\mathrm{sk}(\mathbf{v}_1) + \alpha_2\,\mathrm{sk}(\mathbf{v}_2)$$

and (b) can be computed efficiently in time $O(l \cdot d)$.

In the SketchFDA approach, the salient idea is to employ AMS sketches $\mathrm{sk}(\boldsymbol{\Delta}_t^{(k)}) \in \mathbb{R}^{l\times m}$ as a low-dimensional representation of the local drifts $\boldsymbol{\Delta}_t^{(k)}$.

**Theorem 1.** *Let $l = \mathcal{O}(\log \frac{1}{\delta})$ and $m = \mathcal{O}(\frac{1}{\epsilon^2})$. Define the local state as*

$$\mathbf{A}_t^{(k)} = \left(\left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 , \mathrm{sk}\left(\boldsymbol{\Delta}_t^{(k)}\right)\right) \in \mathbb{R} \times \mathbb{R}^{l\times m}$$

*and the approximation function as*

$$H\left(\overline{\mathbf{A}}_t\right) = \frac{1}{K}\sum_k \left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - \frac{1}{1+\epsilon}\mathcal{M}_2\left(\frac{1}{K}\sum_{k=1}^K \mathrm{sk}\left(\boldsymbol{\Delta}_t^{(k)}\right)\right).$$

*Then, the condition* $H\left(\overline{\mathbf{A}}_t\right) \leq \Theta$ *implies* $\mathrm{Var}\left(\boldsymbol{w}_t\right) \leq \Theta$ *with probability at least* $(1-\delta)$*.*

*Proof.*

$$\begin{aligned}
H\left(\overline{\mathbf{A}}_t\right) &= \frac{1}{K}\sum_{k=1}^K \left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - \frac{1}{1+\epsilon}\mathcal{M}_2\left(\frac{1}{K}\sum_{i=1}^K \mathrm{sk}\left(\boldsymbol{\Delta}_t^{(k)}\right)\right) \\
&\overset{\text{(lin.)}}{=} \frac{1}{K}\sum_{k=1}^K \left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - \frac{1}{1+\epsilon}\mathcal{M}_2\left(\mathrm{sk}\left(\frac{1}{K}\sum_{i=1}^K \boldsymbol{\Delta}_t^{(k)}\right)\right) \\
&= \frac{1}{K}\sum_{k=1}^K \left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - \frac{1}{1+\epsilon}\mathcal{M}_2\left(\mathrm{sk}\left(\overline{\boldsymbol{\Delta}}_t\right)\right) \\
&\overset{(\epsilon\text{-err.})}{\geq} \frac{1}{K}\sum_{k=1}^K \left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - \left\|\overline{\boldsymbol{\Delta}}_t\right\|_2^2 \quad \text{with prob. at least } (1-\delta) \\
&= \mathrm{Var}\left(\boldsymbol{w}_t\right)
\end{aligned}$$

We proved that $H(\overline{\mathbf{A}}_t) \geq \mathrm{Var}\left(\boldsymbol{w}_t\right)$ with probability at least $(1-\delta)$, i.e., we overestimate the model variance with probability at least $(1-\delta)$, completing the proof. □

In Section 3.3.3, we discuss the empirical basis for choosing the values of $l$ and $m$, and how they practically impact the quality of the sketch approximation.

### 3.3.2 LinearFDA: Linear Approximation

Although AMS sketches provide good estimates for variance, their dimension is in the several hundreds, and the communication cost of AllReduce on sketches, performed at each step, may be non-negligible. Therefore, we also introduce a low-cost, ad-hoc estimation variant.

In this approach, instead of an AMS sketch, each local state contains the scalar value $\langle \xi\,,\, \boldsymbol{\Delta}_t^{(k)} \rangle \in \mathbb{R}$, where $\xi \in \mathbb{R}^d$ is a unit vector, known to all workers.

**Theorem 2.** *Define the local state as*

$$\mathbf{A}_t^{(k)} = \left(\left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2\,,\, \left\langle \xi\,,\, \boldsymbol{\Delta}_t^{(k)}\right\rangle\right) \in \mathbb{R}\times\mathbb{R} \quad,\quad \|\xi\|_2 = 1$$

*and the approximation function as*

$$H\left(\overline{\mathbf{A}}_t\right) = \frac{1}{K}\sum_{k=1}^K \left\|\boldsymbol{\Delta}_t^{(k)}\right\|_2^2 - \left|\frac{1}{K}\sum_{i=1}^K \left\langle \xi\,,\, \boldsymbol{\Delta}_t^{(k)}\right\rangle\right|^2$$

*Then, the condition* $H\left(\overline{\mathbf{A}}_t\right) \leq \Theta$ *implies* $\mathrm{Var}\left(\boldsymbol{w}_t\right) \leq \Theta$*.*

*Proof.*

$$
\begin{aligned}
H\left(\overline{\mathbf{\Delta}}_t\right) &= \frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{\Delta}_t^{(k)}\right\|_2^2 - \left|\frac{1}{K}\sum_{i=1}^{K}\left\langle \xi\,,\,\mathbf{\Delta}_t^{(k)}\right\rangle\right|^2 \\
&= \frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{\Delta}_t^{(k)}\right\|_2^2 - \left|\left\langle \xi\,,\,\frac{1}{K}\sum_{i=1}^{K}\mathbf{\Delta}_t^{(k)}\right\rangle\right|^2 \\
&= \frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{\Delta}_t^{(k)}\right\|_2^2 - \left|\left\langle \xi\,,\,\overline{\mathbf{\Delta}}_t\right\rangle\right|^2 \\
&\geq \frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{\Delta}_t^{(k)}\right\|_2^2 - \|\xi\|_2^2\left\|\overline{\mathbf{\Delta}}_t\right\|_2^2 \\
&= \frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{\Delta}_t^{(k)}\right\|_2^2 - \left\|\overline{\mathbf{\Delta}}_t\right\|_2^2 \\
&= \mathrm{Var}\left(\boldsymbol{w}_t\right)
\end{aligned}
$$

We proved that $H(\overline{\mathbf{\Delta}}_t) \geq \mathrm{Var}\left(\boldsymbol{w}_t\right)$, i.e., we always overestimate the model variance, completing the proof. □

An arbitrary choice of $\xi$ (e.g., a random vector) is likely to estimate $\left\|\overline{\mathbf{\Delta}}_t\right\|_2^2$ poorly; if $\xi$ is uncorrelated to $\overline{\mathbf{\Delta}}_t$, then $\left|\left\langle \xi\,,\,\overline{\mathbf{\Delta}}_t\right\rangle\right|^2$ will likely be close to zero. A heuristic choice that might be correlated to $\overline{\mathbf{\Delta}}_t$ is the (normalized) value of $\overline{\mathbf{\Delta}}_{t_0}$, the global drift vector right at the time of last synchronization. All nodes can compute it independently without extra communication, if they take the difference of the models of the last two synchronizations:

$$
\xi = \frac{\overline{\mathbf{\Delta}}_{t_0}}{\left\|\overline{\mathbf{\Delta}}_{t_0}\right\|_2} = \frac{\overline{\boldsymbol{w}}_{t_0} - \overline{\boldsymbol{w}}_{t_{-1}}}{\left\|\overline{\boldsymbol{w}}_{t_0} - \overline{\boldsymbol{w}}_{t_{-1}}\right\|_2}
$$

### 3.3.3 Discussion

**FDA: Intuition.** The main intuition for FDA is summarized in making the decision to synchronize dynamic, based on *model variance* during training. This metric is designed to capture the collective state of the training process. In what follows, we provide intuition on why this is the case. It is important to remember that the global model $\overline{\boldsymbol{w}}_t$ and, by extension, the global drift $\overline{\mathbf{\Delta}}_t$, are ultimately what we care about and evaluate.

Model variance, as defined in Equation equation (3.4), is the difference between the average of the squared local drifts $\frac{1}{K}\sum\|\mathbf{\Delta}_t^{(k)}\|_2^2$ and the squared global drift $\left\|\overline{\mathbf{\Delta}}_t\right\|_2^2$. The first term reflects how far the individual worker models have moved–essentially, how much each worker has learned. The second term indicates how much of this learning is retained in the global model after aggregation.

The interplay between these two quantities is crucial. For example, when the local drifts are high but the global drift is low, the variance increases, signaling the need for synchronization. This scenario suggests that while individual workers have made significant progress (as indicated by high local drifts), this progress is not being effectively captured in the global model (indicated by the low global drift). In other words, the worker models have moved significantly, but the global model has remained relatively stationary in this high-dimensional space. This misalignment indicates that training is no longer progressing optimally, as the workers are moving towards disparate and conflicting local minima, making it crucial to synchronize and realign them. Conversely, when both

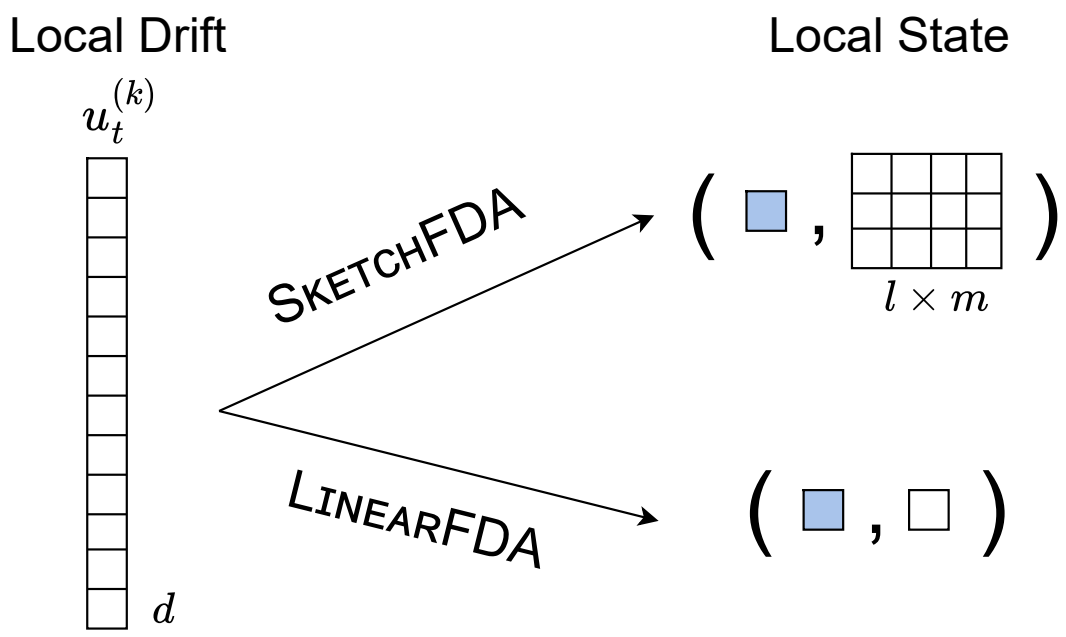


Figure 3.2: SketchFDA & LinearFDA: Local State structure.

the local and global drifts are either low or high, synchronization is not necessary, and the variance naturally remains low.

Neither the average of the local drifts nor the global drift alone provides a complete picture of the collective training progress. Relying solely on one or the other would lead to suboptimal synchronization decisions and likely prove ineffective. In FDA, it is the relationship between these quantities, as captured by the model variance, that offers valuable insights and guides the crucial decision of when to synchronize.

**SketchFDA vs. LinearFDA**: Both methods send the squared norm of the drift $\|\boldsymbol{\Delta}_t^{(k)}\|_2^2$, but differ in the additional accompanying lower-dimensional representation they transmit (Figure 3.2):

1. SketchFDA: An AMS sketch of the local drift.
2. LinearFDA: The dot product of a vector and the local drift.

The key difference between these two variants lies in the fidelity of approximation of the model variance. While both methods conservatively overestimate the variance, SketchFDA provides a provably accurate estimation, which is expected to lead to fewer synchronizations. LinearFDA requires less computational effort and bandwidth to create and communicate the local states, but may overestimate variance by too much, causing unnecessary synchronizations.

**SketchFDA: Choice of $l$ and $m$.** We empirically measured the approximation achieved with sketch dimensions of $l = 5$ rows and $m = 250$ columns (as defined in Section 3.3.1): these settings yield an error bound of $\epsilon \approx 6\%$ and a probabilistic confidence of $(1 - \delta) \approx 95\%$. Based on our experiments, we have adopted these values in our experiments and recommend them. Using these values, the byte-size of a sketch is $l \cdot m \cdot 4\,\text{bytes} = 5\,\text{kB}$, significantly smaller than the size of all our models. Sketches of smaller size could be used, albeit weakening the approximation of the variance. However, given that LinearFDA similarly weakens approximation and avoids using AMS sketches, in the interest of space we do not explore varying AMS sketch sizes in this work.

**FDA: Asynchronous Operation.** As mentioned in Section 3.2, FDA can be readily

Table 3.2: Summary of Experiments

| | | | Hyper-Parameters | | | Training | |
|---|---|---|---|---|---|---|---|
| NN | d | Dataset | $\Theta$ | b | K | Optimizer | Algorithms |
| LeNet-5 | 62K | MNIST | $\{0.5, 1, 1.5, 2, 3, 5, 7\}$ | 32 | { 5, 10, ..., 60 } | Adam | FDA, Synchronous, FedAdam |
| VGG16* | 2.6M | MNIST | $\{20, 25, 30, 50, 75, 90, 100\}$ | 32 | { 5, 10, ..., 60 } | Adam | FDA, Synchronous, FedAdam |
| DenseNet121 | 6.9M | CIFAR-10 | $\{200, 250, 275, 300, 325, 350, 400\}$ | 32 | { 5, 10, ..., 30 } | SGD-NM | FDA, Synchronous, FedAvgM |
| DenseNet201 | 18M | CIFAR-10 | $\{350, 500, 600, 700, 800, 850, 900\}$ | 32 | { 5, 10, ..., 30 } | SGD-NM | FDA, Synchronous, FedAvgM |
| (fine-tuning) ConvNeXtLarge | 198M | CIFAR-100 | $\{25, 50, 100, 150\}$ | 32 | { 3, 5 } | AdamW | FDA, Synchronous |

modified to operate asynchronously. In this setup, one worker-node acts as a coordinator, aggregating local states and determining whether synchronization is needed each time a local state is received. This decision is based on the most recent local states from all workers. It is important to note that, since local states are small in size, asynchronous operation is unlikely to alleviate bandwidth issues. The primary advantage is that it allows training to continue even in the presence of stragglers. Asynchronous operation might also be beneficial in rare cases where the overhead of initializing communication dominates the actual transmission time.

## 3.4 Experiments

### 3.4.1 Setup

Table 3.2 provides a comprehensive overview of our experiments. For each experiment, we detail the Neural Network (NN) architecture, its parameter count ($d$), and the dataset used for training. The table also specifies key hyper-parameters: the batch size ($b$), the number of workers ($K$), and the FDA-specific variance threshold ($\Theta$). Additionally, we indicate the chosen optimizer (as detailed in Section 3.3) and the training algorithms employed for each configuration.

**Platform.** We employ TensorFlow [1], integrated with Keras [11], as the platform for conducting our experiments. We used TensorFlow to implement our FDA variants and all competitive algorithms. All relevant code, figures, and data of this study are available in `https://github.com/miketheologitis/FedL-Sync-FDA`.

**Hardware & Infrastructure.** We conducted our experiments on the ARIS High performance computing (HPC) environment[2], utilizing a cluster of 44 GPU-accelerated worker-nodes. Each worker is equipped with two NVIDIA Tesla K40m GPUs and interconnected via an InfiniBand FDR14 network, providing up to 56 GB/s of bandwidth. Crucially, our evaluation remains agnostic to the underlying infrastructure of the specific workers.

**Datasets & Models.** The core experiments involve training Convolutional Neural Networks (CNNs) of varying sizes and complexities on two datasets: MNIST [15] and CIFAR-10 [35]. For the MNIST dataset, we employ LeNet-5 [37], composed of approximately 62 thousand parameters, and a modified version of VGG16 [63], denoted as VGG16*, consisting of 2.6 million parameters. VGG16* was specifically adapted for the MNIST dataset, a less demanding learning problem compared to ImageNet [60], for which VGG16 was designed. In VGG16*, we omitted the 512-channel convolutional blocks and downscaled the final two fully connected (FC) layers from 4096 to 512 units each. Both models use Glorot uniform initialization [20]. For CIFAR-10, we utilize DenseNet121 and DenseNet201 [30], as implemented in Keras [11], with the addition of dropout regularization layers at rate 0.2 and weight decay of $10^{-4}$, as prescribed in [30]. The DenseNet121 and DenseNet201 models have 6.9 million and 18 million parameters, respectively, and are both initialized

[2] `https://www.hpc.grnet.gr/en/hardware-2/`

with He normal [25].

Lastly, we explore a transfer learning scenario on the dataset CIFAR-100 [35], a choice reflecting the DL community's growing preference of using pre-trained models in such downstream tasks [24]. For example, a pre-trained visual transformer (ViT) on ImageNet, transferred to classify CIFAR-100, is currently on par with the state-of-the-art results for this task [17]. We adopt this exact transfer learning scenario, leveraging the more powerful ConvNeXtLarge model, pre-trained on ImageNet, with 198 million parameters. Following the feature extraction step [22], the testing accuracy on CIFAR-100 stands at 60%. Subsequently, we employ and evaluate our FDA algorithms in the arduous fine-tuning stage, where the entirety of the model is trained [53].

**Algorithms.** We consider five distributed deep learning algorithms: LinearFDA, SketchFDA, Synchronous [3], FedAdam [57], and FedAvgM [28]; the first three are standard in all experiments. Depending on the local optimizer, Adam [34] or SGD with Nesterov momentum (SGD-NM) [70], we also include their communication-efficient federated counterparts FedAdam or FedAvgM, respectively.

**Evaluation Methodology.** Comparing DDL algorithms is not straightforward. For example, comparing DDL algorithms based on the average cost of a training epoch can be misleading, as it does not consider the effects on the trained model's quality. To achieve a comprehensive performance assessment of FDA, we define a *training run* as the process of executing the DDL algorithm under evaluation, on (a) a specific DL model and training dataset, and (b) until a final epoch in which the trained model achieves a specific *testing accuracy* (termed as *Accuracy Target* in figures). Based on this definition, we focus on two performance metrics:

1. **Communication cost**, which is the total data (in bytes) transmitted by all workers. Notably, communication cost is unaffected by the training data volume since only model updates (when synchronizing) and local states (at each step), but not training data, are transmitted. Thus, the communication cost mainly depends on the complexity (number of parameters) of the used model. Translating the communication cost to *wall-clock time* (i.e., the total time required for the computation and communication of the DDL) depends on the network infrastructure connecting the workers and on the overhead of establishing and initializing communication. Its impact is larger in FL scenarios, where workers often use slower Wi-Fi connections.
2. **Computation cost**, which is the number of mini-batch steps (termed as *In-Parallel Learning Steps* in figures) performed by each worker. Translating this cost to *wall-clock time* is determined by the mini-batch size and the computational resources of the worker-nodes. Its impact is larger for workers with lower computational resources.

**Hyper-Parameters & Optimizers.** Hyper-parameters unique to each training dataset and model are detailed in Table 3.2; $\Theta$ is pertinent to FDA algorithms and not applicable to others. Notably, a guideline for setting the parameter $\Theta$ is provided in Section 3.4.3. For experiments involving FedAvgM and FedAdam, we use $E = 1$ local epochs, following [57]. For experiments with LeNet-5 and VGG16*, local optimization employs Adam, using the default settings as per [34]. In these cases, FedAdam also adheres to the default settings for both local and server optimization [57, 11]. For DenseNet121 and DenseNet201, local optimization is performed using SGD with Nesterov momentum (SGD-NM), setting the momentum parameter at 0.9 and learning rate at 0.1 [30]. For FedAvgM, local optimization is conducted with default settings [28, 11], while server optimization employs SGD with momentum, setting the momentum parameter and learn-

[3] The name was derived from the Bulk Synchronous Parallel approach; can be understood as a special case of the FDA Algorithm 6 where $\Theta$ is set to zero.

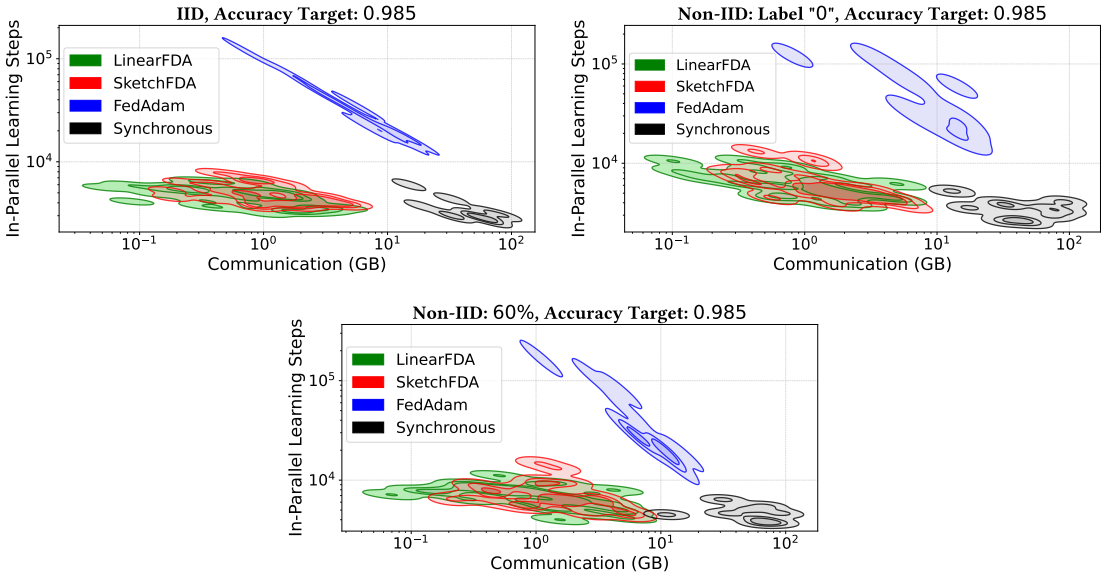


Figure 3.3: LeNet-5 on MNIST. At Non-IID: Label "0", the samples of Label "0" are assigned to few workers. At Non-IID: 60%, 60% of the dataset is sorted and allocated to workers, causing some workers to receive many samples from the same label

ing rate to 0.9 and 0.316, respectively [57]. Lastly, for the transfer learning experiments, local optimization leverages AdamW [49], with the hyper-parameters used for fine-tuning ConvNeXtLarge in the original study [47].

**Data Distribution.** In all experiments, the training dataset is divided into approximately equal parts among the workers. To assess the impact of data heterogeneity, we explore three scenarios:

1. **IID** — Independent and identically distributed.
2. **Non-IID:** $X\%$ — A portion $X\%$ of the dataset is sorted by label and sequentially allocated to workers, with the remainder distributed in an IID fashion.
3. **Non-IID: Label** $Y$ — All samples from label $Y$ are assigned to a few workers, while the rest are distributed in an IID manner.

### 3.4.2 Main Findings

The main findings of our experimental analyses are:

1. LinearFDA and SketchFDA outperform the Synchronous, FedAdam and FedAvgM techniques (their use depends on the local optimizer choice) by 1-2 orders of magnitude in communication, while maintaining equivalent model performance.
2. LinearFDA and SketchFDA also significantly outperform the FedAdam and FedAvgM techniques in terms of computation.
3. The performance of LinearFDA and SketchFDA is comparable in most experiments. SketchFDA provides a more accurate estimator of the variance and leads to fewer synchronizations than LinearFDA, but has a larger communication overhead for its local state (a sketch, compared to two numbers). SketchFDA significantly outperforms LinearFDA at the transfer learning scenario.
4. The FDA variants remain robust at various data heterogeneity settings, maintaining comparable performance to the IID case.

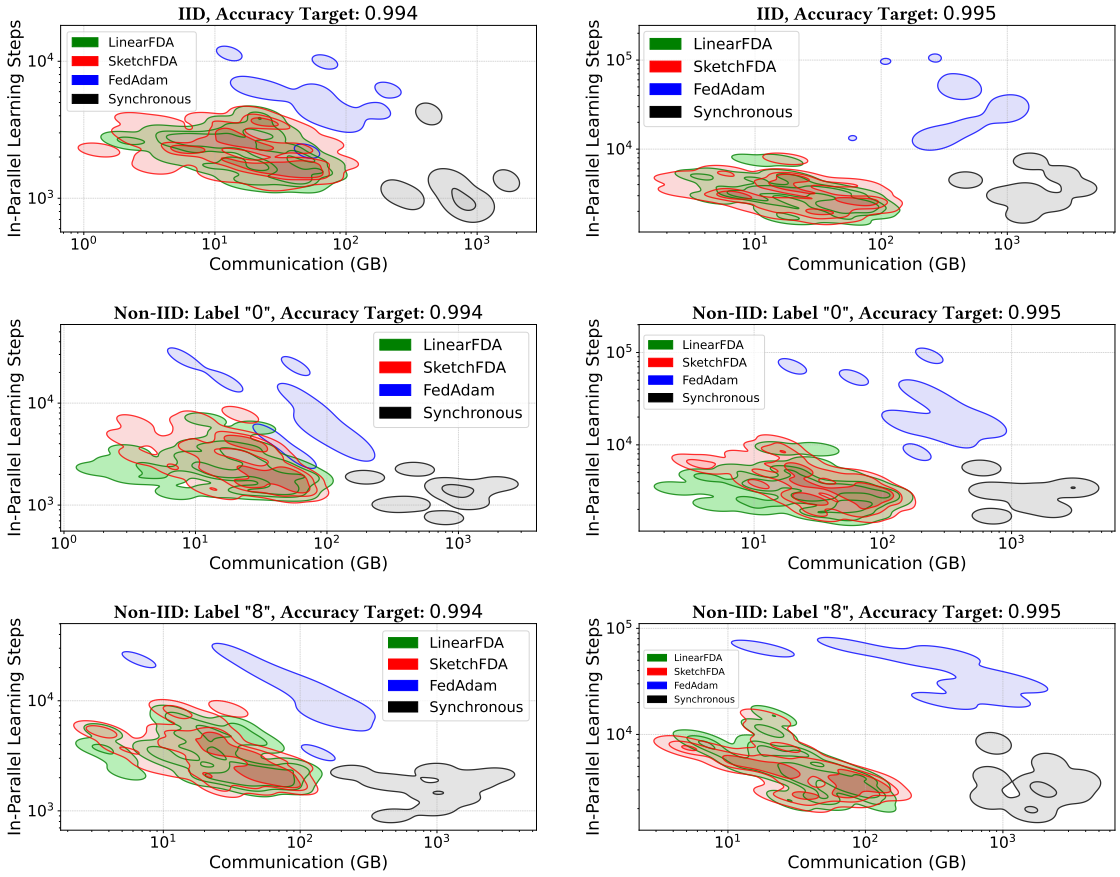


Figure 3.4: VGG16* on MNIST

### 3.4.3 Results

Due to the extensive set of unique experiments (over 1000), as detailed in Table 3.2, we leverage Kernel Density Estimation (KDE) plots [85] to visualize the bivariate distribution of computation and communication costs incurred by each strategy for attaining the *Accuracy Target*. These KDE plots provide a high-level overview of the cost trade-off for training accurate models. The varying levels of opacity in the filled areas of the KDE plots represent the density of the underlying data points: higher opacity indicates areas with a greater concentration of data, whereas lower opacity signifies less dense areas.

As an illustrative example, Figure 3.3 depicts the strategies' bivariate distribution for the LeNet-5 model trained on MNIST with different data heterogeneity setups. In these plots, the SketchFDA distribution is generated from experiments across all hyper-parameter combinations ($\Theta$ and $K$ in Table 3.2) that attained the *Accuracy Target* of 0.985. The observed high variance in the method's distribution stems from the varying $K$ and $\Theta$ values. In subsequent subsections, we elucidate how these hyper-parameters influence the communication and computation costs.

**FDA balances Communication vs. Computation.** DDL algorithms face a fundamental challenge: balancing the competing demands of computation and communication. Frequent communication accelerates convergence and potentially improves model performance, but incurs higher network overhead, an overhead that may be prohibitive when workers communicate through lower speed connections. Conversely, reducing communication saves bandwidth but risks hindering, or even stalling, convergence. Traditional DDL approaches, like Synchronous, require synchronizing model parameters after ev-

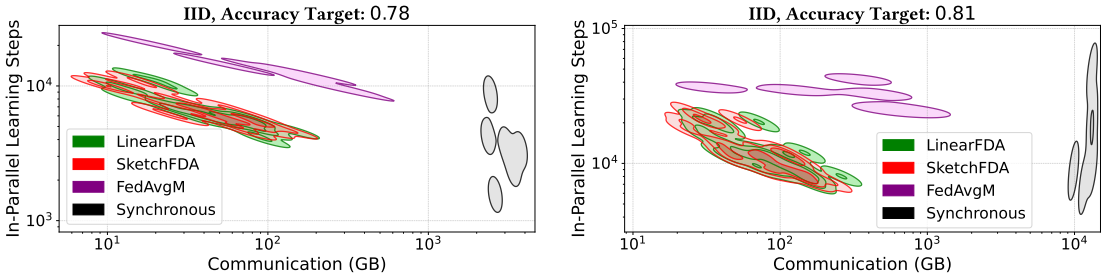


Figure 3.5: DenseNet121 on CIFAR-10

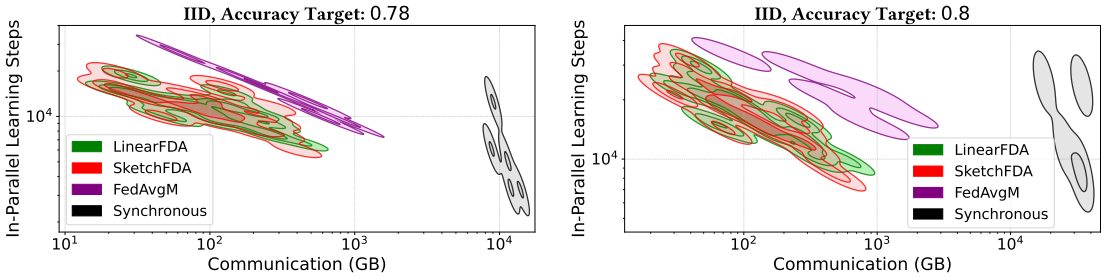


Figure 3.6: DenseNet201 on CIFAR-10

ery learning step, leading to significant communication overhead but facilitating faster convergence (lower computation cost). This is evident in Figures 3.3, 3.4, 3.5, and 3.6 (where SYNCHRONOUS appears in the bottom right — low computation, very high communication). Conversely, Federated Optimization (FEDOPT) methods [57] are designed to be communication-efficient, reducing communication between devices (workers) at the expense of increased local computation. Indeed, as shown in Figures 3.3-3.6, FEDAVGM and FEDADAM reduce communication by orders of magnitude but at the price of a corresponding increase in computation. Our two proposed FDA strategies achieve the best of both worlds: the low computation cost of traditional methods and the communication efficiency of FEDOPT approaches, as seen in Figures 3.3, 3.4, 3.5, and 3.6. In fact, they significantly outperform FEDAVGM and FEDADAM in their element, that is, communication-efficiency. Across all experiments, the FDA methods' distributions lie in the desired bottom left quadrant — low computation, very low communication.

**FDA counters diminishing returns.** The phenomenon of *diminishing returns* states that as a DL model nears its learning limits for a given dataset and architecture, each additional increment in accuracy may necessitate a disproportionate increase in training time, tuning, and resources [22, 74]. We first clearly notice this with VGG16* on MNIST in Figure 3.4 for all three data heterogeneity settings. For a 0.001 increase in accuracy (effectively 10 misclassified testing images), FEDADAM needs approximately 2-7× more communication and 3-7× more computation, respectively. This can be seen by comparing the figures at the left column of Figure 3.4 with the corresponding ones in the right column. Similarly, SYNCHRONOUS requires comparable increases in computation and approximately half an order of magnitude more in communication. On the other hand, the FDA methods suffer a slight (if any) increase in computation and communication for this accuracy enhancement. For DenseNet121 and DenseNet201 on CIFAR-10 (Figures 3.5, and 3.6), FEDAVGM and SYNCHRONOUS require half an order of magnitude more computation and communication to achieve the final marginal accuracy gains (0.78 to 0.81 for DenseNet121, and 0.78 to 0.8 for DenseNet201). In contrast, the FDA methods have almost no increase in communication and comparable increase in computation.

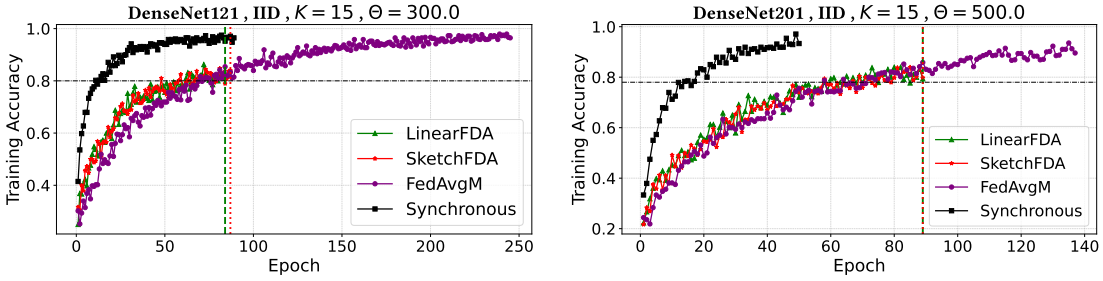


Figure 3.7: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 (top), and 0.78 (bottom). Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

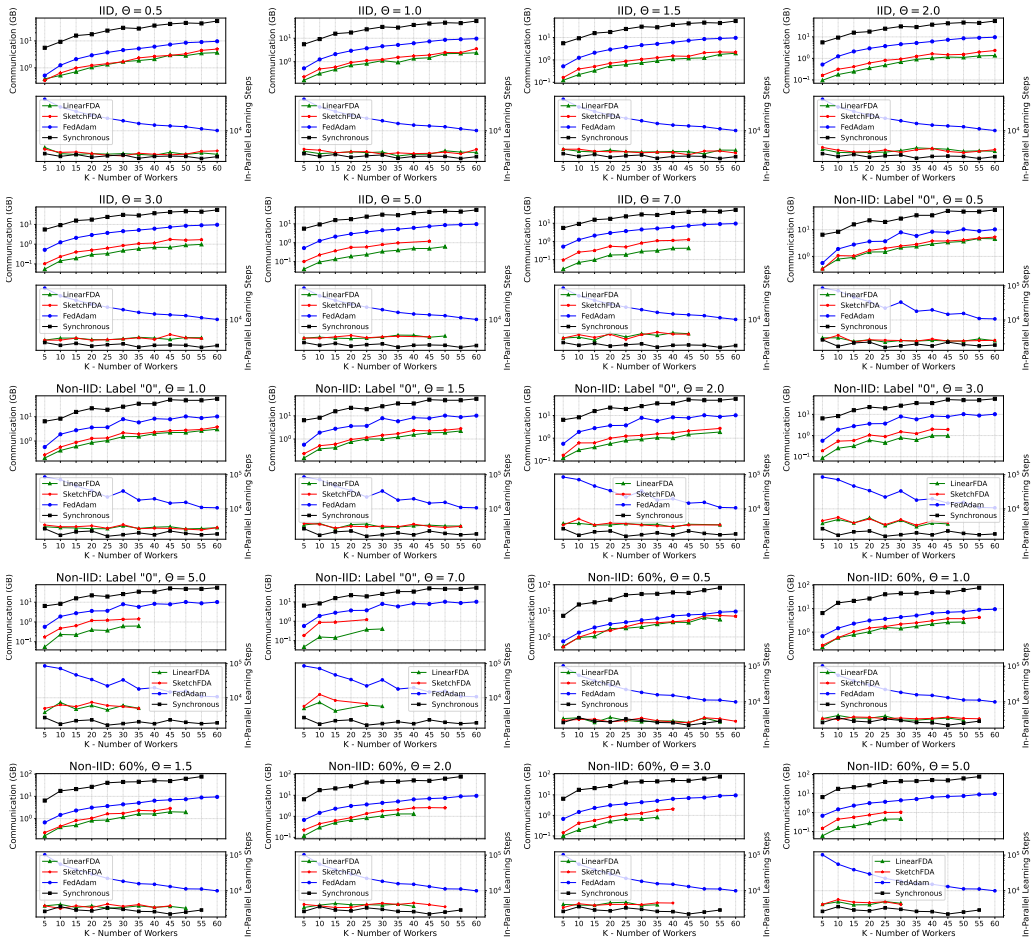


Figure 3.8: LeNet-5 on MNIST: Varying #Workers – Accuracy Target: 0.98

**FDA is resilient to data heterogeneity.** In DDL, data heterogeneity is a prevalent challenge, reflecting the complexity of real-world applications where the IID assumption often does not hold. The ability of DDL algorithms to maintain consistent performance in the face of non-IID data is a critical metric for their effectiveness and adaptability. Our empirical investigation reveals the FDA methods' noteworthy resilience in such heterogeneous environments. For LeNet-5 on MNIST, as illustrated in Figure 3.3, the computation and communication costs required to attain a test accuracy of 0.985 show negligible differences

across the IID and the two Non-IID settings (Label "0", 60%). Similarly, for VGG16* on MNIST, Figure 3.4 demonstrates that achieving a test accuracy of 0.995 incurs comparable computation and communication costs across the IID and the two Non-IID settings (Label "0", Label "8"); while overall costs are aligned, the distributions of the computation costs exhibit greater variability, yet remain closely consistent with the IID scenario.

**FDA has a lower generalization gap.** The factors determining how well a DL algorithm performs are its ability to: (1) make the training accuracy high, and (2) make the gap between training and test accuracy small. These two factors correspond to the two central challenges in DL: underfitting and overfitting [22]. For DenseNet121 on CIFAR-10, with a test accuracy target of 0.8, as illustrated in Figure 3.7, Synchronous and FedAvgM exhibit overfitting, with a noticeable discrepancy between training and test accuracy. In stark contrast, the FDA methods have an almost zero accuracy gap. Please note that LinearFDA and SketchFDA reach the test accuracy target of 0.8 much earlier (at epochs 86 and 91, respectively). Turning our focus to DenseNet201 on CIFAR-10, with a test accuracy target of 0.78, Synchronous again tends towards overfitting, while FedAvgM shows a slight improvement but still does not match the FDA methods, which continue to exhibit exceptional generalization capabilities, evidenced by a minimal training-test accuracy gap, as shown at Figure 3.7. Notably, given the necessity to fix hyper-parameters $\Theta$ and $K$ for the training accuracy plots, we selected two representative examples. The patterns of performance we highlighted are consistent across most of the conducted tests (see Appendix A.1).

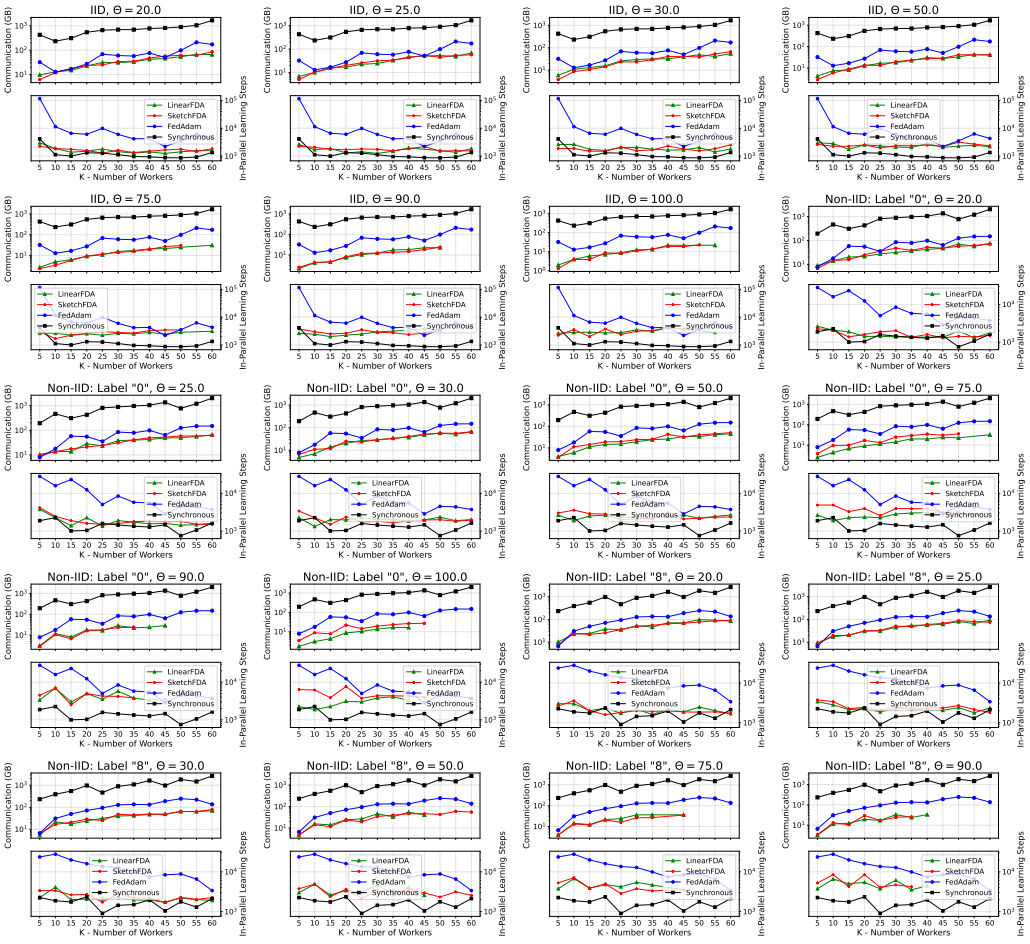


Figure 3.9: VGG16* on MNIST: Varying #Workers – Accuracy Target: 0.994

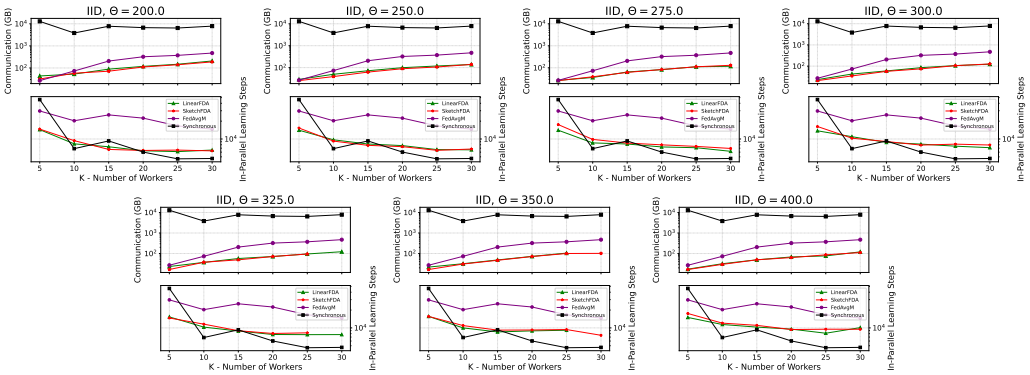


Figure 3.10: DenseNet121 on CIFAR-10: Varying #Workers – Accuracy Target: 0.8

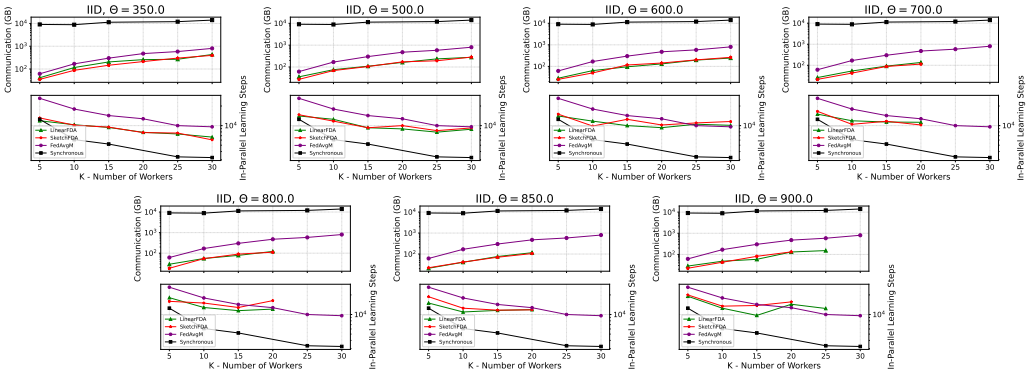


Figure 3.11: DenseNet201 on CIFAR-10: Varying #Workers – Accuracy Target: 0.78

**Dependence on $K$.** In distributed computing, scaling up typically results in proportional speed improvements. In DDL, however, scalability is less predictable due to the nuanced interplay of computation and communication costs with convergence, complicating the expected linear speedup [93]. This unpredictability is starkly illustrated with LeNet-5 and VGG16* on the MNIST dataset across all data heterogeneity settings and all strategies. Figures 3.8, and 3.9 demonstrate that increasing the number of workers does not decrease computation but rather exacerbates communication. These findings are troubling, as they reveal scaling up only hampers training speed and wastes resources. However, for more complex learning tasks like training DenseNet-121 and DenseNet-201 on CIFAR-10 (Figures 3.10, and 3.11), the expected behavior starts to emerge. Especially for DenseNet-121, scaling up ($K$ increase) leads to a decrease in computation cost for all strategies. Communication cost, however, increases with $K$ for all methods except Synchronous, which maintains constant communication irrespective of worker count, but at the expense of orders of magnitude higher communication overhead. Notably, while our findings might, in some cases, suggest potential speed benefits of not scaling up (smaller $K$), DDL is increasingly conducted within federated settings, where there is no other choice but to utilize the high number of workers. Our FDA variants consistently outperform FedAdam, FedAvgM, and Synchronous in communication efficiency, as demonstrated across all experiments in Figures 3.8-3.11. Specifically, they require up to 30 times less communication than FedAdam, 4 times less than FedAvgM, and up to 2.5 orders of magnitude less than Synchronous.

**FDA: Dependence on $\Theta$.** The variance threshold $\Theta$ can be seen as a lever in balancing communication and computation; essentially, it calibrates the trade-off between these two costs. A higher $\Theta$ allows for greater model divergence before synchronization, reducing

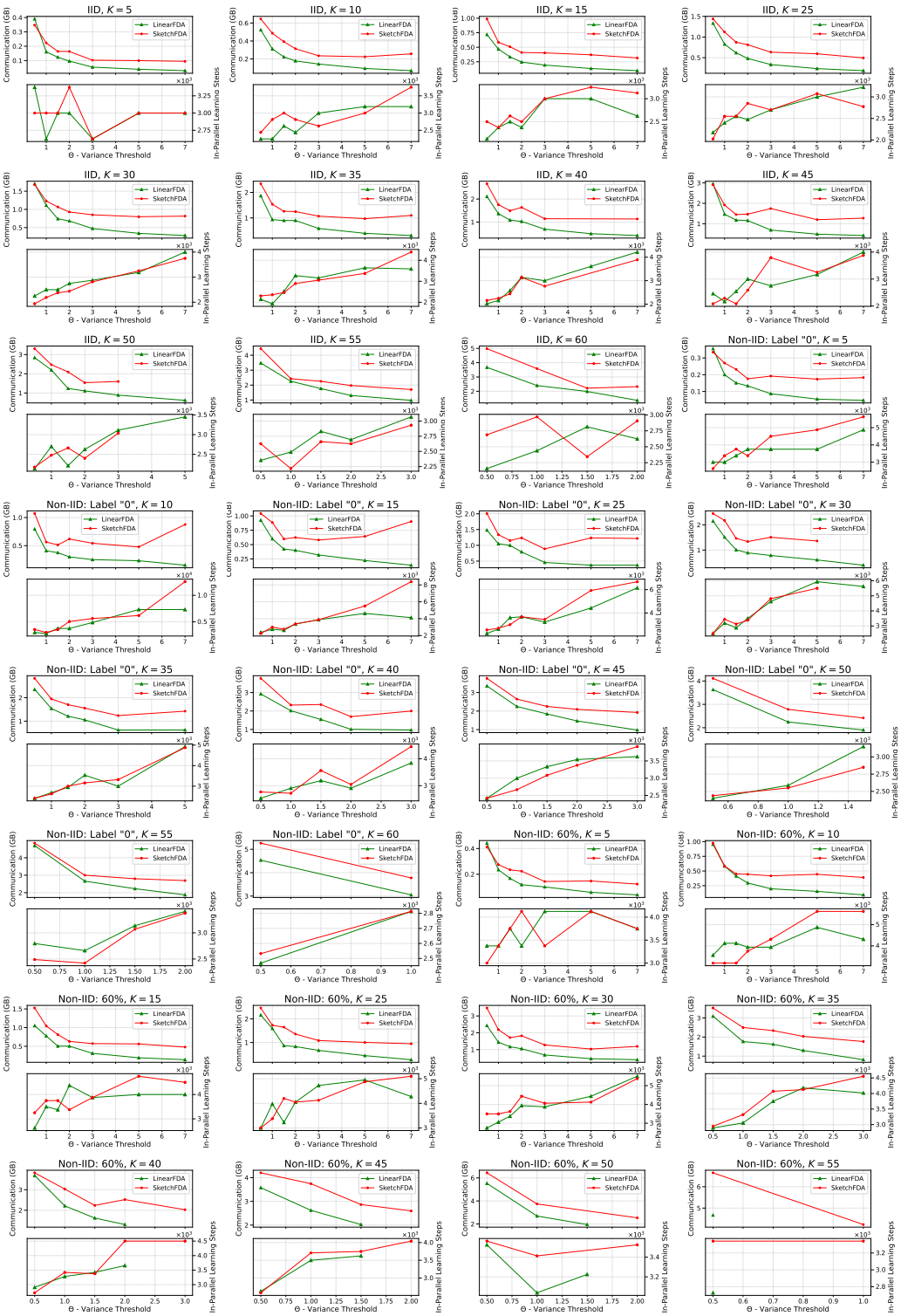


Figure 3.12: LeNet-5 on MNIST: Varying the threshold $\Theta$ — Accuracy Target: 0.98

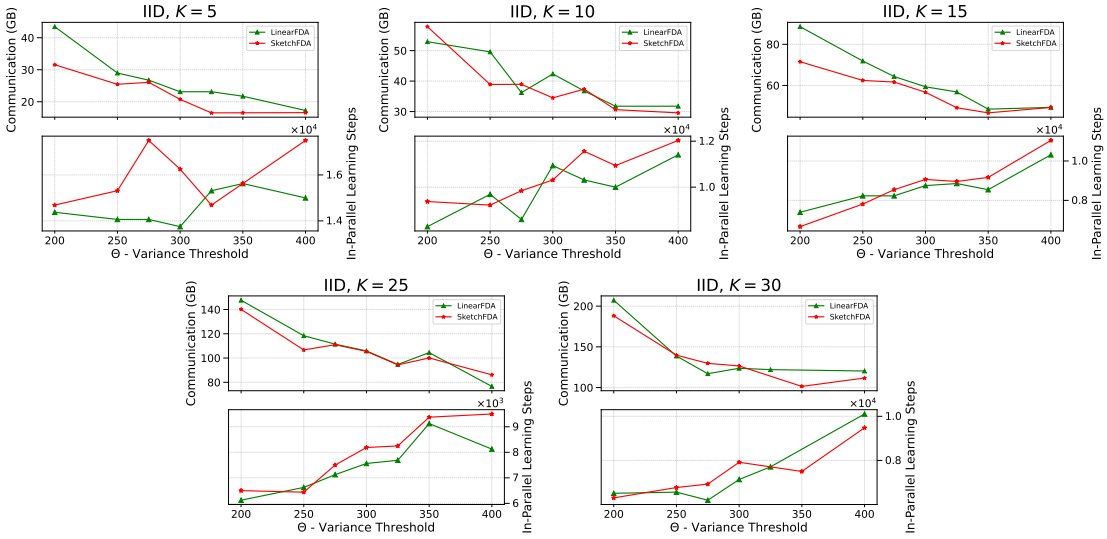


Figure 3.13: DenseNet121 on CIFAR-10: Varying $\Theta$ – Accuracy Target: 0.8

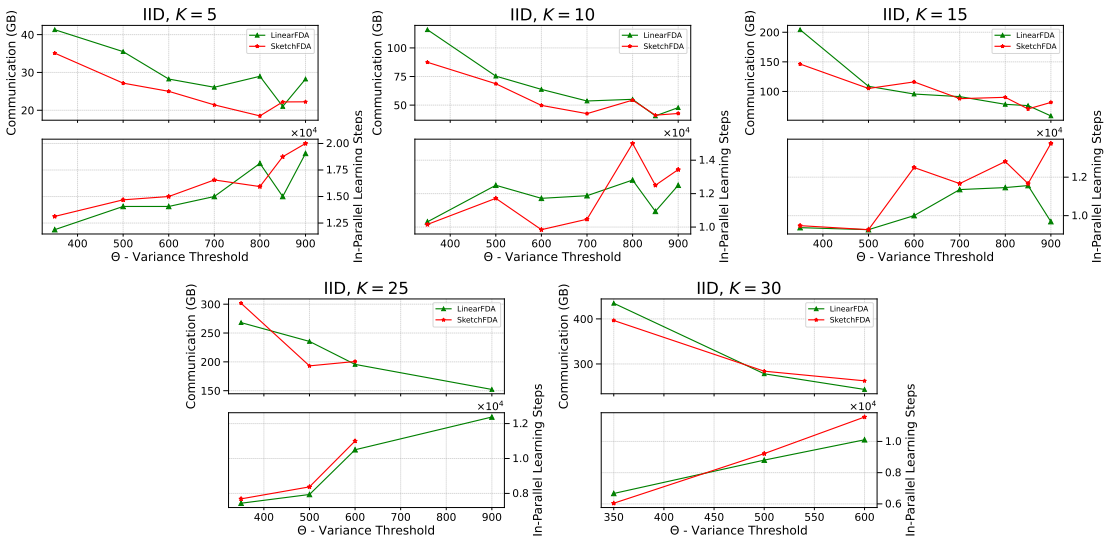


Figure 3.14: DenseNet201 on CIFAR-10: Varying $\Theta$ – Accuracy Target: 0.78

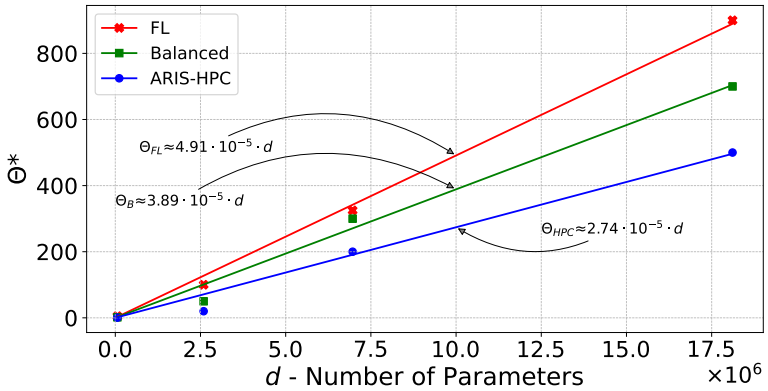


Figure 3.15: Empirical Estimation of the Variance Threshold

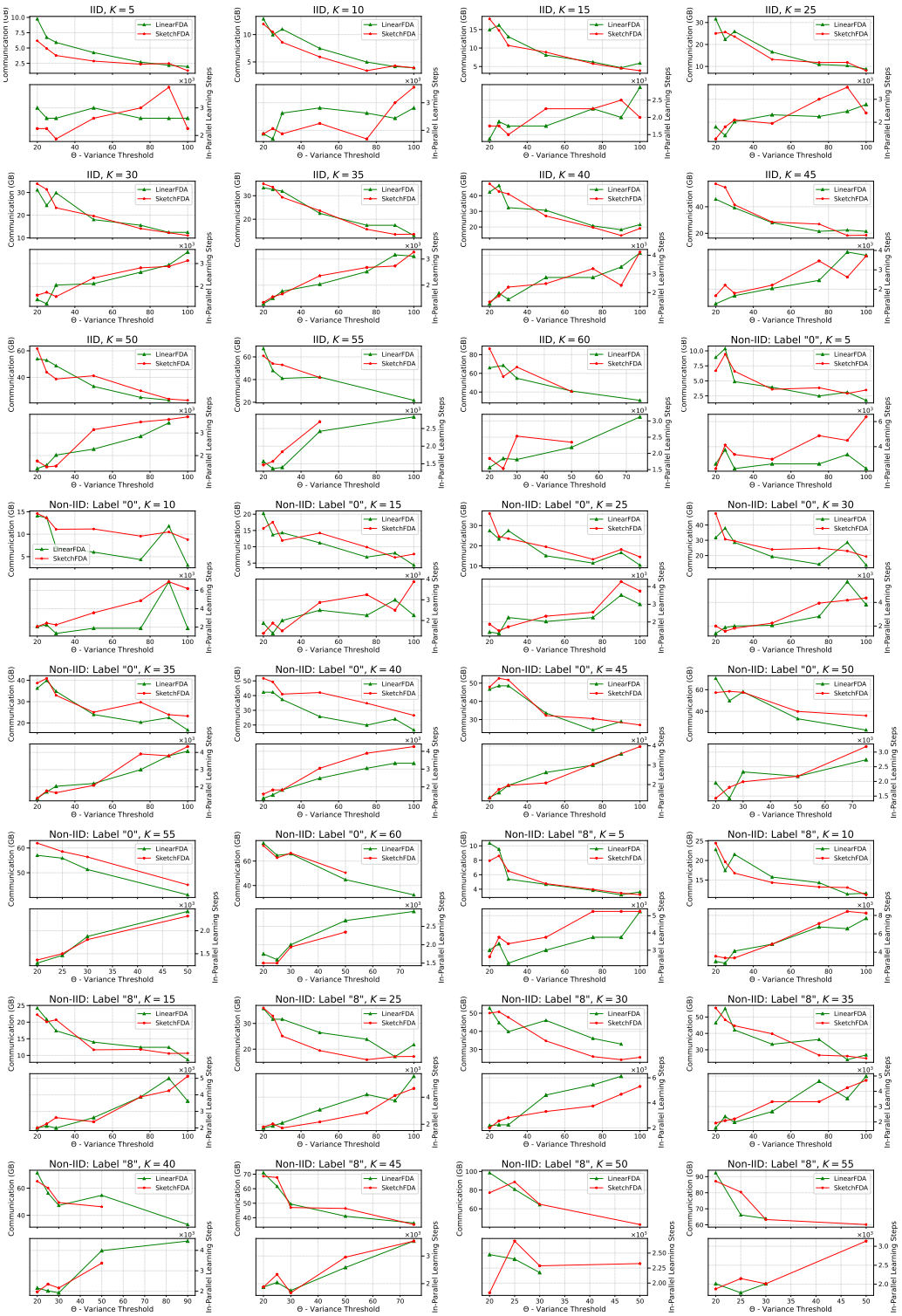


Figure 3.16: VGG16* on MNIST: Varying the threshold $\Theta$ — Accuracy Target: 0.994

communication at the cost of potentially increased computation to achieve convergence. This impact of $\Theta$ is consistently observed across both FDA strategies, and all learning tasks, and data heterogeneity settings (Figures 3.12, 3.13, 3.14, and 3.16). Interestingly, for more complex models like DenseNet121 and DenseNet201 on CIFAR-10, increasing the variance threshold ($\Theta$) does not lead to a significant rise in computation cost, as illustrated in Figures 3.13 and 3.14. It suggests that the FDA methods, by strategically timing synchronizations (monitoring the variance), substantially reduce the number of necessary synchronizations without a proportional increase in computation for the same model performance; this is particularly promising for complex DDL tasks.

**FDA: Choice of $\Theta$.** The experimental results suggest that selecting any $\Theta$ within a specific order of magnitude (e.g., between $10^2$ and $10^3$ for DenseNet201) ensures convergence, as demonstrated in Figures 3.12, 3.16, 3.13, and 3.14. Therefore, identifying this range becomes crucial. To this end, we conducted extensive exploratory testing to estimate the $\Theta$ ranges for each learning task which are predominantly influenced by the number of parameters $d$ of the DNN. Within this context, $\Theta$ values outside the desirable range exhibit notable effects: below this range, the training process mimics Synchronous or Local-SGD approaches with small $\tau$, while exceeding it leads to non-convergence. Subsequently, having identified the optimal ranges for $\Theta$, we selected diverse values within them for our experimental evaluation (Table 3.2), thereby investigating different computation and communication trade-offs. For instance, in the ARIS-HPC environment with an InfiniBand connection (up to 56 Gb/s), experiments show that training wall-time (the total time required for the computation and the communication of the DDL) is predominantly influenced by the computation cost, rendering communication concerns negligible. In such contexts, lower $\Theta$ values are favored due to their computational efficiency. On the contrary, in FL settings, where communication typically poses the greater challenge, opting for higher $\Theta$ values proves advantageous; reduction in communication achieved with higher $\Theta$ values will translate in a large reduction in total wall-time.

To assist researchers in selecting the variance threshold, Figure 3.15 presents empirical estimations for $\Theta$ across three distinct learning settings:

1. FL, assuming a common channel of 0.5Gbps, where

$$\Theta_{FL} = 4.91 \cdot 10^{-5} \cdot d$$

2. Balanced communication-computation equilibrium, where

$$\Theta_{B} = 3.89 \cdot 10^{-5} \cdot d$$

3. Our HPC environment at the ARIS supercomputer, where

$$\Theta_{HPC} = 2.74 \cdot 10^{-5} \cdot d$$

**FDA: Linear vs. Sketch.** In our main body of experiments, across most learning tasks and data heterogeneity settings, the two proposed FDA methods exhibit comparable performance, as illustrated in Figures 3.3, 3.4, 3.5, and 3.6. This suggests that the precision of the variance approximation is not critical. However, in all experiments within the more intricate transfer learning scenario, LinearFDA requires approximately 1.5 times more communication than SketchFDA to fine-tune the deep ConvNeXtLarge model to equivalent performance levels (Figure 3.17). In light of these findings, we conclude the following: for straightforward and less demanding tasks, LinearFDA is the recommended option due to its simplicity and lower complexity per local state computation. On the other hand, for intricate learning tasks and deeper models, SketchFDA becomes the preferred choice, if communication-efficiency is paramount.

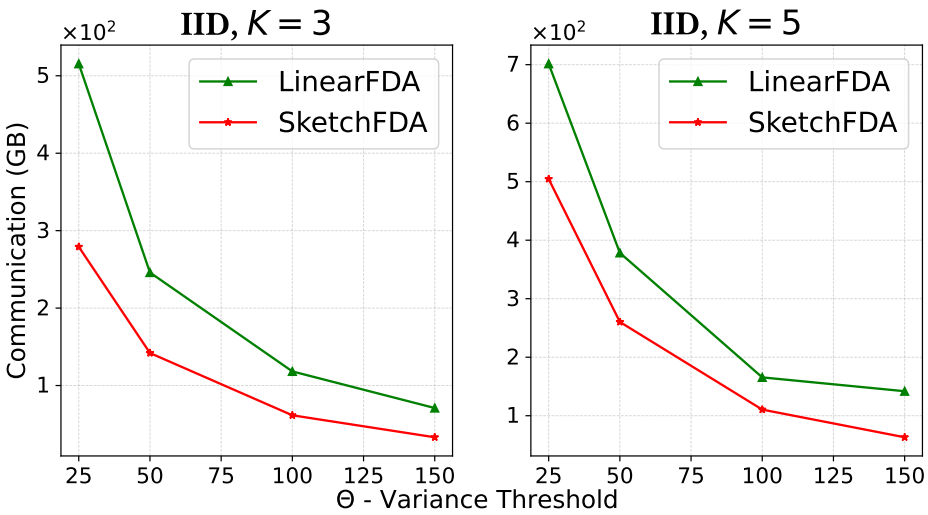


Figure 3.17: ConvNeXtLarge on CIFAR-100 (transfer learning from ImageNet) — Deployment of FDA during the fine-tuning stage with Accuracy Target of 0.76

## 3.5 Key Takeaways

In this chapter, we introduced Federated Dynamic Averaging (FDA), an innovative, adaptive and communication-efficient algorithm for distributed deep learning. Essentially, FDA makes informed, dynamic decisions on when to synchronize the local models based on approximations of the model variance. Through extensive experiments across diverse datasets and learning tasks, we demonstrated that FDA significantly reduces communication overhead (often by orders of magnitude) without a corresponding increase in computation or compromise in model performance—contrary to the typical trade-offs encountered in the literature. Furthermore, we showed that FDA is robust to data heterogeneity and inherently mitigates over-fitting. Our results push the limits of modern communication-efficient distributed deep learning, paving the way for more scalable, dynamic, and broadly applicable strategies.

# 4. Generalized Federated Dynamic Averaging

## 4.1 Motivation

In this chapter, we propose the FDA-Opt family of algorithms that generalizes both FDA and FedOpt and addresses their core shortcomings. Most importantly, we demonstrate that our proposed FDA-Opt algorithms can be directly plugged into modern FL libraries as drop-in replacements for FedOpt. They require no additional configuration and, in fact, perform better even when using hyper-parameters originally tuned for FedOpt. Our contributions are as follows:

- We propose the FDA-Opt family of algorithms, a unified generalization of both FDA and FedOpt. In doing so, we introduce a dynamic scheme for the variance threshold, removing the need for any manual configuration, and completely alleviate the original synchronization bottleneck.
- We show that FDA-Opt outperforms FedOpt in communication-efficiency, achieving improvements of at least 2× while operating under conditions optimized for FedOpt (the competitor). Specifically, we manually identify the optimal hyper-parameter configurations for each FedOpt algorithm, and then apply the same configurations to our proposed FDA-Opt counterparts. Remarkably, even with these "unfair" and "rigged" hyper-paremeters, FDA-Opt achieves 2× greater communication-efficiency for the same model performance. This has the important implication that hyper-parameters proven effective for FedOpt in the literature can be directly leveraged to configure our proposed algorithms.
- We show that all FDA-Opt algorithms converge to 5×–10× lower training loss than their FedOpt counterparts within the same number of rounds.

The remainder of this chapter is organized as follows. Section 4.3 presents our proposed FDA-Opt algorithm. Section 4.4 outlines the experimental approach. Section 4.5 provides a detailed analysis of experimental results. Section 4.6 reviews related work. Finally, Section 4.7 summarizes the main findings of this chapter.

## 4.2 FDA: Revisited

FDA (see Chapter 3) has a key distinction from FedAvg: instead of fixing the number of local steps $\tau$ per round, FDA dynamically decides when clients should stop training and return their models. Specifically, it monitors the *model variance*, and terminates local training once this variance exceeds a predefined threshold $\Theta$. During each round $t$, clients periodically send small pieces of information to the server, which uses it to estimate the model variance and transmit this estimate back to the clients (see Figure 4.1). Specifically, after every local step $i$, each client $k \in \mathcal{S}_t$ computes its local model update $\boldsymbol{\Delta}_{t,i}^{(k)}$ and

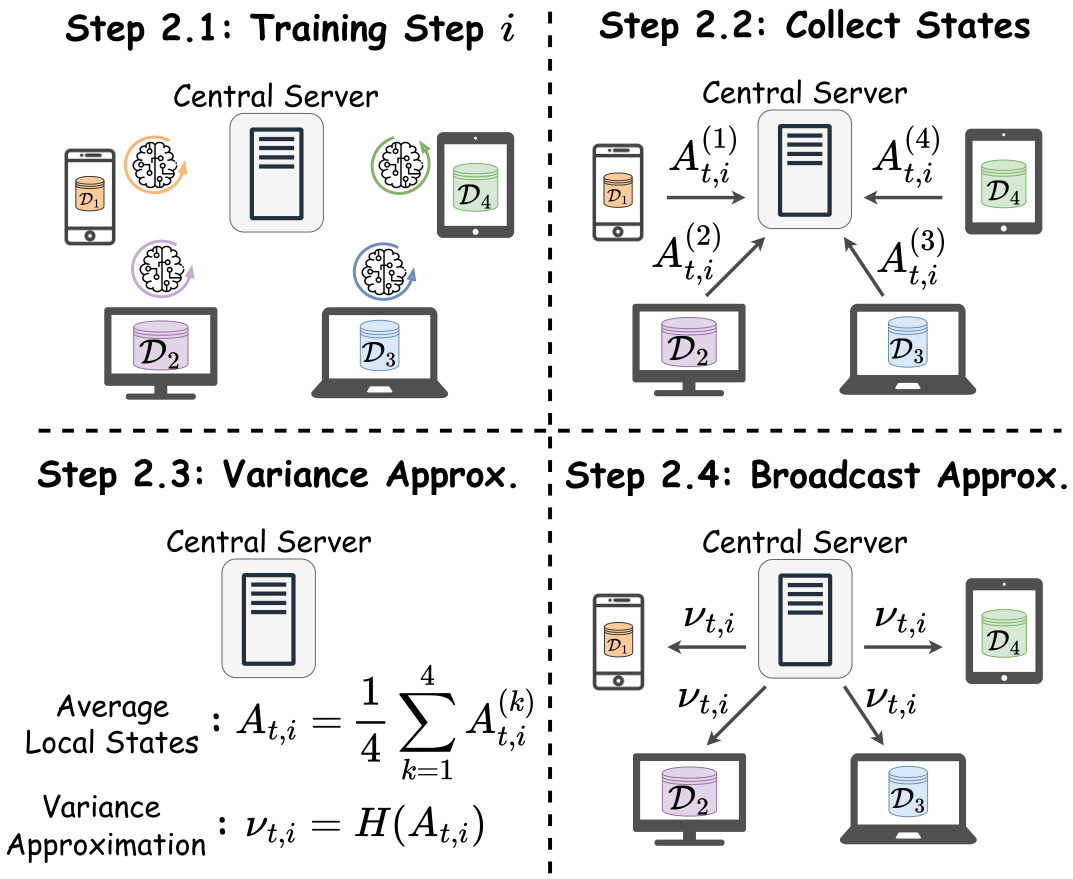


Figure 4.1: Mechanics of local training, in round $t$, under FDA

encodes it into a compact representation:

$$\mathbf{A}_{t,i}^{(k)} = \left[\|\boldsymbol{\Delta}_{t,i}^{(k)}\|_2^2,\ \mathrm{sk}(\boldsymbol{\Delta}_{t,i}^{(k)})\right] \tag{4.1}$$

where $\mathrm{sk}(\cdot)$ is the AMS sketch operator [72, 12]. These updates are highly compressed (e.g., in our experiments, a model might be 1GB, but a sketch is just 10KB). The server averages these to form the global state $\mathbf{A}_{t,i}$, and computes an estimate of the variance $\nu_{t,i}$:

$$\nu_{t,i} = H(\mathbf{A}_{t,i}) \quad , \quad \mathbf{A}_{t,i} = \frac{1}{|\mathcal{S}_t|}\sum_{k\in\mathcal{S}_t} \mathbf{A}_{t,i}^{(k)}$$

Here, $H$ is a function that maps the global state to a variance approximation (we refer to Chapter 3 for the full details). Importantly, if a threshold violation is detected, $\nu_{t,i} > \Theta$, training stops, and the server collects models; otherwise, local training proceeds to the next SGD step, $i+1$, and Steps 2.1–2.4 repeat (Figure 4.1).

In many ways, FDA resembles the original formulation of FedAvg. In the next section, we will extend its capabilities in a manner analogous to how FedOpt generalized FedAvg.

## 4.3 FDA-Opt: Generalized Federated Dynamic Averaging

In this section, we propose FDA-Opt, the unified generalization of both FDA and FedOpt, with the following advancements:

1. **FL Adaptations**: We adapt the algorithm to accommodate FL-specific considerations, including client selection strategies and notation overlooked in the original design.
2. **Generalized Averaging**: We enhance server-side aggregation by incorporating adaptive and accelerated optimizers, extending beyond the original simple averaging scheme.
3. **Dynamic Variance Threshold**: We introduce a novel dynamic mechanism that automatically calibrates the variance threshold during training, eliminating the need for any manual tuning.
4. **Unified Configuration**: We ensure that FDA-Opt shares the exact same hyperparameters as FedOpt, allowing it to be configured using well-established settings from prior work.
5. **Alleviate Synchronization Bottleneck**: We relax the original rigid local-state synchronization requirement by allowing larger intervals between variance approximations.

### 4.3.1 Optimizers

In the original FDA algorithm, the server-side aggregation relied on a simple averaging scheme. However, extensive empirical evidence highlights the effectiveness of adaptive optimizers in FL [58]. Naturally, we extend the server-side aggregation step by introducing the use of arbitrary optimizers in FDA-Opt, denoted as ServerOpt.

Intuitively, the client optimizer, ClientOpt, focuses on minimizing the local objective for each client based on its private data, while the server optimizer, ServerOpt, operates from a global perspective. By employing adaptive or accelerated optimizers at the server, we can now take advantage of meaningful and accurate statistics, which are inherently unavailable at the client level.

### 4.3.2 Model Variance

In this subsection, we formalize the notion of model variance, provide intuition for its components, examine how it evolves during training, and explain how it can be interpreted and used as a signal of meaningful progress or instability.

**Helpful Notation.** Consider a training round at time $t$, involving a sampled subset of participating clients $\mathcal{S}_t = \{k_1, k_2, \ldots, k_N\}$, where $N = |\mathcal{S}_t|$. Moreover, let $i$ be the local training step index. We can organize the client models into a matrix, $\boldsymbol{W}_{t,i} \in \mathbb{R}^{d \times N}$, defined as:

$$\boldsymbol{W}_{t,i} \triangleq \left[\boldsymbol{w}_{t,i}^{(k_1)}, \boldsymbol{w}_{t,i}^{(k_2)}, \ldots, \boldsymbol{w}_{t,i}^{(k_N)}\right]$$

This notation allows us to represent the collective state of the sampled clients in a compact and intuitive manner.

Additionally, as detailed in Section 2.2.2, each client's model change, $\boldsymbol{\Delta}_{t,i}^{(k)}$, captures the drift after the $i$-th local training step. Furthermore, their average, $\boldsymbol{g}_{t,i}$, can be interpreted as a "pseudo"-gradient:

$$\boldsymbol{g}_{t,i} = -\frac{1}{|\mathcal{S}_t|} \sum_{k \in \mathcal{S}_t} \boldsymbol{\Delta}_{t,i}^{(k)} \quad , \quad \boldsymbol{\Delta}_{t,i}^{(k)} = \boldsymbol{w}_{t,i}^{(k)} - \boldsymbol{w}_{t,0}^{(k)}$$

**Definition.** The *model variance* quantifies the spread (or dispersion) of the client models around their average—indicating how compactly they are clustered in parameter space.

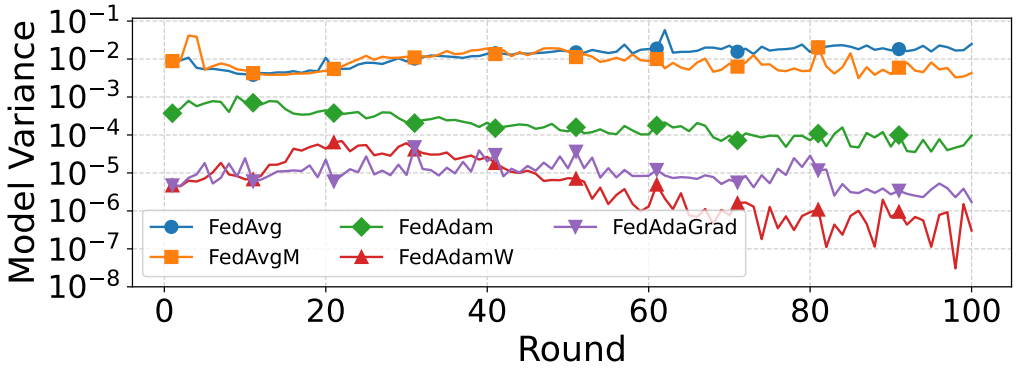


Figure 4.2: Variance progression during the first 100 rounds of training with FedOpt using RoBERTa on the MRPC dataset

At round $t$, after $i$ local training steps, it can be written as [72]:

$$\mathrm{Var}(\boldsymbol{W}_{t,i}) = \overbrace{\frac{1}{|\mathcal{S}_t|}\sum_{k\in\mathcal{S}_t}\|\boldsymbol{\Delta}_{t,i}^{(k)}\|_2^2}^{1^{\text{st}}\text{ term}} - \underbrace{\|\boldsymbol{g}_{t,i}\|_2^2}_{2^{\text{nd}}\text{ term}} \tag{4.2}$$

**Intuition.** During a training round $t$, each client $k$ updates its model, resulting in a local change $\boldsymbol{\Delta}_{t,i}^{(k)}$. The squared norm of this change, $||\boldsymbol{\Delta}_{t,i}^{(k)}||_2^2$, quantifies how far the client's model has moved from the start of the round (i.e., from the original global model). Essentially, this reflects the amount of information the client has learned locally. The first term in Equation (4.2), the average of these quantities across all participating clients, represents the *collective amount of information learned*.

However, a large first term does not necessarily mean that training is progressing well, as clients may be moving in conflicting directions, canceling out their updates by averaging. What ultimately matters is the global change $\boldsymbol{g}_{t,i}$ obtained after averaging—it foreshadows the information that will be retained in the global model. This is captured by the second term, $||\boldsymbol{g}_{t,i}||_2^2$, which inherently accounts for the *direction* of the local changes.

The variance, as defined in Equation (4.2), reflects the interplay between these two quantities serves as an insightful gauge of the state of the training progress:

1. **Low Variance** indicates either minimal local progress (both terms low), or substantial local progress in a cohesive and promising way (both terms high).
2. **High Variance** indicates significant local progress (high first term) but towards conflicting local minima (low second term).

**Trends.** What do we really mean by *low* or *high* variance? The answer is that these terms are inherently *relative* and task-specific, influenced by factors like the dataset, model size, optimizer, and hyper-parameters. To illustrate this, we investigate the value of the variance at the end of each FL round under the fixed-round termination schedule of FedOpt. Figures 4.2-4.6 plot the variance at the exact moment when model updates are collected from the server. While the experimental setup—including the specific model and datasets—will be detailed later, we present these plots here to help readers intuitively grasp the variance as a metric and its trends.

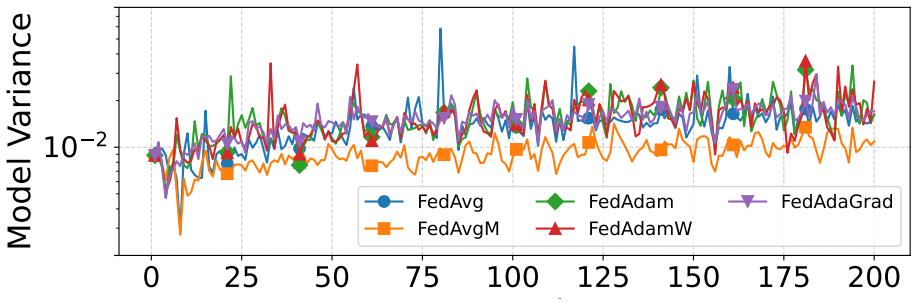


Figure 4.4: Variance progression during the first 200 rounds of training with FedOpt using RoBERTa on the QNLI dataset

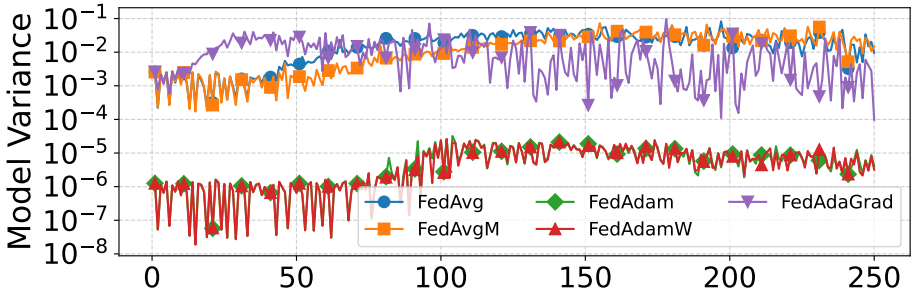


Figure 4.3: Variance progression during the first 250 rounds of training with FedOpt using RoBERTa on the RTE dataset

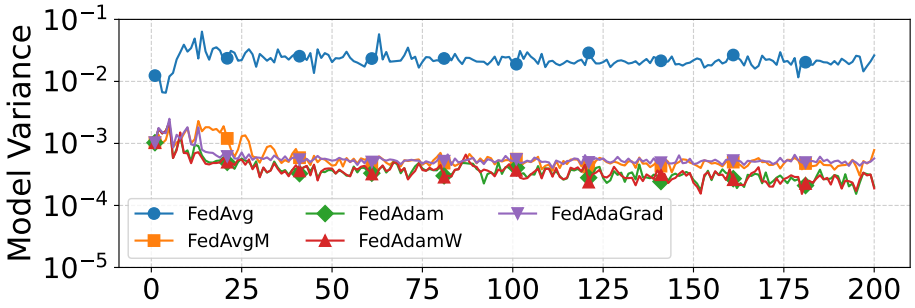


Figure 4.5: Variance progression during the first 200 rounds of training with FedOpt using RoBERTa on the SST-2 dataset

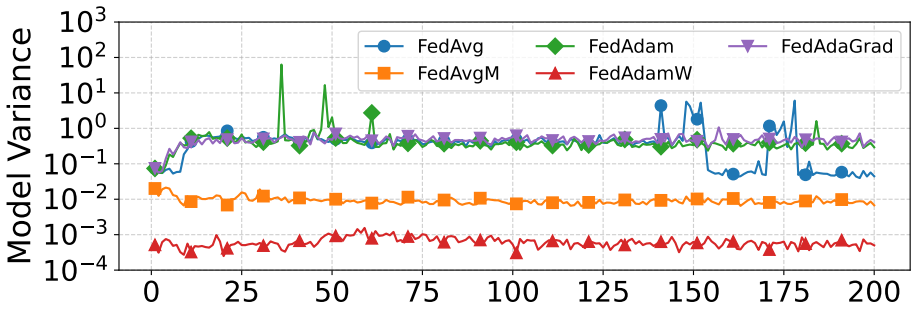


Figure 4.6: Variance progression during the first 200 rounds of training with FedOpt using DeBERTaV3 on the MNLI dataset

These plots underscore the critical need to revisit the originally proposed static variance

monitoring scheme. Variance trends are inherently dynamic, evolving throughout training, and thus require an adaptive approach to be captured effectively. Moreover, different algorithms exhibit vastly different variance magnitudes (please note that the y-axis are logarithmic) and behaviors. Hence, variance trends should only be interpreted within the context of the same training run. Additionally, the magnitude of the variance may evolve even for the same optimizer by 1-3 orders of magnitude, depending on whether the training is in its early, middle, or late stages.

All in all, *high* variance does not necessarily indicate that the model will converge to a suboptimal solution or that training is progressing poorly. Rather, high variance is interpreted as a sign of misalignment and instability when viewed *in relation* to earlier training rounds of the same run/optimizer.

### 4.3.3 The FDA-Opt Algorithm

**High-Level Overview.** Having established model variance as a key indicator of training dynamics—with distinct behaviors and trends—we now leverage these insights for a unique goal: *to extend local training duration beyond traditional limits without compromising convergence*. To this end, we the propose FDA-Opt, presented in Algorithm 7. The highlighted lines represent the core additions that distinguish FDA-Opt: namely, the logic for monitoring model variance and dynamically terminating rounds. These additions act as an adapter to the original FedOpt framework (non-highlighted lines, also detailed in Section 2.2.2), effectively generalizing it with an dynamic round termination scheme.

**Operational Details.** The algorithm begins with a user-provided local training duration $\tau$. This value is expected to be based on well-established and empirically effective configurations from prior FedOpt work. Then, FDA-Opt modifies this internally by extending it to a larger value $\tilde{\tau} \gg \tau$ (e.g., $\tilde{\tau} = 10 \cdot \tau$; see Line 2).

From this point onward, the algorithm proceeds using the standard round-based approach described in Section 2.2.2. At the beginning of each round $t$, a cohort of clients $\mathcal{S}_t$ is sampled. These clients train locally for up to $\tilde{\tau}$ steps (Lines 3–9). At the end of training—whose actual length may vary—we compute the "pseudo"-gradient from model changes and update the global model accordingly (Lines 18 and 21). This process is repeated for a total of $T$ rounds. What sets FDA-Opt apart is its variance monitoring logic.

Specifically, during local training, clients periodically synchronize compact state information (Lines 11–12) to approximate the model variance (Line 13). However, querying variance too frequently (e.g., after every local step) can completely bottleneck a real-world FL system. This is not due to communication volume (the local state states are small), but due to the synchronous nature of the operation. To mitigate this, we introduce a configurable set of step indices, $\mathcal{I}_{\text{query}} \subseteq [1, 2, ..., \tilde{\tau}]$, which determines the exact points where variance queries are performed (Line 10). In contrast to the original FDA formulation [72], which used $\mathcal{I}_{\text{query}} = [1, 2, ..., \tilde{\tau}]$ (i.e., checking after every step), our method leaves this set user-defined. In practice, we encourage it to be significantly sparser. As we will show later, in our setup the variance is queried only once per epoch.

Lastly, following the discussion on variance behavior in Section 4.3.2, we allow the threshold value, $\Theta_t$, to evolve across training rounds. Rather than using a fixed value, the threshold is updated dynamically based on the observed training dynamics. Specifically, we define a function ThresholdAdjust, which takes as input the current threshold $\Theta_t$, the actual variance value at the end of the round, and the final step index $s_t$ where the round terminated (Lines 19–20). Intuitively, these inputs capture the key information needed to answer two questions: (1) *was the current threshold too high or too low?* and (2) *did the round terminate at a reasonable point (e.g., $s_t = 1$ should raise concerns)?* In

**Algorithm 7** FDA-Opt

**Input**: Initial $\boldsymbol{w}_0$; ClientOpt, ServerOpt; Total rounds $T$; Local training steps $\tau$, specifically tuned for FedOpt [58]

1: Set $\Theta_0 = -\infty$
2: Choose extended $\tilde{\tau} \gg \tau$ ▷ FDA-Opt extends local training
3: **for each** round $t = 0, \ldots, T-1$ **do**
4: Sample a subset $\mathcal{S}_t \subseteq \mathcal{K}$ of clients
5: Set $\boldsymbol{w}_{t,0}^{(k)} = \boldsymbol{w}_t$ for all $k \in \mathcal{S}_t$
6: **for each** client $k \in \mathcal{S}_t$ **in parallel do**
7: **for each** local step $i = 1, ..., \tilde{\tau}$ **do**
8: Compute a gradient estimate $\boldsymbol{g}_{t,i-1}^{(k)}$ of $\nabla F_k(\boldsymbol{w}_{t,i-1}^{(k)})$
9: $\boldsymbol{w}_{t,i}^{(k)} = \text{ClientOpt}(\boldsymbol{w}_{t,i-1}^{(k)}, \boldsymbol{g}_{t,i-1}^{(k)})$
10: **If** $i \in \mathcal{I}_{\text{query}}$ **then** ▷ Query variance now
11: Construct local state $\mathbf{A}_{t,i}^{(k)}$ using (4.1)
12: Aggregate states $\mathbf{A}_{t,i} = \frac{1}{|\mathcal{S}_t|} \sum_{k \in \mathcal{S}_t} \mathbf{A}_{t,i}^{(k)}$
13: Variance approx. $\nu_{t,i} = H(\mathbf{A}_{t,i})$ ▷ See [72]
14: **If** $\nu_{t,i} > \Theta_t$ **then** ▷ Threshold violation
15: $s_t = i$ ▷ Round's final step index
16: **break** ▷ Training round terminates
17: $\boldsymbol{\Delta}_{t,s_t}^{(k)} = \boldsymbol{w}_{t,s_t}^{(k)} - \boldsymbol{w}_{t,0}^{(k)}$ ▷ Local model change
18: $\boldsymbol{g}_{t,s_t} = -\frac{1}{|\mathcal{S}_t|} \sum_{k \in \mathcal{S}_t} \boldsymbol{\Delta}_{t,s_t}^{(k)}$ ▷ "Pseudo"-gradient
19: $\nu ar_t = \text{Var}(\boldsymbol{W}_{t,s_t})$ ▷ Compute actual variance using (4.2)
20: $\Theta_{t+1} = \text{ThresholdAdjust}(\nu ar_t, \Theta_t, s_t)$
21: $\boldsymbol{w}_{t+1} = \text{ServerOpt}(\boldsymbol{w}_t, \boldsymbol{g}_{t,s_t})$
22: **return** $\boldsymbol{w}_T$

the next section, we will explore practical choices for this function. Notably, computing the variance in Line 19 requires no additional work, as it reuses quantities—the drifts and "pseudo"-gradient—that are already available (Lines 17–18).

### 4.3.4 Default Configuration for FDA-Opt

We adopt a conservative approach in the configurations below. These are the exact settings used in our experiments, as they yield stable and reliable performance across all tasks. While we believe there is room for improvement—potentially through more aggressive configurations (e.g., a larger linear coefficient of increase of $\tau$)—we leave such tuning as a direction for future work, beyond the scope of this study.

**Local Training Extension.** We set $\tilde{\tau}$ to be twice the original $\tau$ plus a large constant. This constant is chosen to be eight times the size of a typical client dataset—denoted by $e$. Formally:

$$\tilde{\tau} = 2 \cdot \tau + 8 \cdot \lceil e \rceil, \quad \text{where} \quad e \approx \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} |\mathcal{D}_k| \tag{4.3}$$

**Variance Query Indices.** Within each round, we query the variance once per epoch. This completely alleviates the synchronization bottleneck encountered in the original

work [72]. We define:

$$\mathcal{I}_{\text{query}} = \{e, 2e, ..., \lfloor \tilde{\tau}/e \rfloor \cdot e\}$$

ThresholdAdjust. In general, variance within a round $t$ tends to increase as local training progresses. We set the threshold $\Theta_t$ such that it is approximately equal to the expected variance around the midpoint of local training (i.e., at step $\left\lfloor \frac{\tilde{\tau}}{2} \right\rfloor$). Assuming linear growth, we can estimate the midpoint variance of the round $t+1$ using a simple proportion, based on known values from the current round $t$—specifically, the actual variance $\nu ar_t$ and termination step $s_t$ computed in Lines 15 and 19 of Algorithm 7, respectively:

$$\begin{aligned} \nu ar_t &\xrightarrow{\text{after}} s_t \quad \text{local steps} \\ \nu ar_{t+1}^{(est)} &\xrightarrow{\text{after}} \frac{\tilde{\tau}}{2} \quad \text{local steps} \end{aligned}$$

With this linear prediction, we set the next round's threshold to the estimated variance at the midpoint of local training, $\nu ar_{t+1}^{(est)}$:

$$\Theta_{t+1} = \frac{\tilde{\tau}/2}{s_t} \cdot \nu ar_t$$

## 4.4 Experimental Approach

In this section, we outline the methodology and rationale behind evaluating the proposed FDA-Opt family of algorithms against FedOpt. Specifically, we detail the infrastructure, datasets, data partitioning strategies, and models used in our study.

### 4.4.1 Infrastructure

We conduct our experiments using the *transformers* library from Hugging Face [90] with PyTorch [54]. The code for our implementation is available at `https://github.com/miketheologitis/FDA-Opt`. All experiments are performed on a local cluster equipped with 2 NVIDIA A10 GPUs and 46 Intel(R) Xeon(R) Silver 4310 CPUs.

### 4.4.2 Tasks & Datasets

Natural Language Understanding (NLU) tasks are a core component of evaluating machine learning models in NLP. These tasks test a model's ability to understand, reason, and infer relationships between pieces of text. To conduct our experiments, we select six widely used datasets and tasks from the GLUE [78] benchmark: MRPC [16], SST-2 [64], RTE [78], QNLI [78], MNLI-m and MNLI-mm [87]—each corresponding to a distinct NLU task. Since the GLUE test sets are unpublished, we follow prior studies [94] and use the validation sets as the new test sets.

### 4.4.3 Data Partitioning

A critical aspect of FL is the distribution of data across clients, which is characterized by high heterogeneity. Most NLP datasets are designed for centralized machine learning settings. In this subsection, we describe how we partitioned these datasets for FL.

**Clients.** The scale of data federation is determined by the number of clients, which generally falls into two categories: *cross-silo* and *cross-device* [31, 56]. In cross-silo FL, the number of clients is small—such as hospitals or financial institutions—often allowing all clients to participate in each training round. For this setting, we use the MRPC and RTE datasets. In contrast, cross-device FL involves a much larger pool of clients, such as mobile or IoT devices, where only a fraction of them can participate in each round.

Table 4.1: Client Partitioning Strategies per Dataset

| | MRPC | RTE | SST-2 | QNLI | MNLI-m | MNLI-mm |
|---|---|---|---|---|---|---|
| **#Clients** | 10 | 10 | 100 | 250 | 1000 | 1000 |
| **Sample Size** | 10 | 5 | 10 | 10 | 10 | 10 |
| **Cross-Silo** | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| **Cross-Device** | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |

For this setting, we use the SST-2, QNLI, and MNLI datasets. Table 4.1 summarizes the client partitioning and sampling strategies across these datasets.

**Label Distribution.** One of the most common heterogeneity scenarios in FL is the presence of non-IID label distributions, where clients possess data with disproportionately distributed labels. For instance, one client may predominantly have samples of label A, while another may mostly contain label B. This phenomenon frequently occurs in real-world applications and is modeled using the Dirichlet distribution [39, 44, 56]. The degree of label imbalance is controlled by the concentration parameter $\alpha$. Larger values of $\alpha$ result in more uniform distributions, with $\alpha \to \infty$ producing a perfectly uniform distribution. Conversely, as $\alpha$ approaches zero, the distributions become highly imbalanced, with each client predominantly holding samples from a single label. In our experiments, we set $\alpha = 1.0$, introducing a considerable level of heterogeneity.

### 4.4.4 Datasets & Task Descriptions

In this section, we describe the datasets used in our experiments, each corresponding to a distinct NLU task.

**MRPC.** The Microsoft Research Paraphrase Corpus (MRPC) [16] consists of sentence pairs collected from news-wire articles. The goal is to classify whether or not a given pair of sentences are paraphrases of each other. The reported metric is *Accuracy*. MRPC belongs to the category of Sentence Similarity tasks. Notably, the dataset is imbalanced, with 68% of the pairs labeled as paraphrases.

**SST-2.** The Stanford Sentiment Treebank (SST-2) [64] consists of sentences from movie reviews along with human annotations of their sentiment. The goal is to predict the sentiment of a given sentence as either positive or negative. The reported metric is *Accuracy*. SST-2 belongs to the category of Sentiment Analysis tasks.

**RTE.** The Recognizing Textual Entailment (RTE) [78] consists of pairs of sentences where the task is to determine whether or not the meaning of the second sentence (the hypothesis) is entailed by the first sentence (the premise). The reported metric is *Accuracy*. RTE belongs to the category of Natural Language Inference (NLI) tasks.

**QNLI.** The Question-answering Natural Language Inference (QNLI) dataset [78] consists of question-context pairs. The goal is to determine whether or not the context sentence contains the answer to the question. The reported metric is *Accuracy*. QNLI belongs to the category of NLI tasks.

**MNLI-m/mm.** The Multi-Genre Natural Language Inference (MNLI) corpus [87] is a large-scale dataset for evaluating a model's ability to perform NLI across diverse textual domains. Each sample consists of a premise and a hypothesis, and the goal is to classify their relationship as entailment, contradiction, or neutral. The evaluation is conducted on two test sets: matched (MNLI-m) and mismatched (MNLI-mm), representing in-domain and out-of-domain genres, respectively. The reported metric is *Accuracy*.

### 4.4.5 Label Imbalance Scheme

Modeling label imbalance in a way that closely mirrors real-world scenarios has been extensively studied in the literature. The widely adopted approach is to simulate non-IID label distributions using a Dirichlet allocation scheme. The method works as follows [39, 44]:

Assume we have a dataset with $L$ labels. The label distribution for each client is parameterized by a vector $\boldsymbol{q}^{(k)} \in (0,1)^L$ with $||\boldsymbol{q}^{(k)}||_1 = 1$, where $\boldsymbol{q}_l^{(k)}$ represents the proportion of instances of label $l$ for client $k$. We sample $\boldsymbol{q}^{(k)}$ from a Dirichlet distribution, $\boldsymbol{q}^{(k)} \sim Dir_L(\alpha \boldsymbol{p})$, where $\boldsymbol{p}$ is the prior class distribution (assumed to be known) and $\alpha > 0$ is a concentration parameter. After constructing these sampled label distributions, we allocate data to each client accordingly.

The degree of label imbalance can be flexibly controlled by varying a parameter $\alpha$. Larger values of $\alpha$ result in more uniform distributions, with $\alpha \to \infty$ producing a perfectly uniform distribution across clients. Conversely, as $\alpha$ approaches zero, the distributions become highly imbalanced, with each client predominantly holding samples from a single label.

In our experiments we use a concentration parameter $\alpha = 1.0$. Figures 4.8 and 4.7 illustrate its effect on label imbalance across clients for all datasets. Moreover, these figure also showcase the slight quantity imbalance which is inevitably introduced following aforementioned scheme.

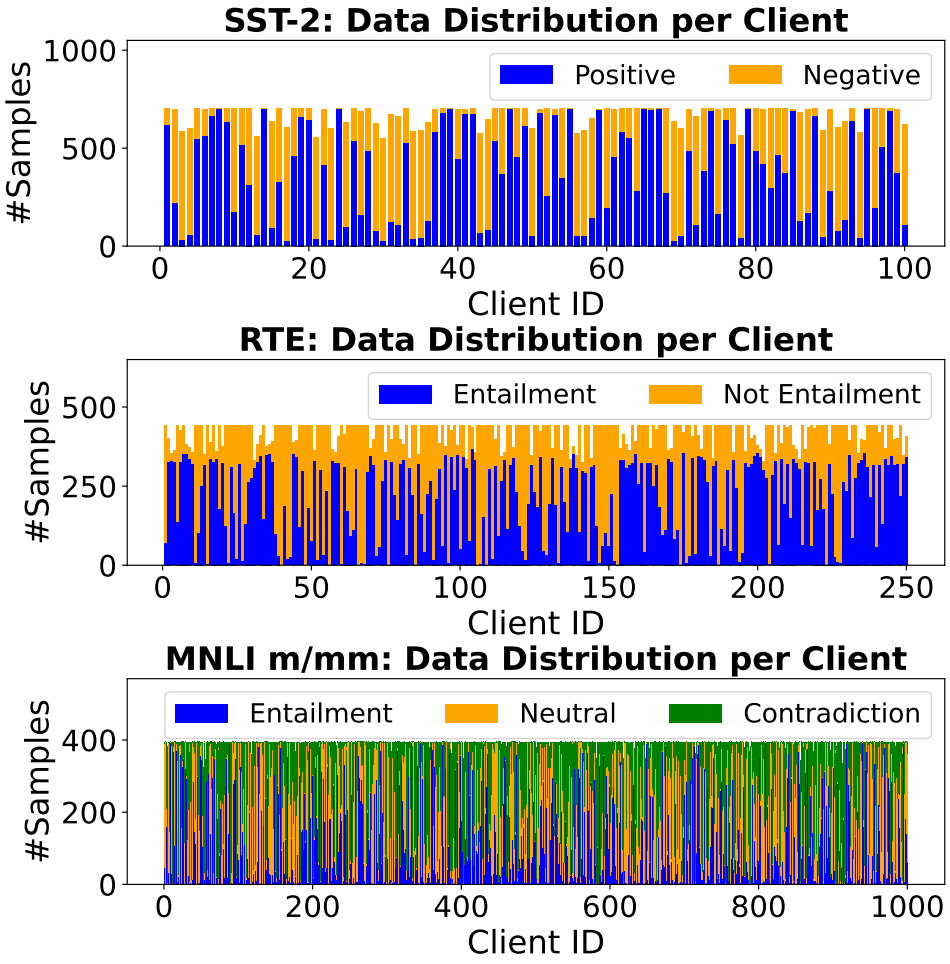


Figure 4.7: Histogram of Label and Quantity data distribution across clients for the three Cross-Device datasets. The non-IID label partitioning is done using a Dirichlet distribution with parameter $\alpha = 1.0$

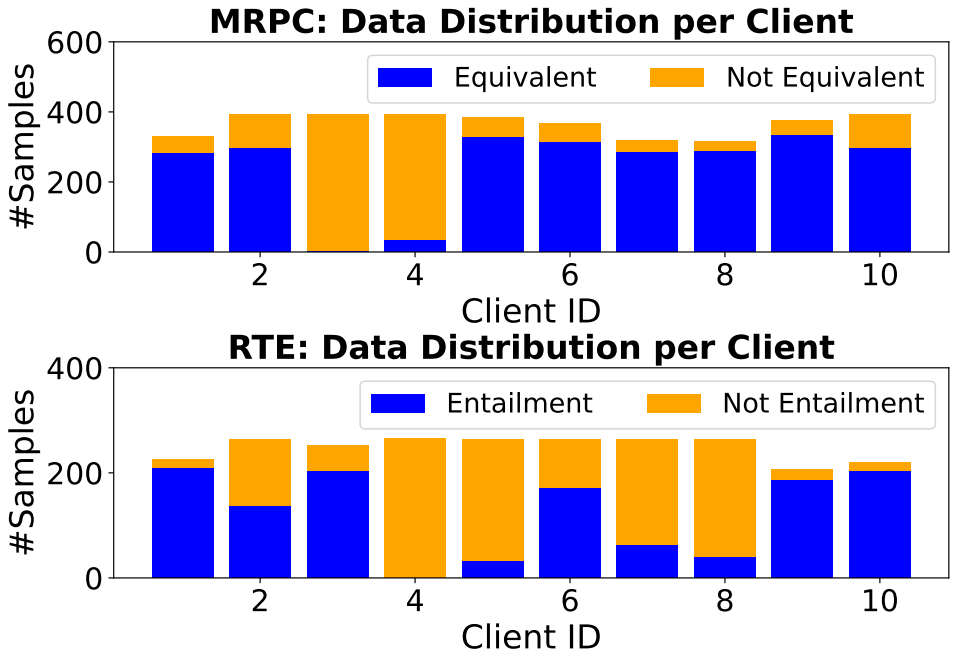


Figure 4.8: Histogram of Label and Quantity data distribution across clients for the two Cross-Silo datasets. The non-IID label partitioning is done using a Dirichlet distribution with parameter $\alpha = 1.0$

### 4.4.6 Hyper-parameter Selection

In this section we outline the approach for identifying the optimal configuration for each FedOpt algorithm when training a specific model on a specific dataset. Algorithms in the FedOpt family are highly sensitive to both client and server learning rates, and their interaction is often task-specific and unpredictable. We conduct a grid search to determine the best hyper-parameters for each dataset and algorithm (see Figures 4.9-4.14). The final selections based on this process are summarized in Table 4.2.

As a representative example, consider the RoBERTa model trained MRPC dataset. We tune learning rates using a commonly adopted search grid [79] (Figure 4.9). Specifically, we investigate server and client learning rates in $\{10^{-5}, 10^{-4}, \ldots, 10^{-1}\}$ for FedAdaGrad, following well-established guidelines [18]. We then train RoBERTa with FedAdaGrad for $T = 100$ rounds, recording the highest accuracy achieved across all rounds. As shown in Figure 4.9, the best-performing configuration for FedAdaGrad on MRPC reached 89% accuracy with server and client learning rates of $10^{-4}$ and $10^{-5}$, respectively. These values are taken as the best hyper-parameters to train RoBERTa with FedAdaGrad on MRPC.

We then apply these optimal learning rates to the FDA-Opt counterpart. Specifically, when training RoBERTa with FDA-AdaGrad on MRPC, we use the server and client learning rates of $10^{-4}$ and $10^{-5}$, respectively, as determined from the above analysis. While these values are most likely not optimal for FDA-AdaGrad, our goal is to conclusively demonstrate that FDA-Opt outperforms FedOpt and that hyper-parameters proven effective for FedOpt not only work for FDA-Opt but also yield better results. This has the powerful implication that we can seamlessly replace all FDA-Opt algorithms with FedOpt.

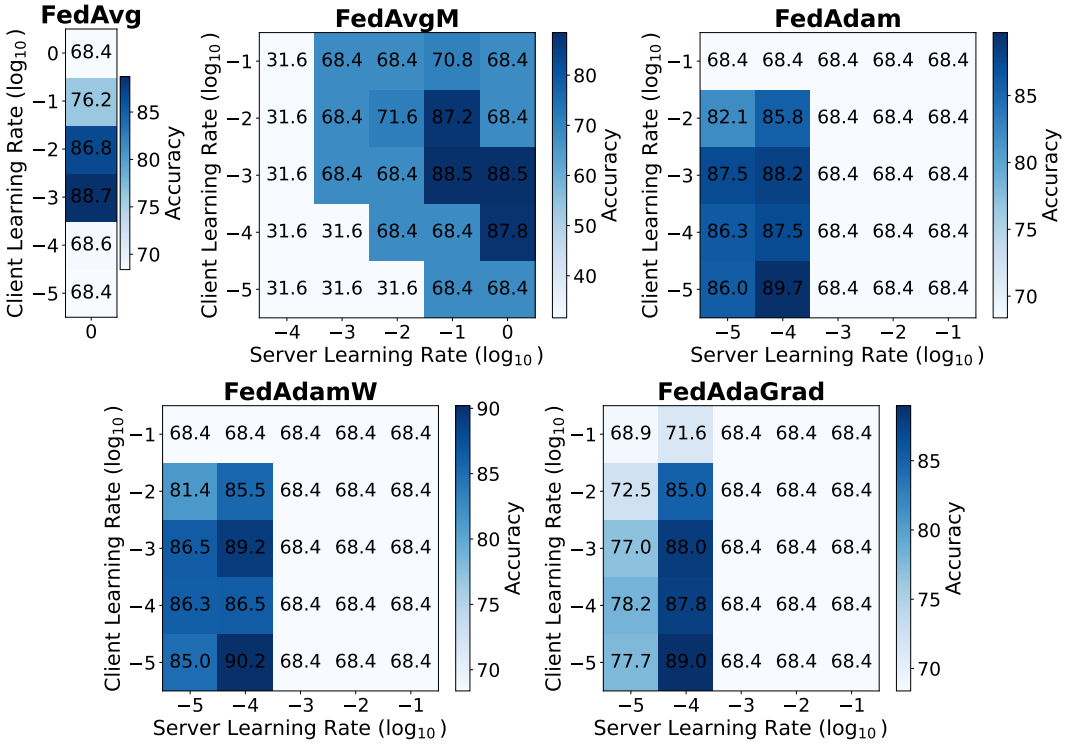


Figure 4.9: The highest testing accuracy of RoBERTa on the MRPC dataset during the first 100 rounds of training, across various client and server optimizer learning rates.

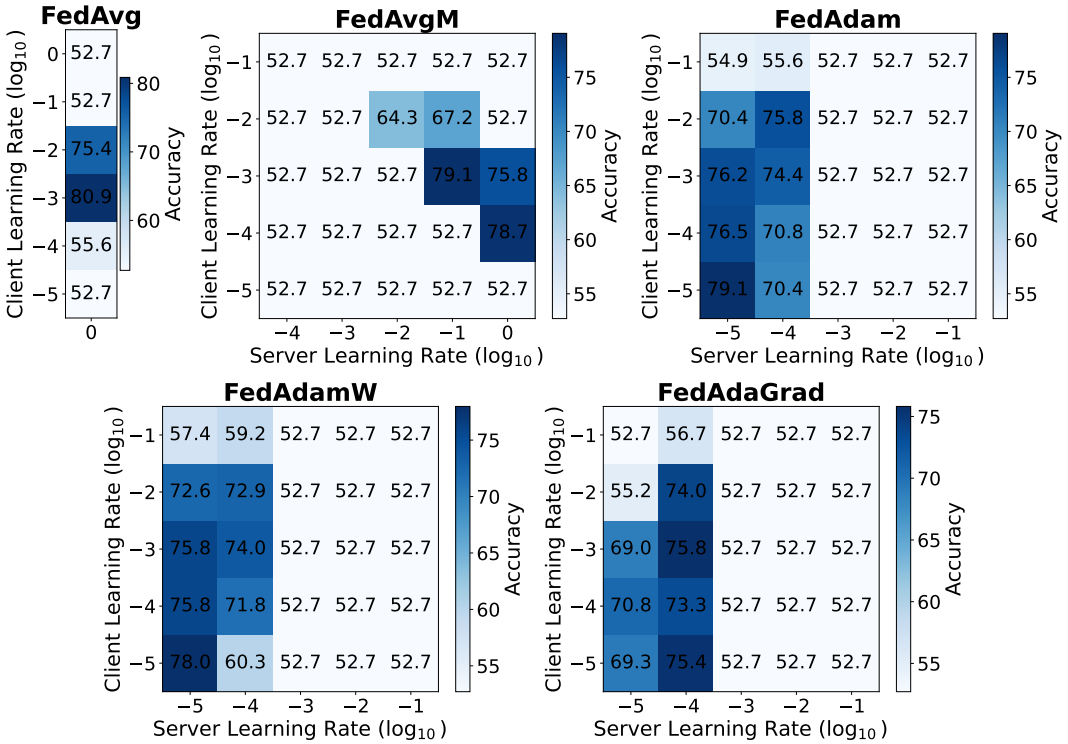


Figure 4.10: The highest testing accuracy of RoBERTa on the RTE dataset during the first 250 rounds of training, across various client and server optimizer learning rates.

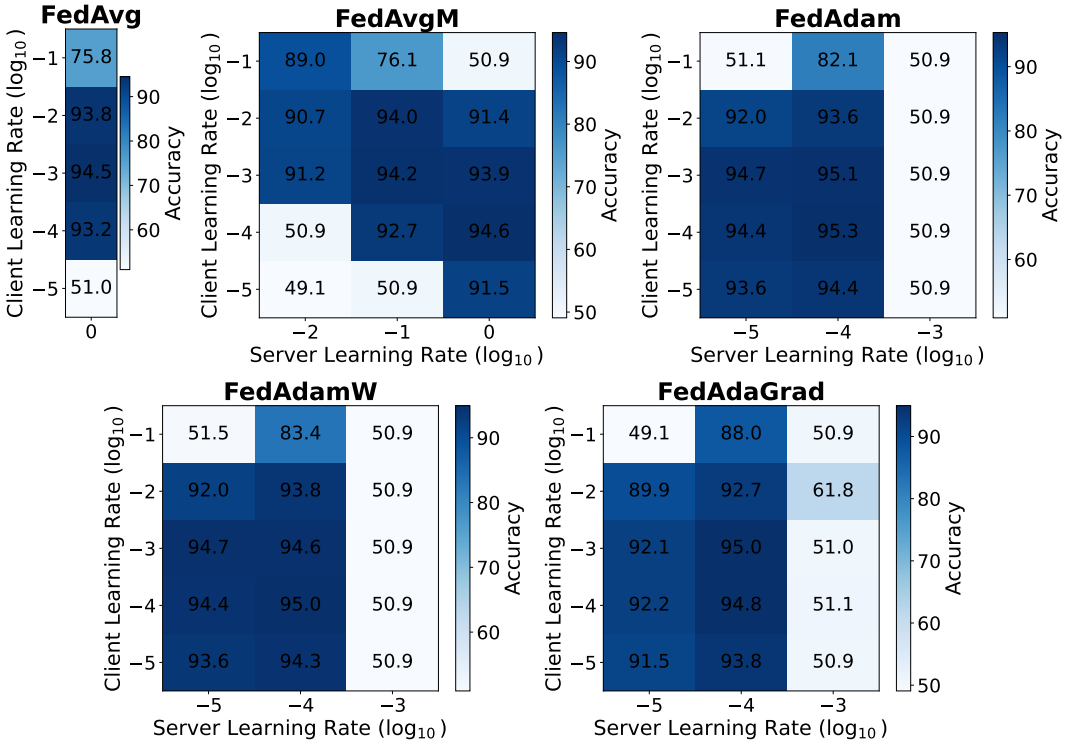


Figure 4.11: The highest testing accuracy of RoBERTa on the SST-2 dataset during the first 200 rounds of training, across various client and server optimizer learning rates.

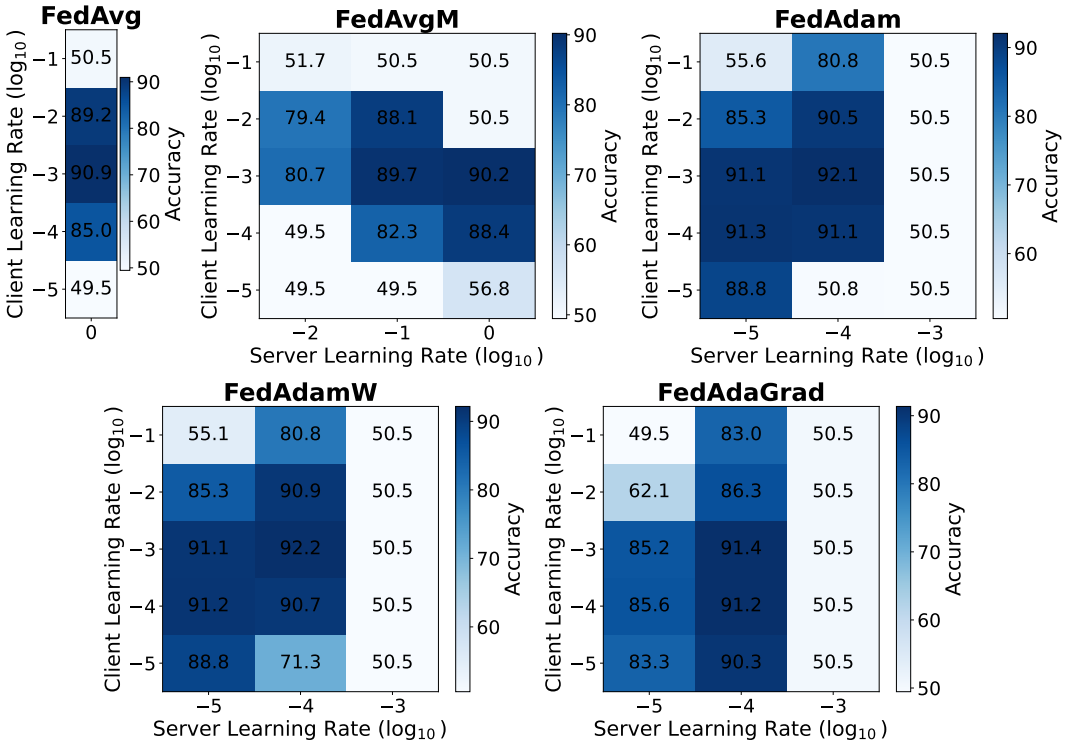


Figure 4.12: The highest testing accuracy of RoBERTa on the QNLI dataset during the first 200 rounds of training, across various client and server optimizer learning rates.

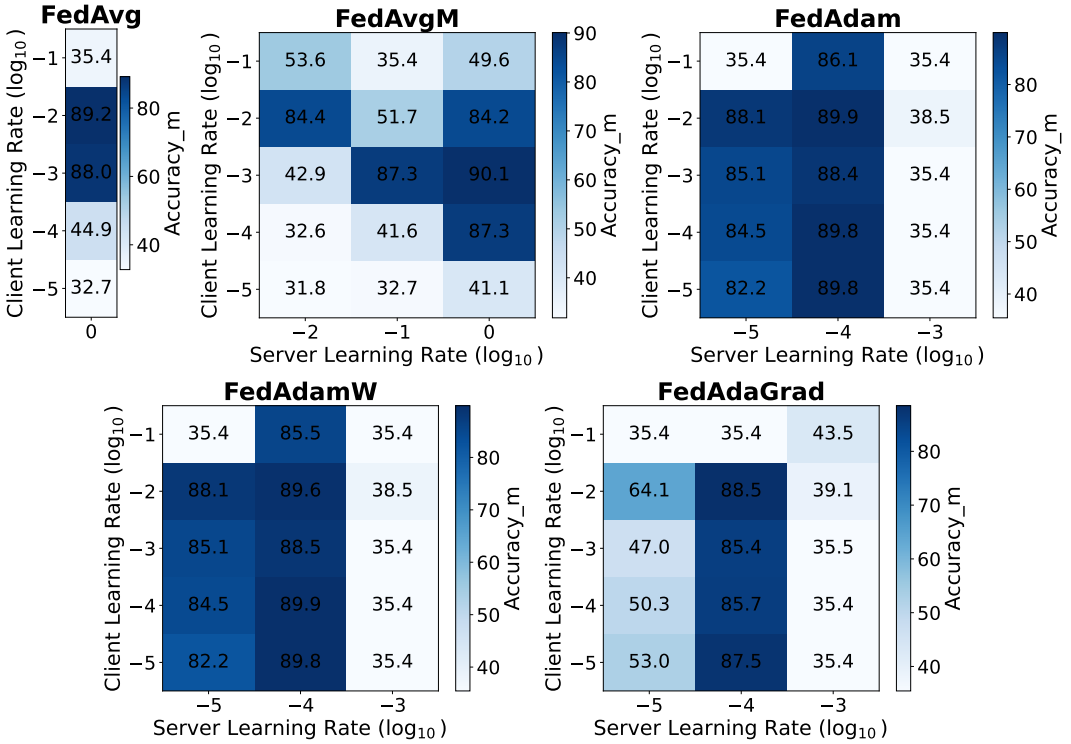


Figure 4.13: The highest testing accuracy of DeBERTaV3 on the MNLI-m dataset during the first 200 rounds of training, across various client and server optimizer learning rates.

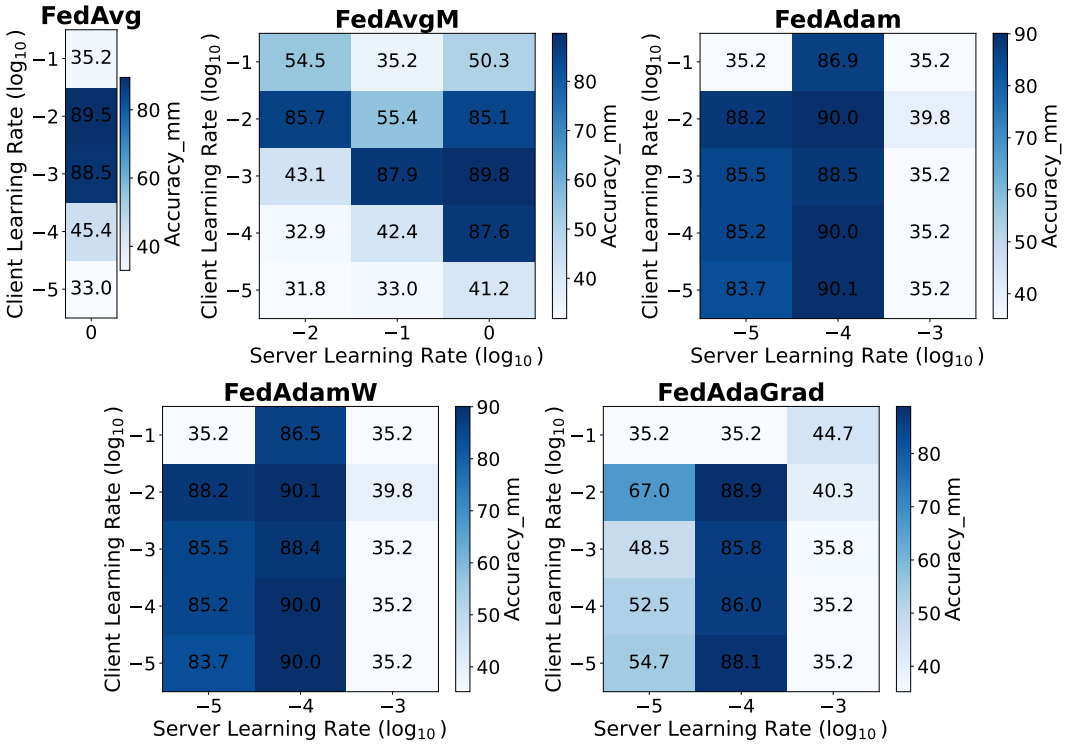


Figure 4.14: The highest testing accuracy of DeBERTaV3 on the MNLI-mm dataset during the first 200 rounds of training, across various client and server optimizer learning rates.

**Quantity Distribution.** Another common heterogeneity scenario in FL is quantity-

Table 4.2: The best-performing learning rate (lr) in $\log_{10}$ for the RoBERTa model with each FedOpt algorithm across datasets. ClientOpt is listed on top and ServerOpt at the bottom. For example, the bottom-left cell shows AdaGrad at the server with lr $= 10^{-4}$ and SGD at the client with lr $= 10^{-5}$

| | | MRPC | RTE | SST-2 | QNLI | MNLI |
|---|---|---|---|---|---|---|
| **FedAvg** | **SGD** | 0 | 0 | 0 | 0 | 0 |
| | **SGD** | −3 | −3 | −3 | −3 | −2 |
| **FedAvgM** | **SGDM** | 0 | −1 | 0 | 0 | 0 |
| | **SGD** | −3 | −3 | −4 | −3 | −3 |
| **FedAdam** | **Adam** | −4 | −5 | −4 | −4 | −4 |
| | **SGD** | −4 | −5 | −4 | −3 | −2 |
| **FedAdamW** | **AdamW** | −4 | −5 | −4 | −4 | −4 |
| | **SGD** | −5 | −5 | −4 | −3 | −4 |
| **FedAdaGrad** | **AdaGrad** | −4 | −4 | −4 | −4 | −4 |
| | **SGD** | −5 | −3 | −4 | −3 | −2 |

Table 4.3: Best-known accuracy scores (↑) for fine-tuning each model on each task in an ideal centralized setting. These values serve as target performance baselines

| **RoBERTa** [46] | | | | **DeBERTa** [26] | |
|---|---|---|---|---|---|
| **MRPC↑** | **RTE↑** | **SST-2↑** | **QNLI↑** | **MNLI-m↑** | **MNLI-mm↑** |
| 90.2% | 78.7% | 94.8% | 92.8% | 90.6% | 90.7% |

based imbalance, where clients possess vastly different amounts of data. We do not explicitly consider this scenario in our experiments. Nevertheless, while we aim to keep quantity distribution balanced, some low degree of quantity imbalance is inevitably introduced due to the label-based non-IID scheme.

### 4.4.7 Models & Quality Baselines

In our experiments, we use the pre-trained transformer-based models RoBERTa [46] and DeBERTaV3 [26]. More specifically, we use the `roberta-base` and `microsoft/deberta-v3-base` checkpoints available through Hugging Face [90], which contain 125 million and 86 million parameters, respectively. Such encoder models are well-suited for FL, as they can be readily fine-tuned for a wide range of practical downstream NLP tasks [44] (e.g., sentiment analysis, textual entailment, etc.). We fine-tune both models with the FedOpt and FDA-Opt algorithms on the datasets outlined in Section 4.4.2.

Of course, our ultimate goal in FL is to train accurate and robust models. Thus, as in most of the FL literature [58], we evaluate algorithms based on how efficiently they manage to do that. In other words, instead of comparing final accuracy scores or communication alone, we ask: *how many communication rounds does it take to train a model with a specific target quality?*

To define those targets in a consistent and task-agnostic way, we express them as percentages of the best-known centralized performance. Given both models' widespread adoption, their top-reported performance when fine-tuned in centralized settings is well-documented (see Table 4.3). Naturally, due to the adverse conditions of data federation, FL performance is typically lower than centralized baselines. For example, the highest accuracy ever reported for RoBERTa when fine-tuned on MRPC is 90.2%; then, a target of 90% means attaining at least 81.18% accuracy ($90\% \cdot 90.2 = 81.18\%$).

### 4.4.8 Algorithms & Hyper-Parameters

We evaluate the most widely used algorithms from the FedOpt family and compare their performance with our proposed FDA-Opt algorithms. To ensure a fair, apples-to-apples comparison, each FedOpt algorithm is paired with its corresponding FDA-Opt variant using identical parameter configurations. For example, we compare FedAdamW with FDA-AdamW, keeping all optimizer settings fixed. The algorithmic pairs we compare are outlined in Table 2.1.

Our goal is to conclusively demonstrate that FDA-Opt outperforms FedOpt and that hyper-parameters proven effective for FedOpt not only work for FDA-Opt but also yield better results. To show this, we first find the optimal hyper-parameter configuration for each FedOpt algorithm[1]. Then, we apply these to our proposed FDA-Opt counterparts. Given this approach, evidence that FDA-Opt outperforms FedOpt is particularly meaningful.

Lastly, unless otherwise specified, we set the number of local steps $\tau$ equal to one epoch—that is, $\tau = \lceil e \rceil$ from Equation (4.3)—as this is the most common and widely accepted choice in the FedOpt literature [44, 58, 28]. Moreover, we use a batch size of 8 and train for an exhaustive number of rounds, $T \in \{100, \ldots, 1000\}$, ensuring that training continues well beyond likely convergence.

## 4.5 Experimental Results & Analysis

In this section, we evaluate the performance of FDA-Opt against FedOpt across a variety of FL tasks, focusing on two key aspects: communication efficiency and convergence behavior, for the same model quality. Notably, our FDA-Opt algorithms operate under configurations optimized for their FedOpt competitors—which makes the following observed improvements all the more noteworthy.

### 4.5.1 Main Findings

The main findings of our experimental analyses are the following:

- **Communication-Efficiency.** FDA-Opt demonstrates significant improvements in communication-efficiency. On average, it is 2.15× more efficient in the cross-silo setting and 1.8× in the cross-device setting to train the highest accuracy models.
- **Convergence.** FDA-Opt converges to 5–10× lower training loss than FedOpt within the same number of rounds. The cross-silo setting is characterized by a sharp initial drop in loss, which is less pronounced in the cross-device setting.
- **Stability**. FDA-Opt is more robust across different initial local training step values, $\tau$, while FedOpt often fails to converge. Moreover, the communication improvements mirror those observed in the default case.

### 4.5.2 Communication-Efficiency

The communication overhead incurred by each algorithm is directly analogous to the number of rounds. For FedOpt, communication between clients and the server occurs only at the end of each round. In FDA-Opt, there is additional communication due to the

[1]For each FedOpt algorithm and dataset/task combination, we perform an exhaustive grid search to identify the best-performing hyper-parameters. For example, to optimize FedAdam training RoBERTa on MRPC, we evaluate a $5 \times 5$ grid of client and server learning rate combinations—25 configurations in total—and select the one achieving the highest accuracy. Across all models, tasks, and optimizers, this process amounts to roughly 700 distinct training runs. While this substantial experimental effort is not emphasized in the main text due to space constraints, it underpins all reported results.

Table 4.4: Comparison of FedOpt and FDA-Opt in communication-efficiency for training RoBERTa and DeBERTaV3 across different datasets to achieve target metrics. The table reports the number of rounds required to reach 85%, 90%, 95%, and 99% of the top-reported results in the literature (see Table 4.3)—naturally, the fewer rounds, the better (↓). For example, in the bottom-left cell, FedAdaGrad requires 29 rounds to achieve 90% of the best-reported accuracy for MRPC (i.e., at least $90\% \cdot 90.2 = 81.18\%$), while FDA-AdaGrad achieves the same in 21 rounds, making it 1.4× more efficient. Similarly, it has a speedup of 1.5× for 95%. The "Avg. Speedup" row is computed by averaging the individual FDA-Opt vs. FedOpt speedup values across each column

| | **RoBERTa** | | | | | | | | **DeBERTaV3** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **MRPC** | | **RTE** | | **SST-2** | | **QNLI** | | **MNLI-m** | | **MNLI-mm** | |
| | **90%↓** | **95%↓** | **90%↓** | **95%↓** | **95%↓** | **99%↓** | **90%↓** | **95%↓** | **85%↓** | **90%↓** | **85%↓** | **90%↓** |
| **FedAvg** | 26 | 30 | 67 | 103 | 8 | 93 | 35 | 71 | 12 | 16 | 12 | 15 |
| **FDA-SGD** | **6** | **10** | **16** | **20** | **7** | **43** | **9** | **23** | **4** | **8** | **4** | **6** |
| **FedAvgM** | 21 | 31 | 111 | 182 | 30 | 91 | 38 | 81 | 29 | 35 | 29 | 35 |
| **FDA-SGDM** | **5** | **16** | **41** | **126** | **12** | **41** | **11** | **44** | **12** | **15** | **12** | **15** |
| **FedAdam** | 20 | 40 | 118 | **153** | 12 | 42 | 11 | 22 | 11 | 12 | 11 | 12 |
| **FDA-Adam** | **7** | **11** | **72** | 188 | **8** | **25** | **6** | **20** | **9** | **10** | **9** | **10** |
| **FedAdamW** | 17 | 25 | 118 | **160** | 12 | 35 | 11 | 22 | 57 | 66 | 57 | 66 |
| **FDA-AdamW** | **14** | **16** | **71** | 176 | **8** | **14** | **6** | **20** | **43** | **54** | **42** | **52** |
| **FedAdaGrad** | 29 | 35 | 20 | 96 | 21 | 67 | 12 | 24 | **8** | 15 | 8 | **11** |
| **FDA-AdaGrad** | **21** | **23** | **18** | **54** | **7** | **24** | **6** | **20** | 14 | 15 | **6** | 14 |
| **Avg. Speedup** | 2.8× | 2.3× | 2.3× | 2× | 1.9× | 2.3× | 2.6× | 1.6× | 1.7× | 1.6× | 1.9× | 1.6× |

transmission of small sketches. However, the size of these sketches is negligible compared to the overall communication cost of transmitting language models (it is $\times 10^6$ smaller).

As is standard in the literature [79], we evaluate our algorithms by comparing their performance against predefined target metrics—such as accuracy (↑). The primary objective of the algorithms is to train a model that attains the target metric. Once they first do, we assess their communication efficiency by examining the number of FL rounds they required (↓). Table 4.4 illustrates this evaluation.

**Cross-Silo.** The cross-silo setting includes RoBERTa trained on the MRPC and RTE datasets, as outlined in Table 4.1.

- **RoBERTa on MRPC.** As per Table 4.4, FDA-Opt demonstrates superior communication-efficiency, with an average speedup of 2.3×-2.8× compared to FedOpt. It is 2.8× more efficient in reaching 90% of the target metric, and 2.3× for 95%.
- **RoBERTa on RTE.** Similar trends are observed for the RTE dataset. FDA-Opt achieves average speedups of 2×-2.3× over FedOpt, with specific gains of 2.3× for 90%, and 2× for 95% of the target metrics.

When the goal is to train the most accurate model—defined here as achieving 95% of the top-reported metrics—FDA-Opt is, on average, 2.15× more communication-efficient in the cross-silo setting.

**Cross-Device.** In the cross-device setting, we analyze performance of training RoBERTa on the SST-2 and QNLI datasets, and DeBERTaV3 on MNLI-m and MNLI-mm, as outlined in Table 4.1.

- **RoBERTa on SST-2.** FDA-Opt consistently outperforms FedOpt, achieving an improvement in communication-efficiency of 1.9×-2.3×. Specifically, FDA-Opt is 1.9× more efficient in achieving 90% of the target metric, and 2.3× for 99%.
- **RoBERTa on QNLI.** Again, FDA-Opt exhibits speedups ranging from 1.6×-2.6×. It is 2.6× more efficient at 90%, and 1.6× at 95% of the target metric.

- **DeBERTaV3 on MNLI-m.** Similarly, FDA-Opt has improvements of 1.7× and 1.6×, corresponding to 85% and 90% of the target metrics, respectively.
- **DeBERTaV3 on MNLI-mm.** Lastly, FDA-Opt is 1.7× and 1.6× more communication-efficient than FedOpt, for attaining 85% and 90% of the target accuracies, respectively.

Similarly, in the cross-device setting, FDA-Opt achieves an average improvement in communication-efficiency of 1.8× when training models to attain the highest possible accuracy targets.

### 4.5.3 Convergence

Analyzing training loss progression is key to understanding the convergence behavior of FL algorithms. It provides insights into how quickly and effectively models learn—essentially, their ability to minimize objectives. In this subsection we analyze Figures 4.15 and 4.16 which compare the performance of FDA-Opt and FedOpt on the four datasets of RoBERTa. The conclusions drawn regarding convergence pace and quality from these two datasets are indicative of the broader trends observed across all experiments.

**Cross-Silo.** On the MRPC dataset (Figure 4.15)—which simulates a cross-silo setting following Table 4.1—all FDA-Opt algorithms demonstrate significantly faster convergence than their FedOpt counterparts (Figure 4.15). For instance, FDA-SGD achieves a training loss 100× smaller than FedAvg within the first 100 rounds. Additionally, both FDA-SGDM and FDA-AdamW exhibit sharp drops within the initial 30–40 rounds, outperforming FedAvgM and FedAdamW, respectively, with FDA-SGDM stabilizing at a loss 10× smaller than FedAvgM. Meanwhile, FDA-Adam converges to a loss 5× smaller than FedAdam, and FDA-AdaGrad consistently maintains a 5× lower loss than FedAdaGrad after the initial rounds. Similar conclusions can be drawn for RTE from Figure 4.16.

**Cross-Device.** On the SST-2 dataset (Figure 4.15)—representing a cross-device setting as outlined in Table 4.1—all FDA-Opt algorithms consistently achieve 5–10× lower training loss compared to their FedOpt counterparts (Figure 4.15). Unlike the cross-silo case, the initial drop in training loss is less pronounced, which is expected since each round involves only a small subset of clients. This leads to initial updates that are less representative of the overall training distribution. Similar conclusions can be drawn for QNLI from Figure 4.16.

### 4.5.4 Stability to Local Training Extension

Empirical evidence suggests that increasing the number of local training steps $\tau$ can lead to degraded performance or non-convergence altogether [79, 93]. Since our scheme extends local training even further ($\tilde{\tau} \gg \tau$), it is crucial to examine whether this extension introduces any instability. To this end, we train RoBERTa on MRPC (cross-silo) and SST-2 (cross-device) using a range of values for $\tau$.

The results, shown in Figure 4.17, reveal two key insights. First, in every one of the 10 subplots, FDA-Opt consistently requires fewer training rounds than FedOpt. Across all 50 individual setups, FDA-Opt outperforms FedOpt in 40 cases—often by a large margin. Conversely, FedOpt shows only marginal improvements in the remaining 10 cases. Second, and most importantly, FDA-Opt converges reliably across all experiments. In contrast, FedOpt fails to converge in 4 out of 50 (specifically, FedAvgM and FedAdaGrad on MRPC). This is particularly problematic in FL, where non-convergence is almost impossible to detect and may lead to severe, and exploding communication cost.

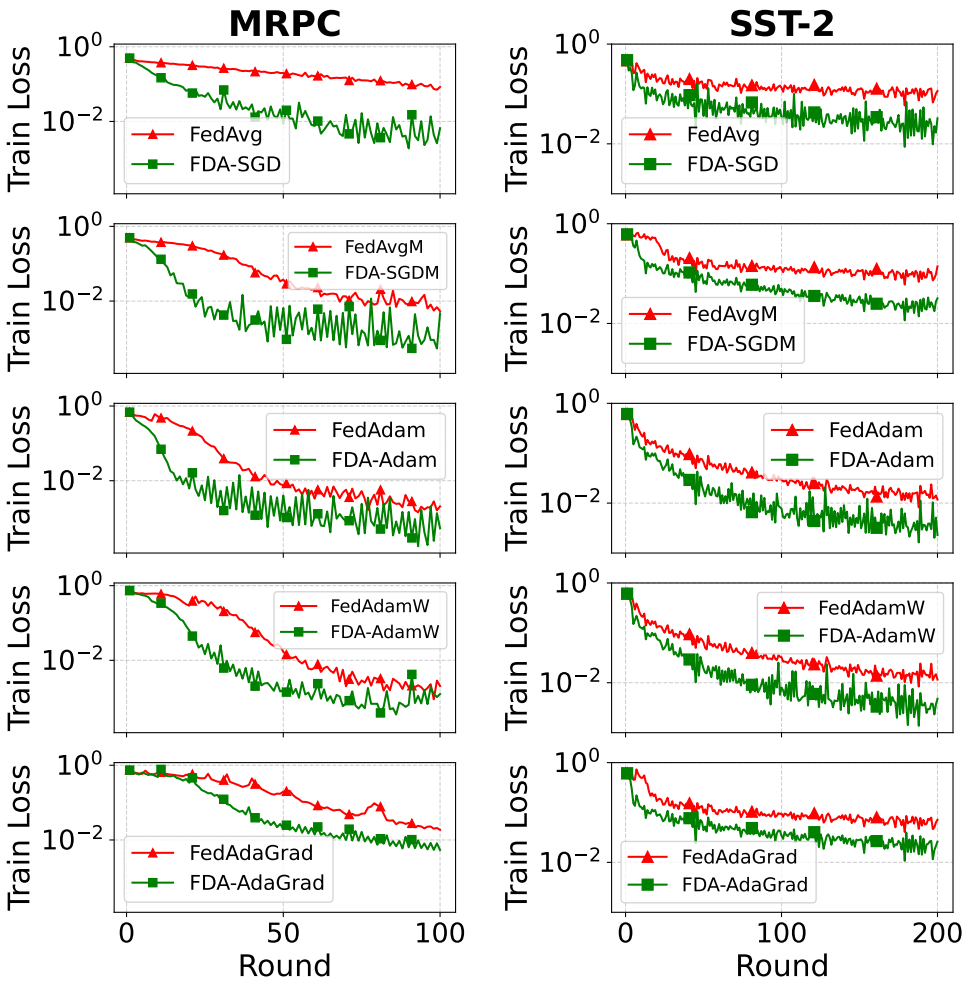


Figure 4.15: Training Loss progression of training RoBERTa with FDA-Opt vs. FedOpt on MRPC (left) and SST-2 (right)

These findings highlight that FDA-Opt is not only more efficient, but also inherently stable and *safe* for extended local training intervals—a direct result of monitoring the variance.

## 4.6 Related Work

This section reviews related work on communication-efficient FL.

**Round Termination.** Several works have explored strategies for determining the round termination intervals in FL. The study in [82] aims to find a fixed, optimal interval by balancing local computation and aggregation constraints. In contrast, [52] analyzes the trade-off between fast convergence and round completion rates under strongly convex client objectives, leading to a decaying round duration scheme. Similarly, [80] minimizes convergence error with respect to wall-clock time by progressively decreasing the round duration. On the other hand, [23] focuses on minimizing communication rounds for a fixed number of model updates, which results in an increasing sequence of round duration intervals. Unlike these approaches, which rely on predefined schedules—whether fixed, decaying, or increasing—FDA-Opt makes round termination decisions dynamically, in real-time, based on the state of the training.

**Convergence.** The most direct way to alleviate the communication burden in FL is

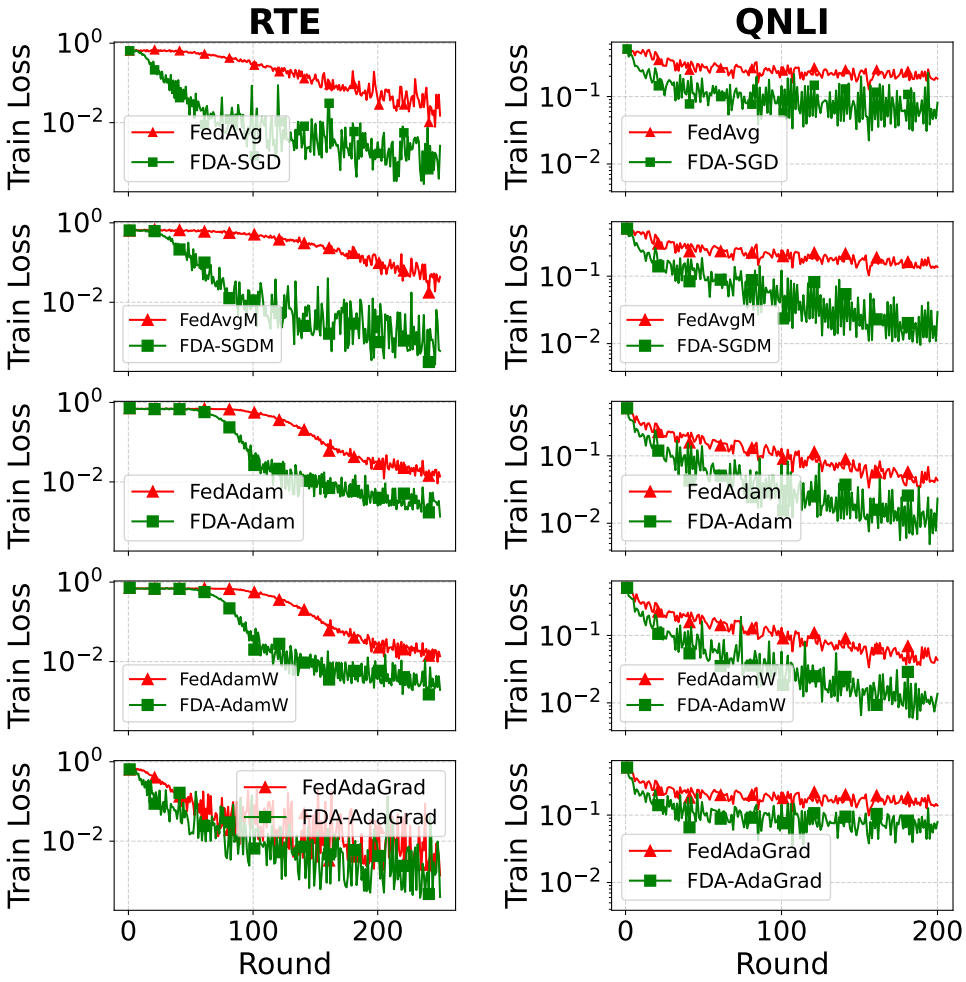


Figure 4.16: Training Loss progression of training RoBERTa with FDA-Opt vs. FedOpt on RTE (left) and QNLI (right)

to accelerate convergence. This motivated the development of the FedOpt family (Section 2.2.2). However, FL algorithms still struggle with high heterogeneity, where different clients converge to disparate and often conflicting local minima [33, 58, 86], ultimately slowing down convergence. To address this, SCAFFOLD introduced control variates to correct client drift [33]. Similarly, the Mime framework [32] leverages both control variates and server statistics. FedProx[41] tackles heterogeneity by adding an $L^2$ regularization term. Notably, it has been demonstrated that FedOpt outperforms FedProx in NLP tasks similar to ours [44]. Lastly, FedDyn [2] introduces a dynamic regularizer, ensuring that client convergence aligns with the stationary point of the global objective. Importantly, FDA-Opt is orthogonal to these optimization methods as we focus on a complementary aspect: when to terminate rounds.

**PEFT.** Parameter-efficient fine-tuning (PEFT) [43, 38, 29, 7] trains only a small subset of a model's parameters while keeping the rest frozen. One of the most widely used approaches, LoRA [29], injects low-rank adaptation matrices into a transformer's frozen attention layers, reducing the number of trainable parameters to $< 1\%$. Given the communication constraints in FL, recent studies have explored the applicability of PEFT methods in this setting [94, 67]. However, techniques like LoRA struggle with the high heterogeneity present in FL [67]. SLoRA [5] addresses this issue by leveraging sparse fine-tuning [4] to initialize the injected parameters more effectively. These PEFT methods can be readily integrated with FDA-Opt, as the choice of trainable parameters, $\mathbf{w}$, remains user-defined.

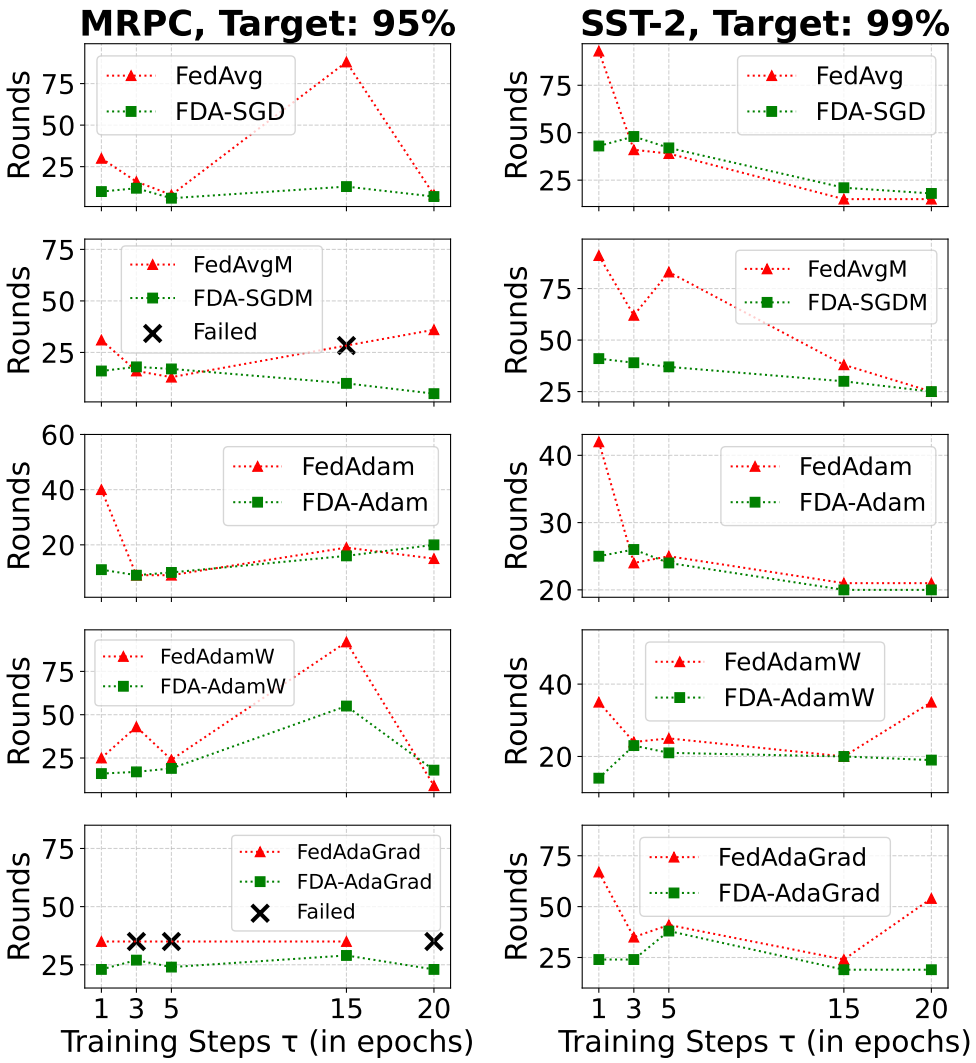


Figure 4.17: Training RoBERTa with FDA-Opt vs. FedOpt for varying initial local training steps $\tau$ (measured in epochs). We report the number of rounds required to attain the target accuracy. Failure to converge is marked with "×"

**Compression.** Another way to reduce communication in FL is to compress transmitted information using techniques like quantization [62] and sparsification [3, 6]. A comprehensive survey of these methods is provided in [84]. Notably, compression is orthogonal to our approach and can be seamlessly integrated with FDA-Opt.

## 4.7 Key Takeaways

In this work, we introduced the FDA-Opt family of algorithms, a unified generalization of both FDA and FedOpt, addressing their core limitations. We empirically prove that FDA-Opt is more communication efficient and reliable than FedOpt. Furthermore, through carefully designed experiments, we demonstrating that FDA-Opt can seamlessly replace FedOpt, as it can be directly configured using settings from the FedOpt literature. To this end, each comparison between the two algorithmic families was conducted using the best-performing configurations for our competitor, FedOpt—showing that well-established settings from the literature can be applied to our algorithms without modification. Importantly, even under these "unfair" conditions, FDA-Opt achieves at least

2× greater communication-efficiency on average and consistently converges to 5×–10× lower training loss. These findings hint at significant practical implications for improving modern FL libraries and real-world deployments.

# 5. Conclusion

This thesis tackled the fundamental challenge of communication-efficiency in Federated Learning by proposing adaptive synchronization strategies based on real-time training dynamics. We introduced Federated Dynamic Averaging (FDA), a novel algorithm that dynamically determines when to synchronize models based on approximations of inter-client model variance. Through extensive experiments on diverse vision datasets and tasks, we demonstrated that FDA significantly reduces communication overhead—often by orders of magnitude—without increasing computational cost or sacrificing model quality. Moreover, FDA showed strong robustness to data heterogeneity and exhibited implicit regularization effects, helping to mitigate overfitting in challenging non-IID settings.

Building on this foundation, we developed the FDA-Opt family of algorithms, a unified generalization of both FDA and FedOpt, addressing their core limitations. We empirically show that FDA-Opt is more communication-efficient and reliable than FedOpt to fine-tune state-of-the-art language models on downstream NLP tasks. Furthermore, through carefully designed experiments, we demonstrate that FDA-Opt can seamlessly replace FedOpt, as it can be directly configured using hyper-parameter settings from the FedOpt literature. To this end, all comparisons were conducted using the best-performing configurations for our competitor, FedOpt, showing that well-established settings can be applied to our algorithms without modification. Importantly, even under these "unfair" conditions, FDA-Opt achieves at least 2× greater communication efficiency on average and consistently converges to 5×–10× lower training loss. These findings suggest significant practical implications for improving modern FL libraries and real-world deployments.

Together, these contributions pave the way for more scalable and practical federated learning frameworks that dynamically adapt to system conditions, data heterogeneity, and training dynamics.

# Bibliography


[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: a system for large-scale machine learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331.

[2] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. "Federated Learning Based on Dynamic Regularization". In: *International Conference on Learning Representations*. 2021.

[3] Alham Fikri Aji and Kenneth Heafield. "Sparse Communication for Distributed Gradient Descent". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 440–445.

[4] Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. "Composable Sparse Fine-Tuning for Cross-Lingual Transfer". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1778–1796.

[5] Sara Babakniya, Ahmed Elkordy, Yahya Ezzeldin, Qingfeng Liu, Kee-Bong Song, MOSTAFA EL-Khamy, and Salman Avestimehr. "SLoRA: Federated Parameter Efficient Fine-Tuning of Language Models". In: *International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023*. 2023.

[6] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. "Qsparse-local-SGD: distributed SGD with quantization, sparsification, and local computations". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[7] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. "BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1–9.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[9] Tianyi Chen, Georgios B. Giannakis, Tao Sun, and Wotao Yin. "LAG: lazily aggregated gradient for communication-efficient distributed learning". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 5055–5065.

[10] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. "Project Adam: Building an Efficient and Scalable Deep Learning Training System". In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 571–582. ISBN: 978-1-931971-16-4.

[11] Francois Chollet et al. *Keras*. 2015.

[12] Graham Cormode and Minos Garofalakis. "Sketching Streams through the Net: Distributed Approximate Query Tracking". In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, pp. 13–24. ISBN: 1595931546.

[13] Angjela Davitkova, Damjan Gjurovski, and Sebastian Michel 0001. "Learning over Sets for Databases". In: *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. Ed. by Letizia Tanca, Qiong Luo 0001, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani. OpenProceedings.org, 2024, pp. 68–80. ISBN: 978-3-89318-091-2.

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[15] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[16] William B. Dolan and Chris Brockett. "Automatically Constructing a Corpus of Sentential Paraphrases". In: *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*. 2005.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[18] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159.

[19] Fangcheng Fu, Xupeng Miao, Jiawei Jiang, Huanran Xue, and Bin Cui. "Towards communication-efficient vertical federated learning training via cache-enabled local updates". In: 15.10 (June 2022), pp. 2111–2120. ISSN: 2150-8097.

[20] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256.

[21] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. Cambridge, MA, USA: MIT Press, 2016.

[22] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[23] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck R. Cadambe. "Local SGD with Periodic Averaging: Tighter Analysis and Adaptive Synchronization". In: *Neural Information Processing Systems*. 2019.

[24] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. "Pre-trained models: Past, present and future". In: *AI Open* 2 (2021), pp. 225–250. ISSN: 2666-6510.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[26] Pengcheng He, Jianfeng Gao, and Weizhu Chen. *DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing*. 2023. arXiv: `2111.09543 [cs.CL]`.

[27] Elad Hoffer, Itay Hubara, and Daniel Soudry. "Train longer, generalize better: closing the generalization gap in large batch training of neural networks". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1729–1739. ISBN: 9781510860964.

[28] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. "Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification". In: *CoRR* (2019).

[29] 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". In: *International Conference on Learning Representations*. 2022.

[30] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 2261–2269.

[31] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. "Advances and Open Problems in Federated Learning". In: *Foundations and Trends in Machine Learning* 14.1–2 (June 2021), pp. 1–210. ISSN: 1935-8237.

[32] Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U Stich, and Ananda Theertha Suresh. *Mime: Mimicking Centralized Stochastic Algorithms in Federated Learning*. 2021.

[33] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 5132–5143.

[34] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. San Diego, CA, USA, 2015.

[35] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto* (May 2012).

[36] Eugenie Yujing Lai, Zainab Zolaktaf, Mostafa Milani, Omar AlOmeir, Jianhao Cao, and Rachel Pottinger. "Workload-Aware Query Recommendation Using Deep Learning". In: *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*. Ed. by Julia Stoyanovich, Jens Teubner, Nikos Mamoulis, Evaggelia Pitoura, and Jan Mühlig. OpenProceedings.org, 2023, pp. 53–65. ISBN: 978-3-89318-088-2.

[37] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[38] Brian Lester, Rami Al-Rfou, and Noah Constant. "The Power of Scale for Parameter-Efficient Prompt Tuning". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3045–3059.

[39] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. "Federated Learning on Non-IID Data Silos: An Experimental Study". In: *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2022, pp. 965–978.

[40] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. "PyTorch distributed: experiences on accelerating data parallel training". In: *Proc. VLDB Endow.* 13.12 (Aug. 2020), pp. 3005–3018. ISSN: 2150-8097.

[41] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. "Federated Optimization in Heterogeneous Networks". In: *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. Ed. by Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze. mlsys.org, 2020.

[42] Xiang Li, Wenhao Yang, Shusen Wang, and Zhihua Zhang. *Communication-Efficient Local Decentralized SGD Methods*. 2021. arXiv: `1910.09126 [stat.ML]`.

[43] Xiang Lisa Li and Percy Liang. "Prefix-Tuning: Optimizing Continuous Prompts for Generation". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli. Online: Association for Computational Linguistics, Aug. 2021, pp. 4582–4597.

[44] Bill Yuchen Lin, Chaoyang He, Zihang Ze, Hulin Wang, Yufen Hua, Christophe Dupuy, Rahul Gupta, Mahdi Soltanolkotabi, Xiang Ren, and Salman Avestimehr. "FedNLP: Benchmarking Federated Learning Methods for Natural Language Processing Tasks". In: *Findings of the Association for Computational Linguistics: NAACL 2022*. Ed. by Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 157–175.

[45] Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi. "Don't Use Large Mini-batches, Use Local SGD". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[46] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692.

[47] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. "A ConvNet for the 2020s". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2022, pp. 11966–11976.

[48] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019.

[49] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019.

[50] Kaihao Ma, Xiao Yan, Zhenkun Cai, Yuzhen Huang, Yidi Wu, and James Cheng. "FEC: Efficient Deep Recommendation Model Training with Flexible Embedding Communication". In: *Proc. ACM Manag. Data* 1.2 (June 2023).

[51] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, Apr. 2017, pp. 1273–1282.

[52] Jed Mills, Jia Hu, and Geyong Min. "Faster Federated Learning With Decaying Number of Local SGD Steps". In: *IEEE Trans. Parallel Distrib. Syst.* 34.7 (July 2023), pp. 2198–2207. ISSN: 1045-9219.

[53] Supun Nakandala and Arun Kumar. "Nautilus: An Optimized System for Deep Transfer Learning over Evolving Training Datasets". In: *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 506–520. ISBN: 9781450392495. DOI: 10.1145/3514221.3517846. URL: https://doi.org/10.1145/3514221.3517846.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: an imperative style, high-performance deep learning library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[55] Tiancheng Qin, S. Rasoul Etesami, and César A. Uribe. "The role of local steps in local SGD". In: *Optimization Methods and Software* (Aug. 2023), pp. 1–27. ISSN: 1029-4937.

[56] Zhen Qin, Shuiguang Deng, Mingyu Zhao, and Xueqiang Yan. "FedAPEN: Personalized Cross-silo Federated Learning with Adaptability to Statistical Heterogeneity". In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '23. Long Beach, CA, USA: Association for Computing Machinery, 2023, pp. 1954–1964. ISBN: 9798400701030.

[57] Sashank Reddi, Zachary Burr Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Brendan McMahan, eds. *Adaptive Federated Optimization*. 2021.

[58] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. "Adaptive Federated Optimization". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 2021.

[59] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. "FetchSGD: communication-efficient federated learning with sketching". In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020.

[60] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252. ISSN: 1573-1405.

[61] Shaohuai Shi, Zhenheng Tang, Xiaowen Chu, Chengjian Liu, Wei Wang, and Bo Li. "A Quantitative Survey of Communication Optimizations in Distributed Deep Learning". In: *IEEE Network* 35.3 (2021), pp. 230–237.

[62] Nir Shlezinger, Mingzhe Chen, Yonina Eldar, H. Vincent Poor, and Shuguang Cui. "UVeQFed: Universal Vector Quantization for Federated Learning". In: *IEEE Transactions on Signal Processing* 69 (Jan. 2021), pp. 500–514.

[63] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[64] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Ed. by David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642.

[65] Ryan Spring, Anastasios Kyrillidis, Vijai Mohan, and Anshumali Shrivastava. "Compressing Gradient Optimizers via Count-Sketches". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 5946–5955.

[66] Sebastian U. Stich. "Local SGD Converges Fast and Communicates Little". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[67] Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. "Improving LoRA in Privacy-preserving Federated Learning". In: *The Twelfth International Conference on Learning Representations*. 2024.

[68] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147.

[69] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147.

[70] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147.

[71] Zhenheng Tang, Shaohuai Shi, Bo Li, and Xiaowen Chu. "GossipFL: A Decentralized Federated Learning Framework With Sparsified and Adaptive Communication". In: *IEEE Transactions on Parallel and Distributed Systems* 34.3 (2023), pp. 909–922.

[72] Michail Theologitis, Georgios Frangias, Georgios Anestis, Vasilis Samoladas, and Antonios Deligiannakis. "Communication-Efficient Distributed Deep Learning via Federated Dynamic Averaging". In: *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25 - March 28*. OpenProceedings.org, 2025, pp. 411–424. ISBN: 978-3-89318-098-1.

[73] Michail Theologitis, Vasilis Samoladas, and Antonios Deligiannakis. *Communication-Efficient Federated Fine-Tuning of Language Models via Dynamic Update Schedules*. 2025. arXiv: 2505.04535 [cs.LG]. URL: https://arxiv.org/abs/2505.04535.

[74] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. "Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable". In: *IEEE Spectrum* 58.10 (2021), pp. 50–55.

[75] Taegeon Um, Byungsoo Oh, Byeongchan Seo, Minhyeok Kweun, Goeun Kim, and Woo-Yeon Lee. "FastFlow: Accelerating Deep Learning Model Training with Smart Offloading of Input Data Pipeline". In: 16.5 (Jan. 2023), pp. 1086–1099. ISSN: 2150-8097.

[76] Leslie G. Valiant. "A Bridging Model for Parallel Computation". In: *Commun. ACM* 33.8 (Aug. 1990), pp. 103–111. ISSN: 0001-0782.

[77] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN: 3319579584.

[78] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Ed. by Tal Linzen, Grzegorz Chrupała, and Afra Alishahi. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355.

[79] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Agüera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas N. Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horváth, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konečný, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtárik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake E. Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. "A Field Guide to Federated Optimization". In: *CoRR* abs/2107.06917 (2021). arXiv: 2107.06917.

[80] Jianyu Wang and Gauri Joshi. "Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-update SGD". In: *Systems and Machine Learning (SysML) Conference* ().

[81] Jianyu Wang and Gauri Joshi. "Cooperative SGD: A Unified Framework for the Design and Analysis of Local-Update SGD Algorithms". In: *Journal of Machine Learning Research* 22.213 (2021), pp. 1–50.

[82] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. "Adaptive Federated Learning in Resource Constrained Edge Computing Systems". In: *IEEE Journal on Selected Areas in Communications* 37.6 (2019), pp. 1205–1221.

[83] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. "Database Meets Deep Learning: Challenges and Opportunities". In: 45.2 (Sept. 2016), pp. 17–22. ISSN: 0163-5808.

[84] Zeqin Wang, Ming Wen, Yuedong Xu, Yipeng Zhou, Jessie Hui Wang, and Liang Zhang. "Communication compression techniques in distributed deep learning: A survey". In: *Journal of Systems Architecture* 142 (2023), p. 102927. ISSN: 1383-7621.

[85] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021.

[86] Phillip Wenig and Thorsten Papenbrock. "DataGossip: A Data Exchange Extension for Distributed Machine Learning Algorithms". In: *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*. Ed. by Julia Stoyanovich, Jens Teubner, Paolo Guagliardo, Milos Nikolic, Andreas Pieris, Jan Mühlig, Fatma Özcan, Sebastian Schelter, H. V. Jagadish, and Meihui Zhang 0001. 2022.

[87] Adina Williams, Nikita Nangia, and Samuel Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. URL: http://aclweb.org/anthology/N18-1101.

[88] Herbert Woisetschläger, Alexander Erben, Shiqiang Wang, Ruben Mayer, and Hans-Arno Jacobsen. "A Survey on Efficient Federated Learning Methods for Foundation Model Training". In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*. Ed. by Kate Larson. Survey Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2024, pp. 8317–8325.

[89] Herbert Woisetschläger, Alexander Erben, Shiqiang Wang, Ruben Mayer, and Hans-Arno Jacobsen. "Federated Fine-Tuning of LLMs on the Very Edge: The Good, the Bad, the Ugly". In: *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning*. DEEM '24. Santiago, AA, Chile: Association for Computing Machinery, 2024, pp. 39–50. ISBN: 9798400706110.

[90] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Qun Liu and David Schlangen. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[91] Gang Yan, Hao Wang, Xu Yuan, and Jian Li. "FedRoLA: Robust Federated Learning Against Model Poisoning via Layer-based Aggregation". In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '24. Barcelona, Spain: Association for Computing Machinery, 2024, pp. 3667–3678. ISBN: 9798400704901.

[92] Hao Yu and Rong Jin. "On the Computation and Communication Complexity of Parallel SGD with Dynamic Batch Sizes for Stochastic Non-Convex Optimization". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7174–7183.

[93] Hao Yu, Sen Yang, and Shenghuo Zhu. "Parallel restarted SGD with faster convergence and less communication: demystifying why model averaging works for deep learning". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1.

[94] Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. "FedPETuning: When Federated Learning Meets the Parameter-Efficient Tuning Methods of Pre-trained Language Models". In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 9963–9977.

[95] Lixi Zhou, Jiaqing Chen, Amitabh Das, Hong Min, Lei Yu, Ming Zhao, and Jia Zou. "Serving deep learning models with deduplication from relational databases". In: *Proc. VLDB Endow.* 15.10 (June 2022), pp. 2230–2243. ISSN: 2150-8097.

[96] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. "Parallelized Stochastic Gradient Descent". In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc., 2010.

# A. Appendix

## A.1 Additional FDA Generalization Gap Results

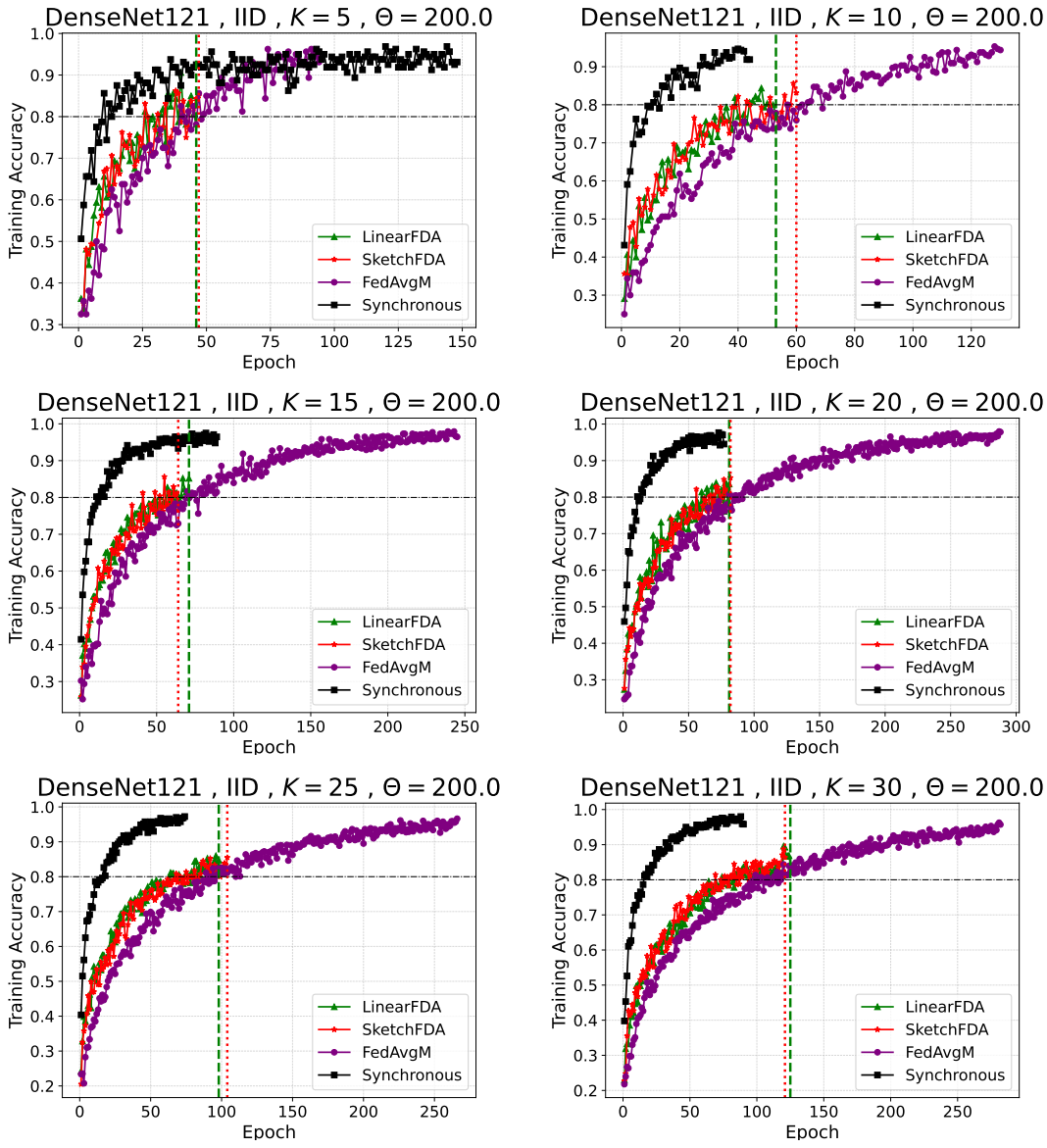


Figure A.1: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

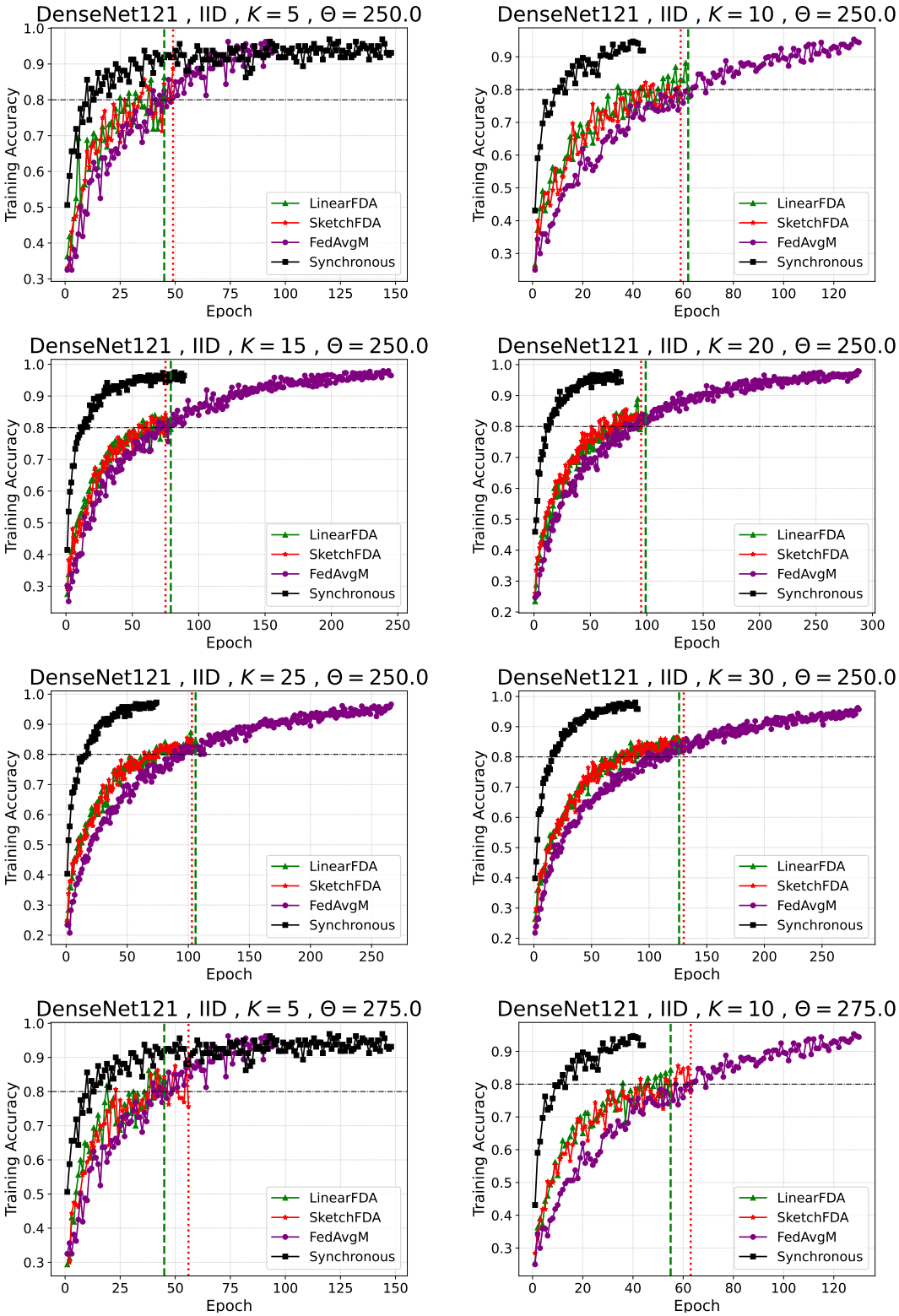


Figure A.2: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

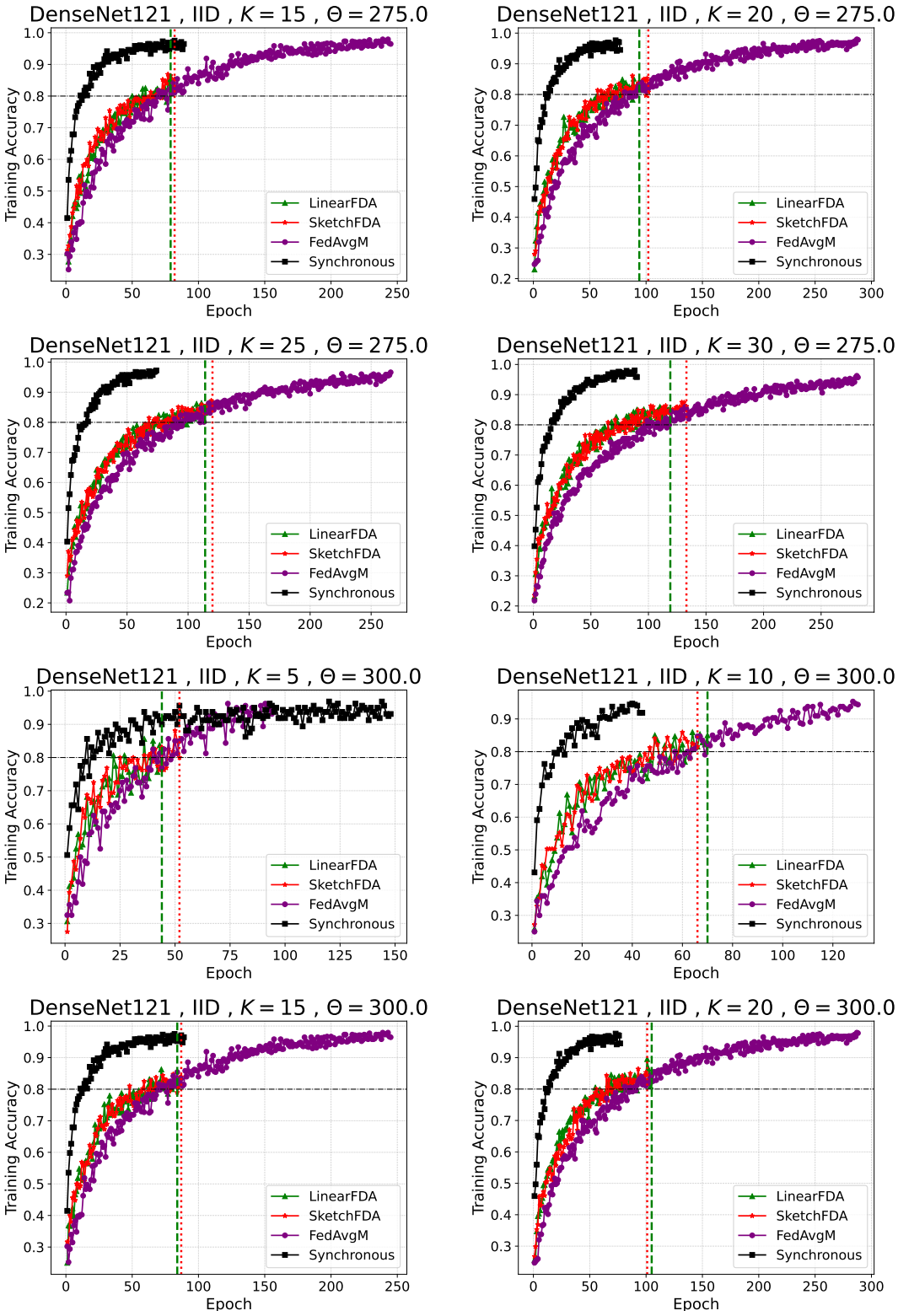


Figure A.3: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

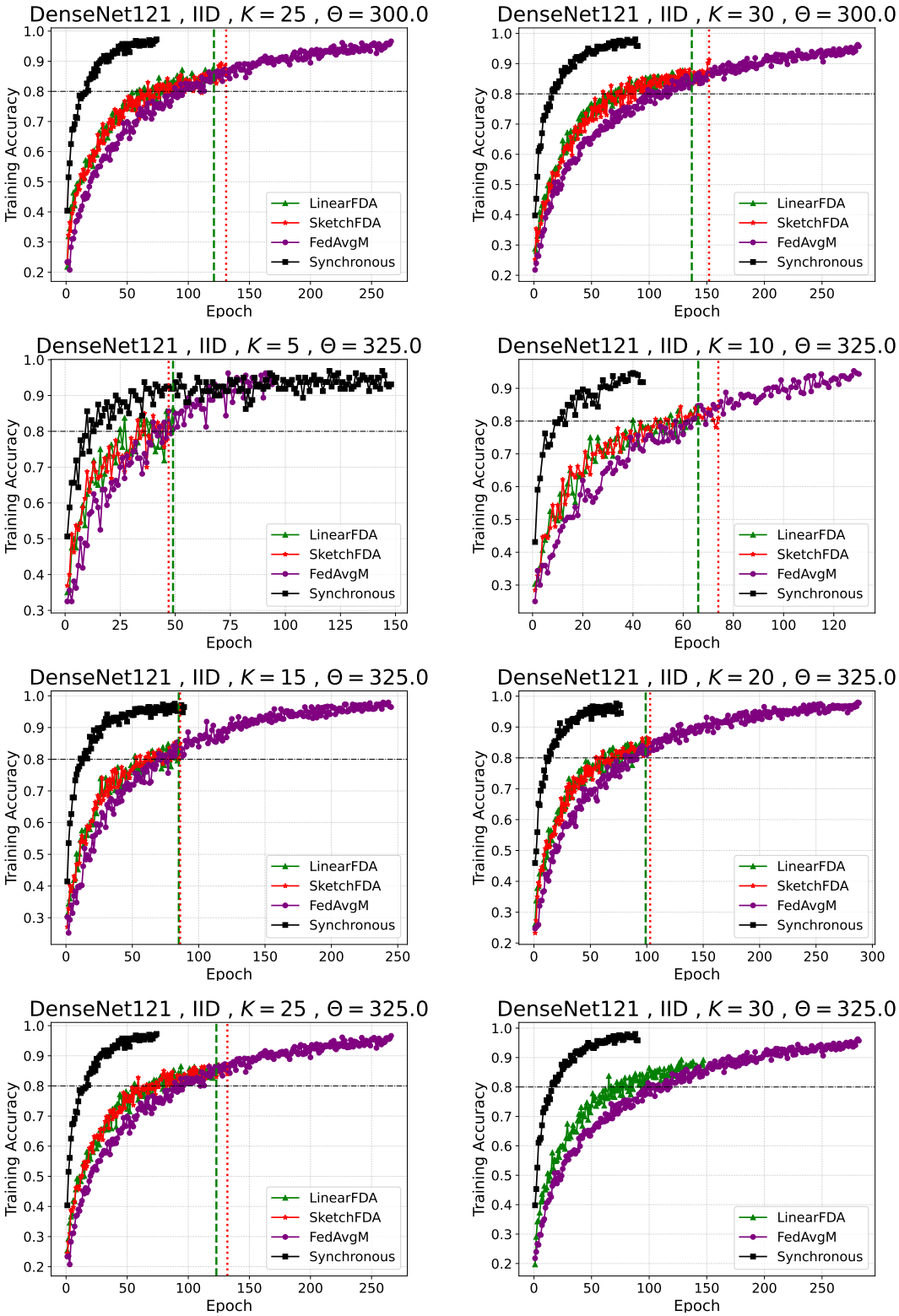


Figure A.4: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

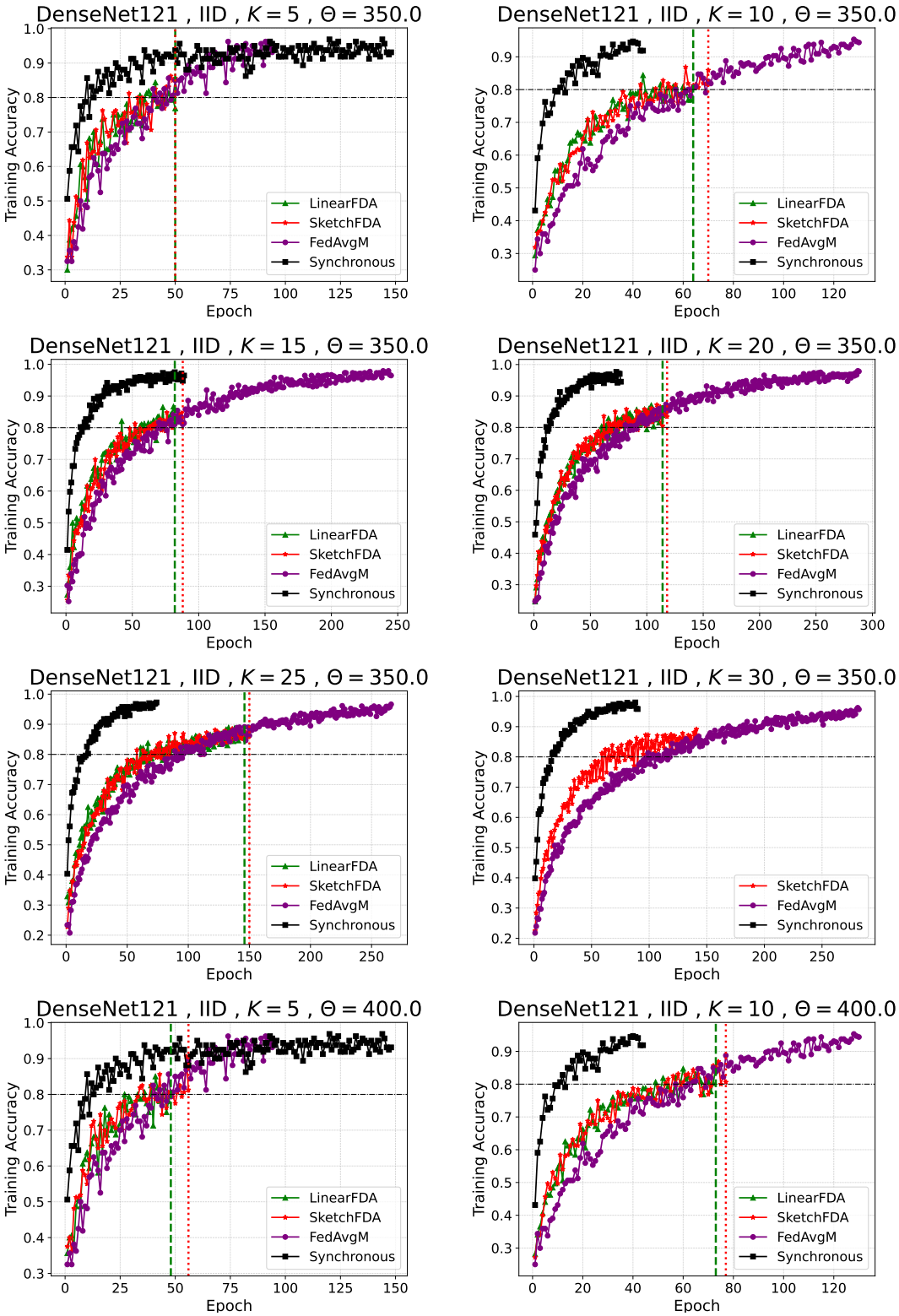


Figure A.5: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

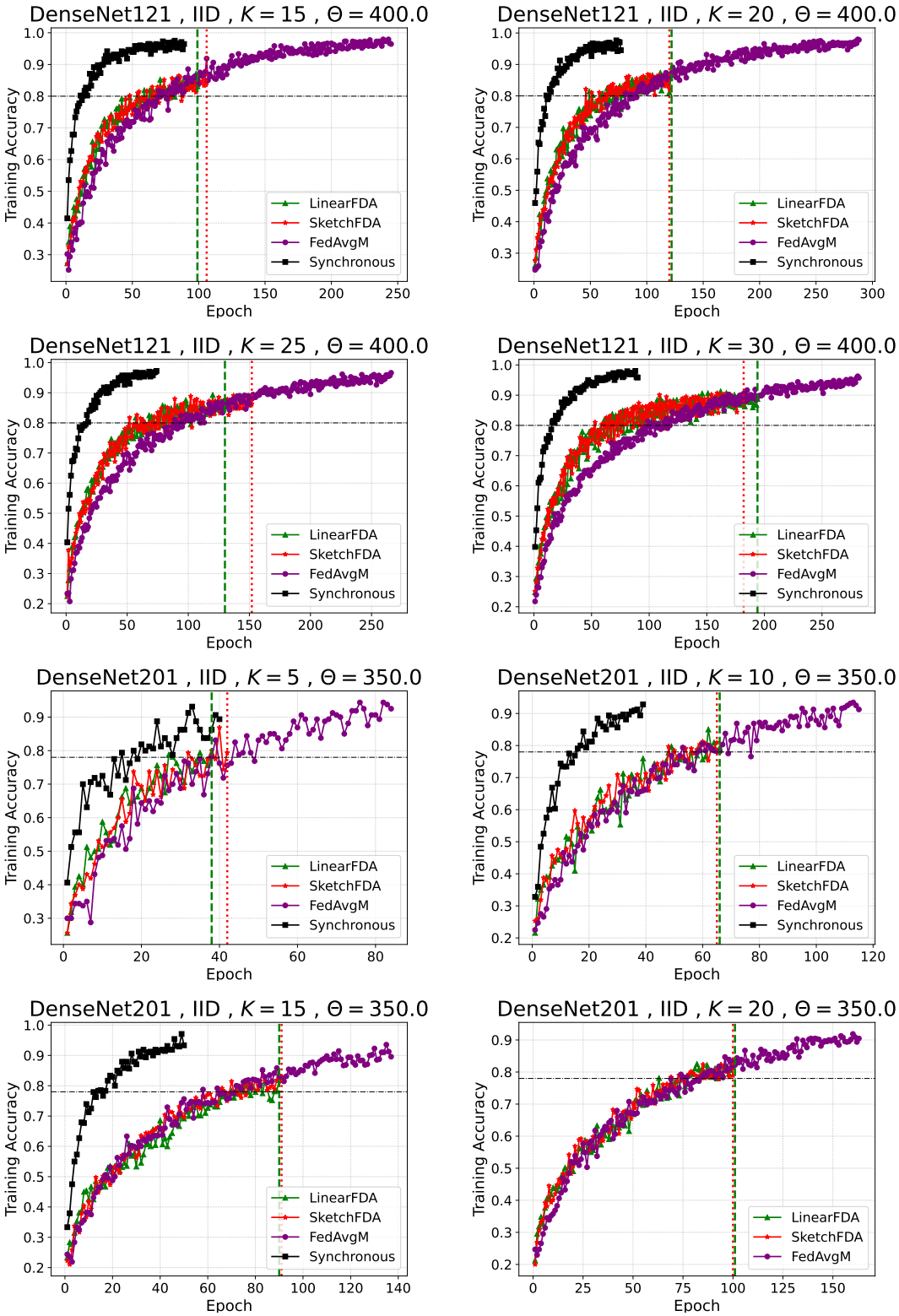


Figure A.6: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

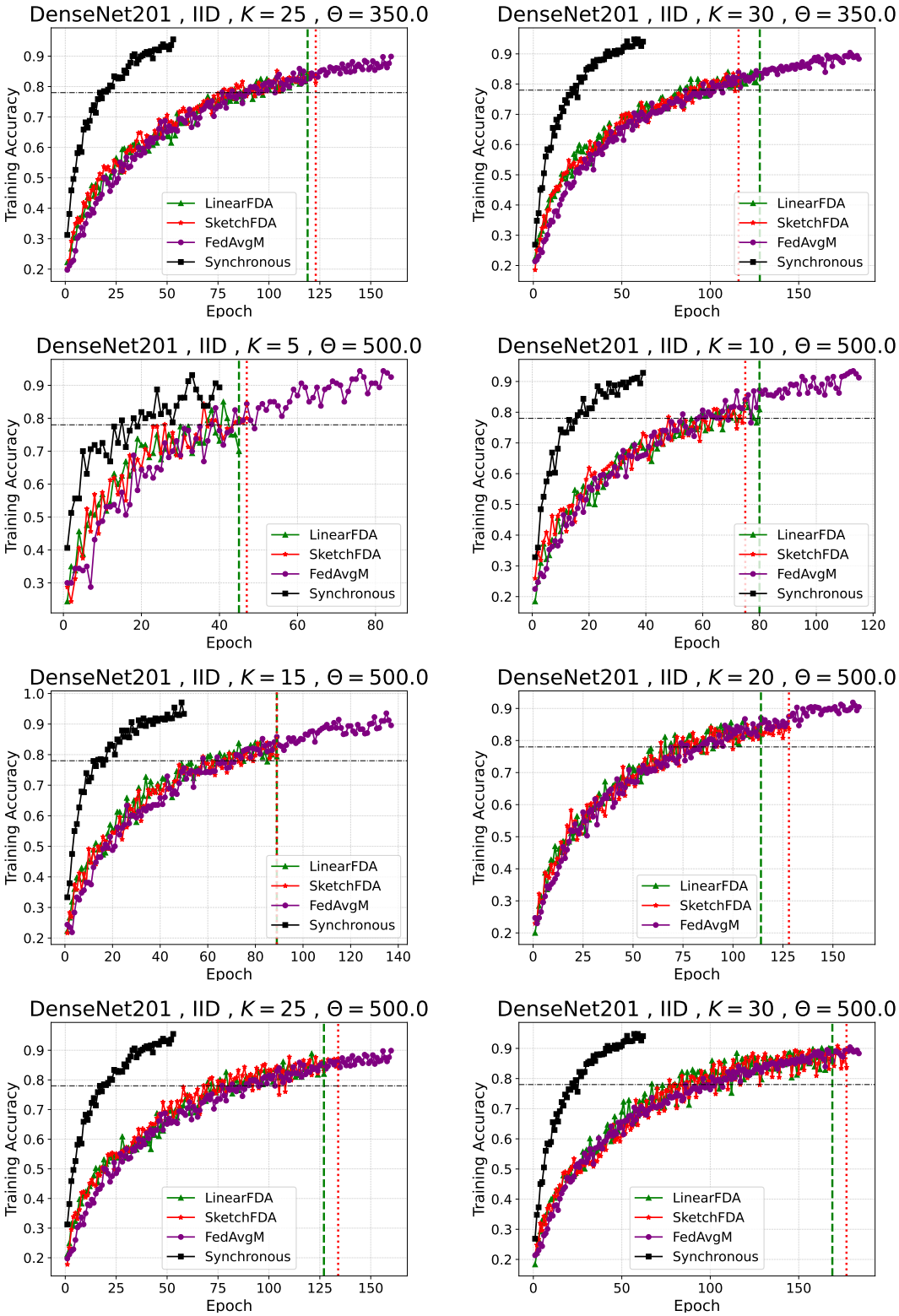


Figure A.7: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

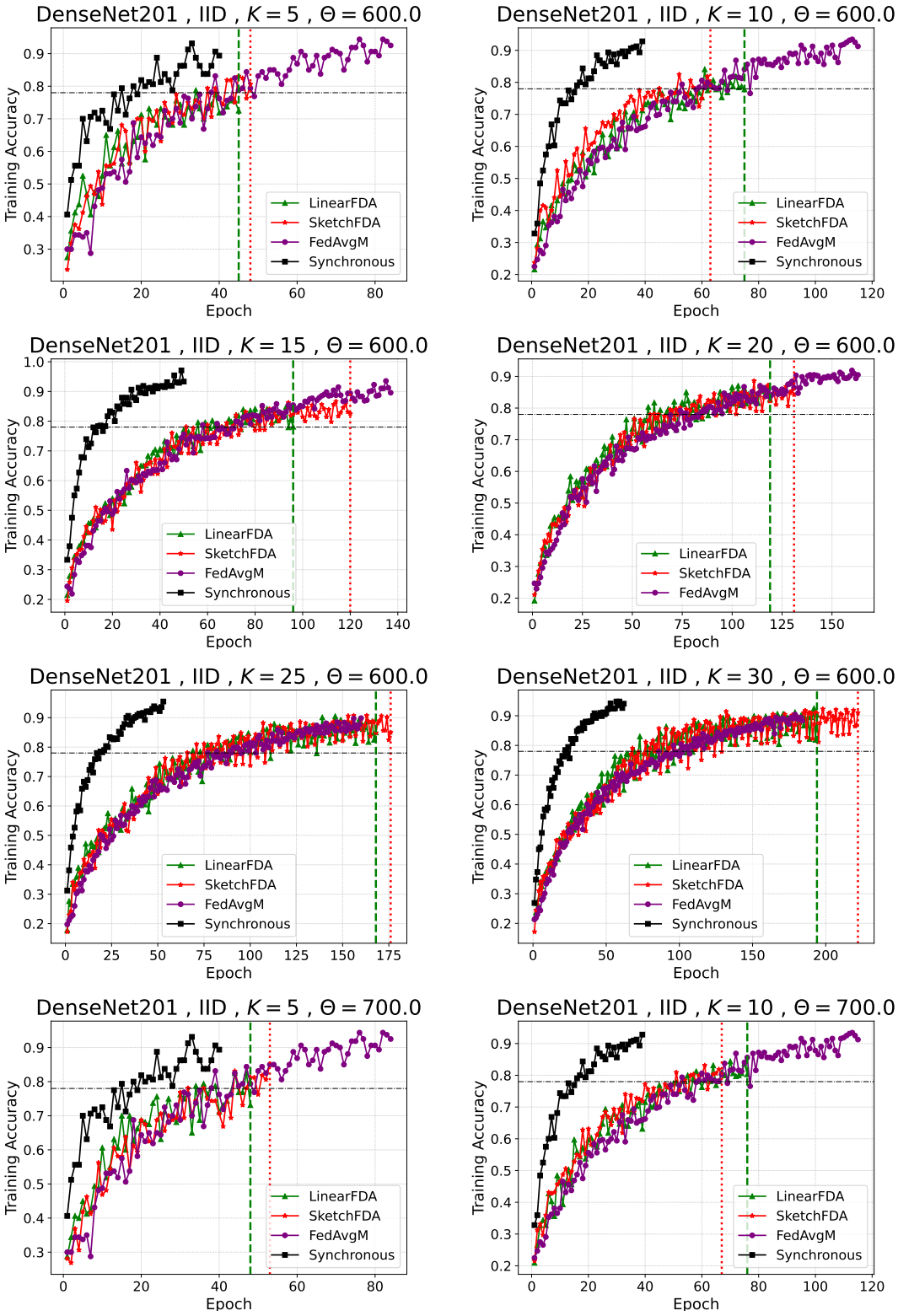


Figure A.8: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

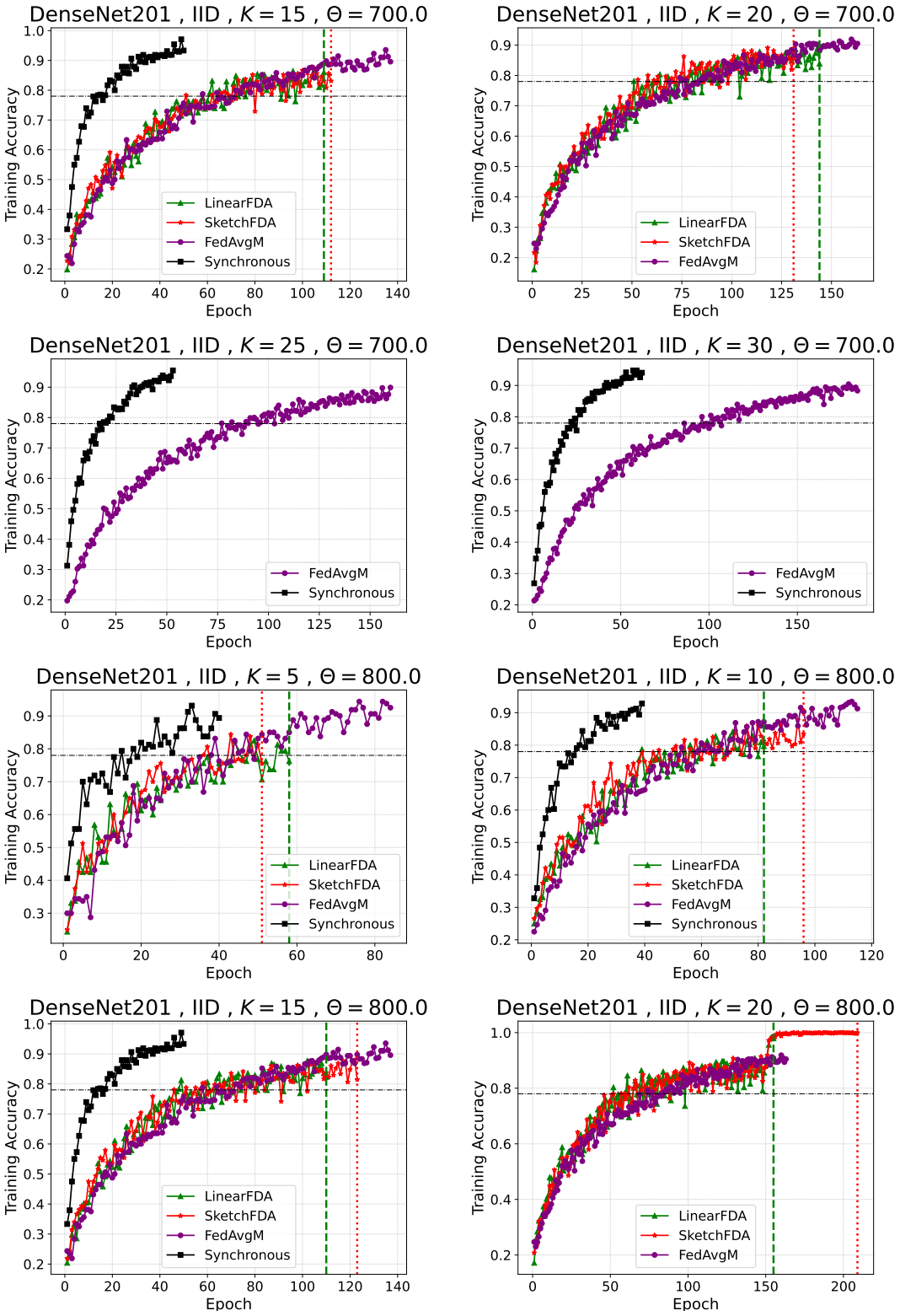


Figure A.9: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

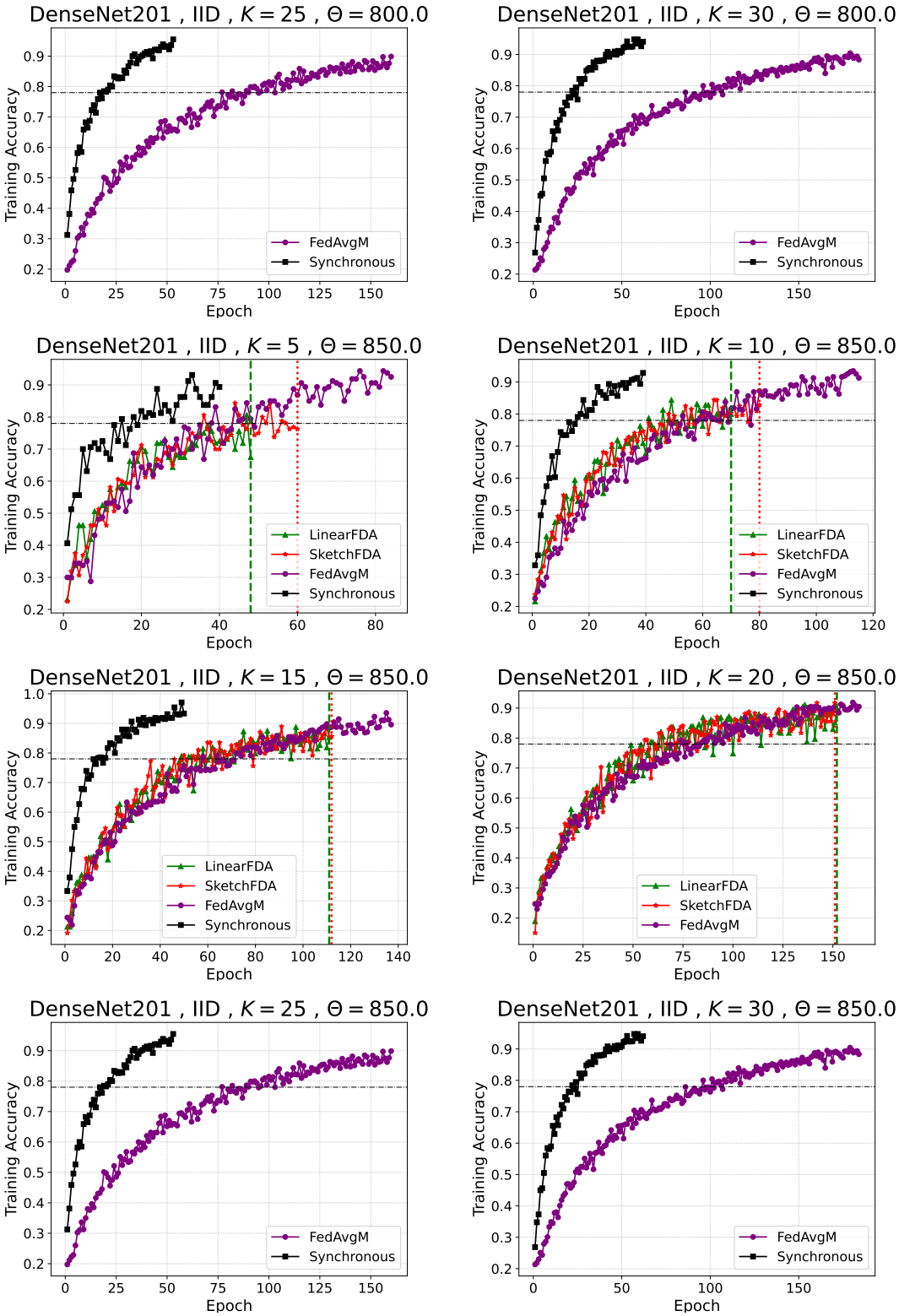


Figure A.10: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model

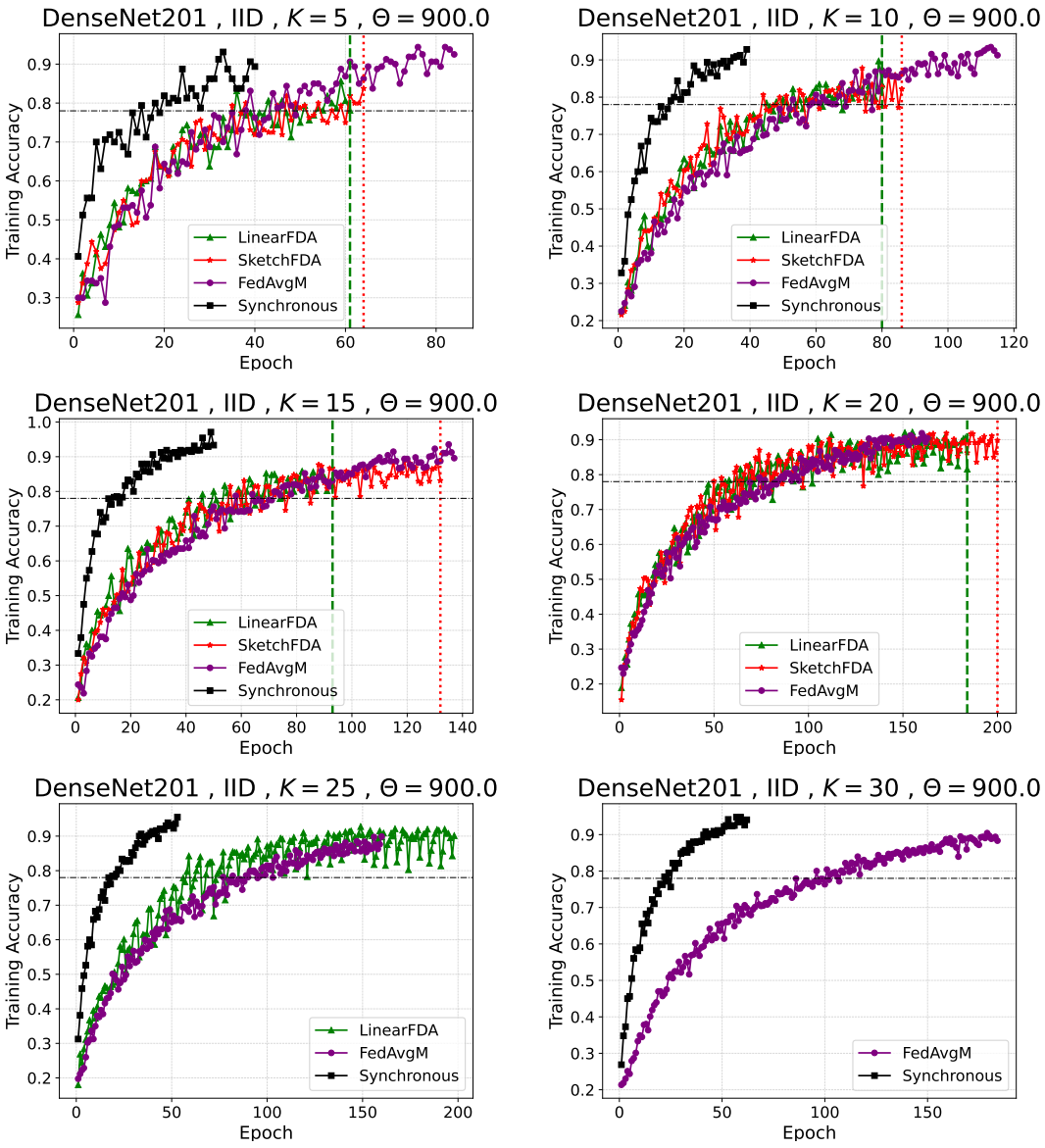


Figure A.11: Training accuracy progression with a test accuracy target (horizontal line) of 0.8 for DenseNet121, and 0.78 for DenseNet201. Dashed and doted lines indicate when LinearFDA and SketchFDA attain the target accuracy, respectively. A smaller final gap between training and target accuracy indicates less overfitting, i.e., better generalization capabilities of the trained model