A Practical Approach to Federated Learning

by

Vaikkunth Mugunthan

B.Tech., Anna University (2017)
S.M., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . . . .
Department of Electrical Engineering and Computer Science
April 29, 2022

Certified by . . . . . .
Lalana Kagal
Principal Research Scientist, MIT CSAIL
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

A Practical Approach to Federated Learning

by

Vaikkunth Mugunthan

Submitted to the Department of Electrical Engineering and Computer Science
on April 29, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Electrical Engineering and Computer Science

Abstract

Machine learning models benefit from large and diverse training datasets. However, it is difficult for an individual organization to collect sufficiently diverse data. Additionally, the sensitivity of the data and government regulations such as GDPR, HIPPA, and CCPA restrict how organizations can share data with other entities. This forces organizations with sensitive datasets to develop models that are only locally optimal. Federated learning (FL) facilitates robust machine learning by enabling the development of global models without sharing sensitive data. However, there are two broad challenges associated with deploying FL systems: privacy challenges and training/performance-related challenges. Privacy challenges pertain to attacks that reveal sensitive information of local client data. Training/Performance-related challenges include high communication costs, data heterogeneity across clients, and lack of personalization techniques. All these concerns have to be addressed to make FL practical, scalable, and useful. In this thesis, I discuss techniques I've designed for addressing these challenges and describe two systems that I've developed to mitigate them - PrivacyFL, a privacy-preserving simulator for FL, and DynamoFL, an easy-to-use production-level system for FL.

Thesis Supervisor: Lalana Kagal
Title: Principal Research Scientist, MIT CSAIL

Dedicated to my parents

Acknowledgments

I would like to start by thanking my amazing advisor Lalana Kagal for her guidance, support, and feedback over the past five years. I honestly could not have asked for a better advisor. She provided me the freedom and flexibility to pursue research topics I found intriguing and helped me identify impactful research questions.

I would next like to thank my committee members Polina Golland and Sam Madden for helping me throughout the thesis submission process and for their insightful comments. I would also like to thank all the mentors I've had during my internships at IBM, Microsoft, JPMorgan, and Intech Investments. I would like to thank Christian Lau, Emile Indik, Eric Lin, and the entire DynamoFL team who trusted my vision for DynamoFL and made my dream come true.

To all my friends and collaborators, I want to thank you for making research at MIT such a great experience. I would like to give special thanks to Doron Hazan, Wanyi Xiao, Prabhakar Kafle, Anton Bueno, Nick Ulven, Pawan Goyal, and Lay Jain for being wonderful UROPs. I would like to thank my other co-authors Antigoni Polychroniadou, Dave Byrd, Vignesh Gokul, Ravi Rahman, Irene Tenison, and Andreas Haupt as well.

Outside of research, I would like to thank the MIT Cricket team for allowing me to play my favorite sport at a professional level for the past 5 years. I want to thank Raj Agrawal, Sarath Pattathil, and Siddhartha Jayanti for their continuous support, and for reminding me that there is a life beyond the computer screen.

Finally, I want to thank my parents who have always been there for me as well as my whole family.

Above all, I would like to thank Lord Ganesha and Lord Krishna for giving me this wonderful opportunity in life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Networks can be used to train high-quality models with state-of-the-art performance in a myriad of applications, including medical image analysis, health informatics, language representation, and many more. However, building such models is not an easy task as it requires access to a large amount of high-quality data. Modern practitioners of machine learning often need to train models from large data sets distributed across many devices. In the past, such data would be centralized for analysis, but that practice has given rise to serious concerns around user privacy, lack of permission to transfer sensitive data, high data transfer costs, and far-reaching consequences when centralized data stores are breached.

Federated learning (FL) is a recent technique that addresses these concerns by training on each local data segment individually, then transmitting and combining only the resulting model parameters [9, 42]. In essence, FL allows decentralized clients to collaboratively learn a model without sharing their private data. It enables clients to coordinate with a central server to train high-quality and robust models by exchanging model parameters and hence keeping their local data private. Potential applications of FL include:

- Predictive Healthcare: Hospitals and healthcare centers contain sensitive patient data for predictive healthcare. However, ethical constraints and privacy regulations like HIPPA, CCPA, and GDPR, require data to remain local. FL is a useful solution as it allows private learning between different healthcare units.

- Smartphones/ Edge Devices: Learning user behavior across numerous smartphones/edge

devices enables statistical models to power applications such as personalized healthcare recommendations, face detection, and next-word prediction, etc. The next generation of intelligent edge devices will need to integrate on-device training as they adapt to new environments and achieve maximum data security. However, individual edge devices will not be able to capture enough data independently to train models from scratch. Federated learning enables connected fleets of edge devices to collaboratively train models without needing to transfer raw sensory data across devices. In addition, FL saves the limited bandwidth/battery power of mobile phones as they don't have to share large volumes of data.

- Financial Fraud Detection: Law enforcement currently catches less than 1% of money laundered globally. Fraud detection methods could be meaningfully improved if financial institutions could combine their financial records to trace fraudulent transactions across institutions. However, data privacy and competitive interests have precluded this possibility. Federated learning provides a non-aggregative approach to developing fraud detection models trained across financial data silos by enabling fraud detection modeling, while keeping financial records behind the firewalls of financial institutions.

However, FL suffers from a myriad of privacy, performance and training, simulation and infrastructure-related challenges.

1.1 Federated Learning Challenges

There are five main challenges associated with FL.

Privacy Concerns: Privacy concerns in FL pertain to attacks that reveal sensitive information on local client data. Though FL seems to provide increased privacy as clients share model updates and not training data, there has been a multitude of privacy attacks on FL systems, including membership inference and model inversion [84, 70, 65, 33]. Recent methods in FL provide privacy guarantees using techniques like differential privacy and multi-party computation. However, these methods provide privacy at the cost of reduced

model performance or system efficacy. It is important to balance these trade-offs to provide an optimal FL solution.

Statistical heterogeneity/Non-identically independently distributed (non-iid) data: When different clients have different data distributions, the performance of vanilla FL degrades and results in slower model convergence. In addition, heterogeneous settings can result in information loss and lead to poor generalization due to the bias induced by dominant clients. Hence, it is important to focus on learning the invariant mechanism that is constant while ignoring spurious mechanisms that differ across clients.

Personalization: Vanilla FL constructs a server model for all clients by averaging their local models, while postulating that all clients share a single common task. However, this scheme does not adapt the model to each client. For example, platforms like Youtube and Netflix require a unique personalized model for each of their clients. Most FL algorithms focus on improving the average performance across clients, aiming to achieve high accuracy for the global server model. However, certain clients might perform poorly while others perform extremely well. This is not the ideal scenario for a fair and optimal FL algorithm. When deployed on edge devices in the real world, local test accuracy is instead a more important metric for success.

Communication Cost: Sending and receiving model parameters is a huge bottleneck in FL protocols as it could be expensive for resource-constrained clients. It is important to reduce the total number of communication rounds and the size of the packets that are transmitted during every round. Unfortunately, there is usually a tradeoff between model accuracy and communication cost accrued during the federation process. For instance, techniques that speed up accuracy convergence or decrease the model size may result in a small decrease in accuracy.

Simulations and Real-World/Production Deployment:

Setting up an easy-to-use federated learning environment, especially with security and privacy guarantees, to evaluate novel approaches is a time-consuming process. In order to help researchers ensure that FL is feasible and to validate if their approach improves model performance, a real-world simulator for privacy-preserving federated learning is required. Recently, there have been a few open source FL simulators such as TensorFlow-Federated

[40], PySyft [82], PrivacyFL [67], etc. These simulators allow system designers to have complete customization control over experimenting novel FL and privacy algorithms without having to focus on low-level communication or backend details for simulation purposes. However, these simulators cannot be used for real-world and production level deployments where clients are spread across different continents.

There are numerous challenges associated with coming up with an easy-to-use production-level FL system:

1. Project Setup: A streamlined process for setting up a project in a couple of minutes is required. This includes - adding collaborators, sharing the global model, configuring/-customizing FL aggregation algorithms, secure integration with clients' existing ML pipelines, etc. A clean interface for setting up the project, monitoring the performance of clients, and tracking their progress is important. This is needed to reduce the burden on the project administrator.

2. Scalability: The system should be able to support millions of clients spread across different continents, process requests at a high speed, and be platform agnostic. For true scalability both I/O operations as well as computational load bearing must be scaled as the platform grows, due to the distinct nature of federated learning (servicing arbitrary number of client connections while performing heavy aggregation jobs on their models). In addition, aggregation of an arbitrarily large number of models starts to run up against memory constraints.

3. Robustness to Client Dropouts and Rejoining: It is important for the system/architecture to carry on and complete the FL process even when a client(s) dropout/rejoin during FL rounds. All participants of a project have to stay in sync - know when to be training, testing, or waiting for a new model to arrive, and preserving state after dropout and reconnection.

4. Privacy-Preserving and Secure Communication: It is important for the system to make models privacy-preserving and establish secure communication between the clients and server during FL rounds.

1.2 Thesis Contributions

In this thesis, I provide solutions to the most pressing challenges in FL.

To address the privacy concerns, I came up with a novel privacy-preserving FL mechanism using *oblivious* distributed differential privacy and multi-party computation(MPC) to protect against any attack based on client collusion, including those where a server preferentially selects compromised clients or simulates fake clients in a so-called "Sybil" attack. In addition, I also show that it is not possible to decipher the exact differentially private result of any client under semi-honest settings.

I propose a gradient masked averaging approach as an alternative to naive averaging of parameters in FL for improved generalization performance under non-iid settings. I hypothesize that to generalize better across non-iid datasets, algorithms should focus on learning the invariant mechanism that is constant while ignoring spurious mechanisms that differ across clients.

For optimal personalization and communication-efficiency under non-IID data settings, I came up with FedLTN - a novel approach motivated by the Lottery Ticket Hypothesis to learn sparse and personalized Lottery Ticket Networks (LTNs).

For simulating FL experiments, I built PrivacyFL - an extensible, easily configurable, and scalable simulator for FL environments. Its key features include latency simulation, robustness to client departure/failure, support for both centralized (with one or more servers) and decentralized (serverless) learning, and configurable privacy and security mechanisms based on differential privacy and secure multiparty computation (MPC).

Finally, I introduce DynamoFL - an easy-to-use, plug-and-play infrastructure that enables organizations to rapidly stand up federated learning across clients/devices in a matter of minutes and can be plugged into existing data and modeling pipelines of ML teams smoothly. It allows machine learning teams to build highly personalized (or) generalized ML solutions that are trained in a federated manner to learn from both individual user data and the global population characteristics in a privacy-preserving manner. In addition, DynamoFL addresses all privacy and training/performance-related challenges associated with FL.

1.3 Thesis Overview

In Chapter 2, I propose an approach to address the privacy concerns. Chapter 3 and Chapter 4 propose novel solutions to address the generalization and personalization challenges assoicated with FL under non-iid settings. Next, Chapter 4 provides a communication-efficient method for personalizing models. In Chapter 5, I present PrivacyFL, an extensible, easily configurable, and scalable simulator for federated learning environments. DyanmoFL, an easy-to-use, plug-and-play FL infrastructure for real-world settings is presented in Chapeter 6. Finally, I wrap-up the thesis with conclusion and future work in Section 7.

Chapter 2

Privacy-Preserving and Secure FL

Privacy-preserving federated learning enables a population of distributed clients to jointly learn a shared model while keeping client training data private, even from an untrusted server. Prior works do not provide efficient solutions that protect against collusion attacks in which parties collaborate to expose an honest client's model parameters. I present an efficient mechanism based on oblivious distributed differential privacy that is the first to protect against such client collusion, including the "Sybil" attack in which a server preferentially selects compromised devices or simulates fake devices. I leverage the novel privacy mechanism to construct a secure federated learning protocol and prove the security of that protocol. I conclude with empirical analysis of the protocol's execution speed, learning accuracy, and privacy performance on two data sets within a realistic simulation of 5,000 distributed network clients.

2.1 Introduction

Individual user privacy can still be compromised under FL settings by using the trained model to infer certain details of the training data set [84, 69]. Two key approaches have been proposed to address this problem. The first is *differential privacy*, which perturbs values to guarantee statistical indistinguishability for individual inputs [25]. This can be applied to federated learning by having each client modify its local model weights by adding randomly-generated values (potentially reducing model accuracy), with the result that a party

obtaining the transmitted weights will still have uncertainty over the original weights. The second approach, which does not compromise accuracy, is *secure multi-party computation* (MPC) [36]. An MPC protocol, which allows parties to collaboratively compute a common function of interest without revealing their private inputs, is considered secure if the parties learn the computational output and nothing else.

I build on a recent line of research that combines differential privacy and MPC to produce a secure federated learning protocol. [9, 41]. These prior works provide strong protection against undesired inference by the server, but the collusion of enough clients can reveal the noisy weights of an honest client, and the scale of that noise is limited by the need for an accurate model.

I propose a novel, efficient mechanism that protects against any attempt to undermine differential privacy by collusion of $n-1$ out of n total clients. Unlike prior works, I offer a protocol where the noise for each party is added in an oblivious way. Obliviousness *can* be achieved by running the noise generation inside the MPC, but such solutions are based on heavy cryptography machinery involving a significant amount of public key operations or incur increased communication complexity [41, 16]. In this work I focus on the concretely efficient aggregation protocol of Bonawitz et al. without drop-out parties which does not involve any public key operations in the learning phase [9]. I, therefore, provide the first practical protection against $n-1$ attacks by constructing an efficient oblivious distributed differentially private aggregation protocol.

2.2 Background

2.2.1 Secure Multiparty Computation

Consider n parties $P_1, \ldots, P_n$ that hold private inputs $x_1, \ldots, x_n$ and wish to compute some arbitrary function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where the output of P_i is y_i. Secure Multi-Party Computation (MPC) enables the parties to compute the function using an interactive protocol such that each party P_i learns exactly y_i and nothing else. [36] (See Appendix A.1 for further detail.)

2.2.2 Differential Privacy

Differential privacy states that if there are two databases that differ by only one element, they are statistically indistinguishable from each other. In this work I use the Laplacian mechanism which preserves ϵ-differential privacy [25]. (See Appendices A.2, A.3, A.4 for further detail.)

Definition 1. (ϵ-differential privacy [26]) A randomized mechanism $\mathcal{A}$ preserves ϵ-differential privacy (ϵ-DP) if for any two neighboring datasets D_1, D_2 that differ by one element, and for all subsets of possible answers $\mathcal{S} \subseteq Range(\mathcal{A})$, Pr $[\mathcal{A}(D_1) \in \mathcal{S}] \leq e^{\epsilon}$ Pr $[\mathcal{A}(D_2) \in \mathcal{S}]$.

2.2.3 Federated Logistic Regression Classifiers

Logistic regression is a machine learning algorithm used to solve the problem of binary linear classification. Assume one of n parties is called P_i and has a local data set consisting of instances $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)})$, where m is the number of features, and their corresponding labels $y^{(i)}$.

Party P_i uses its training examples $(x^{(i)}, y^{(i)})$ to learn a logistic classifier with weights w_i. The weights are obtained by solving the following optimization problem where $f(x_k^{(i)}) = w^T x_k^{(i)}$ and t_i is the number of training examples of P_i:

$$w_i = \arg\min_{w} \frac{1}{t_i} \sum_{k=1}^{t_i} log(1 + e^{-y_k^{(i)} f(x_k^{(i)})}) \tag{2.1}$$

In order to minimize the loss function, I make use of gradient descent, an iterative optimization algorithm, calculating the optimal w iteratively as $w^{j+1} \leftarrow w^j - \alpha \nabla L(w^j)$, where α is the learning rate, j is the iteration, $w^0 = 0$, and ∇L is the gradient of the loss function. Our local logistic regression is a vector-based re-implementation of Jayaraman et al. [41].

Privacy-preserving federated learning allows a large number of parties to learn a model while keeping their local training data private. Parties first train local models on their local data and coordinate with a server to obtain a global model. Given n parties, let w_i, for $i \in 1$ to n, represent the local model estimator after minimizing the objective function. Then $W = \frac{1}{n} \sum_{i=1}^{n} w_i + \eta$, where η is the differentially private noise added to the cumulative

model.

According to Jayaraman et al., for 1-$Lipschitz$ the global sensitivity for a multi-party setting is $\frac{2}{n*k*\alpha}$, where k is the size of the smallest dataset amongst the n parties, and α is the regularization parameter. [41] Hence, $\eta = \mathcal{L}(\frac{2}{n*k*\alpha*\epsilon})$, where ϵ is the privacy loss parameter. In our protocol, each client will add noise to the weights of the trained local model.

2.2.4 Network Topology & Threat Model

As is common in the federated learning setting, I opt for a star network topology, where there is one central party that is connected to all other parties. This central server can be distinct from the n original parties.

The protocols that I describe and compare against are secure in the semi-honest model. A semi-honest adversary follows the protocol correctly but tries to learn as much as possible about the inputs of the uncorrupted parties from the messages it receives. Furthermore, if there are multiple semi-honest corruptions, I allow the adversary to combine the views of the corrupted parties to potentially learn more information. See Appendix A.10 for communication protocol diagrams.

2.3 Approach

Our approach combines secure multi-party aggregation with oblivious distributed differential privacy to better secure federated learning against $n - 1$ collusion attacks. In this work, I consider logistic regression as the local learning method, and each client update includes the weights of that logistic regression. The server receives the weights from all clients at each iteration and computes a new global model using the average of the client updates for each weight. Recall from the Introduction the literature demonstrating that private client data can be inferred from the trained model weights, which is clearly undesirable. The general task, then, is to secure each client's locally trained model weights against discovery while still learning an accurate shared model. I note that the collusion problem can be solved using generic MPC, but such generic solutions are impractical due to computational

inefficiency. Our contribution is a practical and efficient solution to this problem using lightweight cryptographic tools.

2.3.1 Eliminating weight leakage

I use a secure weighted average protocol running across n clients to hide each client's model weights from the server where each weight is sent to the server encrypted/masked. The underlying secure aggregation protocol for online/non- drop-out clients I use appeared in the work of Bonawitz et al. [9], in which clients send individual updates to the server in an encrypted manner.

2.3.2 Eliminating weighted average leakage

Using the secure aggregation protocol of [9] hides all information about client weights from the server, but the final shared model can still reveal information about individual client weights and subsequently a client's local data set. Given the output which is the average of each model weight, $n - 1$ clients working together can remove their weights to discover the exact model weights of the remaining "honest" client.

Previous approaches augment the secure aggregation protocol with differential privacy to mitigate the impact of client data exposure. Under these protocols, each client independently generates and adds random noise to each model weight prior to transmission, so even in the case of $n - 1$ client collusion, only "noisy" weights can be recovered. This is a definite improvement, but unlike MPC it is a lossy one, and the scale of the added noise is limited by a trade-off against model accuracy.

I introduce a novel and efficient *oblivious distributed* differentially private mechanism. In prior works, each client picks its own local noise. By contrast, I offer a protocol where the noise for each party is added in an oblivious way. More specifically: For each weight, each client receives a tuple of encrypted noise terms from each other client and adds only a subset of them. Thus, a party P does not know the cleartext noise added to its weight and the other parties do not know which noise term is chosen by P.

I show that the information leakage on the honest client's weights after the collusion

Protocol 1 Privacy-Preserving Federated Logistic Regression Protocol Π_{PPFL} for a single weight

---

The protocol Π_{PPFL} runs with parties $P_1, \ldots, P_n$ and a server S. It proceeds as follows:
Inputs: For $i \in [n]$, party P_i holds input dataset D_i.
$\boldsymbol{\Pi_{\mathsf{PPFL}}.\mathsf{Setup}(1^{\lambda})}$: Each party P_i for $i \in [n]$ proceeds as follows for all $j \in [n]$ $(i \neq j)$:

- Generate random variables $\gamma_{i,j}^b$ and $\bar{\gamma}_{i,j}^b$ for $b \in \{0,1\}$ from the gamma $\mathcal{G}(1/n, scale)$ distribution with $scale = 2/(n * len(D_i) * \alpha * \epsilon)$. See Section 2.2.3 for the details on $scale$.
- Generate random masks $s_{i,j} \in \mathbb{Z}_q$.
- Compute masked noises $\eta_{i,j}^0 = s_{i,j} + \gamma_{i,j}^0 - \bar{\gamma}_{i,j}^0$ and $\eta_{i,j}^1 = s_{i,j} + \gamma_{i,j}^1 - \bar{\gamma}_{i,j}^1$.
- Run the Diffie-Hellman key Exchange protocol Π_{DH} to obtain a common shared key $r_{i,j}$ with each party P_j which is only known between parties P_i and P_j ($r_{i,j}$ is not known to S).
- Each party P_i sends $\eta_{i,j}^0, \eta_{i,j}^1$ to S who permutes and randomizes them and forwards to party P_j.

Given the above setup, I can compute the federated logistic regression model as follows:
$\boldsymbol{\Pi_{\mathsf{PPFL}}.\mathsf{WeightedAverage}(D_i, \{r_{i,j}\}_{j \in [n]})}$:

Round 1: Each party P_i proceeds as follows:

- Compute the weights w_i, using Equation (1), of the local logistic classifier obtained by implementing regularized logistic regression on input D_i.
  I describe the algorithm for a single weight, denoted by w_i:
- Generate a random bit vector $b = (b_1, \ldots, b_n)$ and send y_i to the server S: $y_i := w_i + \sum_{j=i+1}^{n} r_{i,j} - \sum_{k=1}^{i-1} r_{k,i} + \sum_{j=1}^{n} \eta_{j,i}^{b_j} - \sum_{j=1}^{n} s_{i,j} \bmod p$.

Round 2: The server computes $W = \big(\sum_{i=1}^{n} y_i \bmod p\big)/n$ and sends W to all parties.

$\boldsymbol{\Pi_{\mathsf{PPFL}}.Output(1^{\lambda}, W)}$: Each party P_i upon receiving W repeats WeightedAverage for the next weight or iteration of the logistic regression with locally updated common keys $r'_{i,j}$. (See Section A.11 in the Appendix for further discussion.)

---

attack is smaller than previous approaches. Our task is to enable the parties to calculate the sum of their inputs (i.e., $W = \sum_{i=1}^{n} w_i$), while ensuring privacy for an honest party in

the presence of a collusion attack given W. In prior works if $n-1$ parties collude, they subtract their weights and noise terms from W and then the final noise remaining in the transmitted weight of the honest party w_h is a single value chosen by the honest party. In our case, if $n-1$ parties collaborate then the final noise remaining in the transmitted weight of the honest party w_h is $n-1$ times larger, because the corrupted parties cannot subtract their noise terms.

At a high level, in our scheme, each client sends two encrypted noisy terms (permuted and randomized by the server) per model weight to the other clients, but each receiving client chooses only one of the two to add to each weight. Thus even if parties collude they cannot subtract a significant number of noise terms since they do not know which noise terms the honest client chose.

2.4 Secure Weighted Average Protocol

2.4.1 Our Protocol

I formally describe our weighted average protocol Π_{PPFL}, depicted in Protocol 1, for secure logistic regression performed by a set of clients $(P_1, \ldots, P_n)$ and a server S.

During setup, every pair of parties P_i and P_j will share some common randomness $r_{i,j} = r_{j,i}$. In the online weighted average phase, client P_i sends its weights masked with these common random strings, adding all $r_{i,j}$ for $j > i$ and subtracting all $r_{i,k}$ for $k < i$. That is, P_i sends to server S the following message for its data w_i: $y_i := (w_i + \sum_{j=i+1}^{n} r_{ij} - \sum_{k=1}^{i-1} r_{ki}) \bmod p$. Each weight received by the server is masked by $n-1$ large random numbers r, so it cannot accurately reconstruct any client's true model weight w_i. Because the total randomness applied to the weights sums to zero once the server computes W in round 2 of Π_{PPFL}, the averaged final model W will be identical to one calculated without security.

To establish common randomness r, each pair of parties run the standard Diffie-Hellman Key exchange protocol from the literature [23] communicating via the server. (See Appendix A.15 for description and listing.)

The protocol is given for a single iteration of federated logistic regression. For a detailed explanation of how the parties *locally* update their r masks to be used in the next weight and next iteration of the protocol, see Section A. of the Appendix. I generate the Laplacian noise in a distributed way by the use of gamma distributions $\mathcal{G}$ given that the Laplace distribution $\mathcal{L}$ can be constructed as the sum of differences of iid gamma distributions. To run machine learning algorithms and the DP mechanism which computes on rational values, I use field elements in a finite field $\mathbb{Z}_q$ to represent the *fixed-point values*. Concretely, for a fixed-point value $\bar{x}$ with k bits in the integer part and f bits in the decimal part, we use the field element $x := 2^f \cdot \bar{x} \mod q$ in $\mathbb{Z}_q$ to represent it.

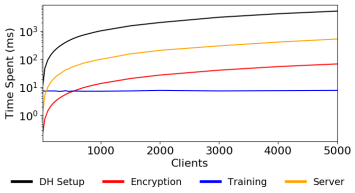


Figure 2-1: Component timing on Graph 1.

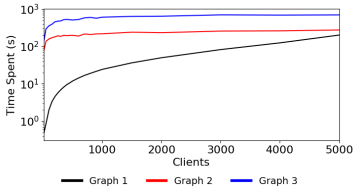


Figure 2-2: Total protocol time for Graphs 1-3.

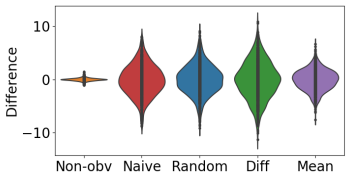
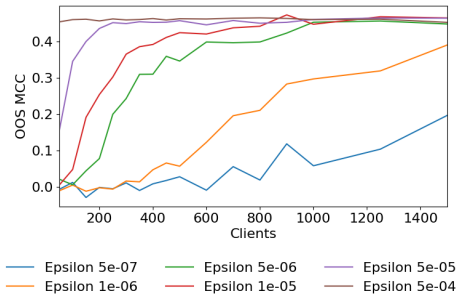
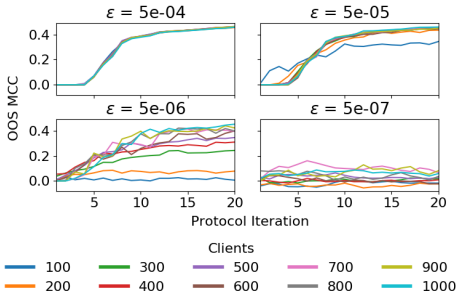


Figure 2-3: Error in attacker weight estimates.



(a) Final shared model by ϵ privacy loss parameter.



(b) Per protocol iteration by ϵ and client count.

Figure 2-4: Out of sample performance (Matthews Correlation Coefficient) for our oblivious distributed differential privacy protocol.

2.4.2 Security of our Protocol

I prove that our protocol protects the privacy of honest users in the semi-honest setting given the topology in Section 2.2.4. In particular I show the following theorem.

Theorem 1. *Suppose n clients $P_1, \ldots, P_n$ each hold private input w_i, and they wish to rely on a server S to compute the sum $f(w_1, \ldots, w_n) = \sum_i w_i$. There exists a protocol Π_{PPFL}, returning the sum which does not leak any information about the other clients' inputs except what can be inferred from the sum and offers collusion-privacy against a coalition of up to $t \leq n-1$ clients.*

In Appendix A.8 and A.9, I further formalize and prove our theorem, and consider security against $t \leq n-1$ semi-honest clients and a curious server, and against t malicious users.

Next I argue that the error term on the honest client's inputs after the collusion attack of t parties is larger than previous approaches. For this, I require the following additional property. Consider the case of $n-1$ collusion; I define collusion privacy as follows:

Collusion-Privacy: An n-party protocol provides *Collusion-Privacy*, for an aggregation function f and a probability distribution $\mathcal{D}$, if any adversary, who controls all parties except client P_h, learns no more than the honest party's values $w_h + \eta$ where $\eta \leftarrow \mathcal{D}$ and $f(w_1, \ldots, w_n)$.

In prior works if $n-1$ parties collude then the final noise left in the weight of the honest party w_h is a single value from $\mathcal{D}$. In our case, if $n-1$ parties collaborate then the final noise left in the weight of the honest party w_h is $n-1$ times larger than $\mathcal{D}$ since the corrupted parties cannot subtract their exact noise terms.

To measure the error, I quantify the difference between $f(D)$ and its perturbed value $\hat{f}(D)$ which is the error introduced by the differential private mechanism of the secure aggregation protocol.

Definition 2. (Error function) Let $D \in \mathcal{D}$, $f : \mathcal{D} \rightarrow \mathbb{R}$, and let $\delta = \frac{|f(D) - \hat{f}(D)|}{|f(D)|+1}$ (i.e., the value of the error). The error function is defined as $\mu = \mathbb{E}(\delta)$. The expectation is taken on the randomness of $\hat{f}(D)$. The standard deviation of the error is $\sigma = \sqrt{Var(\delta)}$.

After the execution of Protocol 1, parties receive the noisy sum of their inputs, i.e., $W = \sum_{i=1}^{n} w_i$, In prior non-oblivious works if $n-1$ parties collaborate and remove their weights from w then the final noise added to the weight of the honest party w_h is a value from $\mathcal{L}(\lambda)$, and hence, the error is $\mu = \frac{1}{|W|+1}\mathbb{E}|\mathcal{L}(\lambda)| = \frac{\lambda}{|W|+1}$.

In our oblivious case, if $n-1$ parties collaborate then the final noise added to the weight of the honest party w_h is $n-1$ times larger than $\mathcal{L}(\lambda)$, and hence, the error is $\mu = \frac{1}{|W|+1}\mathbb{E}|\sum_{i=1}^{n-1}\mathcal{L}(\lambda)| = \frac{(n-1)\cdot\lambda}{|W|+1}$.

However, in practice even if the parties cannot subtract their exact noise terms they can still try to subtract the average of the noise terms, or one of the two noise terms, or use the leakage L to reduce the amount of error. In our protocol I consider the leakage L learned from the difference of the noise terms η^0, η^1. Note that this leakage does not affect the error function given later in Definition 2. In Section 2.5.4 and Figure 2-5 I empirically show that such an attack is little better than the attack of subtracting nothing.

Note that the output of the aggregation protocol, $W+\eta$, is generated such that η follows exactly the same distribution in both non-oblivious and oblivious cases, but the noise left after an $n-1$ attack against the oblivious case is higher. For further discussion, see Appendix A.7 on collusion privacy.

2.5 Experiments

I empirically evaluated our protocol using ABIDES, an open source simulation platform originally designed for financial markets [11] and later adapted for federated learning [12]. Following the agent-based approach of these prior works, I simulated our oblivious protocol for 5,000 distributed clients and analyzed the timing, accuracy, and privacy of the empirical results.

2.5.1 Experimental Dataset and Method

I evaluated our protocol's performance using the Adult Census Income dataset [24], which provides 14 input features such as age, marital status, and occupation, that can be used to predict a categorical output variable identifying whether (*True*) or not (*False*) an individual earns over USD $50K per year. I used a preprocessed version of the dataset from Jayaraman et al. following the method of Chadhuri et al. which transformed each categorical variable into a series of binary features, then normalized both features and examples, resulting in 104 features for consideration. [41, 17] I added a constant intercept feature to permit greater

flexibility in the regression. Of the 45,222 records in our cleaned data set, there were 11,208 positive examples (about 25%), representing a moderately unbalanced dataset.

The dataset was loaded only once per complete simulation of the protocol, after which a randomized train-test split (75% vs 25%) was taken. Once per round of federated learning, each client randomly selected 200 rows from the training data as its "local" data. The holdout test data was the same for all clients, and no client ever trained on it. All clients implemented Protocol 1 as previously described.

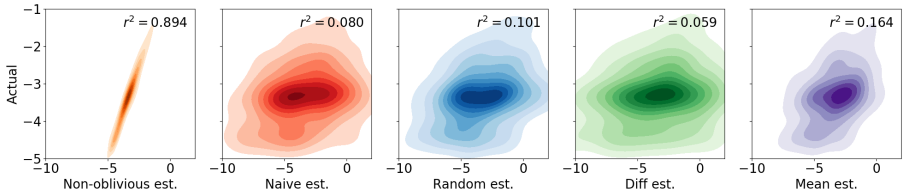


Figure 2-5: Density plot of actual versus estimated honest party weight over 1,000 iterations of the $n - 1$ collusion attack.

2.5.2 Protocol Timing Results

The ABIDES simulation permits construction of an arbitrary network graph with defined pairwise connectivity, minimum latency, and parameters for randomly selected "jitter" with nanosecond resolution. It captures the real elapsed runtime of each client activity and appropriately delays both sent messages and the earliest time at which a client may act again. Using these features, I have estimated the temporal load of Protocol 1. The mean time required to run the protocol simulation on a single Intel Xeon X5650 CPU core (2.6GHz) ranged from 32 seconds to 12 hours for 100 to 5,000 parties.

Figure 2-1 summarizes the time spent performing each section of our protocol on the adult census income data set: **Diffie-Hellman Setup** one time per client, **Encryption** of the weights and local model, **Training** per client per protocol iteration, and **Server** aggregation time per protocol iteration. Figure 2-2 shows the estimated time required to run the full protocol (not the simulation) for three different network graphs: **Graph 1** places all participants around New York City, **Graph 2** places the server in New York City and clients around London, **Graph 3** places the server in New York City and clients all over the world.

Tabular data is presented in Appendix A.122.

All experiments comprised 20 rounds of secure federated learning, with each client running 50 iterations of local regression training at each round. Latency is the most significant time component for small participant networks, but as the population size grows, computation effort surpasses it. Fortunately, the two largest components of computational time growth represent work performed only once per client for the entire protocol, and work performed only by the server.

2.5.3 Protocol Accuracy Results

The secure multi-party aggregation component of Protocol 1 is lossless, because the MPC encryption elements sum to zero in each shared model. Differential privacy introduces shared model accuracy loss inversely proportional to the ϵ privacy loss parameter. Smaller selections of ϵ result in more uncertainty about a client's local weights when other parties collude to reveal them, but increasingly confound learning. For example, in our protocol experiments with 200 clients, final model accuracy worsens dramatically once $\epsilon < 5e-5$.

Matthews Correlation Coefficient: Because of the significant (3:1) class imbalance in our data, I assess accuracy using the Matthews Correlation Coefficient (MCC) [63], a contingency method of calculating the Pearson product-moment correlation coefficient (with the same interpretation), that is appropriate for imbalanced classification problems. [5, 75, 77, 28] (See Appendix A.12 for more detail.)

In Figure 2-4a I show the MCC of our protocol's final shared model predictions against the correct values for a range of ϵ. As expected, smaller ϵ harms the accuracy of the learned model. Thus there is a dynamic lower bound, varying with population size, on useful values of ϵ. For example when considering out of sample $MCC(n)$, with n being the client population size, in our experiments with $\epsilon = 1e-5$: $MCC(100) = 0.005$, $MCC(200) = 0.254$, and $MCC(500) = 0.423$. For all evaluated client population sizes (50 to 5,000), models trained under Protocol 1 with $\epsilon \geq 5e-4$ had similar accuracy to unsecured federated learning. Figure 2-4b shows the impact ϵ can have on each round of federated learning: with $\epsilon = 5e-4$ or $\epsilon = 5e-7$, population size does not matter because

either all sizes succeed at learning or none do; but with $\epsilon = 5e-6$, varying client population sizes learn at vastly different rates.

In our simulated network environment, with 1000 clients implementing the described protocol for federated logistic regression using privacy loss parameter $\epsilon = 5e-4$, I found an out of sample final iteration relative accuracy loss (MSE versus learning in the clear) of $1.1e-6$ and an out of sample final iteration relative MCC loss of 0.0018.

2.5.4 Adversarial Data Recovery

Prior works like Bonawitz et al. discuss attacks from a "snooping" server which attempts to infer the unencrypted weights of a particular client. [9] The attacks fail since the server does not have the common random values r.

Collusion attack: I consider the $n-1$ attack, in which *all other clients* conspire to recover the unencrypted model weights of a single "honest" client. Let the honest client be h and the set of $n-1$ colluding clients be C. For a single model weight, let $F = w_h + W_C + T_h + T_C$ be the output of the aggregation protocol at the end of each iteration, where w_h is the honest party's original weight, T_h is the honest party's noise sum, $W_C = \sum_{c \in C} w_c$, $T_C = \sum_{c \in C} T_c$. Note that the sum of the randomness r, s is removed by design from the output when the computation is performed, so MPC cannot defend against this type of attack.

Under prior non-oblivious protocols in which each party generates and adds its own noise locally, the colluding parties know W_C and T_C and can therefore recover the honest party's noisy weights $W_h + T_h$. Under our oblivious protocol, the noise which cannot be subtracted in F has a more dispersed distribution that cannot be much narrowed by the colluding parties since h will have received from each colluding party c a choice of two difference of gamma privacy noises $\hat{\gamma}^0_{ch}$ and $\hat{\gamma}^1_{ch}$ and c will not know which was selected by h. Moreover, since the server permutes and randomizes the encrypted noise terms, T_C is also not precisely known to the colluding parties. For an extreme Sybil attack, the server will have to run a secure shuffle protocol which I have not implemented in this version (see discussion in Appendix A.9).

I empirically illustrate the dramatic improvement in privacy against an $n-1$ distributed attack by considering five cases. In each case, the colluding parties attempt to recover W_h and always remove W_C. In the **Non-Oblivious** case followed by prior works, the corrupted parties can accurately remove T_C. In the other four cases, Protocol 1 is attacked, and the corrupted parties must decide how to deal with the unknown noise choices by other parties i: under **Naive** they do nothing additional (i.e., they do not remove any noise terms); under **Random** each corrupted party c removes either $\hat{\gamma}_{ci}^0$ or $\hat{\gamma}_{ci}^1$ at random; and under **Diff** and **Mean** each corrupted party c removes the difference or mean of $\hat{\gamma}_{ci}^0$ and $\hat{\gamma}_{ci}^1$ respectively.

I consider the recovery attempt across 1,000 full iterations of Protocol 1 with 100 clients participating. At the end of every iteration, 99 clients share information in an attempt to recover the unencrypted model weights of the one honest client. Privacy loss parameter $\epsilon = 5e-4$ was selected because it did not cause significant shared model accuracy loss for any tested number of parties.

Figure 2-5 shows a density plot of the honest party's actual model weight versus the collaborators' estimate of that weight. Figure 2-3 summarizes the distribution of the difference between estimated and actual weights for each attack scenario. The $n-1$ attack is successful ($r^2 = 0.894$) against the prior non-oblivious protocol, but not successful ($r^2 = 0.164$ or worse) against our new oblivious protocol. (For additional discussion of the attacks, see Appendix A)

To further validate the new approach for oblivious distributed differential privacy, I also ran Protocol 1 and a previous non-oblivious protocol against the Kaggle Credit Card Fraud data set [22] and evaluated the success of the same cases of $n-1$ attacks. This data set provides transformed features that represent the first 26 principal components of unknown original features. Two original features are provided without transformation: the elapsed time from the start time of the dataset and the amount of the transaction. I used the Amount column without transformation, but excluded the Time column because our learning method does not attempt to identify temporal clusters or patterns. The dataset provides a categorical y variable identifying whether the transaction was judged fraudulent (True) or not (False). Of the 284,807 records, only 492 (less than 0.2%) are labelled fraudulent, representing an extremely unbalanced dataset. The data set was otherwise handled in exactly the same

manner as the adult census data set.

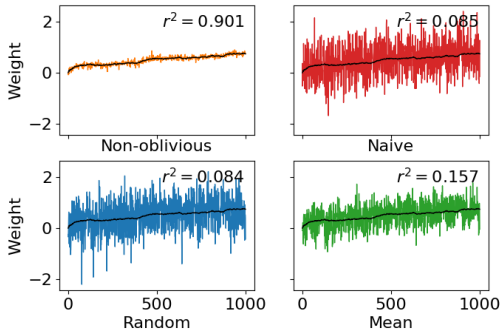


Figure 2-6: Actual (black) versus estimated (color) honest party weight over 1,000 iterations of $n - 1$ collusion attack.

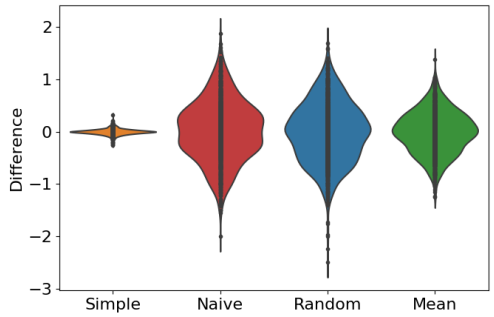


Figure 2-7: Violin plot showing distribution of difference between estimated and actual honest party weight.

In Figures 2-6 and 2-7, I show the result of 1,000 attempted $n - 1$ attacks against an arbitrarily-selected honest party weight, using the credit card fraud data set. Because of the randomness involved, the magnitude of difference varies across the model weights.

2.6 Summary

In this chapter, I presented an efficient mechanism for oblivious distributed differential privacy that is the first to secure against collusion attacks on the clients' model parameters, and leveraged that mechanism to construct a secure federated learning protocol. I also detailed the protocol and proved its security.

To empirically evaluate the protocol in a practical setting, I implemented it for a common data set with 5,000 parties in an open source simulation that has been adapted to the domain of privacy-preserving federated learning, and estimated its accuracy and running time for various client counts and values of the ϵ privacy loss parameter. I also conducted an $n - 1$ attack and showed that it is effective against prior non-oblivious protocols, but not against our new protocol.

I have not considered the case where clients drop off during the protocol as future work. Our mechanism is therefore well suited to cross-silo federated learning applications where clients are different organizations (e.g. medical or financial) or geodistributed datacenters, as opposed to mobile or IoT devices which can possibly go offline.

Chapter 3

Tackling Statistical Heterogeneity in FL

Standard FL algorithms involve averaging of model parameters or gradient updates to approximate the global model at the server. However, in heterogeneous settings (different clients have different data distributions) averaging can result in information loss and lead to poor generalization due to the bias induced by dominant clients. I hypothesize that to generalize better across non-iid datasets as in FL settings, the algorithms should focus on learning the invariant mechanism that is constant while ignoring spurious mechanisms that differ across clients. Inspired from recent work in the Out-of-Distribution (OOD) literature, I propose a gradient masked averaging approach for federated learning as an alternative to the standard averaging of client updates. This client update aggregation technique can be adapted as a drop-in replacement in most existing federated algorithms. I perform extensive experiments with gradient masked approach on multiple FL algorithms with in-distribution, real-world, and OOD (as the worst case scenario) test dataset and show that it provides consistent improvements, particularly in the case of heterogeneous clients.

3.1 Introduction

A challenge in FL is heterogeneity in the data distributed across clients. The non-iid data distribution degrades the performance of federated learning models [58, 94, 106]. One of the reasons for this is the loss of information regarding invariances across clients induced by the averaging of model parameters or updates. This is further exacerbated by the multiple local

steps taken by each client with the aim of reducing communication rounds which results in "client drift"[45]. Each client after multiple local steps can progress too far towards minimizing their local objective which may deviate from that of the global objective.

Recently [74] proposed an approach for improving generalizing across "environments" in OOD. In this work, I draw connections between the OOD setting and the federated learning setting, proposing to adapt the approach of [74] to the FL. Specifically, I propose a new aggregation method called gradient masked averaging with the goal of improving generalization across clients and of the global model. The gradient masked averaging can be plugged into any FL algorithm as an alternative to naive averaging of model parameters at the server. Intuitively, gradient masking prioritizes gradient components that are aligned with the overall dominant direction across clients while the inconsistent components of the gradient are given lesser importance. The proposed gradient masking approach improves the convergence of adaptive and non-adaptive FL algorithms.

3.1.1 Related Work

Federated Learning In FedAVG [64] for each communication round, all selected B fraction of clients perform E local steps of gradient descent with their local datasets. The model parameters from participating clients are averaged at the server to obtain the global model. It is equivalent to FedSGD [64] when $E = 1$ and each client performs stochastic gradient descent. Multiple local steps help minimize communication costs, which is a major bottleneck in FL. Quantization methods [80] and gradient descent acceleration [103] methods have been proposed to reduce communication overhead.

Convergence of FedAVG under i.i.d settings have been analyzed widely [86, 102, 91]. The convergence rate of FedAVG worsens with increasing heterogeneity among client datasets and this has been analyzed by several works [58, 94, 58]. Multiple variations of FedAVG have been proposed to improve convergence in non-iid data distribution settings, including adding regularization to the client objective [58], normalized averaging of model parameters [92], and introducing server momentum [38]. [45] uses control variates to reduce client drift. Adaptive optimizers like Adam and Yogi have been introduced to the federated

setting by [79]. Algorithms like PerFedAVG [29], Ditto [57], FedBABU [72] focus on personalization of clients. Differential privacy and blockchain have been used in FL to enhance data privacy in federated learning [95, 68]. Probabilistic Federated Neural Matching (PFNM) [105] and FedMA[90] addresses the problems due to permutation variances in the neural networks. [104] studied new approaches for evaluation FL in more realistic heterogeneous setting, proposing a notion of participation gap.

Out of Distribution Generalization In traditional machine learning, a model is evaluated based on its test performance on an unseen dataset drawn i.i.d from the train data distribution. However, this assumption may not hold true in real-world datasets and many supervised learning models do not perform well on related but non-iid test datasets. This problem is often referred to as the OOD generalization or the closely related domain generalization problem [50, 1].

This problem has been addressed in several works like Invariant Risk Minimization(IRM) [4], Risk Extrapolation(REx) [52], and Gradient Starvation [76]. These approaches typically focus on introducing penalties that learn invariant representations in a setting with known variations in the data (corresponding to environments). However, this idea cannot be easily ported to a federated learning setting as the clients performing the optimization steps would require access to the data of other clients. On the other hand [74] proposed a gradient agreement method based on gradient directions to learn features that agree across environments. This was extended by [83] to include gradient magnitude. In this chapter, I focus on these methods that utilize gradients from environments to learn invariant features. Distinct from the prior work, which considers the case of individual samples and single global updates, I consider and adapt this approach to a federated setting, where each client produces an aggregate update based on multiple gradient iterations.

3.2 Connections of FL to OOD Generalization

OOD generalization is often formalized using the notion of domains or environments. Under the formalism of [4] an environment corresponds to a data generating distribution that can be related through underlying (potentially unknown) causal variables to a set of other

environments. Different environments can arise during model training and testing, while it is typically assumed all environments (train and test) share some invariant mechanisms. They can however have spurious mechanisms that differ across environments [74, 13]. The concept of environments can be related to the federated learning setting by considering each client as producing a set of data generated from a different environment. All clients however have underlying invariant mechanisms to be considered for training a global model. Each client also has their specific spurious mechanisms or data distributions. For example consider a scenario of different clients corresponding to smartphone users taking pictures of fruit, to build a model that identifies fruit items. Each smartphone may have a different camera and each user may take pictures of different subsets of fruit. Thus the clients may differ in terms of the label distribution of their local data and the camera related image characteristics, which can be a spurious mechanism while the overall set of food items is invariant across clients.

The objective of OOD generalization is to improve the performance of a model on data from distributions that are related yet different from the training data distribution. [3] quantifies the following objective for $R^{OOD}(f) = \max_{e \in \xi_{all}} R^e(f)$ where $R^e(f)$ is the risk or expected loss for data from environment e, which belongs to ξ_{all}, a large (often infinite) family of distinct yet related environments $\xi_{tr} \subset \xi_{all}$. In practical federated learning, one of the major objectives of the global model is to improve its performance on non-participating clients (clients that do not contribute to global model training [43]) and on new train clients (participating clients that are new to the federated network). The data at these clients will be from related distributions having the same invariant mechanism but may differ from data distributions at train clients. Hence, one way to frame the goal for FL global models is to enhance the performance across a large set of related clients which may have different data distributions. I can quantify this as $\min R_{gFL}(f) = \max_{n \in N_{all}} R^n(f)$ where $R^n(f)$ is the risk at client $n \in N_{all}$, and N_{all} is a large set of distinct yet related clients which have different data distributions.

3.3 Methods

In the following section I introduce the notations used, review standard federated aggregation, and then introduce our gradient masked averaging for global model approximation.

3.3.1 Federated Aggregation

Consider a federated setting having N clients where the data at each client, $n \in N$ is $D^n = (x_i^n, y_i^n)$. The clients collectively learn a function, $f : X \rightarrow Y$ $f(x \in X; w)$ that, in our case, corresponds to a neural network model with parameters, w, and $(X, Y) = \{(x_n, y_n) : \forall n \in N\}$ is the entire set of data distributed across clients. At each communication round, k, the parameters of the global model, w_k, are sent to participating clients who perform multiple local gradient steps to obtain an update, Δ_n^k, corresponding to the difference between the clients model after multiple updates and w_k. In most FL algorithms the global model update at k^{th} communication round is then obtained as

$$w^{k+1} = w^k - \eta_g \Delta^k \tag{3.1}$$

Where, η_g is the global learning rate. Δ^k is the update or "pseudo-gradient" at the k^{th} global communication round obtained by aggregating the updates from the participating clients $(\Delta_n^k; n \forall N)$ as

$$\Delta^k = \frac{1}{|N|} \sum_{n \in N} \Delta_n^k; \tag{3.2}$$

In the case of a single gradient step at each client the update, Δ^k, corresponds the gradient of the global objective. Each client has a different data distribution and thus different loss surface. [74] show that averaging of gradients across environments leads to poor consistency of solutions, and reduced generalization, particularly to unseen environments. Indeed naive averaging of parameters fails to capture the consistencies in the loss landscapes due to the bias that may be induced by dominant features in the environments as explained by [83]. This is further exacerbated in real world federated settings as there are multiple possible scenarios where some clients dominates over others. For example, some clients may have better computational resources, connectivity, or more data and contribute disproportionately

to the global update.

3.3.2 Gradient Masked Aggregation

[74] highlights several issues associated with the standard arithmetic mean, which they equate to a logical OR, used combine gradients. They propose to use an analog of the logical AND operation resulting in taking geometric mean of sample gradients. The gradient components that are "inconsistent" in sign across environments are set to 0. Specifically they construct a binary vector, $m_\tau(\delta)$ based on the agreement of gradient components among environments. The components j of the mask, m_τ is computed as

$[m_\tau]_j = \mathbb{1}[\frac{1}{|N|}\sum_{e\in\eta}\text{sign}([\nabla L_e]_j) \geq \tau]$.

Here ∇L_e is the gradient of the loss with respect to environment e, and $\tau \in [0, 1]$ is a hyperparameter.

Direct application of this idea to FL setting is however challenging. [74] applies the rule assuming each sample represents an environment, whereas each client more naturally corresponds to the environment in FL. Furthermore, they show that this can lead to a slower convergence rate in practice as too many components can be masked at each iteration. This would be impractical in the federated setting as I would not want to sacrifice convergence speed alone for generalization. In the federated setting I propose a variant of this mask that doesn't sacrifice convergence speed while retaining some of the improved generalization properties. Specifically I propose to use masking at the aggregation stage of standard FL, with a mask computed based on each client update (which arises from multiple local gradient steps). The mask is calculated based on sign agreement among client updates Δ_n and it is applied on the global model update Δ_n^k. This masking controls the parameter update based on the agreement of direction among the gradients across clients or environments. To provide rapid convergence I apply a soft masking procedure instead of the hard binary mask.

I define an agreement score, $A \in (0, 1]$, given as a function of all the client updates,

$$\text{A} = \& \left| \frac{1}{|N|} \sum_{n\in N} \text{sign}(\Delta^n) \right|$$

The mask $\tilde{m}_\tau$ is now defined element element-wise as

$$[\tilde{m}_\tau]_j = \&1 \textbf{ if } \mathrm{A}_j \geq \tau \textbf{ else } \mathrm{A}_j \tag{3.3}$$

The global model update is given by $\tilde{m}_\tau \odot \Delta_t$. This ensures that the updates to the global model are with respect to their agreement across clients. When the agreement across clients is greater than the hyperparameter τ, it would be assigned 1 and when the agreement is lesser than τ, the mask value would be equivalent to the agreement score. This real mask ensures that each parameter updates but the magnitude is adjusted to be proportional to the agreement across clients.

This masking can be easily plugged in to any FL algorithm as part of the aggregation at the server that involves gradient or parameter averaging. In Alg. 1 the full algorithm for Gradient Masked Aggregation is given for the case of FedAVG, and for adaptive methods FedADAM and FedYogi.

3.4 Method Analysis

I now analyze the convergence properties of the masking, focusing on the case of FedAVG and gradient masked aggregation. In our setting, I define the global objective function in terms of the local objective, F_n, of each client

$$\min_w f(w) = \min_w \sum_{n=1}^{N} F_n(w)$$

and I assume that $\mathbb{E}[f_n(w)] = F_n(w)$.

Following [79], I make the following standard assumptions. I write $\mathcal{F}_t$ the filtration adapted over our stochastic process at time t.

Assumption 2 (Lipschitz gradient)**.** I assume that each client objective has Lipschitz gradient with constant L, meaning that there exists $L > 0$, $\forall n, \forall w, v$, $\|\nabla F_n(w) - \nabla F_n(v)\| \leq L\|w - v\|$

Algorithm 1 Gradient Masked FedAVG [64], FedADAM, and FedYogi [79]

Initialize w_0
for each server epoch, t = 1,2,3,... **do**
  Choose C clients at random
  for each client in C, n **do**
    $w_t^n = \text{ClientUpdate}(w_{t-1})$
    $\Delta_t^n = \frac{n_k}{\sum_{n=1}^{N} n_k}(w_t^n - w_{t-1})$
  end for
  $\Delta_t = \sum_{n=1}^{N} \Delta_t^n$
  $z_t = \beta_1 z_{t-1} + (1-\beta_1)\Delta_t$
  $v_t = v_{t-1} - (1-\beta_2)\Delta_t^2 \text{sign}(v_{t-1} - \Delta_t^2)$
  $v_t = \beta_2 v_{t-1} + (1-\beta_2)\Delta_t^2$
  $\Delta_t = \frac{z_t}{\sqrt{v_t}+e^{-3}}$
  $b = \tilde{m}_\tau(\{\Delta_t^n\}_{n=1..C})$
  $w_t = w_{t-1} - \eta_g * b \odot \Delta_t$
end for

ClientUpdate(w):

Initialize $w_0 = w$
for each local client iteration, i=0,1,2,3,..,n **do**
  $g_i = \nabla_{w_i} L(w_i)$
  $w_{i+1} = w_i - \eta_c \, g_i$
end for
return w_{i+1} to server

Assumption 3 (Bounded gradients)**.** I assume that each client has a bounded gradient by G, leading to: $\exists G > 0, \forall n, \forall w, \|\nabla F_n(w)\| \leq G$.

Assumption 4 (Finite variance)**.** I assume a global bound on the variance of the gradient estimate of each individual client, meaning that: $\exists \sigma > 0, \forall n, \forall w, \mathbb{E}\|\nabla F_n(w) - \nabla f_n(w)\|^2 \leq \sigma^2$.

Lemma 1 (Bounded drift from client update, Appendix A, Lemma 3 of [79])**.** . *Given Assumptions 2, 3,4 there exists $C > 0$ such that for any time step t and x_t, Δ_t obtained from Alg. 1, for any client $n \leq N$:*

$$\mathbb{E}[\|\Delta_t - \nabla F_n(w_t)\|^2] \leq C(\sigma^2 + \mathbb{E}[\|\nabla f(w_t)\|^2])$$

This Lemma involves the aggregation at every step t of the local client updates obtained individually on each client. In particular, it does not depend on the server's algorithm. Due to this the proof from Appendix A, Lemma 3 of [79], which gives the explicit C, applies directly.

The next proposition derives a rate of convergence on the masked gradient which is similar to [79], and in the order of $\mathcal{O}(\frac{1}{T})$.

Proposition 5 (Convergence analysis)**.** *Given Assumptions 2, 3,4. If $\eta_g \leq \frac{1}{2L}$, then, one has the following rate over the masked gradients given by the FedAVG algorithm in Alg. 1:*

$$\mathbb{E}[\min_{t<T} \|b \odot \nabla f(w_t)\|^2] \leq 2L \left(\frac{f(w_0) - f(w_T)}{T} + C\eta_g \left(\sigma^2 + G^2\right) \right)$$

Proof. I consider the optimization path given by Alg 1. Let us write $\tilde{\Delta}_t = b_t \odot \Delta_t$. First, I note that given that $0 \leq b_t^j \leq 1$, I get $\|\tilde{\Delta}_t\| \leq \|\Delta_t\|$. Next I follow the approach of [10] for

obtaining optimal non-convex bounds. Each f_n is L-smooth, thus:

$$\begin{aligned}
&F_n(w_{t+1}) \\
&\quad \leq F_n(w_t) + \langle \nabla F_n(w_t), w_{t+1} - w_t \rangle + \frac{L}{2}\|w_{t+1} - w_t\|^2 \\
&\quad = F_n(w_t) - \eta_g \langle \nabla F_n(w_t), b_t \odot \Delta_t \rangle + \frac{L}{2}\|b_t \odot \Delta_t\|^2 \\
&\quad \leq F_n(w_t) - \eta_g \langle \nabla F_n(w_t), b \odot \Delta_t \rangle + \frac{L}{2}\eta_g^2\|b_t \odot \Delta_t\|^2
\end{aligned}$$

Averaging over $1 \leq n \leq N$ and conditioning over $\mathcal{F}_t$ leads to:

$$\begin{aligned}
&\mathbb{E}[f(w_{t+1})|\mathcal{F}_t] \\
&\quad \leq f(w_t) - \eta_g \langle \nabla f(w_t), b_t \odot \Delta_t \rangle + \frac{L}{2}\eta_g^2\|b_t \odot \Delta_t\|^2 \\
&\quad = f(w_t) - \eta_g \langle \nabla f(w_t), b_t \odot (\nabla f(w_t) - \nabla f(w_t) + \Delta_t) \rangle \\
&\qquad\qquad + \frac{L}{2}\eta_g^2\|b_t \odot \Delta_t\|^2
\end{aligned}$$

Now, I use the inequality: $\langle a, b \rangle \leq \|a\|\|b\| \leq \frac{1}{2}(\|a\|^2 + \|b\|^2)$ and noting the masking can be seen as multiplication by diagonal matrix, I obtain:

$$\begin{aligned}
&\langle \nabla f(w_t), b \odot (\nabla f(w_t) - \Delta_t) \rangle \\
&\qquad = \langle b \odot \nabla f(w_t), \nabla f(w_t) - \Delta_t \rangle \\
&\qquad \leq \frac{1}{2}\|b \odot \nabla f(w_t)\|^2 + \frac{1}{2}\|\nabla f(w_t) - \Delta_t\|^2 .
\end{aligned}$$

From the Bounded gradients and Lemma 1, I get:

$$\begin{aligned}
\mathbb{E}[\|\nabla f(w_t) - \Delta_t\|] &\leq C(\sigma^2 + \mathbb{E}[\|\nabla f(w_t)\|^2] \\
&\leq C(\sigma^2 + G^2)
\end{aligned}$$

Since $0 \leq b^j \leq 1$, I get:

$$-\nabla f(w_t)^j \times b_t^j \times \nabla f(w_t)^j \leq -(b^j)^2 (\nabla f(w_t)^j)^2 \,,$$

which implies that:

$$-\langle \nabla f(w_t), b \odot \nabla f(w_t) \rangle \leq -\|b \odot \nabla f(w_t)\|^2$$

Taking the expectation, and summing, I have:

$$\frac{1}{2}(\eta_g - L\eta_g^2) \sum_{t=0}^{T-1} \mathbb{E}[\|b \odot \nabla f(w_t)\|^2] \\ \leq f(x_0) - f(x_T) + \eta_g T C(\sigma^2 + G^2)$$

In particular, this implies for a learning rate $\eta_g = \frac{1}{2L}$ small enough such that $\eta_g - L\eta_g^2 = \frac{1}{2L} > 0$

$$\mathbb{E}[\min_{t<T} \|b \odot \nabla f(w_t)\|^2] \\ \leq 2L \left(\frac{f(w_0) - f(w_T)}{T} + C\eta_g \left(\sigma^2 + G^2\right) \right)$$

□

I now observe that under assumptions similar to those proposed in [74], the distribution of updates will match the true underlying distribution.

Proposition 6 (Mask stability)**.** *Denote δ, δ^n the r.v. corresponding respectively to a coordinate of Δ, Δ^n. Furthermore consider $\tilde{\delta}$ the r.v. for each coordinate of $\tilde{\Delta}$, where $\tilde{\Delta} = b \odot \Delta$. Assume that δ^n is σ-subGaussian, that the δ_n are mutually independent and write $\mu^n = \mathbb{E}[\delta^n]$. If $\frac{1}{N}\texttt{card}(\{n|\mu^n > 0\}) > \tau$, then, with probability $1 - \mathcal{O}\left(e^{-\frac{(\inf_{\mu_n > 0} \mu_n)^2}{\sigma^2}}\right)$, I obtain $\tilde{\delta} = \delta$.*

Proof. I show a lower bound on δ^n. With probability $1 - e^{-\frac{t^2}{\sigma^2}}$, I get:

$$|\delta^n - \mu^n| < t \tag{3.4}$$

Table 3.1: Average in-distribution test performance(%) over the last 10 communication rounds of FedAVG, FedProx, SCAFFOLD, FedAdam, FedYogi and their GMA versions on i.i.d and non-iid distributions of MNIST, FMNIST, FEMNIST, and CIFAR-10. The best result among AVG and GMA versions of each algorithm and dataset is shown in bold.

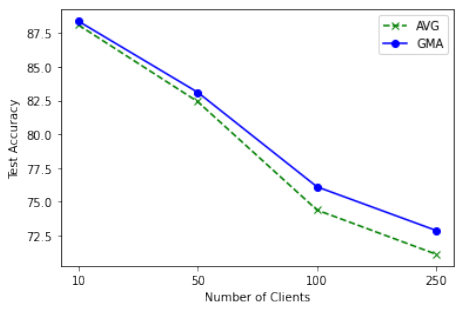
Dataset (Model)		**FedAVG**		**FedProx**		**SCAFFOLD**		**FedADAM**		**FedYogi**	
		AVG	**GMA**	**AVG**	**GMA**	**AVG**	**GMA**	**AVG**	**GMA**	**AVG**	**GMA**
MNIST	**IID**	99.1	**99.16**	99.13	**99.17**	99.13	**99.21**	98.71	**98.88**	98.71	**98.88**
(LeNet)	**Non-IID**	**98.9**	98.87	98.78	**98.88**	**99.01**	98.96	98.48	**98.6**	98.54	**98.9**
FMNIST	**IID**	89.14	**90.52**	90.07	**91.39**	90.04	**90.77**	88.62	**90.41**	88.58	**90.02**
(LeNet)	**Non-IID**	88.1	**88.38**	87.61	**87.7**	87.67	**88.31**	86.8	**87.28**	86.61	**87.5**
FEMNIST	**IID**	**99.7**	99.68	99.44	**99.5**	99.6	**99.62**	99.5	**99.75**	**99.71**	99.69
(LeNet)	**Non-IID**	94.2	**96.04**	95.18	**95.61**	**95.12**	94.88	95.82	**97.26**	95.6	**96.67**
CIFAR-10	**IID**	87.3	**87.61**	87.18	**87.5**	86.58	**86.72**	86.9	**87.7**	87.53	**87.78**
(ResNet)	**Non-IID**	83.25	**83.95**	83.87	**84.4**	84.01	**85.36**	83.53	**84.84**	83.17	**84.55**

Let's thus pick $\tilde{t} = \inf_{\{\mu^n > 0\}} \frac{\mu^n}{2}$. By considering the intersection of those events, it implies that with probability at least $1 - e^{-\frac{\tilde{t}^2}{\sigma^2}}$, $\delta^n > \mu^n - \frac{\mu^n}{2} = \frac{1}{2}\mu^n > 0$. Consequently, the mask is equal to 1 and $\delta^n = \tilde{\delta}^n$. Now, I can note that $\inf_{\mu_n > 0} \mu_n > \inf_{\mu_n \neq 0} |\mu_n|$, which allows to conclude the proof. □

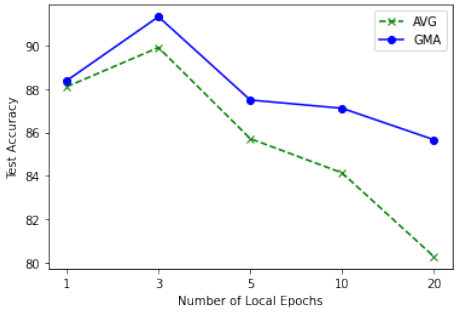
Informally I can see that if w_t is far from a local minimum then the masked gradient $b_t \odot \Delta_t$ is likely to not be equal to 0 thanks to Prop 5.6. Thus, Prop 5.5 suggests that the norm of the gradient is decreasing.

3.5 Experiments

In this section, I show empirically that the proposed GMA tends to outperform standard aggregation (AVG), converging at similar or better than standard aggregation, while enhancing the global model generalization. I observe this for multiple FL algorithms with respect to multiple datasets and data distributions (i.i.d and non-iid). Evaluations were conducted on in-distribution settings, real-world, and OOD test datasets.



(a) Final shared model by ϵ privacy loss parameter.



(b) Per protocol iteration by ϵ and client count.

Figure 3-1: (a) Test accuracy vs. Number of selected clients in the federated network. (b) Test accuracy vs. number of local epochs per client in each communication round. The experiment was on non-iid distributed FMNIST using a LeNet model. In all cases, GMA outperforms naive averaging.

Implementation I conduct experiments on gradient masked and naive versions of non-adaptive federated optimizers like FedAVG [64], FedProx [59], and SCAFFOLD [45] and adaptive optimizers like FedADAM and FedYogi [79] across a variety of datasets. Experiments involve both i.i.d and non-iid distributions of data. The heterogeneity simulated is label distribution skew similar to that in [64]. All algorithms were trained until convergence and the average test accuracies over the last 10 communication rounds are reported. All algorithms were implemented on Pytorch.

Hyperparameters An SGD optimizer with a momentum ($\rho = 0.9$) and cross-entropy loss was used to train each client for $E = 1, 3, 5, 10$ or 20 client epochs before aggregation at the server in all our experiments unless specified. For experiments with non-convex objectives, LeNet architecture was employed at all clients and at the global model for all datasets except CIFAR-10, which used a ResNet18 with group norm [79]. The momentum parameters of adaptive federated optimizers are fixed at $\beta_1 = 0.9$ and $\beta_2 = 0.99$ as per [79]. For each of the considered algorithms I tune the local client model learning rates and global model learning rates to consider the best performances of the algorithms. More details on the hyperparameters used are given in Appendix B.3.7.

3.5.1 In-Distribution Evaluation

This is the most widely considered setting in the FL literature. The global model is evaluated on a test dataset sampled from data at all clients irrespective of the data distribution across clients. This test dataset is a representation of all participating clients. The datasets used for in-distribution testing are MNIST [54], Fashion MNIST [97], FEMNIST (Federated EMNIST) [15, 21], and CIFAR-10 [51], distributed i.i.d and non-iid across clients.

Table B.4 shows the test performance of the algorithms and their GMA versions for the case $E = 1$ and $N = 10$. It was observed that in a majority of the cases across algorithms and datasets, GMA outperforms naive averaging. The difference in improvement is more significant when the data distribution is non-iid. The major reason for this is that gradient masking is capable of focusing on learning the invariances even under increased spuriousness of non-iid data distribution. Across datasets, I can observe that the improvement with gradient masking is more prominant on CIFAR10 and FMNIST, which are relatively complex datasets than other datasets considered. Furthermore, in Appendix B.3.6 I show an ablation comparing GMA and AVG when the same client and global rates are used, showing that for nearly any hyperparameter choice GMA outperforms AVG, suggesting it is highly robust to the choice of hyperparameters.

Using the non-iid distributed FMNIST data I further study how the performance is affected as the number of clients grows and the number of local epochs increases. The results are shown in Figure 3-1a and Figure 3-1b. It can be observed that gradient masking increasingly outperforms naive averaging in these more complex scenarios. In Figure 3-1a I observe that with increasing number of clients, difference between the test accuracies corresponding to GMA and naive averaging increases. This validates the enhanced invulnerability of gradient masking to the bias that could be induced by one or more clients in the network. In Figure 3-1b I observe increasing local epochs beyond 3, the test accuracy decreases due to client drift [45]. Gradient masking is however much more robust in this (challenging) scenario.

Table 3.2: Real-World and Out-of-Distribution Evaluations. Average test performance(%) over the last 10 communication rounds of FedAVG, FedProx, SCAFFOLD, FedAdam, FedYogi and their GMA versions on real-world distribution of FEMNIST and i.i.d and non-iid distributions of FedCMNIST and FedRotMNIST. The best result among AVG and GMA of each algorithm is shown in bold.

Dataset (Model)		**FedAVG**		**FedProx**		**SCAFFOLD**		**FedADAM**		**FedYogi**	
		AVG	GMA	AVG	GMA	AVG	GMA	AVG	GMA	AVG	GMA
FEMNIST (LeNet)	Real-World	99.34	**99.4**	99.23	**99.31**	98.88	**99.3**	99.12	**99.16**	99.17	**99.18**
FedCMNIST (LeNet)	IID	89.73	**90.07**	**90.15**	90.12	90.06	**90.93**	89.88	**90.22**	89.91	**90.03**
	Non-IID	85.92	**88.59**	85.62	**90.02**	83.22	**85.57**	87.06	**88.93**	85.33	**88.04**
FedRotMNIST (LeNet)	IID	98.69	**98.87**	98.67	**98.86**	97.78	**98.85**	98.1	**98.39**	**98.52**	98.44
	Non-IID	96.18	**96.78**	96.87	**97.29**	91.28	**94.13**	95.11	**96.56**	95.51	**97.38**

3.5.2 Real-World Evaluation

Though in-distribution evaluation is the most common global model evaluation in FL, it does not capture some key aspects of practical settings. In the practical world, the data across clients is heterogeneous and the clients which deploy the global model (including test clients, non-participating clients, and new clients in the federated network) can have data distribution different from that at any train clients. Hence, evaluating on samples from data at train clients is less meaningful in a federated setting with these additional clients containing OOD data. The practical setting can be simulated by using a realistic federated data distribution provided by Federated EMNIST [15]. Specifically I use test data consisting of data from the same domain as the train dataset across clients but from one user (or a set of users) not included in the set of train clients. The test performance of the algorithms and their gradient masked alternatives are given in Table 3.2. I observe that gradient masking outperforms naive averaging. In the next section I consider another type of more systematic OOD situation to further evaluate GMA.

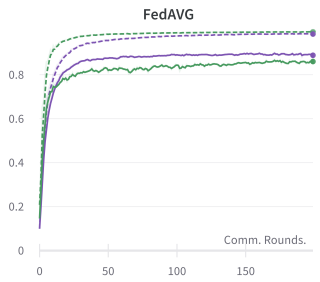


Figure 3-2: FedAvg

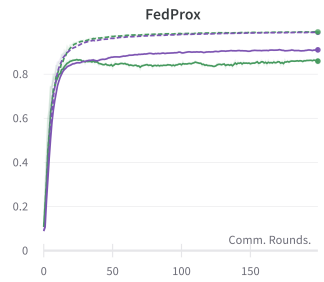


Figure 3-3: FedProx

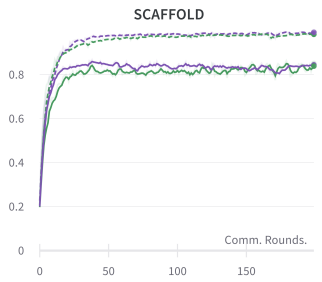
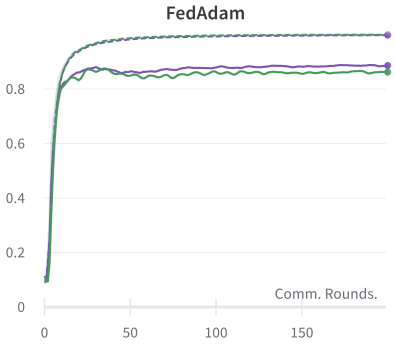
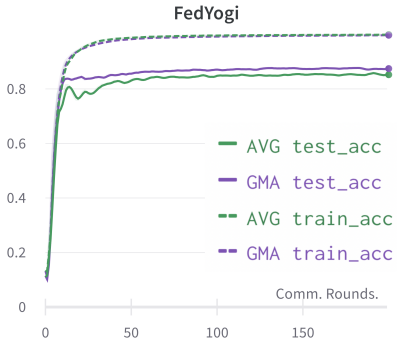


Figure 3-4: Scaffold



(a) FedAdam



(b) FedYogi

Figure 3-5: Train accuracy and test accuracy vs. communication rounds of gradient masked and naive averaging versions of the algorithms on FedCMNIST distributed non-iid across clients. I observe that GMA versions generalize better in all algorithms.

3.5.3 Out-of-Distribution Evaluation

A more complex OOD test can be implemented to better understand the performance of gradient masking in a worst case scenario. For this I induce a spurious mechanism in the train clients and in the test besides the class label based heterogeneity for non-iid data distribution. The global model would be tested on a dataset having a spurious mechanism that was not present in any of the train clients.

I use FedCMNIST, a federated multiclass version of CMNIST [4]. The invariant mechanism here is the digit. There also exists a spurious mechanism marked by a color given to the numbers. The color is digit specific to induce correlation to the label. I also use FedRotMNIST, inspired by [30], where an angle of rotation is the spurious mechanism. A more detailed discussion on the datasets is given in Appendix B.2. The performance of the various algorithms and their gradient masked averaging counterparts on these OOD test

datasets is given in Table 3.2. It is to be noted that across all datasets, data distribution, and algorithms gradient masking outperforms naive averaging. Figure 2 shows train and test curves of the algorithms and their gradient masked alternatives on non-iid distribution of FedCMNIST. It is to be noted that although GMA train accuracies are less than or equal to that of naive averaging, the GMA test accuracies are higher. This indicates that gradient masked versions generalize better than the naive versions of the algorithms.

3.5.4 Convex Objective

To further understand the performance of gradient masking in the convex setting, I experiment with MNIST and FedAVG on both i.i.d and non-iid data distributions and it was observed that GMA outperforms naive averaging in the non-iid setting. A logistic regression model with SGD with momentum optimizer was used at the clients for these experiments. When data distribution was i.i.d, GMA was converging to an average (over last 10 communication rounds) of 92.5% test accuracy while naive averaging obtains to 92.4%. Furthermore, when the data distribution across clients was non-iid, the enhancement in performance was more significant with gradient masking. While naive averaging was converging to 87.0% test and 92.0% train, while GMA reached 88.5% test and 92.2% train. Further demonstrating GMA can generalized better in the non-iid case.

Applications to privacy [89] suggests that algorithms focusing on learning the causal mechanisms provide stronger privacy guarantees in certain cases, for example they can be more robust to membership inference attacks and model inversion attacks. I make a preliminary assessment of the privacy benefits of the proposed GMA on membership inference attacks. Specifically, I simulate the membership inference adversarial attacker model on the naive averaging and gradient masked FedAVG models on Non-IID CIFAR-10 (ResNet) that reported the performances shown in Table B.4. I observe that the attack accuracy with respect to the GMA model is 55% while that of naive averaging model is 57%. This suggests that gradient masking can potentially enhance robustness to membership inference attacks. More details is given in Appendix B.4.3.

3.6 Summary

In this chapter, I proposed a new aggregation scheme applicable to a wide variety of federated learning algorithms. The proposed method, gradient masking enhances generalization performance of the global model in FL by focusing on learning the invariances across clients. The simple masking outperforms earlier naive averaging versions across a variety of algorithms and datasets. The theoretical analysis shows the convergence of the proposed masking algorithm and the stability of the proposed mask. Future directions include exploration of masks incorporating magnitude and other methods to better capture the invariances, thus leading to better generalization at the global model.

Chapter 4

Personalized and Communication-Efficient FL

High communication costs, data heterogeneity across clients, and lack of personalization techniques hinder the development of FL. In this chapter, I propose FedLTN, a novel approach motivated by the well-known Lottery Ticket Hypothesis to learn sparse and personalized lottery ticket networks (LTNs) for communication-efficient and personalized FL under non-identically and independently distributed (non-IID) data settings. Preserving batch-norm statistics of local clients, postpruning without rewinding, and aggregation of LTNs using server momentum ensures that our approach significantly outperforms existing state-of-the-art solutions. Experiments on CIFAR-10 and TinyImageNet datasets show the efficacy of our approach in learning personalized models while significantly reducing communication costs.

4.1 Introduction

Vanilla FL constructs a server model for all clients by averaging their local models, while postulating that all clients share a single common task. However, this scheme does not adapt the model to each client. For example, platforms like Youtube and Netflix require a unique personalized model for each of their clients. Most FL algorithms focus on improving the average performance across clients, aiming to achieve high accuracy for the global server

model. However, certain clients might perform poorly while others perform extremely well. This is not the ideal scenario for a fair and optimal FL algorithm. When deployed on edge devices in the real world, local test accuracy is instead a more important metric for success.

In addition to the above-mentioned personalization problem, sending and receiving model parameters is a huge bottleneck in FL protocols as it could be expensive for resource-constrained clients. It is important to reduce the total number of communication rounds and the size of the packets that are transmitted during every round. Unfortunately, there is usually a tradeoff between model accuracy and communication cost accrued during the federation process. For instance, techniques that speed up accuracy convergence or decrease the model size may result in a small decrease in accuracy.

While there have been numerous papers that address each of these challenges individually (See Section 4.1.1 for related work), finding one approach that provides solutions for all of them has proven to be difficult. LotteryFL [56] provided the first attempt at addressing these issues in one protocol. It is motivated by the Lottery Ticket hypothesis (LTH), which states that there exist subnetworks in an initialized model that provides the same performance as an unpruned model. Finding these high-performing subnetworks are referred to as finding winning lottery tickets. LotteryFL obtains personalized models by averaging the lottery tickets at every federated round. This also improves the communication costs as only the lottery tickets are communicated across the client and server instead of the whole model. However, LotteryFL fails to achieve the same performance in terms of pruning as obtained in Lottery Ticket Hypothesis (LTH). LotteryFL models are pruned only up to 50% compared to 90% or more in the non-federated LTH setting. This is due to the fact that pruning in LotteryFL takes a lot of time – the authors claim that it takes around 2000 federated rounds to prune around 50% of the model.

The slow pruning process of LotteryFL presents major drawbacks. In many experimental settings, local clients have difficulty reaching the necessary threshold for accuracy to prune for most rounds. On more difficult tasks, some clients may never reach the threshold accuracy. Consequently, they fail to find winning lottery tickets and do not reach personalized and more cost-efficient models. To avoid this in the LotteryFL approach, the accuracy threshold must be lowered, which subsequently lowers the efficacy of finding the right lottery ticket

networks for each client.

Moreover, LotteryFL uses evaluation measures such as average test accuracy to compare their work with baselines. I argue that these measures can easily misrepresent the performance of a federated training paradigm. In FL, it is also essential that each client achieves a fair performance, an aspect that is not captured by the average test accuracy. I introduce an evaluation metric based on the minimum client test accuracy to solve this problem. I find that LotteryFL sometimes achieves a lower minimum client accuracy than FedAvg.

To address the above-mentioned challenges, I present FedLTN, a novel approach motivated by the Lottery Ticket Hypothesis to learn sparse and personalized Lottery Ticket Networks (LTNs) for communication-efficient and personalized FL under non-IID data settings.

4.1.1 Related Work

FedAvg is a simple and commonly used algorithm in federated learning proposed in the seminal work of [64]. Most existing FL algorithms are derived from FedAvg where the goal is to train a server model that tries to perform well on most FL clients. However, FL still faces numerous challenges, among which convergence on heterogeneous data, personalization, and communication cost are the most pressing problems.

Performance on heterogeneous (non-IID) data

FedAvg is successful when clients have independent and identically distributed data. However, in many real-world settings, data is distributed in a non-IID manner to edge clients and hence leads to client drift [44]. That is, gradients computed by different clients are skewed and consequently local models move away from globally optimal models. This substantially affects the performance of FedAvg as simple averaging leads to conflicting updates that hinder learning, especially in scenarios where clients have a large number of local training steps and a high degree of variance in their data distributions. Under such scenarios, introducing adaptive learning rates [78] and momentum [38, 99, 98, 73, 46] to the server aggregation process are beneficial as they incorporate knowledge from prior iterations.

Gradient masking [88] and weighted averaging [81, 101] of models have also been shown to be useful.

Personalization

Under the traditional FL setting, a ”one-fit-for-all" single server model is trained and updated by all clients. However, this model may underperform for specific clients if their local distribution differs drastically from the global distribution. For example, if there is extreme non-IID data distribution skew amongst clients, the server model trained through FL may only reach a mediocre performance for each local test set. Another failure scenario occurs when there are uneven data distributions amongst clients. In these cases, the federated server model may learn to perform well only on a subset of data. Clients with data distributions that are different from this subset would then have subpar performance. Consequently, it is important to evaluate FL frameworks not only for their global performance but also for their average and worst-case (minimum) local performance.

A variety of papers have aimed to introduce personalized FL, where each client can learn a model more properly finetuned based on their data distribution. In [29], the authors apply the use of a model-agnostic meta-learning framework in the federated setting. In this framework, clients first find an initial shared model and then update it in a decentralized manner via gradient descent with different loss functions specific to their data. Other papers have proposed similar finetuning approaches based on the use of transfer learning [93, 62].

Another category of personalization techniques relies on user clustering and multi-task learning [85, 48]. These techniques cluster together clients with similar data distributions and train personalized models for each cluster. Then, they use multi-task learning techniques to arrive at one model that may perform well for all clients.

Lastly, preserving local batch normalization while averaging models have been used to address the domain and feature shift non-IID problems [19, 60, 18, 39]. Since these batch normalization layers are unique to each client, they help personalize each model to the underlying local data distribution.

However, [53] mentions drawbacks to these various personalization approaches in the FL setting. Namely, most of these approaches incur increased communication costs, such as

a greater number of federation rounds – both transfer learning and user clustering techniques require clients to learn a shared base model first. Furthermore, many personalization approaches result in greater model sizes, such as the addition of batch normalization layers.

In our work, clients can learn personalized lottery ticket networks (LTNs) similar to user clustering techniques without the overhead of communication costs. I show in this chapter that by preserving local batch norm properties while learning these LTNs, clients can improve their accuracy (i.e. achieve better personalization) while compressing model sizes.

Communication Cost

Communication cost is a huge problem for FL as clients frequently communicate with the server. There are three major components of cost during federation – model size, gradient compression (limiting the data needed to be transmitted between each local edge device and the server), and an overall number of rounds of the federation. Model compression techniques [6, 2, 37] like quantization, pruning, and sparsification are usually taken from the classic single centralized setting and applied to FL. In particular, sparse models not only lead to lower memory footprints but also result in faster training times.

For gradient compression, [47, 87, 2] have proposed various update methods to reduce uplink communication costs. These include structured updates, which restrict the parameter space used to learn an update, and sketched updates, which compress model updates through a combination of quantization, rotations, subsampling, and other techniques.

Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis (LTH) [31] states that there exists a subnetwork in a randomly initialized network, such that the subnetwork when trained in isolation can equal the performance of the original network in atmost the same number of iterations. The steps involved in LTH usually include the following: First, the randomly initialized model is trained for a few iterations. Second, the model is pruned based on the magnitude of its weights. Then, the unpruned weights are reinitialized to the weights at the start of the training (rewinding to round 0). This process continues iteratively until the target pruning is achieved.

Though LTH initially showed promising results only for small datasets and network architectures, recent works have expanded LTH to more complex networks and datasets. [32] notably demonstrates better results by rewinding weights to a previous iteration rather than that of the initial iteration (round 0).

4.2 FedLTN: Federated Learning for Sparse and Personalized Lottery Ticket Networks

In this section, I present the motivation and reasoning behind various components of our FedLTN framework that achieve higher degrees of personalization, pruning, and communication efficiency in finding lottery ticket networks (LTNs). These components focus on improving the averaged server LTN to inform better localized performance in terms of accuracy and memory footprint. Along with FedLTN, I propose Jump-Start, a technique for drastically reducing the number of federated communication rounds needed to learn LTNs:

Algorithm 4-2 (FedLTN): To learn complex image classification tasks with greater non-IID feature skew, FedLTN utilizes batch-norm preserved LTN, postpruning, no rewinding, and accelerated global momentum. This combination uses batch-norm preserved averaging to find a server LTN that helps individual clients learn without imposing on their personalized accuracy. Postpruning without rewinding parameters significantly increases the rate of pruning and also helps networks find more personalized LTNs. Finally, our accelerated aggregation of LTNs helps speed up convergence.

Algorithm 4-2 (FedLTN with Jump-Start): Jump-Start can be utilized to skip the communication costs of the first few rounds of training by replacing them with local training, without causing a loss in local test accuracy. This technique is especially useful for scenarios where there are resource constraints on local client devices. One client can be first trained and pruned, then all other clients transfer-learn off this smaller model.

Algorithm 1 FedLTN

function SERVEREXECUTE(θ_0):
  $\theta_g \leftarrow \theta_0$ ▷ random init model unless Algorithm 2 is used
  $k \leftarrow max(N \cdot K, 1)$ ▷ N available clients, participation rate K
  $S_t \leftarrow \{C_1, \ldots, C_k\}$ ▷ k randomly sampled clients
  for each $k = 1, 2, \ldots, N$ **do**
    $\theta_k^t \leftarrow \theta_g$ ▷ each client starts with same init model
  end for
  for each round $t = 1, 2, \ldots, T$ **do**
    for each client $k \in S_t$ **in parallel do**
      $\theta_k^t = \theta_g^t \odot m_k^t$ ▷ m_k is the mask of client k and indicates its LTN
      $\theta_k^{t+1} \leftarrow$ ClientUpdate$(C_k, \theta_k^t, \theta_0)$
    end for
    // BN-preserved aggregation: simple average of non-BN layers of LTNs
    $\theta_g^{t+1} \leftarrow$ aggregate(θ_k^{t+1}, ignore_batchnorm=True)
    // Aggregation of LTNs using server momentum
    $\theta_g^{t+1} \leftarrow \tau\theta_g^{t+1} + (1-\tau)(\theta_g^t - \lambda\Delta^t)$ ▷ τ is an hyperparameter
    $\Delta^{t+1} \leftarrow -(\theta_g^{t+1} - \theta_g^t)$
  end for
end function
function CLIENTUPDATE$(C_k, \theta_k^t, \theta_0)$:
  // BN-preserved LTN
  $C_k^t, \theta_k^t \leftarrow$ copy_ignore_batchnorm$(C_k, \theta_k^t, C_k^{t-1}, \theta_k^{t-1})$
    ▷ Retain previous round's batch normalization layers
  $\mathcal{B} \leftarrow$ split (local data D_k^{train} into batches)
  // Regularization when using server momentum
  $\theta_{init} \leftarrow \theta_k^t$
  for each local epoch i from 1 to E **do**
    for batch b $\in \mathcal{B}$ **do**
      $\theta_k^{t+1} \leftarrow \theta_k^t - \eta\nabla_{\theta_k^t}\ell(\theta_k^t; b) + \|\theta_{init} - \theta_k^t\|$ ▷ η is learning rate, $\ell(\cdot)$ loss function
    end for
  end for
  // Postpruning without rewinding
  $acc \leftarrow$ eval(θ_k^t, local val_set D_k^{val})
  // r_{target} is the target pruning rate, r_k^t is the current pruning rate for client k at round t
  if $acc > acc_{threshold}$ and $r_k^t < r_{target}$ **then**
    $m_k^{t+1} \leftarrow$ prune(θ_k^t, pruning step r_p) ▷ new mask for LTN
    $\theta_k^{t+1} \leftarrow \theta_k^{t+1} \odot m_k^{t+1}$ ▷ do not rewind parameters
    $\mathcal{B} \leftarrow$ split (local data D_k^{train} into batches) ▷ train again after pruning
    for each local epoch i from 1 to E **do**
      for batch b $\in \mathcal{B}$ **do**
        $\theta_k^{t+1} \leftarrow \theta_k^t - \eta\nabla_{\theta_k^t}\ell(\theta_k^t; b)$ ▷ η is learning rate, $\ell(\cdot)$ loss function
      end for
    end for
  end if
  return $\theta_k^{t+1}, m_k^{t+1}$
end function

Figure 4-1: FedLTN

4.2.1 Personalization

Batch normalization-preserved LTN:

In order to personalize LTNs, I introduce a batch normalization-preserved protocol. During the federated aggregation process, batch normalization (BN) layers are not averaged while computing the server model nor uploaded/downloaded by local clients. Since BN layers have been commonly used for domain shift adaptation in single client settings, these layers help personalize each client to its individual data distribution. Preserving BN layers during aggregation leads to higher personalized accuracy by avoiding conflicting updates to clients' individualized BN. It also decreases communication costs as batch normalization layers are not transmitted to the server.

4.2.2 Smaller memory footprint / Faster pruning

Postpruning:

As reported in Section 4.1, one problem with the LotteryFL's naive approach in applying LTH to FL is that the server model sent back to clients each round suffers drastic losses in accuracy before any local training. Moreover, each client decides to prune the model immediately after receiving it from the server, **prior** to any local training. If the server model reaches a certain threshold for the client's local test accuracy, they prune $r_p\%$ of the parameters that have the least L1-norm magnitude. For clients with relatively low levels of pruning, this means that most of the parameters pruned will be the same amongst all the clients that decide to prune that federated round. This approach hopes that only clients with the same archetype (who have the same data distribution) will prune on the same round. However, due to the above-stated challenges in slow prune rates, LotteryFL sets the accuracy threshold to be 50% for clients trained on a binary classification problem. Consequently, models merely need to be slightly better than random chance – which means that clients of different archetypes pruning on the same round (on mostly the same parameters) are a common occurrence. This process hinders the degree of personalization achieved via LotteryFL.

Here, I present an alternative pruning protocol. Instead of pruning *before* any local training, I prune the client model based on the magnitude of the weights *after* n local_epochs of training. This speeds up the pruning process since it is much more likely for the validation accuracy threshold to be reached. Furthermore, postpruning actively encourages diversity in pruned parameters, as each client's parameters will be different after they locally train. Since a freshly pruned model needs to be retrained, I stipulate that if a client prunes it retrains for another n epochs.

Rewinding:

Conventionally, pruned models rewind weights and learning rate to a previous round T. LotteryFL rewinds to $T = 0$ (global initial model). Although in the Lottery Ticket Hypothesis rewinding is needed to avoid convergence to a local minima in the loss function, I hypothesize that this isn't needed during federation, since averaging across multiple models helps mitigate overfitting. That is, instead of resetting all parameters back to the global_init model after pruning, the non-pruned weights stay the same.

Aggregation of LTNs using server momentum:

One of our main contributions is to fasten the convergence of each client model. This is crucial to obtain a performance greater than the threshold so that the client model can prune at a faster rate. This, in turn, reduces the number of communication rounds and hence the overall communication cost, as fewer parameters have to be sent each round. In our problem setting, our server 'model' is an aggregate of all the clients' winning ticket networks. In order to improve the convergence speed, the server sends an anticipatory update of the ticket networks. I outline the steps Algorithm 4-2 follows to achieve faster convergence.

- At each round, the server sends an accelerated update (θ_t) to each of the clients. This means that the clients receive an accelerated winning lottery ticket update at every round. This initialization helps each client to train faster. More formally, I have in the

server, for round t:

$$\theta_g^{t+1} = \tau \sum_k \text{LTNs}(\theta_k^t) + (1-\tau)(\theta_g^t - \lambda\Delta^t)$$

$$\Delta^{t+1} = \theta_g^{t+1} - \theta_g^t$$

where τ is an hyperparameter.

- The server sends the corresponding parameters to each of the clients based on the client mask, I have for client k at iteration 0:

$$\theta_{k0}^{t+1} = m_k.\theta_g^{t+1}$$

- The initial weights of the client is used in the regularization term while training client to align the local gradients with the accelerated global updates. For client k at iteration i, I have

$$L(\theta_{ki}^{t+1}) = l(\theta_{ki}^{t+1}) + \beta\|\theta_{ki}^{t+1} - \theta_{k0}^{t+1}\|$$

FedLTN with Jump-Start

In real-world FL, conducting thousands of communication rounds may be infeasible due to unreliable communication with edge clients. Furthermore, clients with resource-constrained devices may only be able to locally store and train a model after a certain degree of sparsification already takes place. I present FedLTN with Jump-Start in Algorithm 4-2 as an extension of FedLTN to address these challenges

Before federated training, k clients (usually, $k = N$ all clients) locally train for T_{jump} rounds without communicating with the server nor with each other. During local training, they prune to a small degree (e.g. 30%). Then, I choose the model with the highest validation accuracy from the local training Jump-Start and send it to all clients as a model for transfer learning. Then, FedLTN begins with much fewer communication rounds required. Jump-Start is motivated by the work of [66], which found that an LTN trained on one image

Algorithm 2 FedLTN with Jump-Start

function JUMPSTART(T_{jump}):
  $\theta_g \leftarrow \theta_0$ ▷ init random global model
  $k \leftarrow max(N \cdot K, 1)$ ▷ N available clients, participation rate K
  $S_t \leftarrow \{C_1, \ldots, C_k\}$ ▷ k randomly sampled clients
  for each round $t = 1, 2, \ldots, T_{jump}$ **do**
    $\theta_k^t = \theta_k^{t-1} \odot m_k^t$ ▷ m_k is the mask of client k and indicates its LTN
    $\theta_k^{t+1}, m_k^{t+1} \leftarrow \text{ClientUpdate}(C_k, \theta_k^t, \theta_0)$ ▷ local update only
  end for
  $\theta_g \leftarrow \arg\max_{\theta_k^t} \frac{\Sigma_n \text{eval}(\theta_k^t, D_n^{val})}{N}$ ▷ choose client with highest val acc
  ServerExecute(θ_g) ▷ pass θ_g to FedLTN for clients to transfer learn
end function

Figure 4-2: FedLTN-JumpStart

classification dataset can be finetuned to other datasets without much loss in accuracy.

4.3 Experiments

I evaluate the performance of FedLTN against different baselines in several different types of experiments, where I vary the task (image classification), environment setting (large vs. small number of clients), heterogeneity settings (non-IID distributions), and client participation rates.

4.3.1 Experiment Setup

Datasets

I use the CIFAR-10 and Tiny ImageNet datasets for our experiments. To simulate the non-IID scenario, each client consists of 2 classes and 25 datapoints per class for training, 25 datapoints for validation, and 200 datapoints as the test set. For Tiny ImageNet, I randomly sample 10 classes from the 200 classes as our dataset. To simulate a more challenging scenario, I also consider the Dirichlet non-IID data skew. Each client consists of all 10 classes, with the proportions of data volume per class based on the Dirichlet distribution.

Compared Methods:

FedAvg: This baseline indicates performance of an unpruned federated model in our settings. FedAvg computes the federated model by averaging the weights (including BN layers) of all the participating clients' models at every round.

FedBN: FedBN proposes an improvement over FedAvg to obtain better personalization for clients. The server model in FedBN does not aggregate the batch-norm parameters of the participating clients. I use this as a baseline to compare our personalization performance.

LotteryFL: LotteryFL presented the first attempt of using the Lottery Ticket Hypothesis to achieve personalized submodels for each client. I use this as a baseline to analyze the performance of a pruned federated model.

Model architecture

I utilize ResNet18 as the standard architecture for our experiments. Since LotteryFL does not account for batch normalization (BN) layers, I also conduct experiments on a custom CNN model without BN layers on CIFAR-10 for a baseline comparison. Results for the high-client setting is shown below, whereas custom CNN results for the low-client setting and other implementation details can be found in our supplementary material.

Hyperparameters

I set the hyperparameters local epochs (E) = 3, batch size (B) =8, τ = 0.5, accuracy threshold ($acc_{threshold}$) = 0.6, prune step(r_p) = 0.1 for FedLTN. For LotteryFL, I use the same hyperparameters the authors mentioned in [56]. I fix the number of communication rounds to 50 for the low-client (10) setting and 2000 rounds for the high-client (100) setting. I denote the models with the target pruning rate within parenthesis. For example, FedLTN(0.9) refers to FedLTN paradigm with 90% as the target pruning percent. For experiments using FedLTN with Jump-Start, I use 25 Jump-Start and 25 FedLTN rounds with $K = 50\%$ participation and 10% r_p prune step. I set a max prune of 30% for Jump-Start and 90% for FedLTN.

Evaluation Metrics

To evaluate the personalization achieved by our models, I compute the average classification accuracy achieved by each client model on its corresponding test set. Although average test accuracy gives an indication of overall test performance across all clients, it is also important to measure the minimum client test accuracy. This is to ensure that all clients participating in the federated training paradigm learn a personalized LTN that performs well on their local data distribution. Hence, I also use the minimum client test accuracy as an evaluative measure. To compute communication costs, I sum the data volume communicated at every round during the training process.

4.3.2 Evaluation

I demonstrate the success of FedLTN in learning personalized, sparse models in this section. I analyze the test accuracy performance, maximum pruning achieved, communication costs, and the convergence speed of FedLTN with baselines. I report the results of all the baselines and our method in Table 4.1.

Pruning:

In our experiments, I evaluate FedLTN's performance while using different target pruning ranging from 10-90%. As shown in Table 4.1, I can see that our method improves average test accuracy despite pruning 40% more parameters compared to baseline LotteryFL. Our pruning achieves substantial reductions in memory footprint as seen in Table 4.2. This degree of sparsity is important as it makes it feasible to train on even resource-constrained edge devices. I also compare the rate of pruning between our method and the baselines. As seen in figure 4-3, I observe that our method prunes around 70% in the first 20 rounds, while LotteryFL(0.9) prunes around 10%. Our postpruning method can achieve larger pruning rates quickly in a few rounds, even though I set a higher accuracy threshold than LotteryFL.

Dataset	Algorithm	Avg Test Acc (%)	Min Test Acc (%)	Comm. Cost (MB)
CIFAR-10	FedAvg	72.8	<u>64.9</u>	11,150.0
	FedBN	78.72	**65.75**	11,137.5
	LotteryFL(0.1)	72.0	56.3	10,613.95
	LotteryFL(0.5)	75.8	52.5	8,675.7
	FedLTN(0.1)	74.5	59.8	10,134.6
	FedLTN(0.5)	81.1	61.5	6,563.3
	FedLTN(0.9)	**82.6**	63.0	<u>3,753.9</u>
	FedLTN(0.9; jumpstart)	<u>82.2</u>	64.8	**1,846.0**
	FedLTN(0.9; rewind)	71.5	53.0	4,940.7
TinyImageNet	FedAvg	68.9	51.8	11,150.0
	FedBN	73.0	55.5	11,137.5
	LotteryFL(0.1)	72.6	41.3	10,370.7
	LotteryFL(0.5)	71.3	50.8	6,885.5
	FedLTN(0.1)	68.4	38.3	10,169.0
	FedLTN(0.5)	73.3	50.0	6,885.5
	FedLTN(0.9)	<u>74.5</u>	<u>59.8</u>	<u>4,650.9</u>
	FedLTN(0.9; jumpstart)	**83.1**	**61.8**	**1,778.4**
	FedLTN(0.9; rewind)	71.8	53.8	5,144.0

Table 4.1: Comparison of performance of FedLTN with all the baselines on the CIFAR-10 and Tiny ImageNet datasets in the low-client setting with ResNet18. FedLTN(0.9; jumpstart) refers to 90% target pruning with 25 rounds of Jump-Start and 25 rounds of FedLTN. Rewinding resets model parameters to randomly initialized model in round 0. **Bolded** numbers represent best performance and <u>underlined</u> numbers represent the second best.

Communication Costs:

Since our method prunes more than LotteryFL, the communication costs are 2.3x and 3x times lower than that of LotteryFL and FedAvg. As our method prunes faster than LotteryFL, I send lower parameters during each communication round, reducing the overall communication cost. I observe that the communication costs reduce even more when using jumpstart. For example, with jumpstart, I obtain 4.7x and 6x lower communication costs than LotteryFL(0.5) and FedAvg.

Personalization:

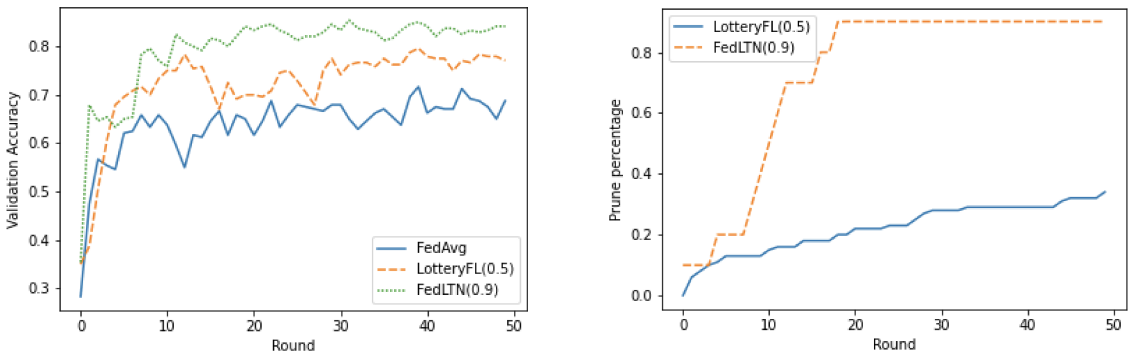
I analyze the level of personalization achieved by each client participating in FL. I consider the average test accuracy across all clients, i.e the average of performance for each client model on the test datasets. I find that our method achieves better accuracy than an unpruned FedAvg baseline and 50% pruned LotteryFL baseline models. For example, in CIFAR-10, FedLTN(0.9) achieves average test accuracy of 82.6% when compared to LotteryFL(0.5), which achieves 75.8% and FedBN which achieves 78.72%. For both CIFAR-10 and TinyImageNet, FedLTN(0.9) with and without Jump-Start performs better than all the baselines, even when pruning 40% more parameters.

I find that our method achieves the highest minimum test accuracy compared with all the other baselines. In TinyImageNet, FedLTN(0.9) with 25 Jump-Start rounds achieves 59.8% minimum test accuracy compared to 50.8% in the same setting using LotteryFL(0.5).

I observe that while increasing the target pruning rate of the experiment, the overall test accuracy increases. Our highest pruned models give the best overall test accuracy. This is due to clients learning more personalized models as the pruning rate increases, which leads to better test performance.

Convergence:

One of our objectives is to achieve faster convergence to facilitate faster pruning. Figure 4-3 shows the performance obtained by the clients on their validation set in each training round. Our method converges faster than FedAvg and LotteryFL. This is due to the server



(a) (b)

Figure 4-3: Left (a): Comparision of validation accuracies at each round. I observe that our method converges faster than other baselines. Right (b): Comparison of pruning rate at each round. our method prunes around 70% in 20 rounds while baseline LotteryFL prunes around 10%.

Model	Size of parameters at 90% prune (MB)
Custom CNN	0.031
ResNet18	4.46

Table 4.2: Memory footprint of 90% pruned model parameters.

broadcasting accelerated aggregated LTNs during each round. This anticipatory update and the client initialization, along with the modified regularization term align the client gradients to avoid any local minima. This boosts the convergence speed of our method compared to the baselines.

Impact of number of clients on performance:

I run experiments to compare the performance of FedLTN in a high-client setting (100 clients) with low client participation of 5%. Table 4.3 shows the performance of FedLTN compared with other baselines. I observe that FedLTN achieves the best average and minimum local test accuracy. FedAvg performs the worst among all baselines. I also note that FedLTN(0.9) achieves better performance than LotteryFL(0.5) even if FedLTN is pruning 40% more parameters than the latter.

Impact of number of classes on performance:

I run experiments with each client consisting of all 10 classes in CIFAR-10. The data volume of these classes is given by the Dirichlet distribution. The Dirichlet distribution is controlled

Setup	Algorithm	Avg Test Acc (%)	Min Test Acc (%)	Comm. Cost (MB)
CIFAR-10 Custom CNN No BN Layers 100 clients	FedAvg	50.6	42.0	2398.3
	LotteryFL(0.1)	75.1	47.5	2173.1
	LotteryFL(0.5)	74.9	<u>51.3</u>	1304.4
	FedLTN (0.1)	**77.9**	50.5	2166.1
	FedLTN(0.5)	<u>76.3</u>	**54.3**	1254.7
	FedLTN(0.9)	75.4	50.5	**472.5**

Table 4.3: Performance of FedLTN and other baselines on CIFAR-10 in the high-client setting with 100 clients over 2000 rounds.

Algorithm	$\alpha = 0.5$		$\alpha = 0.7$	
	Avg Test	Min	Avg Test	Min
LotteryFL(0.5)	61.19	24.00	54.00	36.00
FedLTN(0.5)	67.20	40.00	56.00	40.00
FedLTN(0.9)	64.80	40.00	55.60	36.00

Table 4.4: Performance on the CIFAR-10 dataset with dirichlet $\alpha = \{0.5, 0.7\}$

by the parameter α. For low α (close to 0) the data volume is heavily skewed towards one particular class, while as α increases (close to 1), the data volume is distributed in an iid manner across all classes. Since our goal is to learn more personalized models, higher values of α pose a challenge. Table 4.4 shows the performance of FedLTN(0.9) and FedLTN(0.5) compared with LotteryFL(0.5) for α values of 0.5 and 0.7. As I can see, FedLTN(0.9) learns better-personalized models for high values of α. For example, FedLTN(0.5) achieves 6% more test accuracy than LotteryFL(0.5). I also observe that FedLTN(0.9) despite pruning more parameters, is able to achieve better overall performance than LotteryFL(0.5). This means that our method is capable of learning more personalized models for high values of α when I have more number of classes.

Impact of rewinding on performance:

Table 4.1 shows that rewinding to round 0 after pruning leads to significant drops in FedLTN's accuracy to similar levels as LotteryFL. Moreover, rewinding leads to slower pruning and thus convergence, as seen in our supplementary material.

Impact of Architecture on Performance:

When training on the custom CNN, our BN-preserved LTN aggregation cannot be applied and leads to lower performance on sparser models in Table 4.3. Comparing the results from Table 4.1 and the equivalent CIFAR-10 low-client experiment in the supplementary material, BN-preserved LTN provides a major boost in our test accuracy (especially for sparser models). Despite this, I see that FedLTN still performs better than LotteryFL in Table 4.3. On the other hand, I find that LotteryFL does not benefit from the additional BN layers and greater model depth of ResNet18, which points to the efficacy of our BN-preserved LTN aggregation.

Impact of Jump-Start on Performance and Communication Cost:

I observe that using Jump-Start drastically reduces the communication cost without any compromise on the performance. For example, I see from Table 4.1 and the supplementary material that FedLTN(0.9) achieves 74.5% on Tiny ImageNet, while FedLTN(0.9; jumpstart) achieves 83.1% with up to 60% lower communication costs. This is due to the efficacy of transfer learning from the best-performing model after local training. Consequently, Jump-Start allows clients to skip the communication cost of the initial FL rounds.

4.4 Summary

The Lottery Ticket Hypothesis has shown promising results in reducing the model size without loss of accuracy for models trained on a single client. In this chapter, I addressed the most pressing challenges of applying LTH to the FL setting – slow model pruning and convergence. I proposed a new framework, FedLTN, for learning Lottery Ticket Networks via postpruning without rewinding, preserving batch normalization layers, and aggregation using server momentum. FedLTN and FedLTN with Jump-Start achieve higher local test accuracies, significantly accelerate model pruning, and reduce communication cost by 4.7x compared to existing FL approaches.

Chapter 5

PrivacyFL: A Simulator for Privacy-Preserving and Secure Federated Learning

Setting up a federated learning environment, especially with security and privacy guarantees, is a time-consuming process with numerous configurations and parameters that can be manipulated. In order to help clients ensure that collaboration is feasible and to check that it improves their model accuracy, a real-world simulator for privacy-preserving and secure federated learning is required.

In this chapter, I introduce PrivacyFL, which is an extensible, easily configurable, and scalable simulator for federated learning environments. Its key features include latency simulation, robustness to client departure/failure, support for both centralized (with one or more servers) and decentralized (serverless) learning, and configurable privacy and security mechanisms based on differential privacy and secure multiparty computation (MPC).

I describe the architecture of the simulator and associated protocols, and discuss its evaluation in numerous scenarios that highlight its wide range of functionality and its advantages. This chapter addresses a significant real-world problem: checking the feasibility of participating in a federated learning environment under a variety of circumstances.

5.1 Introduction

While designing a federated learning environment, it is important to understand the trade-offs between privacy, efficiency, and accuracy. System designers should be able to choose different parameters and compare multiple scenarios to find the right trade-offs for their environment. Simulators are essential for federated learning environments for the following reasons:

- To evaluate accuracy: It should be possible to simulate federated models and compare their accuracy with local models.
- To evaluate total time taken: Communication between distant clients can become expensive. Simulations are useful for evaluating if client-client and client-server communications are beneficial.
- To evaluate approximate bounds on convergence and time take for convergence.
- To simulate real-time dropouts: Clients in a federated environment may drop out at any time.

the simulator, PrivacyFL, is designed to specifically address these issues. It is efficient in terms of speed and execution time as client and server operations happen in a parallelized manner. It provides numerous differentially private mechanisms including the Laplacian mechanism [26], the Gaussian mechanism [26], and the Staircase mechanism [34]. System designers can choose from any of these mechanisms to ensure that the learning is privacy-preserving. the system also provides the option for clients to use a Diffie-Hellman key exchange for inbuilt security. The simulator can also easily support the addition of new privacy-preserving algorithms. It is composed of generalized classes and functions that make it easy for developers to add/modify new algorithms and protocols. Along with simulating federated environments that use one or more central servers, the simulator is also able to simulate a completely decentralized environment, i.e., a federated learning system without a centralized server. In the real world, environments are dynamic with clients failing, dropping off, or being added at run-time. Being able to model this is an important requirement of

a simulator for federated learning and PrivacyFL can handle such dynamic cases. It can simulate computation and communication times to evaluate how long a client would have to wait for each step of the protocol to complete in a real world setting. the system provides necessary guarantees for the "honest-but-curious" threat model. In this threat model, all the clients in the system follow the protocol but try their best to identify the data/model of other clients.

5.1.1 Related Work

There have been a few open source and industry-based solutions for FL over the last couple of years. Open source FL libraries such as LEAF [14] and TensorFlow-Federated [40] support only horizontal topology-based FL algorithms. These libraries are unsuitable for customized training procedures and protocols. In addition, TFF supports only one specific implementation of differentially-private SGD. They do not provide customizable privacy features. Though, PySyft [82] provides functions to implement FL clients, handle client communications, secret sharing, homomorphic encryption, etc., it doesn't provide any documentation on its system architecture or interfaces. In addition, their code is not generalized and their libraries are highly unstable. Industry-based real-world FL solutions such as FATE [100] and PaddleFL [61] lack flexible frameworks (i.e.) developers have to modify the source code in order to run new algorithms. They don't support novel FL algorithms which may require support from external libraries.

5.2 Architecture

The purpose of PrivacyFL is to provide a light-weight but efficient Python framework that enables clients to simulate privacy-preserving secure federated learning. In a simulation, a client corresponds to an instance of the **ClientAgent** class. Each client agent is capable of interacting with the other client agents to establish common keys before the simulation. Client agents also communicate with an instance of the **ServerAgent** class if configured to be a centralized system. The server agent is an agent that is not training its models but instead serves to run the federated learning algorithm. It is responsible for requesting

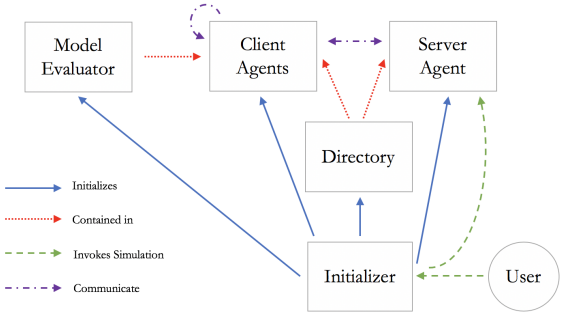


Figure 5-1: System diagram containing relationships between the important classes in PrivacyFL

weights from the clients, averaging the weights, and returning to the clients the federated weights. Many applications will only require one server agent, but it is worth noting that with some modifications the framework can handle any number of server agents.

To run the simulation, an instance of the **Initializer** class must be created. The initializer will create the agents in the simulation and also invoke the client agents to perform any offline-stage logic such as key exchanges. The initializer also gives each agent a directory, which contains a mapping of agent names to agent instances so that agents can invoke each other's methods, thereby enabling all agents to communicate. The initializer also gives each client agent an instance of the **ModelEvaluator** class, which it uses to evaluate the federated learning on a test dataset.

5.2.1 Simulation Lifecycle

This section provides an overview of the steps carried out in a simulation.

1. System designer wishing to simulate a federated learning environment specifies simulation parameters in `config.py` and runs `run_simulation.py`.

2. `run_simulation.py` creates an instance of Initializer, which instantiates the client agents and server agent. It also creates a mapping of agent names to instances and passes those to each client.

3. `run_simulation.py` invokes the initializer's `run_simulation()`, which subsequently calls the server agent's `request_values()` method.

4. Repeat for $i = 1, 2, 3, ...$ num_iterations :

    4.1 Server agent requests weights from clients in parallel by invoking each client's `produce_weights()` method.

    4.2 In `produce_weights()`, each client trains its machine learning model on D_i, the dataset for that iteration. Each client saves the weights locally and then creates a copy of its weights to which it adds a security offset and differentially private noise. It returns the modified weights to the server.

    4.3 Upon receiving the weights from each client, the server averages the weights and returns the federated weights to each client in parallel through their `receive_weights()`.

    4.4 Upon receiving the federated weights for iteration i from the server, each client computes the accuracy of the federated model vs its local model on the test set. Each client then computes whether its weights have converged and tells the server whether it is dropping out after this iteration.

    4.5 To conclude the iteration, the server tells all the clients which other clients have dropped out of the simulation.

5.2.2 Classes

There are 6 main classes in the simulator namely **Message**, **Agent**, **ClientAgent**, **ServerAgent**, **Initializer**, and **Directory**.

The Message Class

All agent-agent communications occur by one agent invoking another agent's method with a message as the sole argument. The message class contains metadata about the communication as well as a `body` attribute, which is a dictionary containing all the values an agent is sending.

The Agent Class

The Agent class is not meant to be initialized directly. Rather, its intended usage is to be sub-classed to create agents with more specific behavior. It is the base class for the ClientAgent and ServerAgent classes I provide but can be sub-classed to create a different kind of agent if ClientAgent and ServerAgent are not easily modified to fit the needs of ythe simulation.

The ClientAgent Class

An instance of the ClientAgent class is an entity that is training a machine learning model on the same task as the other client agents. Client agents are assigned an `agent_number`, which is then appended to the string `client_agent` to create their name. For example, if there are three clients they are named client_agent0, client_agent1, and client_agent2. There are two important public methods of ClientAgent that are invoked by the ServerAgent in the online portion of the simulation.

- `produce_weights(self,message)` is called every iteration and prompts the client to train its machine learning model on its dataset for that iteration. The message contains the iteration, a mutex lock used for multi-threading, and information about the simulated time.

- `receive_weights(self,message)` is called every iteration when the server has federated weights to return to the client. The message contains the iteration, weights, and the simulated time this message is received. This method returns `True` if the client agent's weights have converged with the federated weights and `False` otherwise.

- `remove_active_clients(self,message)` is called at the end of the iteration. The message contains the iteration, list of clients that have dropped out, and the simulated time the message is received. This method is only used if the simulation is configured with client drop out.

In addition, client agents have some methods for the client-client communication that are used to establish shared common keys with the other clients.

- The `send_pubkeys(self)` of each client is invoked by the initializer once all client agents have been initialized. In this method, each client computes secrets $a_{i,1}, \ldots, a_{i,n}$ and sends $a_{i,j}$ to client j by invoking that client's `receive_pubkey()` method.

- In `receive_pubkey(message)`, each client saves the public key sent to it by the other clients.

- `initialize_common_keys(self)` is called once all clients have exchanged public keys using the above two methods. In this method, the common key list is initialized. This common key list will be used to produce security offsets for the online portion of the simulation.

The ServerAgent Class

An instance of the ServerAgent class represents a third-party entity that coordinates the online portion of the simulation. ServerAgent has one method:

- `request_values(num_iterations)` is called by an instance of Initializer to signal the server agent to start requesting values from clients. `request_values()` first requests weights in parallel from the clients by calling their `produce_weights()`. It then averages these weights and returns them to the clients in parallel by calling their `receive_weights()` method. Finally, if the simulation is configured to allow client dropout, the server invokes the clients' `remove_active_clients()` to indicate which clients have dropped out at the end of this iteration.

The Initializer Class

An instance of the Initializer class is used to initialize the agents and model evaluator. In addition, any offline stage logic, such as prompting the clients to perform a Diffie-Hellman key exchange, should occur in this class. In the example, it loads the MNIST

dataset [55], partitions it, and distributes the correct amount of data to each client. To commence the simulation, one creates an instance of the Initializer class and invokes its `run_simulation()` method, which then invokes the server agent's `request_values()` to commence the online portion of the simulation.

The Directory Class

An instance of the Directory class contains a mapping of agent names to agent instances that allows agents to invoke other agents' methods by only having their name. An instance of Directory is created in the `__init__` method of the Initializer class after all the agents have been created. It is then passed on to all the agents using their `set_directory()` method.

5.2.3 Configurations and Features

PrivacyFL can be configured to simulate a variety of scenarios with the configuration parameters available in `config.py`. These configuration parameters are described in Table 5.1.

In addition, config.py allows system designers to set several other parameters such as the number of clients, custom ϵ and dataset sizes for each client, and thread-safe random seed for reproducibility.

5.3 Algorithms

To evaluate PrivacyFL, I developed a secure and differentially private federated logistic regression algorithm.

5.3.1 Differentially Private Federated Averaging

The federated averaging algorithm is one of the most popular approaches for federated learning. It is a customized version of parallel Stochastic Gradient Descent. In the approach, Algorithm 2, each client runs U rounds of local SGD and returns the trained weights.

USE_SECURITY	If True, the clients perform a Diffie-Hellman key exchange in the offline portion of the simulation to establish common keys for encryption. Has no effect on federated accuracy.
USE_DP_PRIVACY	If True, clients will add differentially private noise with parameters specified in the config file.
SUBTRACT_DP_NOISE	If True, clients will subtract the noise they added to the federated model upon receiving it from the server. When False, clients use the federated model computed by the server.
CLIENT_DROPOUT	If True, a client drops out of simulation when each weight in the federated model is within config.tolerance of the client's weight. The simulation continues without that client.
SIMULATE_LATENCIES	If True, the system simulates how long it would take for each step in the protocol to complete using the user-defined communication latencies in config.LATENCY_DICT. If False, this information is not displayed.
USING_CUMULATIVE	If using the data partitioning module, this flag is useful for experimenting between dataset options. If False, the dataset for each iteration includes only the new data available that iteration. This makes sense for Algorithm 2. If True, the size of the dataset grows each iteration by the amount of new data. This configuration makes more sense for Algorithm 3, where the weights of the ith iteration are not used in producing the weights of the $i+1$th iteration.

Table 5.1: Configuration Parameters

Clients make use of new data every round. Here, each client adds the difference of gamma random variables to their unperturbed weights. In the aggregation phase, the server adds the perturbed weights it receives from all values. The server aggregates the weights into the final model, which are differentially private because

$$L(\mu, \lambda) = \frac{\mu}{n} + \sum_{k=1}^{n} \gamma_k - \gamma'_k$$

γ_k and γ'_k are Gamma distributed random variables.

The client-server communication happens R times unless a client drops out.

I propose Algorithm 3, where clients reuse old data used in previous rounds and also make use of any new data available.

The main difference between Algorithm 2 and Algorithm 3 is that in Algorithm 3 clients retrain their model from scratch on their entire local dataset, which may or may not get updated every round, and don't use the weights received from the server while running local gradient descent. In Algorithm 2, clients compare their weights from the previous iteration with that of the global weights they receive from the server. With all algorithms, if local convergence is achieved then the client can drop out of the system.

In all the algorithms, when clients receive the federated weights from the server they are able to subtract the differentially private noise that they contributed to the federated weights.

Algorithm 2 Differentially Private Federated Averaging - Each client contributes a portion of the total differentially private noise

---

Server Implements:

Initialize w_0
for $r \leftarrow 1$ to R **do**
  for every client $k \in n$ **in parallel do**
    $w_{r+1}^k \leftarrow ClientProcedure(k, w_r)$
  end for
  $w_{r+1} \leftarrow \frac{1}{n} \sum_{i=1}^{n} w_{r+1}^i$
end for

ClientProcedure(k,w):

for local gradient update $u \leftarrow 1$ to U **do**
  $w \leftarrow w - \eta \nabla g(w)$
end for
Return $w + \gamma - \gamma'$

Algorithm 3 Differentially Private Weighted Averaging

Server Implements:

for $r \leftarrow 1$ to R **do**
  for every client $k \in n$ **in parallel do**
    $w_{r+1}^k \leftarrow ClientProcedure(k, w_r)$
  end for
  $w_{r+1} \leftarrow \frac{1}{n} \sum_{i=1}^{n} w_{r+1}^i$
end for

ClientProcedure(k, w_s)**:**

if Round 1 **then**
  Initialize $w_c^{dp} = 0$
end if
if $w_s < w_c^{dp}$ **then**
  Initialize w
  for local gradient update $u \leftarrow 1$ to U **do**
    $w \leftarrow w - \eta \nabla g(w)$
  end for
  $w_c^{dp} \leftarrow w + \gamma - \gamma'$
end if
Return w_c^{dp}

5.3.2 Secure Aggregation

To provide the necessary security guarantees, I use a combination Diffie-Hellman key exchange and Pseudo-Random Generators to ensure a secure aggregation protocol. the protocols ensure that clients only need to communicate with each other once at the start of the simulation to ensure security for all iterations, thereby reducing the amount of client-client communications and overall communication time.

In the offline phase, every pair of clients P_i and P_j share n bit secret random values $r_{i,j} = r_{j,i}$ which is known only to them. The clients generate the common random values by running the Diffie-Hellman Key exchange protocol [23].

Every client splits their secret random values into a k bit offset $r'_{i,j}$ and a $n - k$ bit seed $s_{i,j}$. client P_i sends its weight w_i masked with the random offsets. For every $j > i$, clients add $r'_{i,j}$ and for every $j < i$ clients subtract $r'_{i,j}$. In order to generate new n bit secret random values for the next round, each client makes use of the $n - k$ bit seed of the current round. This process of generating random offsets and seeds continue as long as clients are sending weights to the server, i.e., until clients drop out. Hence, clients only need to communicate

with each other once at the start of the simulation to ensure security for all iterations, thereby reducing the amount of client-client communications and overall communication time. If desired, the clients can perform the Diffie-Helman key exchange again at any time in the simulation.

5.4 Experiments

For the evaluation, I configured the clients to train on UCI's Machine Learning MNIST dataset [55] [71] for eight iterations. This dataset is easily loaded using scikit-learn's `datasets.load_digits()` module. I ran the experiments on a MacBook Pro 2.3 GHz Intel Core i5, using Python's multithreading module for parallelization. All experiments were initialized with the same thread-safe random seeds, ensuring fair comparisons where possible.

5.4.1 Experiment 1: Accuracy vs Privacy vs Number of Clients Trade-offs

One of the principal uses of the simulator is to enable clients to assess whether the accuracy of their model would increase if they participated in the federated learning. For that reason, I compare the mean accuracy of the clients' model with the accuracy of the federated global model for each iteration. I used Algorithm 3. When computing a client's accuracy on iteration i, I consider its weights with no differential privacy since this scenario corresponds to clients not participating in the federated learning, thus having no need to add differentially private noise. Figure 5-2 compares the mean of three clients' accuracy with the federated accuracy for different values of ϵ, where a smaller value of ϵ corresponds to less noise. As expected, the federated model's accuracy increases as ϵ increases, but there is not a significant difference between $\epsilon = 1$ and $\epsilon = 8$. It is also important to note that the federated accuracy for the $\epsilon = 0.1$ line increases more rapidly than the mean client accuracy. This is expected behavior since the amount of noise added per client is smaller as the size of each client's dataset increases by thirty samples each iteration.

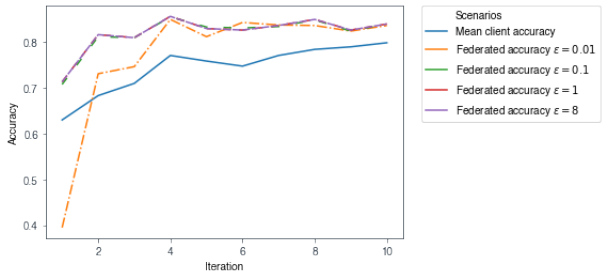


Figure 5-2: Mean client accuracy on test dataset versus federated model accuracy for different ϵ. Clients follow Algorithm 3.

In the following example, I simulate a larger number of clients. I used the KDDCup99 dataset [7] and Algorithm 2. In Figure 5-3, one can also see the result of varying the number of clients and ϵ. As expected, the accuracy of the federated model increases with ϵ since less noise is added. Also, accuracy increases as the number of clients increases. This behavior is expected since more clients contributing to the federated model should increase the accuracy of the federated model. I can also see that convergence happens at a faster rate when the number of parties increases.

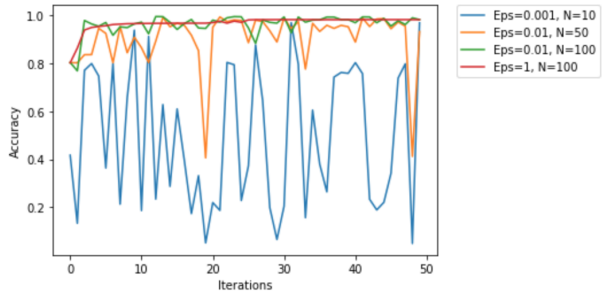


Figure 5-3: Accuracy vs Iterations for different values of ϵ and number of parties

5.4.2 Experiment 2: Privacy Constraints

This experiment illustrates a use case where each client can have different amounts of data and privacy requirements. Suppose Clients 1 and 2 each have 150 data points and $\epsilon = 1.0$. However, Client 3 has 250 data points but also a more stringent privacy requirement, $\epsilon = 0.1$. Clients 1 and 2 may want to decide whether they should only collaborate among themselves, include Client 3, or participate in federated learning at all.

Scenarios	Client 1	Client 2	Client 3
Scenario 1	0.750	0.793	0.823
Scenario 2	0.806	0.810	-
Scenario 3	0.767	0.800	0.813
Scenario 4	0.827	0.830	0.850

Table 5.2: Each client's accuracy using Algorithm 3 in the the different scenarios

- Scenario 1. No clients participate in federated learning so their accuracy is their own model's accuracy.
- Scenario 2. Clients 1 and 2 participate in the federated learning but do not include Client 3 because Client 3 has $\epsilon = 0.1$ while Clients 1 and 2 have $\epsilon = 1.0$.
- Scenario 3. All three clients participate in the federated learning and use Client 3's privacy requirement, $\epsilon = 0.1$.
- Scenario 4. All three clients participate in federated learning, but Clients 1 and 2 use their privacy requirement $\epsilon = 1.0$, and Client 3 uses its privacy requirement, $\epsilon = 0.1$

Table 5.2 shows the clients' accuracy in each scenario. These values are obtained by using the configuration options that correspond to Algorithm 3. As is apparent from the table, Clients 1 and 2 do not benefit from including Client 3 in the simulation unless each client can use its own value of ϵ. In that case (Scenario 4), each client benefits from participating in federated learning.

5.4.3 Experiment 3: Decentralized (Serverless) Federated Learning

The simulation can also be easily modified beyond the configuration parameters. In the following example, I sub-classed the Agent class to create a new kind of Agent that is able to perform federated machine learning without a server. In this case, clients send their masked weights directly to the other clients. Once a client has received the weights from all the other clients, it is able to average them to create a federated model. Figure 5-4 shows the results of three clients each training according to Algorithm 3, with thirty data points per iteration for seven iterations. In this example, clients use the security protocol but no differential privacy. This behavior is achieved by setting the USE_DP_PRIVACY flag to False and the

USE_SECURITY flag to True. As one can see from Figure 5-4, all clients clearly benefit from participating in the federated learning in this situation.

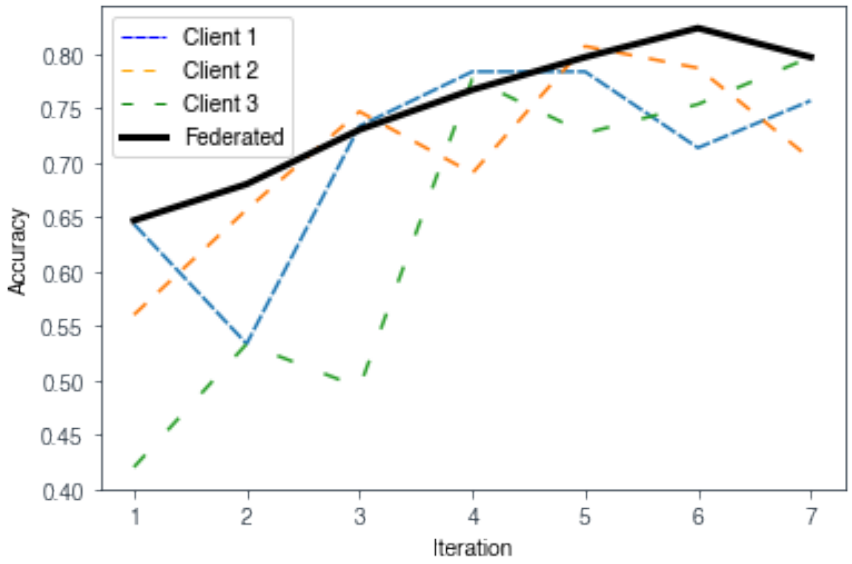


Figure 5-4: Simulation with Algorithm 3 but modified so that no server is required.

5.4.4 Experiment 4: Real-World Latency Simulation

For this experiment, I simulated the situation where the clients are in different geographic locations. I configured three clients as if they were in Boston, Singapore, NYC, with the server located closest to the NYC client. As such, I designated the following communication latencies: (i) Boston-Server: 0.3 seconds, (ii) Singapore-Server: 2.0 seconds, (iii) NYC-Server: 0.1 seconds. One can also set client-client latencies. In this example, those only come into effect for the offline portion of the simulation since there is no client-client communication during the online portion of the simulation.

Table 5.3 shows the simulated time, which measures the time from the start of each iteration to the time when clients would receive the federated weights from the server. The simulated time is available for any step in the protocol since it is included in every message between clients. Upon receiving a message, an agent computes how long its logic has taken. The agent then adds the time that its logic with the simulated communication time for

Iteration	Simulated Time to Receive Federated Weights (seconds)		
	Boston	Singapore	NYC
Iteration 1	4.310	6.010	4.110
Iteration 2	4.306	6.006	4.106
Iteration 3	0.909	Dropped Out	0.709

Table 5.3: Simulated time to receive federated weights by iteration in an example where Singapore, the farthest client, drops out after the second iteration.

whomever the client is sending the message to. This produces a new simulated time for the receiving agent. In Table 5.3 one can also see that the simulated times for Boston and NYC are significantly lower on the third iteration. This is because clients were permitted to drop out since the CLIENT_DROPOUT flag was set to True. The Singapore-Server latency is the highest, which means that once Singapore drops out at the end of the second simulation the other clients receive the federated weights faster since the server does not need to wait on the Singaporean client's weights. Similarly, the time taken by each client to compute its weights on each iteration is also displayed by the simulator.

5.5 Availability

PrivacyFL is available on GitHub at https://github.com/vaikkunth/PrivacyFL/ under the MIT License. To run the simulator, the repository should be cloned locally and a conda environment with the necessary dependencies should be created as shown in Figure 5-5. The config.py file contains a variety of parameters as described in Section 5.2.3 that allow the simulation to be customized. `utils/data_formatting.py` provides an example of how to pre-process the dataset for the simulation using the MNIST dataset as an example.

```
conda env create -f environment.yml -n YourEnvironmentName
cd src
python run_simulation.py
```

Figure 5-5: Steps to execute the code

5.6 Summary

In this chapter, I presented PrivacyFL, a simulator for privacy-preserving, and secure federated learning. The primary features of the system include latency simulation, robustness to client departure/failure, support for both server-based and serverless federated learning, and tunable parameters for differential privacy and MPC.

I evaluated PrivacyFL by running multiple experiments on federated logistic regression to demonstrate the flexibility of the framework. PrivacyFL and its protocols can be easily applied to other federated machine learning algorithms as well. PrivacyFL is highly customizable and system designers can easily configure parameters such as setting different latencies, choosing appropriate differential privacy mechanisms, and simulating clients dropping out or being added dynamically.

Chapter 6

DynamoFL: A Production Level FL System

DynamoFL [1] is an easy-to-use, plug-and-play infrastructure that enables organizations to rapidly stand up federated learning across clients/devices in a matter of minutes and can be plugged into existing data and modeling pipelines of ML teams smoothly. It allows machine learning teams to build highly personalized (or) generalized ML solutions that are federatively trained to learn from both individual user data and the global population characteristics in a privacy-preserving manner. In addition, DynamoFL addresses all privacy and training/performance related challenges associated with FL.

6.1 Introduction

DynamoFL supports both traditional federated learning (building a generalized machine learning model trained across siloed datasets) and a powerful variation of federated learning for personalized model development.

At its core, DynamoFL supports:

1. The algorithmic toolkit to build highly personalized models (or alternatively, highly generalized models) that are federatively trained to learn from both individual user

[1] www.dynamofl.com - Work done at DynamoFL, Inc. and DynamoFL, Inc. holds all rights to the work in this chapter

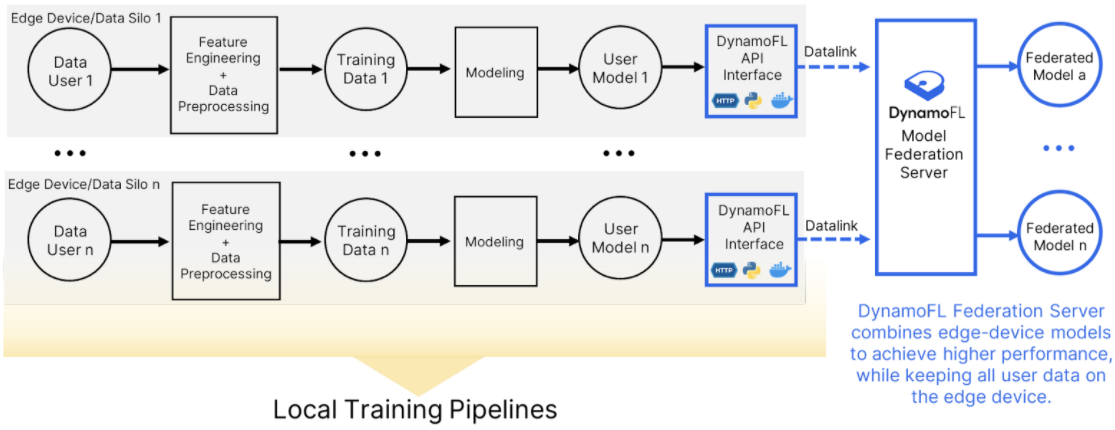


Figure 6-1: DynamoFL's Decentralized Federated Learning Workflow

data and the global population characteristics.

2. The plug-and-play infrastructure that enables an organization to rapidly stand up federated learning across their clients/devices in a matter of minutes.

6.1.1 DynamoFL's Decentralized Federated Learning Workflow

DynamoFL supports traditional decentralized federated learning workloads where federated models are trained across edge devices or data silos without centrally collecting data. Federated learning participants locally train their models on their datasets and submit models to DynamoFL's Model Federation Server through our API interface / using an inbuilt docker based solution (see next section for information on DynamoFL API integrations).

6.1.2 Challenges: Federated Learning Infrastructure

Building out a seamless federated learning infrastructure spanning multiple clients has been extremely challenging in practice. DynamoFL was built to systematically address these challenges so that any organization can deploy federated learning with ease.

Challenge 1: As federated learning projects scale, organizations need to coordinate and synchronize many federating clients (often 100+ clients or 1M+ devices) that are each running their own persistent training processes (federating runs can often last weeks). This is

even more challenging in cases where clients belong to different organizations with diverse underlying IT infrastructures, varying administrative controls, and complex data pipelines.

Solution: DynamoFL provides a unified framework for efficiently coordinating actions between federated learning participants. DynamoFL can rapidly scale to support 100+ to 1M+ federated clients whose actions need to be synchronized in lock-step to ensure a successful federation. We've worked with our partners to build a solution that plugs readily into a wide array of complex data pipelines.

Challenge 2: Federated learning requires building a cross-organizational or inter-departmental data infrastructure that needs to be closely maintained to ensure federating clients can seamlessly and cohesively collaborate.

Solution: DynamoFL was designed to be the simplest possible method for standing up federated learning in any organization. With DynamoFL an IT professional with no background in data science or machine learning can set up their organization's federated learning infrastructure in 3 minutes. Our solution was built to be easily portable to different IT environments and users don't have to worry about the interoperability of the federated learning infrastructure.

Challenge 3: New security and privacy attacks emerge every week. Sustainable federated learning endeavors need to adopt the up-to-date data privacy postures to convince data stakeholders why their federated learning infrastructure meets the rapidly evolving data privacy and security challenges of tomorrow.

Solution: DynamoFL is continually updated with the latest privacy-preserving techniques and algorithms for dealing with the newest wave of privacy and security threats.

6.2 How DynamoFL plugs seamlessly into AI/ML pipelines

DynamoFL supports three methods for standing up federated learning in an organization: a Docker-based solution(Datapods), Python Client Package, and HTTP API. None of these solutions require the client team to abandon or revamp their existing in-house ML or data pipelines.

```python
import dynamofl
from my_training_code import train, test

dynamofl.init('<API_KEY>')
project = dynamofl.get_project('<PROJECT_KEY>')

for round in range(project.current_round, project.rounds):
    # Pulls federated model from server and saves to disk.
    project.pull_model(round=round)

    # Train a new model and save to path (new_model.pt).
    train(project.get_server_model(), 'models/new_model.pt')

    # Push trained model to be aggregated on the server
    project.push_model('models/new_model.pt', 'datalink_21432')
```

Figure 6-2: Client Package

6.2.1 Docker-based Datapod Client

DynamoFL's Datapod solution automates the entire federated learning process in 3 simple steps. It is the most hands-off way of interacting with the API. Simply provide the train and test methods, as well as the data file (or logic to access data if hosted remotely) to stand up federated learning on any client's machine in three minutes [2]. The Datapod was designed to be the simplest solution that exists for setting up federated learning. The datapod can also be easily customized to be flexible and extensible, offering optional callbacks where more complex logic might be desired.

6.2.2 Python Client Package

The Python Client Package was created to streamline interfacing with the API. The goal of the client package is to make the most common API interactions as simple as possible. These common interactions include creating a DynamoFL project, initializing Datapods, pulling and pushing models, as well as reporting validation scores.

[2]https://www.youtube.com/watch?v=XbOttWQ7Q_M&ab_channel=ChristianLau

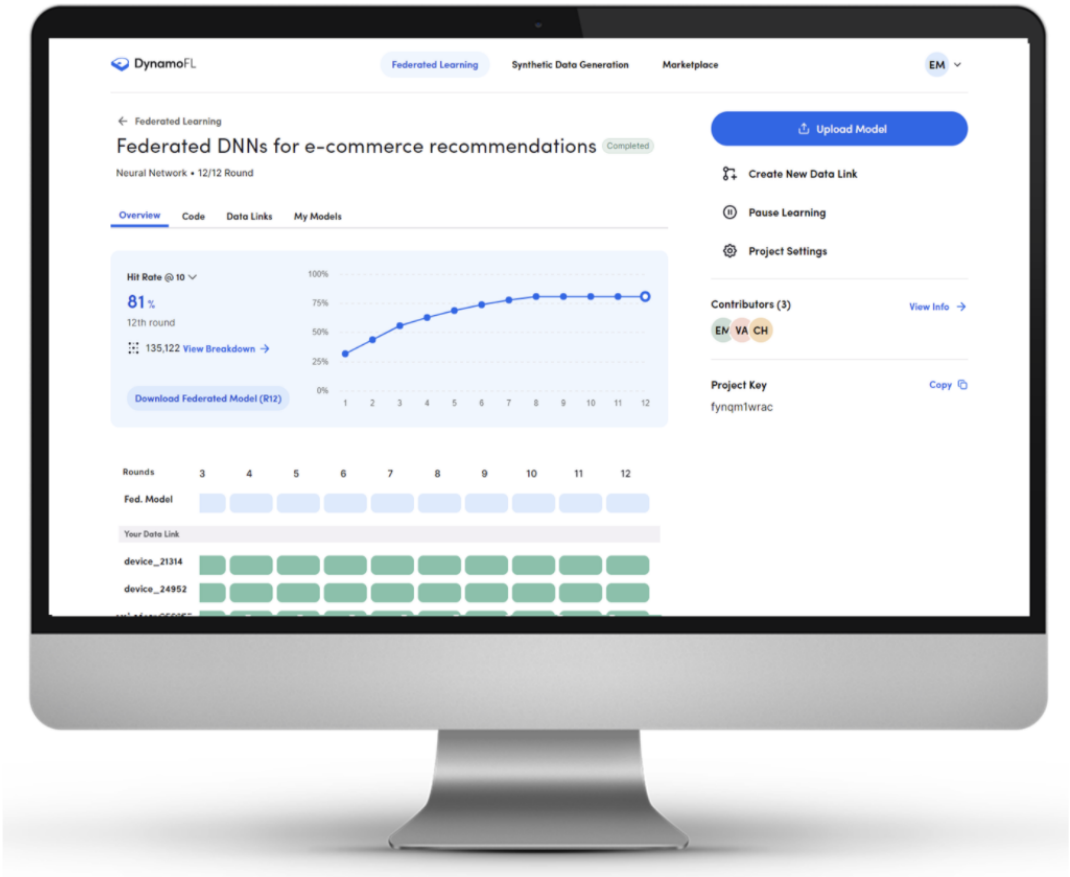


Figure 6-3: DynamoFL UI

6.2.3 HTTP API

The HTTP API was created to make DynamoFL federated learning capabilities accessible using CURL or any other language's built-in HTTP client.

In addition to these three integration methods, our users can deploy and monitor their federated learning projects from our web-app user interface as shown in Figure 6-3.

6.3 Use Cases of DynamoFL

6.3.1 Health AI

Advances in AI for healthcare promise to improve and streamline clinical diagnosis by assisting clinicians in identifying pathologies. Effective AI for healthcare needs to have access to diverse and abundant training images held by different medical institutions. DynamoFL was

built by working closely with researchers at leading universities and healthcare institutions to realize a federated learning solution that can be stood up in minutes rather than months.

6.3.2 Insurtech

The most advanced insurtech solutions draw from a diversity of data sources to perform risk stratification and population health analytics. This requires access to claims data, EHRs, and multivariate signals for social determinants of health (SDOH). Federated learning provides a scalable solution for developing analytics on these privacy-critical data sources. Moreover, DynamoFL's personalized federated learning capabilities enable insurtech firms the ability to provide powerful modeling solutions designed for independent patient cohorts, while ensuring equitable model performance across various subgroups.

6.3.3 Financial Fraud Detection

Law enforcement currently catches less than 1% of money laundered globally. Fraud detection methods could be meaningfully improved if financial institutions could combine their financial records to trace fraudulent transactions across institutions. However, data privacy and competitive interests have precluded this possibility. Federated learning provides a non-aggregative approach to developing fraud detection models trained across financial data silos by enabling fraud detection modeling, while keeping financial records behind the firewalls of financial institutions.

6.3.4 Cohesive Interdepartmental data and ML pipelines

Large organizations developing advanced technologies and internal R&D often host heavily siloed datasets within departments. DynamoFL federated learning offers these organizations the critical capabilities to enable cross-departmental analytics across siloed data, without ever needing to expose raw data between departments or groups.

6.3.5 Manufacturing, Supply-Chain, and Logistics

The global supply chain relies on an interconnected network of manufacturers, shippers, ports, distributors, warehouses, etc. to operate. Efforts to consolidate critical tracking and operations data generated at each of these nodes are impeded by the fact that operational data is often highly protected by each involved party. Federated learning enables analytics across the supply chain since it keeps data at the source it was generated.

6.3.6 Machine Learning on the Edge

The next generation of intelligent edge devices will need to integrate on-device training as they adapt to new environments and achieve maximum data security. However, individual edge devices will not be able to capture enough data independently to train models from scratch. Federated learning enables connected fleets of edge devices to collaboratively train models without needing to transfer raw sensory data across devices.

6.4 Summary

In this chapter, I discussed DynamoFL, an easy-to-use, plug-and-play infrastructure that enables organizations to rapidly stand up federated learning across clients/devices in a matter of minutes. DynamoFL can be plugged into existing data and modeling pipelines of ML teams smoothly and can be used for generalized and personalized FL.

Chapter 7

Conclusion

Federated learning enables clients to collaboratively train a model, while keeping their local training data decentralized. However, there are numerous privacy and training/performance related challenges associated with FL. To address privacy concerns, I proposed a novel privacy-preserving FL mechanism using *oblivious* distributed differential privacy and multi-party computation to prevent model inversion and collusion based attacks. To overcome issues under non-iid/heterogenous FL settings, I proposed a gradient masked averaging approach as an alternative to naive averaging of parameters in FL for improved generalization performance. To come up with optimal personalized and communication-efficient models for clients (especially edge devices), I presented FedLTN, a personalization approach motivated by the lottery ticket hypothesis.

I also presented two systems: PrivacyFL, an extensible, easily configurable, and scalable simulator for federated learning environments and DyanmoFL, an easy-to-use, plug-and-play FL infrastructure for real-world settings. DynamoFL enables organizations to rapidly stand up federated learning across clients/devices in a matter of minutes and can be plugged into existing data and modeling pipelines of ML teams smoothly.

I see tremendous potential for DynamoFL in the future as it can be used by leading organizations such as Facebook, Samsung, Apple, Target, etc. to address their privacy concerns while improving their recommendation models. In addition, as FL lies in the intersection of AI and Web3, a platform like DynamoFL can address numerous problems pertaining to training models in a decentralized manner. DynamoFL can be extended into

an accountable FL system that is fully decentralized and privacy-preserving. It can also be extended to use Ethereum smart contracts to incentivize clients to contribute high-quality models.

FL is an active and ongoing area of research. Although I addressed several challenges, there are still some open problems yet to be explored. These include: (i) Concept Drift: This occurs when the underlying data-generation model changes over time, (ii) Malicious Adversaries: Participating clients may not be honest and might feed bad models, (iii) Fairness: FL algorithms should generate fair/unbiased models with respect to individual/group fairness metrics, (iv) Incentivizing Clients: Clients participating in FL should be appropriately incentivized according to the value of the models they provide, and (v) Vertical FL: In this setting, participating clients have different feature spaces.

Appendix A

Appendix for "Collusion Resistant Federated Learning with Oblivious Distributed Differential Privacy"

A.1 Secure Multi-Party Computation

Consider a group of mutually distrustful parties who need to compute a function jointly over their inputs, while maintaining the privacy of those inputs and ensuring the correctness of the result. If there is some trusted third party, the other parties can simply send their inputs to it for computation and communication of the results.

Secure Multi-Party Computation (MPC) eliminates the need for such a trusted party. Seminal results established in the 1980s [36] show that *any* computation can be emulated by a secure protocol, and the protocol effectively replaces the trusted party. More concretely, consider n parties $P_1, \ldots, P_n$ that hold private inputs $x_1, \ldots, x_n$ and wish to compute some arbitrary function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where the output of P_i is y_i. MPC enables the parties to compute the function using an interactive protocol, where each party P_i learns exactly y_i, and nothing else.

Security of MPC The security of the protocol should be preserved even in the presence of some adversarial entity that corrupts some of the participating parties, combines their

transcripts and coordinates their behaviors.

Several different types of adversaries have been studied, where the complexity and the efficiency of the protocols depend on the level of security:

A semi-honest adversary (also known as "honest-but-curious" or "passive"), follows the protocol specification but may attempt to learn secret information about the private information of the honest parties from the messages it receives. In this scenario, parties can also collude with each other to learn the private information of the other parties.

A malicious adversary (also known as "active") may, in addition, deviate from the protocol specification and follow any arbitrary behavior. This is the strongest level of security possible. Such protocols usually work by adding many checks throughout the protocol execution that parties follow the protocol as specified.

In this work, I focus on semi-honest security (as in prior works in our setting) and leave the malicious case as a future work. However, I provide a mechanism for misbehaved colluding parties who may refuse to add the encrypted noise terms received from their peers (see Section A.9). Note that the work of [9] does not provide malicious security for the case where parties misbehave and do not follow the protocol instructions. They rather offer a weaker form of malicious security. In particular, they only show input privacy for honest users, they do not guarantee correctness.

A.2 Global Sensitivity

Definition 3. (Global Sensitivity [26]) For a real-valued query function $f : \mathcal{D} \rightarrow \mathbb{R}$, where $\mathcal{D}$ denotes the set of all possible datasets, the global sensitivity of q, denoted by Δ, is defined as

$$\Delta = \max_{D_1 \sim D_2} |f(D_1) - f(D_2)|, \tag{A.1}$$

for all $D_1 \in \mathcal{D}$ and $D_2 \in \mathcal{D}$.

The sensitivity is defined as the maximum effect of any single input of the function on the output.

A.3 Laplacian Mechanism

One of the most well-known techniques in differential privacy is the Laplacian mechanism, which uses random noise X drawn from the symmetric Laplacian distribution. The zero-mean Laplacian distribution has a symmetric probability density function $f(x)$ with a scale parameter λ defined as:

$$f(x, \lambda) = \frac{1}{2\lambda} e^{-\frac{|x|}{\lambda}}. \tag{A.2}$$

Given the global sensitivity, Δ, of the query function f, and the privacy parameter ϵ, the *Laplacian mechanism* uses random noise X drawn from the Laplacian distribution with scale $\lambda = \frac{\Delta}{\epsilon}$. The Laplacian mechanism preserves ϵ-differential privacy [25].

A.4 Generating Laplace Random Variable from Gamma Random Variables

A Laplace random variable can be generated from the sum of n random variables as follows:

Lemma 2. *(Divisibility of Laplace distribution [49]) Let $\mathcal{L}(\lambda)$ denote a random variable which has a Laplace distribution with probability density function*

$$f(x, \lambda) = \frac{1}{2\lambda} e^{\frac{|x|}{\lambda}}.$$

Then, for every integer $n \geq 1, \mathcal{L}(\lambda) = \sum_{i=1}^{n} (\mathcal{G}_1(n, \lambda) - \mathcal{G}_2(n, \lambda))$, where $\mathcal{G}_1(n, \lambda)$ and $\mathcal{G}_2(n, \lambda)$ are i.i.d. random variables having gamma distribution with probability density function

$$g(x, n, \lambda) = \frac{(1/\lambda)^{1/n}}{\Gamma(1/n)} x^{1/n-1} e^{-x/\lambda}$$

where $x \geq 0$.

Moreover, the sum of i.i.d. gamma random variables follows gamma distribution (i.e., $\sum_{i=1}^{n} \mathcal{G}(k_i, \lambda) = \mathcal{G}(1/\sum_{i=1}^{n} \frac{1}{k_i}, \lambda)$.

A.5 Supplementary Material: Security Proof

A.6 Security Proofs

Proof of Theorem 1 In the following I show security against a semi-honest adversary who can corrupt $t = n - 1$ parties. Following the standard idea-real paradigm, I show that when executing the protocol with a set of parties $\mathcal{U}$ of size n with threshold $t < n$, the joint view of the server S and any set of less than t users does not leak any information about the other users' inputs except what can be inferred from the output of the computation.

The view of a party P_i consists of its internal state (including its input w_i and randomness) and all messages this party received from other parties. The messages *sent* by this party do not need to be part of the view because they can be determined using the other elements of its view. Given any subset $\mathcal{C}$ of corrupted parties out of the n parties in $\mathcal{U}$, let $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(w_1, \ldots, w_n)$ be a random variable representing the combined views of all parties in $\mathcal{C}$ in the above protocol execution, where the randomness is over the internal randomness of all parties. I am going to show that there exists an efficient simulator that, for every choice of the honest clients' inputs, outputs a simulation of the adversarial participants' view of the protocol run whose distribution is computationally indistinguishable from the distribution of the adversaries' view of the real protocol run.

The following theorem shows that the joint view of any subset of less than t users and the server can be simulated without knowing the secret input of any other users. In other words, the adversary controlling less than t users cannot learn anything other than the output of the computation. In our protocol I consider the leakage L learned from the difference of the noise terms η^0, η^1. Note that this leakage does not affect the error function given later in Definition 2.

Theorem 7 (Semi-honest security against t clients)**.** *For security parameter λ, an n-party protocol for an aggregation function f and all leakage L of size $O(n)$, is L-secure if there exists a PPT simulator* SIM *such that for all $k, t, \mathcal{U}$ where $t < |\mathcal{U}|$, all corrupted parties $\mathcal{C} \subseteq \mathcal{U}$ and all $w_{\mathcal{U}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of* SIM *is computationally indistinguishable from the output of*

$\mathsf{REAL}^{\mathcal{U},t,k}$*:*

$$\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{U}}, L) \approx_c \mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{C}}, L)$$

Proof. In this case since the server is honest, its view is omitted and the joint view of the parties in $\mathcal{C}$ does not depend on the inputs of the parties not in $\mathcal{C}$. The simulator can therefore produce a simulation by running the honest but curious users on their true inputs, and all other honest parties not in $\mathcal{C}$ on dummy values for the input, and output the simulated view of the users in $\mathcal{C}$. That said, the joint view of users in $\mathcal{C}$ will be identical to that in $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}$. Note that the leakage L which is the difference between $\alpha - \beta$ where $\alpha = \gamma_{i,j}^0 - \bar{\gamma}_{i,j}^0$ and $\beta = \gamma_{i,j}^1 - \bar{\gamma}_{i,j}^1$ is given to the simulator to accordingly generate the messages η^0, η^1.

□

Security with a curious server Next I consider security against an honest-but-curious server, who can additionally combine knowledge with some honest-but-curious clients. I show that any such group of corrupted users can be simulated given the inputs of all users in this group, and only the sum of the values of the honest users. That said, the corrupted clients and the server learn nothing more than their own inputs and the sum of the inputs of the honest users.

Theorem 8 (Semi-honest security with curious server)**.** *For security parameter λ, an n-party protocol for an aggregation function f and all leakage L of size $O(n)$, is L-secure if there exists a PPT simulator* SIM *such that for all $k, t, \mathcal{U}$ where $t < |\mathcal{U}|$ and Server $\mathcal{S}$, all corrupted parties $\mathcal{C} \subseteq \mathcal{U} \cup \mathcal{S}$ and all $w_{\mathcal{U}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of* SIM *is computationally indistinguishable from the output of* $\mathsf{REAL}^{\mathcal{U},t,k}$*:*

$$\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{U}}, L) \approx_c \mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{C}}, L, z)$$

where $z = \sum_{w_u \in \mathcal{U} \setminus \mathcal{C}}$

Proof. To prove the theorem I follow a standard hybrid argument. In particular, I define the simulator SIM through a series of modifications to the random variable REAL, so that any two subsequent random variables are computationally indistinguishable.

Hyb_0: In this hybrid I consider the joint view of the parties $\mathcal{C}$ in a real execution of the protocol.

Hyb_1: In this hybrid, instead of running the Diffie-Hellman Key exchange protocol, the simulated honest parties use a uniformly random value r chosen by the simulator. The Decisional Diffie-Hellman assumption guarantees that this hybrid is indistinguishable from Hyb_0.

Hyb_2: In this hybrid, the simulated honest parties use a uniformly random value $\tilde{s}$ instead of s chosen by the simulator and given the knowledge of the leakage L, the messages η^0, η^1 are generated. This hybrid is indistinguishable from Hyb_1 since s is uniformly random value in the real execution.

Hyb_3: In this hybrid, instead of using real input w_i for the y_i message sent to the server, for each honest user $i \in \mathcal{U} \backslash \mathcal{C}$, SIM randomly picks $\tilde{w}_i$ under the constraint that $\sum_{i \in \mathcal{U} \backslash \mathcal{C}} \tilde{w}_i = \sum_{i \in \mathcal{U} \backslash \mathcal{C}} w_i = z$.

This hybrid is indistinguishable from the previous hybrid (see Lemma 6.1 in [9]), since w_i is uniformly random value in the real execution.

The distribution of this hybrid is exactly the same as the distribution of the previous hybrid. In this hybrid, the simulator does not know w_i for any user i.

Now I have proved that the joint view of $\mathcal{C}$ in the real execution is computationally indistinguishable from the view in the simulated execution.

□

Security against malicious users Next I show privacy of the honest parties inputs for the case where t users are maliciously corrupted.

Theorem 9 (Privacy against t malicious users)**.** *For security parameter λ, an n-party protocol for an aggregation function f and all leakage L of size $O(n)$, is L-secure if there exists a PPT simulator SIM such that for all PPT adversaries $M_{\mathcal{C}}$ and $k, t, \mathcal{U}$ where $t < |\mathcal{U}|$, all corrupted parties $\mathcal{C} \subseteq \mathcal{U}$ and all $w_{\mathcal{U} \backslash \mathcal{C}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of SIM is computationally indistinguishable*

from the output of $\mathsf{REAL}^{\mathcal{U},t,k}$*:*

$$\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, w_{\mathcal{U}\setminus\mathcal{C}}, L) \approx_c \mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,k}(\lambda, M_{\mathcal{C}}, L)$$

Proof. As in the proof of Theorem 1, even though the adversarial users get to send arbitrary messages, the messages they receive from honest users never depend on the inputs of those users. That said, the simulator can emulate the real view of the corrupted parties $\mathcal{C}$ by running the honest users on dummy inputs and using $M_{\mathcal{C}}$ for the adversarial users.

□

In the following sections I argue about security given the noisy sum of inputs of the honest parties.

A.7 Collusion Privacy

Recall the definition of Collusion-Privacy in the presence of an $n-1$ attack, in our case the probability distribution $\mathcal{D}$ will be the Laplace distribution.

Collusion-Privacy: An n-party protocol provides *Collusion-Privacy*, for an aggregation function f and a probability distribution $\mathcal{D}$, if any adversary, who controls all parties except client P_h, learns no more than the honest party's values $w_h + \eta$ where $\eta \leftarrow \mathcal{D}$ and $f(w_1, \ldots, w_n)$.

In our oblivious protocol implementation each party P_i sends $\eta_{i,j}^b = s_{i,j} + \gamma_{i,j}^b - \bar{\gamma}_{i,j}^b$ for $b \in \{0, 1\}$ to every party P_j where $\gamma_{i,j}^b$ and $\bar{\gamma}_{i,j}^b$ denote two random values independently drawn from the same gamma distribution $\mathcal{G}(n, \lambda)$. Then, the server sums up all values received from the users such that $W = \sum_{i=1}^{n} w_i + \sum_{i=1}^{N}(\mathcal{G}_1(n, \lambda) - \mathcal{G}_2(n, \lambda)) = \sum_{i=1}^{n} w_i + \sum_{i=1}^{N} \mathcal{L}(\lambda)$ where $N = n^2$.

In our oblivious protocol, if $n-1$ parties collaborate against party P_h, then the final noise added to the weight of the honest party w_h consists of $n^2 - n + 1$ noise terms (the $n-1$ parties can only subtract the $n-1$ noise terms $\eta_{i,j}$ for the case where $i = j$). Since every group of size n of them constitutes a Laplace distribution, the attack leaves in the end $(n^2 - n + 1)/n > n$ i.i.d noise terms from a Laplace distribution. For the case where the

colluding parties try to remove $n^2 - 2n - 1$ noise terms by guessing which noise terms are actually chosen, the error left in w_h is on average the same as in the case where the parties subtract $n^2 - n + 1$ noise terms (the probability of guessing the correct chosen noise is $1/2$). For the case where the colluding parties try to remove $n^2 - 2n - 1$ noise terms by taking the mean of $\gamma_{i,j}^b$ and $\bar{\gamma}_{i,j}^b$ then the variance of the error left is reduced to almost half. See Figure 3 for the empirical results.

A.8 Misbehaved Colluding Parties

Note that the scope of this work is not malicious security. However, colluding parties could misbehave and choose to not add any encrypted noise term received from their peers to their weights in order to learn more information about the weights of the honest parties. In what follows I show a simple modification to Protocol 1 which avoids such a behavior. In particular, the parties choose the random masks s according to an additive secret sharing scheme.

Additive Secret-Sharing. In an additive secret-sharing scheme, n parties hold shares, the sum of which yields the desired secret. By setting all but a single share to be a random element, I ensure that any subset of $n - 1$ parties cannot recover the initial secret.

Definition 4 (Additive secret-sharing)**.** Consider the secret-sharing scheme $\mathrm{A}^n = (\mathsf{Share}, \mathsf{Recover})$ defined below.

- The algorithm Share on input (s, n) performs the following:
  1. Generate $(s_1, \ldots, s_{n-1})$ uniformly at random and define $s_n = s - \sum_{i=1}^{n-1} s_i$.
  2. Output $(s_1, \ldots, s_n)$ where s_i is the share of the i-th party.
- The recovery algorithm $\mathsf{Recover}$ on input $(s_1, \cdots, s_n)$, outputs $\sum_{i=1}^{n} s_i$.

It is easy to show that the distribution of any $n - 1$ of the shares is the uniform one and hence independent of s.

Secret-sharing Notation. Denote by $[s]$ a random sharing of s for the secret-sharing scheme A^n where $[s] = (s_1, \ldots, s_n)$.

Π_{PPFL} secure against misbehaved colluding parties. During the setup phase $\Pi_{\mathsf{PPFL}}.\mathsf{Setup}$ of Protocol 1, each party P_i proceeds as follows:

1. Generate a secret sharing of the zero value using the additive secret-sharing scheme A^{n+1} such that $[0] = (s_{i,1}, \ldots, s_{i,n}, s_{i,n+1})$.

2. For all $j \in [n]$, compute noise $\eta^0_{i,j} = s_{i,j} + \gamma^0_{i,j} - \bar{\gamma}^0_{i,j}$ and $\eta^1_{i,j} = s_{i,j} + \gamma^1_{i,j} - \bar{\gamma}^1_{i,j}$.

During the computation phase $\Pi_{\mathsf{PPFL}}.\mathsf{WeightedAverage}$ of Protocol 1, each party P_i sends to the server:

$$y_i := w_i + \sum_{j=i+1}^{n} r_{i,j} - \sum_{k=1}^{i-1} r_{k,i} + \sum_{j=1}^{n} \eta^{b_i}_{i,j} + s_{i,n+1}.$$

If the $n-1$ colluding parties misbehave and do not add the prescribed η values to the message sent to the server, then the final output of the aggregation protocol will be incorrect. More specifically, since the honest party P_h generated a secret sharing of the masks such that $\sum_{j=1}^{n+1} s_{h,j} = 0$, the masks will not cancel out during the aggregation and the attack will be unsuccessful.

A.9 Security against Sybil Attacks

In the case of a sybil attack, the server collaborates with (or fakes) $n-1$ parties in an effort to leak more information about the weights of the honest party given the output of the aggregation protocol. To accommodate such an attack in Protocol 1, the parties could run a shuffle protocol where the encrypted noise terms are shuffled and randomized collectively (not only by the server). As mentioned in the main body, I have not implemented such a secure shuffle protocol in this version. Using generic MPC the parties could run once at the onset of the protocol during the setup phase a secure shuffle protocol where the parties randomize the encrypted noise terms and generate a secret permutation (which cannot be

revealed even if $n-1$ parties collaborate). The permutation can be applied to the randomized encrypted noise terms. I leave the implementation of such a protocol as future work.

Note that protocols in the shuffle model [8, 20, 27] do not apply in this case since I do not need to just shuffle the encrypted noise terms I also need to randomize them. Furthermore, sybil attacks are still possible in the shuffle model.

A.10 Communication Protocol Diagrams

Our protocol includes the ability to reuse the common randomness for each iteration of logistic regression, so the clients will only require pairwise communication once via the server, at the start of the protocol. In subsequent iterations, each client only has to communicate with the server. The left side of Figure A-1 shows a secure 3-party weighted average protocol where $\bar{w}_1 = w_1 + r_{12} + r_{13}, \bar{w}_2 = w_2 - r_{21} + r_{23}, \bar{w}_3 = w_3 - r_{31} - r_{32}$. Note that the weights are encrypted via the use of the common randomness r. The values $\bar{w}$ reveal nothing about the weights w. In the right side of Figure A-1, client P_1 receives encrypted noise terms $\bar{\eta}_{21} = (\bar{\eta}^0_{21}, \bar{\eta}^1_{21})$ and $\bar{\eta}_{31} = (\bar{\eta}^0_{31}, \bar{\eta}^1_{31})$ and it decides to add only 1-out-of-2 of them. For example, client P_1 will add to his encrypted weights $\bar{x}_1$, the encrypted noise term $\bar{\eta}^1_{2,1}$ and $\bar{\eta}^0_{3,1}$. The final weight sent to the server is $\overline{W}_1 = \bar{w}_1 + \bar{\eta}^1_{2,1} + \bar{\eta}^0_{3,1}$.

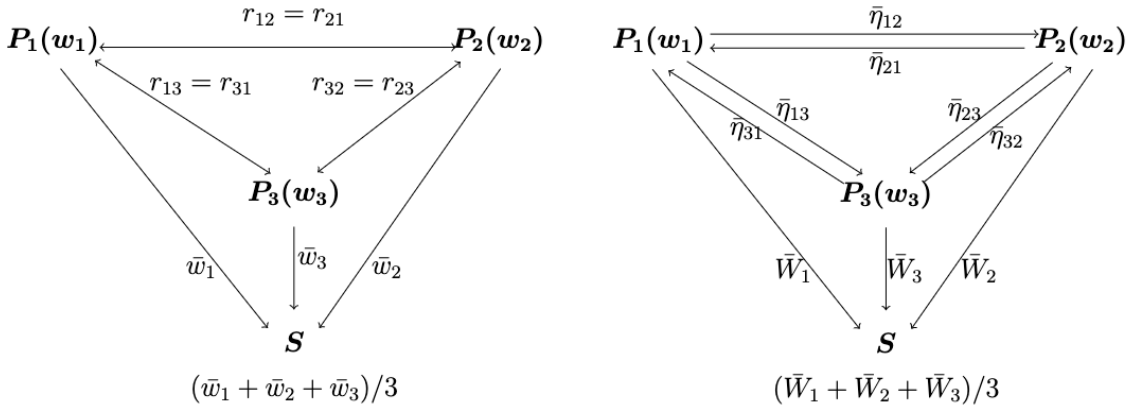


Figure A-1: Toy example of 3-party secure weighted average protocol with server S.

A.11 Secure Aggregation for Multiple Iterations

In Protocol 1, in order to handle the next iteration without the need to re-run the Diffie-Hellman key exchange I follow the same approach as in [9]. More specifically, in the first iteration I use a pseudorandom generator[1] $(r'_{i,j}, s) = PRG(r_{i,j})$ to both update the common randomness $r_{i,j} := r'_{i,j}$ and obtain a new random seed s. For subsequent iterations, instead of executing $\Pi_{\mathsf{PPFL}}.\mathsf{Setup}$, parties can run $PRG(s)$ to obtain a new $r'_{i,j}$ and the seed for the next iteration and so on. Thus the parties need to run the exchange only once at the onset of the training.

A.12 Supplementary Material: Experiments

A.12.1 Matthews Correlation Coefficient

For our adult census income data set, the proper classification for 75% of the examples is False (<= $50,000 income), therefore a naive classifier that always returns False would achieve a misleading 75% accuracy of prediction.

I instead assess our approach using the Matthews Correlation Coefficient (MCC).[63] MCC assesses binary classification performance even in the face of unbalanced output classes by accounting for the size of the true negative prediction set: Information not captured by precision, recall, and the F-score.[5] MCC is a contingency method of calculating the Pearson product-moment correlation coefficient and therefore has the same interpretation. [75, 77, 28] For example, the MCC of the aforementioned naive classifier would be *zero*, indicating no correlation between the predicted and actual values.

Let $C(M, D)$ represent the confusion matrix between binary classification model M and data D, and recall that for classification output variable y, $True$ indicates the higher income bracket and $False$ indicates the lower bracket. I can then define $MCC(M, D)$ for this problem as:

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \tag{A.3}$$

[1] A pseudorandom generator with double expansion is defined as follows: $PRG : \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$.

where matrix entries $TP = C(True, True)$, $FP = C(True, False)$, $TN = C(False, False)$, and $FN = C(False, True)$.

A.12.2 Additional Timing Results

In Table A.1, I present categorized millisecond timing for Protocol 1. The table outlines four categories:

- *Server* includes all data reception, storage, and aggregation tasks by the server presented as mean time per protocol iteration.
- *DH Setup* includes the one-time Diffie-Hellman Key Exchange and noise generation presented as mean time per client party.
- *Training* includes the time spent performing logistic regression on local training data presented as mean time per client per protocol iteration.
- *Encryption* includes the time spent applying encryption randomness, generating the next encryption randomness, and applying privacy noise presented as mean time per client per protocol iteration.

The total number of messages sent by the parties grows quadratically with n for the one-time *DH Setup* phase, and linearly with n for each *Encryption* phase. No messages are sent for the *Training* phase.

		User		
Users	Server	DH Setup	Training	Encrypt
200	20.3	193.0	7.5	2.9
400	42.7	404.0	7.7	5.6
600	61.4	631.6	7.4	8.3
800	81.5	832.1	7.4	11.1
1000	101.1	1,046.5	7.3	13.9
3000	303.8	3,185.0	7.5	41.3
5000	532.5	5,277.9	7.9	68.9

Table A.1: Categorized protocol time in milliseconds.

These timings were generated using network graph 1, which is the case that places all parties including the server at random locations around New York City. I note, however, that choice of network graph should not affect these component times, only the communication latency for the full protocol running times shown in the main body.

A.13 Attacks against the protocol

Here I consider in further depth the security of the protocol against attacks from within the participant population. The first attack is defended well by previous non-oblivious protocols, but the second is not. Our oblivious protocol defends well from both attacks.

A.13.1 Snooping Server

First suppose that the untrusted server, acting alone, attempts to infer the unencrypted weights of a particular client. For a single model weight M, the value transmitted by the honest party h to the server is:

$$M_h = w_h + T_h + R_h \tag{A.4}$$

where w_h is the honest party's original weight, T_h is the total privacy noise added by h, and $R_h = \sum_{c \in C} \pm r_{ch}$ for the set of all other clients C. Recall that for each client pair (h, c), one of them will add R_{ch} and the other will subtract it. In the case of the server acting alone, I do not need to be concerned with the details of T_h, because the weights are already heavily encrypted against the server by MPC.

When the server tries to solve the problem of reconstructing w_h from M_h, it does not possess any of the pairwise client values r_{ch} that compose R_h. With just 100 participating clients, a single client's R_h is a summation of 99 values randomly generated (in our case) from the range $(0, 2^{32})$. The value of R_h is therefore orders of magnitude greater than the range of w_h, leaving the server with no meaningful information about the honest client's private model weights.

As noted, this relatively simple case is handled well by the secure aggregation protocol

alone and successfully defended by prior works in our area. The next case is not.

A.13.2 Collusion attack

This attack is described in the main body. Here I discuss an additional failure mode of the attack.

Additional Residual Noise: In the main body, I discuss a key aspect of our protocol's security against an $n-1$ collusion attack, that given final model output for a single weight:

$$F = w_h + W_C + T_h + T_C \tag{A.5}$$

where w_h is the honest party's original weight, T_h is the honest party's noise sum, $W_C = \sum_{c \in C} w_c$, and $T_C = \sum_{c \in C} T_c$, the conspirators C cannot accurately infer T_h because each conspirator c does not know whether h added $\hat{\gamma}^0_{ch}$ or $\hat{\gamma}^1_{ch}$ to its weight.

However, I only briefly touched on a second aspect of our protocol's resistance to the $n-1$ attack. Not only can the conspirators not remove T_h, they actually cannot accurately remove the T_C either. This is because Protocol 1 also encrypts the oblivious distributed differential privacy noise choices. When h generates $\hat{\gamma}^0_{hc}$, it is a composition:

$$\hat{\gamma}^0_{hc} = \gamma^0_{hc} + s_{hc} \tag{A.6}$$

where γ^0_{hc} is the initial noise choice generated from a difference of gamma distributions and s_{hc} is a large value. Honest party h adds the same s_{hc} to $\hat{\gamma}^0_{hc}$ and $\hat{\gamma}^1_{hc}$. It later subtracts s_{hc} from its own transmitted weights to avoid disturbing the calculation. For the case where s_{hc} is generated based on additive secret sharing (see Section A.9), the honest party uses a different s_{hc} for its own transmitted weights and this attack does not apply because the sum of all the s's is zero.

Thus conspirator c knows $\hat{\gamma}^0_{hc}$ and $\hat{\gamma}^1_{hc}$, and knows which one it selected. However the encryption randomness s_{hc} has already been removed from the final shared output weight, leaving only the unencrypted γ^0_{hc} or γ^1_{hc}. Conspirator c must therefore remove only the appropriate γ_{hc} and *not* $\hat{\gamma}_{hc}$, but it cannot do this, because it does not know what s_{hc} was.

I did not simulate this additional error in the current work. With n parties, this will cause the collaborators C in our attack cases to be *even more wrong* since they can only approximate the $n-1$ values of s's chosen by the honest party for each weight of h, when c removes values based on encrypted $\hat{\gamma}_{hc}$ instead of unencrypted γ_{hc}.

Thus under our Protocol 1, for each collaborator c, the additional error (beyond basic differential privacy noise) in the estimate of w_h will compose two factors:

1. Not knowing whether h added $\hat{\gamma}^0_{ch}$ or $\hat{\gamma}^1_{ch}$.
2. Being off by large encryption value s_{hc} when subtracting $\hat{\gamma}^0_{hc}$ or $\hat{\gamma}^1_{hc}$.

I evaluated the first error in this work and left detailed analysis of the second for future work, although it is already part of the additional security of our protocol.

A.14 Diffie-Hellman Key Exchange Protocol

The cryptographic primitives used in this protocol include:

- An algorithm $G(1^\lambda)$, where λ is the security parameter, that outputs a representation of a cyclic group $\mathbb{G}$ of order q (with $||q|| = \lambda$) for which the discrete logarithm problem is believed to be hard (in other words, it is hard to guess x given g^x for particular groups $\mathbb{G}$).
- A key derivation function $H : \mathbb{G} \to \{0,1\}^\lambda$. It is assumed that if h is distributed uniformly in $\mathbb{G}$, then $H(h)$ is distributed uniformly in $\{0,1\}^\lambda$.
- A pseudorandom generator with double expansion, i.e., $PRG : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$.

(See **Protocol 1 Detail** Diffie-Hellman Key Exchange listing on following page.)

Protocol 2 Detail Diffie-Hellman Key Exchange Protocol Π_{DH}

The protocol Π_{PPFL} runs with parties $P_1, \ldots, P_n$ and a server S. It proceeds as follows:
Public Parameters: $(\mathbb{G}, g, q)$ generated by $G(1^\lambda)$ and modulo p.

Round 1: Each party P_i for $i \in [n]$ proceeds as follows:

- Choose n secrets $a_{i,1}, \ldots, a_{i,n}$ uniformly and independently at random from $\mathbb{Z}_q$ and computes $(pk_{i,1}, \ldots, pk_{i,n}) = (g^{a_{i,1}} \bmod p, \ldots, g^{a_{i,n}} \bmod p)$.
- Each party P_i sends $pk_{i,j}$ to S who forwards to party P_j.

Round 2: Each party P_j for $j \in [n]$ proceeds as follows:

- Upon receiving all values $(pk_{1,j}, \ldots, pk_{n,j})$, compute the shared common keys $r_{i,j}$ for all $i \in [n]$ as follows:
  (a) Using the secret $a_{j,i}$ compute $c_{i,j} = c_{j,i} = (pk_{i,j})^{a_{j,i}} = (g^{a_{i,j}})^{a_{j,i}} \bmod p$.
  (b) Let $c_{1,j}, \ldots, c_{n,j}$ be the set of all common keys. Use a key-derivation function and set $r_{i,j} = r_{j,i} = H(c_{i,j})$

Output: Each party P_i obtains a shared common key $r_{i,j}$ with every other party P_j for $i \neq j$.

Appendix B

Appendix for "Gradient Masked Averaging for Federated Learning"

B.1 GMA on SCAFFOLD

In this section I show SCAFFOLD with the proposed gradient masking.

B.2 Datasets and Models

In this section I discuss more about the datasets used for different test settings, the distribution of the dataset across the clients, and the models used for our experiments in detail.

B.2.1 IID and Non-IID data distribution

I simulate IID distribution of data by assigning samples from all classes to each client such that the representation of each class on a client is approximately equal. This ensures that all labels are equally represented on each client. The same idea of i.i.d-ness is used in all experiments and datasets of in-distribution and out-of-distribution test settings. Table B.1 shows a sample distribution of classes across 10 clients in IID setting.

For the Non-IID distribution, I use the label distribution skew with 10% noise. That is, each client will have a majority or 90% of samples from any 2 classes randomly chosen from

Algorithm 4 Gradient Masked SCAFFOLD [45]

Initialize w_0
for each server epoch, t = 1,2,3,... **do**
  Choose C clients at random
  for each client in C, n **do**
    $w_t^n = \text{ClientUpdate}(w_{t-1})$
    $w_t^n, \Delta_c^n = \text{ClientUpdate}(w_{t-1}, \Delta_c)$
    $\Delta_t^n = \frac{n_k}{\sum_{n=1}^{N} n_k}(w_t^n - w_{t-1})$
  end for
  $\Delta_t = \sum_{n=1}^{N} \Delta_t^n$
  $\Delta_c = \frac{1}{N} \sum_{n=1}^{N} \Delta_c^n$
  $mask = \tilde{m}_\tau(\{\Delta_t^n\}_{n=1..C})$
  $w_t = w_{t-1} - \eta_g * mask \odot \Delta_t$
end for

ClientUpdate(w):

Initialize $w_0 = w$
$c_i = c_i^+$
for each local client epoch, i=0,1,2,3,..,n **do**
  $g_i = \nabla_{w_i} L(w_i)$
  $w_{i+1} = w_i - \eta_c \, g_i - c_i + c$
end for
$c_i^+ = (i) g_i(x)$or $(ii) c_i - c + \frac{1}{K\eta_l}(x - y_i)$
return w_{i+1} , $c_i^+ - c_i$ to server

the set of all classes and the rest 10% of samples would include data samples from all other 8 classes. This induces a heterogeneity in the label distribution. The label distribution skew ensures that the distribution varies across clients. Table B.2 shows a sample distribution of classes across 10 clients in non-IID setting. It is to be noted that this IID and Non-IID distribution of data is specific to the distribution of the train data across the clients. It is independent of the distribution of the test dataset or the dataset used to evaluate the performance of the global model. In both these cases the global model test dataset contains samples from all classes in equal proportions.

Table B.1: This table shows the label distribution across clients for an IID setting. Each client will have randomly chosen examples from all 10 classes. This represent the 10 class setting in MNIST. I have considered 3 clients for the table. The same pattern would be present across all clients.

	0	1	2	3	4	5	6	7	8	9
Client 1	585	643	591	550	571	561	631	628	620	620
Client 2	589	691	593	628	553	526	588	640	602	590
Client 3	531	697	595	627	557	557	596	626	581	633
.....										

Table B.2: This table shows the label distribution skew for experiments on the non-IID data distribution across clients. This represents the 10 class setting in MNIST. I have taken 3 clients. The same pattern would be present across all clients.

	0	1	2	3	4	5	6	7	8	9
Client 1	2894	2247	51	48	50	53	52	47	47	47
Client 2	44	2246	1962	40	42	42	42	41	41	46
Client 3	31	31	33	1962	33	1371	35	31	31	32
.....										

B.2.2 Datasets

FedCMNIST Colored MNIST was introduced [4] to evaluate OOD generalization. It was used for binary classification based on the class label. The digit were colored red or green depending on its correlation with the label. Precisely, numbers 0-4 (label 0) were colored red and 5-9 (label 1) were colored green in the train environment with a noise of probability 0.2 to 0.25. The colors were reversed on the test environment. This was ported naively to a federated setting by [30]. In this setting, the task was binary classification and the federated network was limited to 2 clients. Each client involved all class labels, however, there was a difference in the probability of noises induced in the train clients and the test client had a

reversed color based spuriousness. In this paper I use a complex version of colored MNIST. I design a multi-class classification problem where the task is to identify the digit. I induce a spuriousness based on correlation with the class label. Specifically, each digit would have one or more color that remains the same across examples in the train set or in the data at the participating clients. The color of the same digit in the test set would be different from that at any of the train clients. I call this dataset FedCMNIST. This spuriousness is in addition to the label distribution based skew with non-iid distribution of data across clients.

FedRotMNIST The rotated MNIST dataset was introduced by [35] for domain generalization. It was adapted to a federated setting by [30] where all data samples at a client would be rotated at a specified angle irrespective of the class label. I use a slightly different spuriousness that is more correlated to the class label or digit. I rotate each digit at an angle such that a label based correlation is induced. In our experiments the digits were rotated at 10, -10, 20, -20, 30, -30, 40, -40, 50, and -50 respectively. The test images are not rotated at any angle irrespective of the digit or label. The preprocessing includes padding on rotation and cropping.

FEMNIST is a federated version of Extended MNIST [21] with labels to identify the writer of the character. A real world distribution of this dataset was introduced in [15] where the data is partitioned across clients such that the data from a writer is always clustered. The test data is reserved for samples from writers who were not included in any of the train clients. This ensures a different data distribution for evaluation of the global model. This is a simpler version of out-of-distribution generalization performance evaluation of the global model.

B.3 Hyperparameters

In this section I discuss the various hyperparameters involved in the experiments and their effects.

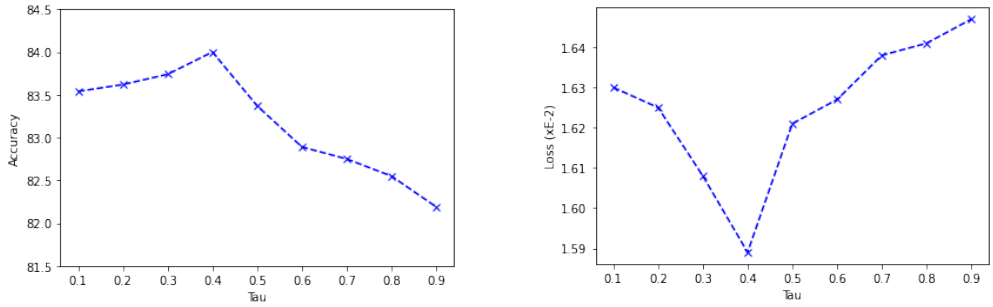
B.3.1 Effects of τ

$\tau \in \{0, 1\}$ is a hyperparameter introduced to threshold agreement across clients in Equation 3. It marks the minimum agreement required to consider the gradient for aggregation. When $\tau = 0$ or negligible, the gradients of all parameters will have an agreement score greater than or equal to τ. This makes all gradients consistent across clients. The agreement score would be over-written by 1 and the equation becomes equal to that of naive federated aggregation where all gradients are averaged with the same importance. This is equivalent to an underfit condition. The opposite overfit criterion can happen with a high τ value. In this case, no gradient would be considered dominant and all parameters updates would be diminished corresponding to their agreement score. The agreement score can be 1 when the data distribution across clients is an ideal i.i.d distribution where all gradients across clients would be along the same direction. But in practical scenario, such data distributions are rare in a federated setting. When agreement $= 1$, $w^k = w^{k-1} - \eta_g \, 1 \odot \Delta^k = w^k - \eta_g \Delta^k$; equivalent to naive federated aggregation. This implies that the naive federated aggregation is a case of the proposed gradient masked averaging.

Throughout the experiments I have maintained $\tau = 0.4$. This value was observed to give the best performance across models and datasets on searching on $\{0, 1.0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. $\tau = 0.4$ implies that the gradient being considered have a 40% excess or a total of 60% of the client gradients along the dominant direction. From our experiments it was observed that when τ is low, the model underfits. The accuracy was best at $\tau = 0.4$ and on further increasing τ, it was overfitting. This is visible from the test accuracy vs. τ plot in Figure B-1a and test loss vs. τ plot in Figure B-1b.

B.3.2 Effect of Client Momentum

In contrast to the experiments in [79], I use an SGD optimizer with momentum ($\rho = 0.9$) at each client for all our experiments. This was primarily because of the increase in test accuracy observed during our experiments on FedAVG with and without momentum. However, the enhancements due to gradient masking was independent of the momentum induced at the client optimizer. In both cases (with and without momentum), gradient



(a) (b)

Figure B-1: (a) Test accuracy vs. τ (b) Test loss vs. τ. The experiment was on non-i.i.d distributed CIFAR-10 using a ResNet model.

masking was outperforming naive averaging in most of the algorithms and datasets. Table B.3 shows the performance of the algorithms and their GMA versions on non-i.i.d distributed FMNIST using an LeNet model. The client optimizers used in our experiments is a naive SGD optimizer with momentum parameter and it does not involve the correction parameter introduced in [99].

Table B.3: Performance of the algorithms and their GMA versions with and without momentum(ρ) on non-i.i.d distributed FMNIST using an LeNet model. Momentum improves performance of the algorithms. Irrespective of momentum, GMA outperforms AVG.

Dataset (Model)		**FedAVG** ($\rho = 0$)		**FedAVG** ($\rho = 0.9$)	
		AVG	**GMA**	**AVG**	**GMA**
MNIST	**IID**	**99.01**	98.96	99.1	**99.16**
(LeNet)	**Non-IID**	98.43	**98.55**	98.87	**98.9**
FMNIST	**IID**	**88.61**	88.49	89.14	**90.52**
(LeNet)	**Non-IID**	86.95	**87.8**	88.1	**88.38**
FEMNIST	**IID**	98.8	**98.92**	**99.7**	99.68
(LeNet)	**Non-IID**	92.17	**94.61**	94.2	**96.04**
CIFAR-10	**IID**	85.8	**86.31**	87.3	**87.61**
(ResNet)	**Non-IID**	81.1	**82.28**	83.25	**83.95**

B.3.3 Effect of GroupNorm

The test accuracies reported in paper corresponding to CIFAR-10 used a ResNet18 model with batch normalization layers replaced by group normalization[96] similar to the experiments in [79]. Our initial experiments involved batch normalization as in the original ResNet and it was observed that the replacement of batch norm with group norm improved the test accuracies. Table B.4 shows the comparison of the algorithms and their GMA versions on CIFAR-10 using ResNet model having batch normalization and group normalization layers.

It is to be noted that irrespective of the normalization layer used, gradient masking was outperforming naive averaging across all algorithms and data distributions. This further validates the capabilities of the proposed GMA.

Table B.4: Average in-distribution test performance(%) over the last 10 communication rounds of FedAVG, FedProx, SCAFFOLD, FedAdam, FedYogi and their GMA versions on i.i.d and non-i.i.d distributions of CIFAR-10 on ResNet18 models using batch normalization and group normalization. The best result among AVG and GMA versions of each algorithm is shown in bold.

Dataset		**FedAVG**		**FedProx**		**SCAFFOLD**		**FedADAM**		**FedYogi**	
(Model)		**AVG**	**GMA**	**AVG**	**GMA**	**AVG**	**GMA**	**AVG**	**GMA**	**AVG**	**GMA**
ResNet	**IID**	87.11	**87.42**	87.2	**87.38**	87.56	**87.52**	77.32	**80.65**	78.78	**80.55**
BatchNorm	**Non-IID**	77.3	**79.9**	78.4	**80.2**	75.82	**78.83**	69.82	**74.05**	67.29	**71.51**
ResNet	**IID**	87.3	**87.61**	87.18	**87.5**	86.58	**86.72**	86.9	**87.7**	87.53	**87.78**
GroupNorm	**Non-IID**	83.25	**83.66**	83.87	**84.4**	84.01	**85.36**	83.53	**84.84**	83.17	**84.55**

B.3.4 Grid Search Range

For all our experiments a search for client learning rate (η_l) and global learning (η_g) rate across the grid specified below was conducted and the best performing learning rates of each algorithm was used for experiments that reported the test performances in the paper. The grid was fixed the same for all datasets and algorithms for easy experimentation.

$$\eta_{\mathbf{l}} \in \{10^{-3}, 10^{-2}, 5.10^{-2}, 10^{-1}\}$$
$$\eta_{\mathbf{g}} \in \{10^{-2}, 10^{-1}, 1, 1.5, 2\}$$
$$\tau \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$$

Table B.5: The best learning rates corresponding to the performances of the algorithms and datasets as reported in Table B.4

Dataset		FedAVG		FedProx		SCAFFOLD		FedAdam		FedYogi	
		η_g	η_l	η_g	η_l	η_g	η_l	η_g	η_l	η_g	η_l
MNIST	**IID**	1.0	0.01	1.0	0.01	1.0	0.01	0.05	0.001	0.05	0.001
	Non-IID	1.0	0.01	1.0	0.01	1.0	0.01	0.05	0.001	0.05	0.001
FMNIST	**IID**	1.0	0.1	1.0	0.1	1.0	0.01	0.05	0.001	0.05	0.001
	Non-IID	1.0	0.01	1.0	0.01	1.0	0.01	0.05	0.001	0.05	0.001
FEMNIST	**IID**	1.0	0.01	1.0	0.01	1.5	0.1	0.05	0.001	0.05	0.001
	Non-IID	1.0	0.01	1.5	0.01	1.0	0.01	0.05	0.001	0.05	0.001
CIFAR-10	**IID**	1.0	0.01	1.0	0.01	1.5	0.01	0.01	0.001	0.01	0.001
	Non-IID	1.5	0.01	1.0	0.001	1.5	0.001	0.05	0.001	0.05	0.001

The values for global learning rate and client learning rate was observed to vary across algorithms and datasets. However, it was observed that the same combination was giving the best performance for GMA and AVG versions of the same algorithm for the same dataset. This enables better comparison of naive averaging and gradient masked versions of algorithms better. For τ, it was observed that $\tau = 0.4$ was giving the best result acriss algorithms and datasets. More details about hyperparameter τ is given in Appendix C.1.

B.3.5 Best Performing Learning Rates

The best performing learning rates corresponding to the Tables B.4 and 3.2 are given below in Tables B.5 and B.6 respectively.

Table B.6: The best learning rates corresponding to the performances of the algorithms and datasets as reported in Table 3.2

Dataset		FedAVG		FedProx		SCAFFOLD		FedAdam		FedYogi	
		η_g	η_l	η_g	η_l	η_g	η_l	η_g	η_l	η_g	η_l
FEMNIST	**Real World**	1.0	0.01	1.0	0.01	1.0	0.01	0.05	0.001	0.05	0.001
FedCMNIST	**IID**	1.0	0.01	1.0	0.01	1.0	0.01	0.05	0.001	0.05	0.001
	Non-IID	1.0	0.01	1.0	0.01	1.0	0.01	0.05	0.001	0.05	0.001
FedRotMNIST	**IID**	1.0	0.01	1.0	0.01	1.5	0.01	0.05	0.001	0.05	0.001
	Non-IID	1.0	0.01	1.5	0.01	1.0	0.01	0.05	0.001	0.05	0.001

B.3.6 Performance for same learning rates

Table B.7 shows the in-distribution test performance of naive and gradient masked versions of FedAVG on non-iid FMNIST dataset. Performance corresponding to the entire range of hyperparameters mentioned in Appendix C.4 has been given here. It can be noted that across learning rates where the model learns, gradient masking outperforms naive averaging. This suggest the gains are highly robust to hyperparameter choices.

Table B.7: Performance of FedAVG across a range of global learning rate and client rate on non-iid FMNIST. It can be observed that GMA outperforms AVG in most of the cases where the algorithms learn and converge.

Global Learning Rate		0.001	0.01	0.05	0.1	1.0	
	0.01	0.1	56.06	61.66	68.02	0.1	AVG
		0.1	**57.72**	**65.13**	**72.56**	0.1	GMA
	0.1	56.93	73.66	83.39	85.22	0.1	AVG
		57.51	**73.9**	**83.79**	**87.22**	0.1	GMA
	1.0	72.69	86.49	87.76	88.31	0.1	AVG
		73.34	**87.0**	**88.14**	**88.4**	0.1	GMA
	1.5	**77.11**	86.29	87.82	86.96	0.1	AVG
		75.63	**87.04**	**88.21**	**88.3**	0.1	GMA
	2.0	71.53	82.42	86.93	87.63	0.1	AVG
		77.59	**86.9**	**87.6**	**88.1**	0.1	GMA

Client Learning Rate

B.3.7 Additional Hyperparameters

In this section I mention the additional hyperparameters that I have used in our experiments. These hyperparameters were maintained the same across all experiments considering the large number of hyperparameters to tune. Batch size was maintained at 32 for all our experiments across datasets and algorithms. An L2 regularization of strength $1e^{-5}$ was used in all the loss functions. $\mu = 1$ for FedProx. Additionally, all models start from the same

initialization of parameters considering the results in [64].

B.4 Details of Experiments

In this section I give more details about the various experiments included in the paper.

B.4.1 Increased Clients and Local Epochs

Table B.8 and Table B.9 shows average test accuracies corresponding to Figures 3-1a and 3-1b. These report the average test performance over last 10 communication rounds on convergence of FedAVG and its GMA version across increasing number of selected clients and number of local epochs at each client per communication round.

Table B.8: Average test performance values of non-i.i.d FMNIST on varying number of clients

Number of Selected Clients		**10**	**50**	**100**	**250**
FedAVG	**AVG**	88.1	82.42	74.39	71.11
	GMA	**88.38**	**83.11**	**76.11**	**72.87**

Table B.9: Average test performance values of non-i.i.d FMNIST with varying number of local client epochs per communication round

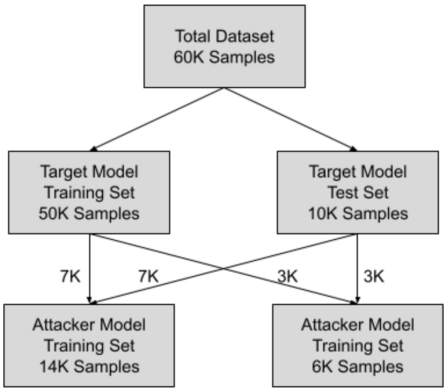
Number of Local Epochs		**3**	**5**	**10**	**20**
FedAVG	**AVG**	89.91	85.71	84.13	80.27
	GMA	**91.73**	**87.29**	**87.11**	**86.66**

B.4.2 Convex Objective

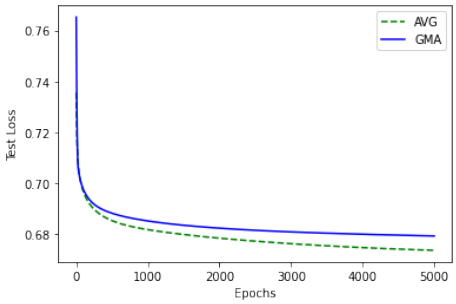
For the experiments on convex objective I employed a logistic regression model on MNIST with FedAVG for global model approximation at the server. The model was created using a single linear layer in a perceptron model without any non-linear activation functions. A cross-entropy loss was used and the model was updated by an SGD optimizer with momentum at each client. Each client undergoes 1 update step before aggregation (gradient masked or naive avergaing) to approximate the global model. Experiments included i.i.d and non-i.i.d data distribution of data across clients with in-distribution test for global model evaluation. The number of clients selected at each round was set to 10 and the models were run until convergence. The various hyperparameters involved in the experiments are global model learning rate of 1.0, client model learning rate of 0.01, and $\tau = 0.4$.

B.4.3 Membership Inference Attack

Most machine learning models tends to overfit on their training data and such models are susceptible to membership inference attacks that can accurately predict whether a data sample was present in the training set of the model given the model output logits [89]. This is a major privacy breach and it can be simulated by using a black-box adversarial attacker model. The attacker model I have employed is a binary logistic regression model with binary cross-entropy loss. The input to this attacker model is the logits of the converged gradient masked averaging model and naive averaging model. The attacker model is supposed to identify whether the input of the global model corresponding to the logit given was present in the global model's training set or not. For our experiments, I used CIFAR-10 and ResNet models. Firstly, the GMA and AVG models were trained and tested. For each model, the logits corresponding to the train and test set data and their labels (whether train data or test data) were stored. The data is split as shown in Figure B-2a [89] and the attacker model is trained for 5000 rounds. The accuracy is as reported in the paper and loss as shown in Figure B-2b. A lower attack accuracy of gradient masked implies that GMA has better immunity to membership inference attacks than naive averaging global models.



(a)



(b)

Figure B-2: (a) Data split and creation for attacker model (b) Test loss vs. epochs of the logistic regression attacker model.

Bibliography

[1] Kartik Ahuja, Karthikeyan Shanmugam, Kush R. Varshney, and Amit Dhurandhar. Invariant risk minimization games, 2020.

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30, 2017.

[3] Martin Arjovsky. Out of distribution generalization in machine learning, 2021.

[4] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization, 2020.

[5] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus AF Andersen, and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[6] Leighton Pate Barnes, Huseyin A Inan, Berivan Isik, and Ayfer Özgür. rtop-k: A statistical estimation approach to distributed sgd. *IEEE Journal on Selected Areas in Information Theory*, 1(3):897–907, 2020.

[7] Stephen D Bay, Dennis Kibler, Michael J Pazzani, and Padhraic Smyth. The uci kdd archive of large data sets for data mining research and experimentation. *ACM SIGKDD explorations newsletter*, 2(2):81–85, 2000.

[8] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 441–459. ACM, 2017.

[9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[11] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. ABIDES: towards high-fidelity market simulation for AI research. *CoRR*, abs/1904.12066, 2019.

[12] David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the 2020 ACM International Conference on AI in Finance*, ACM ICAIF '20, New York, NY, USA, 2020. Association for Computing Machinery.

[13] Peter Bühlmann. Invariance, causality and robustness, 2018.

[14] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[15] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2019.

[16] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. Securely sampling biased coins with applications to differential privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 603–614, New York, NY, USA, 2019. Association for Computing Machinery.

[17] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109, 2011.

[18] Yiqiang Chen, Wang Lu, Jindong Wang, and Xin Qin. Fedhealth 2: Weighted federated transfer learning via batch normalization for personalized healthcare. *arXiv preprint arXiv:2106.01009*, 2021.

[19] Yiqiang Chen, Wang Lu, Jindong Wang, Xin Qin, and Tao Qin. Federated learning with adaptive batchnorm for personalized healthcare. *arXiv preprint arXiv:2112.00734*, 2021.

[20] Albert Cheu, Adam D. Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 375–403. Springer, 2019.

[21] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

[22] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE, 2015.

[23] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[24] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[25] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[26] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[27] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2468–2479. SIAM, 2019.

[28] James D Evans. *Straightforward statistics for the behavioral sciences*. Brooks/Cole, 1996.

[29] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach, 2020.

[30] Sreya Francis, Irene Tenison, and Irina Rish. Towards causal federated learning for enhanced robustness and privacy, 2021.

[31] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[32] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[33] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

[34] Quan Geng, Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The staircase mechanism in differential privacy. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1176–1184, 2015.

[35] Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. Domain generalization for object recognition with multi-task autoencoders, 2015.

[36] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

[37] Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. Fedboost: A communication-efficient algorithm for federated learning. In *International Conference on Machine Learning*, pages 3973–3983. PMLR, 2020.

[38] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

[39] Meryem Janati Idrissi, Ismail Berrada, and Guevara Noubir. Fedbs: Learning on non-iid data in federated learning using batch normalization. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 861–867. IEEE, 2021.

[40] A Ingerman and K Ostrowski. Introducing tensorflow federated. *External Links: Link Cited by*, 4, 2019.

[41] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *Advances in Neural Information Processing Systems*, pages 6343–6354, 2018.

[42] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[43] 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 Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, 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, 2021.

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

[45] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning, 2021.

[46] Geeho Kim, Jinkyu Kim, and Bohyung Han. Communication-efficient federated learning with acceleration of global momentum. *arXiv preprint arXiv:2201.03172*, 2022.

[47] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[48] Kavya Kopparapu and Eric Lin. Fedfmc: Sequential efficient federated learning on non-iid data. *arXiv preprint arXiv:2006.10937*, 2020.

[49] S. Kotz, T. Kozubowski, and K. Podgorski. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Progress in Mathematics. Birkhäuser Boston, 2001.

[50] Masanori Koyama and Shoichiro Yamaguchi. When is invariance useful in an out-of-distribution generalization problem ?, 2021.

[51] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[52] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex), 2021.

[53] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Survey of personalization techniques for federated learning. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 794–797. IEEE, 2020.

[54] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[55] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998. *URL http://yann. lecun. com/exdb/mnist*, 10:34, 1998.

[56] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371*, 2020.

[57] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization, 2021.

[58] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, May 2020.

[59] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020.

[60] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fedbn: Federated learning on non-iid features via local batch normalization. *arXiv preprint arXiv:2102.07623*, 2021.

[61] Yanjun Ma, Dianhai Yu, Tian Wu, and Haifeng Wang. Paddlepaddle: An open-source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105–115, 2019.

[62] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020.

[63] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

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

[65] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[66] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *Advances in neural information processing systems*, 32, 2019.

[67] Vaikkunth Mugunthan, Anton Peraire-Bueno, and Lalana Kagal. Privacyfl: A simulator for privacy-preserving and secure federated learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 3085–3092, 2020.

[68] Vaikkunth Mugunthan, Ravi Rahman, and Lalana Kagal. Blockflow: An accountable and privacy-preserving solution for federated learning, 2020.

[69] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910*, 2018.

[70] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753. IEEE, 2019.

[71] DJ Newman, S Hettich, CL Blake, and CJ Merz. Uci repository of machine learning databases. dept. information and computer sciences, univ. california, irvine, 1998.

[72] Jaehoon Oh, Sangmook Kim, and Se-Young Yun. Fedbabu: Towards enhanced representation for federated image classification, 2021.

[73] Emre Ozfatura, Kerem Ozfatura, and Deniz Gündüz. Fedadc: Accelerated federated learning with drift control. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 467–472. IEEE, 2021.

[74] Giambattista Parascandolo, Alexander Neitz, Antonio Orvieto, Luigi Gresele, and Bernhard Schölkopf. Learning explanations that are hard to vary, 2020.

[75] Karl Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

[76] Mohammad Pezeshki, Sékou-Oumar Kaba, Yoshua Bengio, Aaron Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[77] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2011.

[78] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[79] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2021.

[80] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization, 2020.

[81] Jonatan Reyes, Lisa Di Jorio, Cecile Low-Kam, and Marta Kersten-Oertel. Precision-weighted federated learning. *arXiv preprint arXiv:2107.09627*, 2021.

[82] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[83] Soroosh Shahtalebi, Jean-Christophe Gagnon-Audet, Touraj Laleh, Mojtaba Faramarzi, Kartik Ahuja, and Irina Rish. Sand-mask: An enhanced gradient masking strategy for the discovery of invariances in domain generalization, 2021.

[84] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[85] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

[86] Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

[87] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *International Conference on Machine Learning*, pages 3329–3337. PMLR, 2017.

[88] Irene Tenison, Sai Aravind Sreeramadas, Vaikkunth Mugunthan, Edouard Oyallon, Eugene Belilovsky, and Irina Rish. Gradient masked averaging for federated learning. *arXiv preprint arXiv:2201.11986*, 2022.

[89] Shruti Tople, Amit Sharma, and Aditya Nori. Alleviating privacy attacks via causal learning, 2020.

[90] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging, 2020.

[91] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms, 2019.

[92] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization, 2020.

[93] Kangkang Wang, Rajiv Mathews, Chloé Kiddon, Hubert Eichner, Françoise Beaufays, and Daniel Ramage. Federated evaluation of on-device personalization. *arXiv preprint arXiv:1910.10252*, 2019.

[94] 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, 2019.

[95] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farokhi Farhad, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis, 2019.

[96] Yuxin Wu and Kaiming He. Group normalization, 2018.

[97] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[98] An Xu and Heng Huang. Double momentum sgd for federated learning. *arXiv preprint arXiv:2102.03970*, 2021.

[99] Jing Xu, Sen Wang, Liwei Wang, and Andrew Chi-Chih Yao. Fedcm: Federated learning with client-level momentum, 2021.

[100] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(3):1–207, 2019.

[101] Yousef Yeganeh, Azade Farshad, Nassir Navab, and Shadi Albarqouni. Inverse distance aggregation for federated learning with non-iid data. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 150–159. Springer, 2020.

[102] Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning, 2018.

[103] Honglin Yuan and Tengyu Ma. Federated accelerated stochastic gradient descent, 2021.

[104] Honglin Yuan, Warren Morningstar, Lin Ning, and Karan Singhal. What do we mean by generalization in federated learning?, 2021.

[105] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Probabilistic federated neural matching, 2019.

[106] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.