August 2023

Emerging Trustworthiness Issues in Distributed Learning Systems

Hamid Mozaffari
University of Massachusetts Amherst

EMERGING TRUSTWORTHINESS ISSUES IN DISTRIBUTED LEARNING SYSTEMS

A Dissertation Presented

by

HAMID MOZAFFARI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2023

Manning College of Information and Computer Sciences

EMERGING TRUSTWORTHINESS ISSUES IN DISTRIBUTED LEARNING SYSTEMS

A Dissertation Presented

by

HAMID MOZAFFARI

Approved as to style and content by:

Amir Houmansadr, Chair

Daniel Sheldon, Member

Yair Zick, Member

Hossein Pishro-Nik, Outside Member

Ramesh K. Sitaraman, Associate Dean for
Educational Programs and Teaching
Manning College of Information and Computer
Sciences

DEDICATION

To the intricate dance of neurotransmitters, with a focus on the leading role played by dopamine in our lives.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my PhD advisor, Dr. Amir Houmansadr, for his continuous guidance, encouragement, and mentorship throughout this journey. His expertise and dedication have been invaluable in shaping this research, and I am grateful for his support. I would also like to extend my appreciation to my committee members, Dr. Daniel Sheldon, Dr. Yair Zick, and Dr. Hossein Pishro-Nik, for their insightful feedback on the work presented in this thesis. Their comments and suggestions have contributed significantly to the quality of this dissertation.

I am incredibly grateful for the friendships that have been made throughout this journey. To Anahita, Alireza, Hamed, Arian, Soha, Pegah, Sadegh, AliBana, Virat, Milad, Shahrzad, Hadi, Amirhossein, and many others who have become my family away from home, thank you for the laughter, support, and camaraderie that made this experience all the more enjoyable and memorable.

Lastly, I would like to dedicate this work to my family, who has been a constant source of love and support throughout my life. To my Dad, Mom, and Amir, I cannot thank you enough for your unwavering belief in me and for always being there to provide encouragement and advice. Your love and support have been the pillars that have held me up during this journey, and I am eternally grateful for everything you have done for me.

ABSTRACT

EMERGING TRUSTWORTHINESS ISSUES IN DISTRIBUTED LEARNING SYSTEMS

MAY 2023

HAMID MOZAFFARI

B.Sc., K. N. TOOSI UNIVERSITY OF TECHNOLOGY

M.Sc., AMIRKABIR UNIVERSITY OF TECHNOLOGY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Amir Houmansadr

A distributed learning system allocates learning processes onto several workstations to enable faster learning algorithms. Federated Learning (FL) is an increasingly popular type of distributed learning which allows mutually untrusted clients to collaboratively train a common machine learning model without sharing their private/proprietary training data with each other. In this dissertation, we aim to address emerging trustworthiness issues in distributed learning systems, particularly in the field of FL.

First, we tackle the issue of robustness in FL and demonstrate its susceptibility by presenting a comprehensive analysis of the various poisoning attacks and defensive aggregation rules proposed in the literature and connecting them under a common framework. To address this issue, we propose Federated Rank Learning (FRL) which reduces the space of client updates from a continuous space of float numbers in

standard FL to a discrete space of integer values, limiting the adversary's options for poisoning attacks.

Next, we address the privacy concerns in FL, including access privacy and data privacy. An adversarial server in FL gets information about the data distribution of a target client by monitoring either I) local updates that the target submits throughout the FL training or II) the access pattern of the target, which can be privacy sensitive in many real-world scenarios. To preserve access privacy, we design Heterogeneous Private Information Retrieval (HPIR), which allows clients to fetch their specific model parameters from untrusted servers without leaking any information. We believe that HPIR will enable new application scenarios for private distributed learning systems, as well as improve the usability of some of the known applications of PIR. To preserve data privacy, we show that local rankings leak less information about private training data. We conduct a comprehensive investigation on the privacy of rankings in FRL to measure data leakage compared to weight parameter updates in standard FL in presence of the state-of-the-art white-box membership inference attack.

Finally, we address the issue of fairness in FL where a single model cannot represent all clients equally due to heterogeneity in their data distributions. To alleviate this issue, we propose Equal and Equitable Federated Learning (E2FL). E2FL produces fair federated learning models by preserving both equity and equality among the participating clients based on learning on parameter rankings where multiple global models are learned so that each group of clients can benefit from their personalized model.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The accuracy of machine learning models is directly proportional to the amount of training data utilized. With the exponential growth of available data, there has been a proliferation of new applications of machine learning models, including but not limited to next word prediction [126], autonomous driving [105], and medical diagnosis systems [96]. To enhance the efficiency of the learning process, distributed learning systems have been developed that allocate the learning process across multiple workstations. These systems distribute the data among the workstations for parallel learning.

In real-world settings, data is often dispersed across different organizations or clients, making it necessary to preserve privacy restrictions. To address this, Federated Learning (FL) has emerged as a prominent research topic in recent years. FL is a distributed learning paradigm that enables mutually untrusted *clients*, such as Android devices, to collaboratively train a shared model, known as the *global model*, without explicitly sharing their local training data. Figure 1.1 illustrates one round of FL training, where the FL *server* (e.g., a Google server) repeatedly collects model updates computed by the clients using their local private data. In each round of FL training, first, the FL server broadcasts the global model to the clients (Figure 1.1-①). Then the clients begin to train a local model based on their local private data starting from the global model (Figure 1.1-②). After local training, in Figure 1.1-③, the server collects clients' updates, aggregates them using an *aggregation rule* (AGR), and finally

uses the aggregated updates to tune the jointly trained model (i.e., global model for next round).

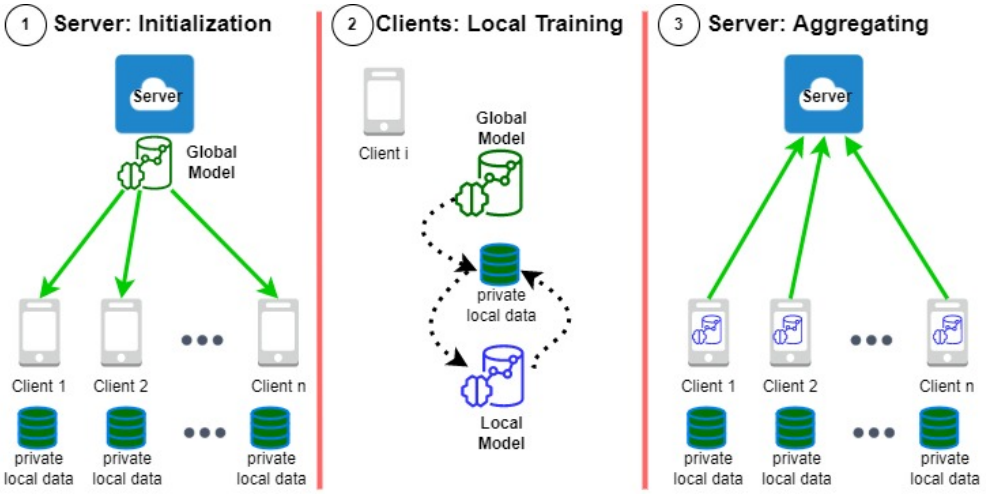


Figure 1.1. One round of Federated Learning (FL), including server initialization, client local training, and server aggregation.

The shift from centralized architectures to distributed systems has introduced new trustworthiness challenges. This thesis aims to explore these emerging issues in distributed learning systems, and FL in particular, and design mechanisms to mitigate them. To achieve this, I built privacy-preserving, robust, and fair mechanisms on top of distributed learning systems, and FL in particular, to enhance their trustworthiness.

1.1 Trustworthiness of distributed learning systems

In this section, I identify the emerging trustworthiness challenges in distributed learning systems in real-world settings. For a distributed learning system to be trusted, it must meet the following criteria: I) robustness against adversarial inputs, II) preservation of the privacy of training data, and III) fairness to all participants.

1.1.1 Robustness

The threat of poisoning is a significant obstacle to the real-world adoption of FL in critical tasks [90, 114, 149]. This is the first focus of my work. Federated Learning algorithms, including FedAvg [126] and FedProx [115], operate on mutually untrusted clients and servers, making them susceptible to poisoning attacks [90, 149, 28]. A poisoning adversary can either own or control a few FL clients, known as malicious clients, and instruct them to share malicious updates with the central server, reducing the performance of the global model. There are three forms of poisoning attacks in FL: *targeted attacks* [23, 157] aim to reduce the utility of the global FL model on specific test inputs of the adversary's choice; *untargeted attacks* [16, 63, 148] aim to reduce the utility of the global model on arbitrary test inputs; and *backdoor attacks* [15, 159, 165] aim to reduce the utility on test inputs containing a specific signal called the trigger. In my research, I focus on the more severe threat of untargeted poisoning [149], which affects the majority of FL clients.

Fake or compromised? A fork in the literature Based on how the adversary introduces malicious clients in the FL ecosystem, existing works on FL poisoning can be categorized into two major lines of work: 1) a small percentage (<1%) of "actual" clients are *compromised* by an adversary, e.g., by taking control of some compromised mobile devices; 2) a large percentage (>10%) of *fake* clients are created and injected into the FL ecosystem, e.g., by creating Sybil accounts or using botnets. The former category (compromised clients), exemplified by works such as [16, 148], targets sophisticated, large-scale applications such as Gboard and Siri that have deployed proper protections against Sybil attacks and botnets. However, these attacks require compromising actual FL devices, which is costly in practice.

On the other hand, the latter category (fake clients), exemplified by works such as [35], assumes that the adversary can introduce large numbers of fake clients, such as spam bots, into the FL ecosystem. Such fake clients cannot be injected in sophisticated

applications such as Gboard and Siri as thoroughly discussed by [149]; however, many FL applications are built on third-party code/software, and hence, vulnerable to such fake clients.

These two major lines of study are significantly different in terms of their assumptions about the adversary, threat model, and the practical settings they represent. This stark disconnect between the threat model assumptions in current works leads to confusion about the applicability of a given threat model to the FL setting of interest.

1.1.2 Privacy

In FL, data is located at different clients (e.g., organizations or application users) who train a common model on their private data while preserving their privacy restrictions. FL aims to preserve privacy by sharing the trained models instead of the actual data. There are two aspects of privacy for FL users:

Data Privacy: Membership inference attacks are the most fundamental form of privacy leakage in FL training. In these attacks, the adversary can determine whether a specific data sample was used in the private training dataset of a particular user based on the model updates the target user provides throughout the FL learning process. It is essential that the trusted distributed learning framework assures the participating clients that their private data is safe by training a local model on top of them.

Access Privacy: In some FL frameworks, the user must ask the server to retrieve personal data. For example, LotteryFL [109] proposes using a personalized subnetwork training in a large neural network, so each FL client needs only the parameters of a subnetwork. In this scenario, each client must retrieve a specific subset of data parameters related to their customized subnetwork instead of retrieving all parameters. Another example would be the recent line of research [70, 125] where it proposes learning multiple global models because, due to heterogeneity in clients'

data distributions, a single model cannot represent all clients equally. To address this issue, multiple global models can be learned, and each client asks for the specific global model that represents its local data better so that each group of clients benefits from their personalized model. Another aspect of privacy is information leakage about the access pattern of the target user. The adversary can determine the client data distribution by monitoring the target's access pattern. Private access to models in a distributed learning system is a crucial aspect of trustworthiness, as clients do not want to leak any information about what they are seeking.

1.1.3 Fairness

The study of algorithmic fairness in FL algorithms is a relatively under-researched area. In FL, the performance of the global model varies across the network due to heterogeneity in the data each client possesses. This concern is referred to as *representation disparity* [78] and results in unfair performance gaps for participating clients, i.e., a tail user whose data distribution differs from the majority of the clients does not receive similar performance guarantees. The majority of existing research in this area [116, 174, 131, 170, 113, 152] defines FL fairness as providing similar accuracy across FL clients. Specifically, Li et al. [116] proposed a formal definition for fairness in FL: a model is more fair when its performance distribution across clients is more uniform, i.e., when $\text{std}\{F_k(w)\}_{k\in[K]}$ is smaller where std{.} is the standard deviation and $F_k(.)$ denotes the local objective function of client $k \in [K]$.

1.2 Contributions

This dissertation addresses the emerging trustworthiness challenges in distributed learning systems, with a focus on FL. In this section, I outline my contributions to each aspect of trustworthiness in these systems.

My first contribution focuses on the robustness of FL. I propose a comprehensive framework for understanding the various poisoning attacks and defensive aggregation rules (AGRs) in the FL literature. I present a spectrum of adversarial models, rather than focusing on the extremes of small percentages of compromised clients or large percentages of fake clients. My framework aims to bridge the gap between existing works and provide practitioners and researchers with a clear understanding of the various threat models that need to be considered when designing FL systems. Moreover, I propose the federated rank learning (FRL) approach, where clients are asked to rank edges in a randomly initialized network. My results demonstrate that training on ranks reduces the adversary's choices to poison the global model.

Next, I investigate the issue of access privacy in FL. To preserve access privacy, I design the heterogeneous private information retrieval (HPIR) mechanism, allowing clients to fetch their specific model parameters from untrusted servers without leaking any information. I also investigate the privacy leakage of local rankings in FRL by mounting membership inference attacks on them.

The final aspect I investigate is the fairness of the globally trained model in FL. Due to heterogeneity in clients' data distributions, a single model cannot represent all clients equally. To address this challenge, I propose to learn multiple global models, so that each group of clients benefits from their personalized model. I design a fair FL based on learning on parameter ranks, ensuring that the global model performs similarly across different clients.

Specifically, I make the following contributions for each aspect of trustworthiness in distributed systems:

1.2.1 Robustness: Fake vs. compromised clients in federated learning

The field of FL security is plagued with confusion due to the proliferation of research that makes different assumptions about the capabilities of adversaries and

the threat models they operate under. My work aims to clarify this confusion by presenting a comprehensive analysis of the various poisoning attacks and defensive aggregation rules (AGRs) proposed in the literature, and connecting them under a common framework. To connect existing threat models, I present a hybrid threat model, which lies in the middle of the spectrum of adversaries, where the adversary compromises a few clients, trains a denoising diffusion probability model (DDPM) with their compromised samples, and generates new synthetic data to solve an optimization for a better attack against different robust aggregation rules. By presenting the spectrum of FL adversaries, I aim to provide practitioners and researchers with a clear understanding of the different types of threats they need to consider when designing FL systems, and identify areas where further research is needed. My work is vital in providing a clear and concise overview of the current state of the art in FL security and helping advance the field.

1.2.2 Robustness: Robust federated learning by training on parameter ranks

I argue that the key factor to the success of poisoning attacks against existing FL systems is the large space of model updates available to the clients, allowing malicious clients to search for the most poisonous model updates, e.g., by solving an optimization problem.

Federated Rank Learning (FRL): I present FRL, a novel FL algorithm that concurrently achieves the two goals of robustness against poisoning attacks and communication efficiency. FRL uses a novel learning paradigm called *supermasks* training [176, 143] to create edge rankings, which, as we will show, allows FRL to reduce communication costs while achieving significantly stronger robustness. Specifically, in FRL, clients collaborate to find a *subnetwork* within a *randomly initialized* neural network which we call the *supernetwork* (this is in contrast to conventional FL where

clients collaborate to *train* a neural network). The goal of training in FRL is to collaboratively rank the supernetwork's edges based on the importance of each edge and find a *global ranking*. The global ranking can be converted to a supermask, which is a binary mask of 1's and 0's, that is superimposed on the random neural network (the supernetwork) to obtain the final subnetwork. For example, in our experiments, the final subnetwork is constructed using the top 50% of all edges. The subnetwork is then used for downstream tasks, e.g., image classification, hence it is equivalent to the global model in conventional FL. Note that in entire FRL training, weights of the supernetwork *do not* change.

More specifically, each FRL client computes the importance of the edges of the supernetwork based on their local data. The importance of the edges is represented as a ranking vector. Each FRL client will use the *edge popup* algorithm [143] and their data to compute their local rankings (the edge popup algorithm aims at learning which edges in a supernetwork are more important over the other edges by minimizing the loss of the subnetwork on their local data). Each client then will send their local edge ranking to the server. Finally, the FRL server uses a novel *voting* mechanism to aggregate client rankings into a global ranking vector, which represents which edges of the random neural network (the supernetwork) will form the global subnetwork.

Intuitions on FRL's robustness: In traditional FL algorithms, clients send large-dimension model updates $\in \mathbb{R}^d$ (real numbers) to the server, providing malicious clients significant flexibility in fabricating malicious updates. By contrast, FRL clients merely share the rankings of the edges of the supernetwork, i.e., integers $\in [1, d]$, where d is the size of the supernetwork. This allows the FRL server to use a voting mechanism to aggregate client updates (i.e., ranks), therefore, providing high resistance to adversarial ranks submitted by poisoning clients, since each client can only cast a single vote! Therefore, as we will show both theoretically and empirically, FRL provides robustness by design and reduces the impact of untargeted poisoning attacks.

Furthermore, unlike most existing robust FL frameworks, FRL does not require any knowledge about the percentages of malicious clients.

Intuitions on FRL's communication efficiency: In FRL, the clients and the server communicate just the rankings of the edges in the supernetwork, i.e., a permutation of indices in $[1, d]$. Ranking vectors are generally significantly smaller than the global model. This, as we will show, significantly reduces the upload and download communication in FRL compared to Federated Averaging (FedAvg) [126], where clients communicate model parameters, each of 32/64 bits.

1.2.3 Private access: Heterogeneous Private Information Retrieval

Private information retrieval (PIR) is a technique to provide query privacy to users when fetching sensitive records from untrusted databases. That is, PIR enables users to query and retrieve specific records from untrusted database(s) in a way that the serving databases can not identify the records retrieved. PIR algorithms have been suggested to be used in various application scenarios involving untrusted database servers [136, 130, 75, 83, 76, 36, 29, 141, 111], from retrieving Tor relay information [130] to privacy-preserving querying of location services [69] to registering Internet domains [136].

For example, a hospital company (say, hospital.com) wants to learn multiple models on the training data of different patients.These models are representing different diseases or different groups for each disease. It also wants to leverage a CDN provider (say, Akamai) for hosting its models. The hospital will need to give Akamai access to the contents of its communications (e.g., by providing Akamai with hospital's TLS private keys); while this is necessary to enable Akamai serve hospital's patients, unfortunately, this exposes the private information of hospital's patients to Akamai. Now Akamai server can get more information about the each patient training data distribution by models that they are asking to retrieve. A potential solution is using

a two-server PIR protocol, in which one PIR server is a CDN edge server (e.g., an Akamai server), and the other one is hospital's origin server. However, one of the main reasons that content publishers (e.g., hospital) use CDNs is to refrain from processing large volumes of traffic on their own servers. Therefore, for the presented two-server PIR model to be practical, it needs to impose much lower computation/communication overheads on hospital's origin servers, compared to the overheads imposed on the CDN (Akamai) edge servers. Figure 1.2 shows two steps for retrieving the global model using HPIR, and submitting the model update for each client.

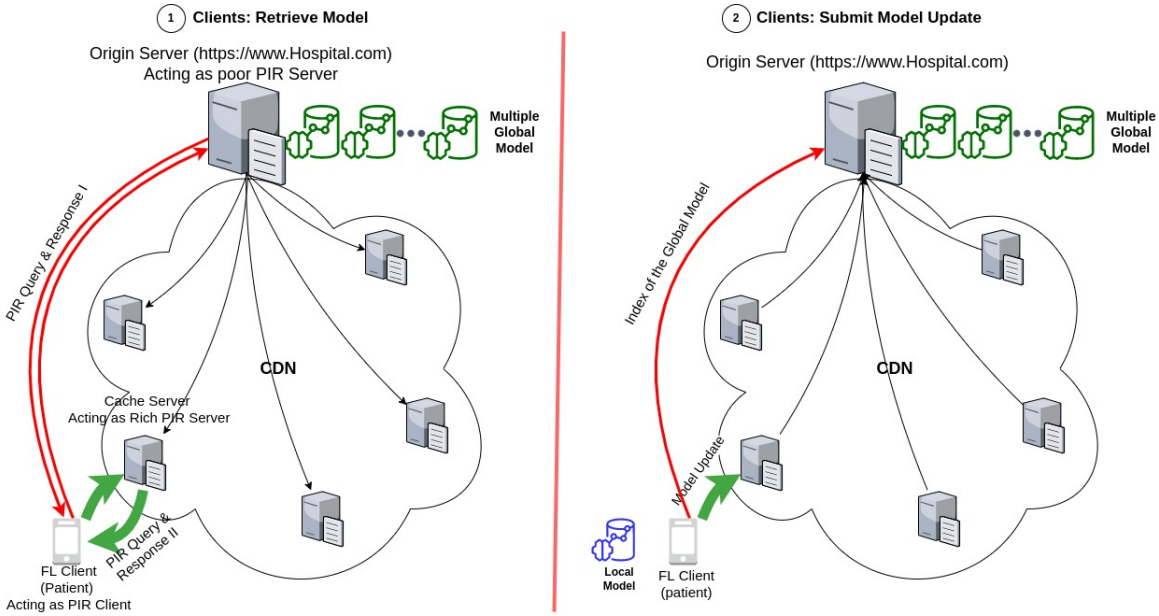


Figure 1.2. Heterogeneous private information retrieval in federated learning with multiple global models.

Existing multi-server PIR protocols are homogeneous! The existing body of work on multi-server PIR considers a setting in which *the non-colluding PIR servers have similar computation and communication constraints*. We call such traditional multi-server PIR protocols *homogeneous*.

Introducing heterogeneous multi-server PIR. In this thesis, we introduce a new class of multi-server PIR, which we call *heterogeneous PIR (HPIR)*. An HPIR

protocol is a multi-server PIR protocol with *asymmetric computation and communication constraints on its servers*, i.e., some of its servers handle higher computation/communication overheads than the others. We argue that *HPIR algorithms enable new applications for PIR, as well as improve the utility of some of the existing applications of PIR*; this is because HPIR allows the participation of low-resource entities in running private services.

1.2.4 Privacy analysis of federated rank learning

There are different types of privacy leakages via sharing the trained local model. I focus on membership inference attack because it is the most fundamental measurement of privacy leakage. Although there is a large number of research works studying the privacy leakage of FL with different types of membership inference attacks, there is no investigation on the information leakage of rankings trained in federate rank learning. I investigate the privacy leakage of ranking in an FL trained on parameter ranks. My results show that local rankings of FRL leak less information about the training data compared to weight parameters in standard FL. This holds true across all scenarios, from observing the local models (when the FL server is adversary) and aggregated global model (when the FL clients are adversary) for initial to end and small distance FL rounds to large-distance FL rounds.

1.2.5 Fair federated learning by training on parameter ranks

A trustworthy system must not only ensure security and privacy preservation but also promote fairness. Existing fairness definitions and algorithms in centralized ML cannot be directly extended to FL due to: (I) FL being a multi-party framework that introduces unique notions of algorithmic bias, and (II) the distinct nature of data bias and minority groups in FL. To address the well-known accuracy unfairness/bias among collaborating users, I propose a ranking-based FL approach similar to FRL.

In this work, FL fairness is examined from two perspectives: **a)** *Equality*, which aims to provide similar performances for all individual clients, and **b)** *Equity*, which strives to offer comparable performances across all groups of clients (i.e., majority and minority groups), where a group is defined as a set of clients with similar data distributions.

Due to the heterogeneity in clients' data distributions, a single model cannot adequately represent all clients. A *trade-off* exists between training one global model and multiple global models. If one global model is trained, all clients can benefit from each other's knowledge, but the model will be biased towards the majority population. Conversely, training multiple models (e.g., as in IFCA [70], HypCluster [125], and MOCHA [153]) improves fairness but each global model loses knowledge from excluded clients. To strike a balance, I present **E**qual and **E**quitable **F**ederated **L**earning (E2FL) as a novel FL algorithm that achieves both equality and equity. In E2FL, multiple global models are trained, but in each round, these models are combined into one global model to leverage the knowledge from all client groups.

The key insight in E2FL involves converting the problem of model weight optimization (in standard FL) into the problem of ranking model edges (as in FRL). In E2FL, each client computes the importance of randomized neural network edges using their local data, which is represented by a ranking vector. The E2FL server then employs a majority voting mechanism to aggregate the collected local rankings into multiple global rankings based on the group index they belong to. Finally, the E2FL server combines all group rankings into one global ranking for the next round of training. Applying majority vote on group rankings rather than local rankings aids E2FL in enforcing equity, as each group has equal influence on the global model. To ensure equality in E2FL, clients use their group's global ranking, rather than the overall global ranking, for downstream tasks, as it better represents the client and its group members.

CHAPTER 2

FAKE OR COMPROMISED? MAKING SENSE OF MALICIOUS CLIENTS IN FEDERATED LEARNING

FL is susceptible to *poisoning* by malicious clients who aim to hamper the accuracy of the global model by contributing malicious updates during FL's training process. The literature on FL poisoning and defensive AGRs is vast, with tens of new papers being published every month. However, this proliferation of research has created confusion for practitioners and researchers alike, as each paper makes different assumptions about the threat model and the capabilities of the adversary.

Presenting a spectrum of adversaries: As opposed to considering two (extreme) threat models, i.e., compromised and fake (defined in Section 1.1.1), in this chapter, we fill the gap between these two threat models and present a spectrum of threat models, as sketched in Figure 2.1. We believe that this is essential to truly understand the poisoning threat to various types of FL deployments in the real world. We provide a comprehensive analysis of the different types of poisoning attacks and defensive AGRs that have been proposed in the literature, and we show how these different works can be connected by a common framework.

The two ends of the spectrum consist of just compromised and just fake clients. However, in the middle, there exist adversaries who compromise a few real users and use their data to mount a larger scale attack using large percentages of fake clients. We call this a *hybrid threat model*. Given the quick and broad adoption of FL in various applications, we believe that the hybrid threat model will be representative of a very large percentage of FL applications in future. Under the hybrid threat, we

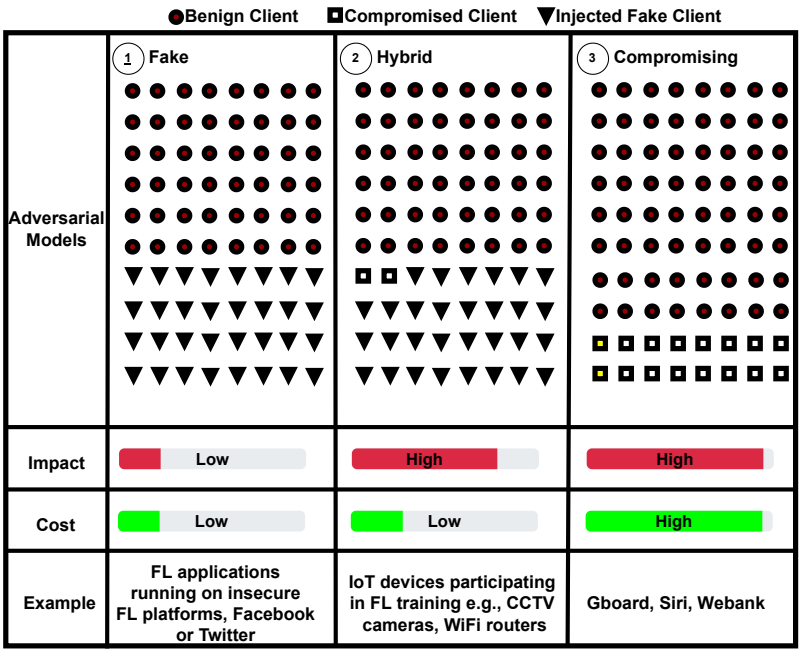


Figure 2.1. Spectrum of the adversarial models that vary in the number of compromised clients and the number of fake clients injected into the FL system. **(1)** A scenario of fake clients may occur when FL applications are running on insecure FL platforms, or if we learn an FL model on Facebook or Twitter users, which can have a large number of fake accounts. In this scenario, the adversary can easily introduce fake clients, such as spam bots, into the FL ecosystem; these fake clients do not have any real data and can manipulate the updates they send to the central server. **(2)** A scenario of hybrid attack may occur when IoT devices participating in FL training, such as CCTV cameras or WiFi routers. An adversary can buy zombies from botnets for compromised and fake clients (more details in Section 2.3.4). **(3)** A scenario of compromised clients may occur in FL applications such as Google's Gboard, Apple's Siri, and Webank. In this scenario, the adversary may use sophisticated techniques such as social engineering, malware injection, or exploiting software vulnerabilities to compromise a small percentage of clients.

propose a novel model poisoning attack, called *hybrid attack*, that first leverages the data of compromised clients to *generate* more data using state-of-the-art denoising diffusion probabilistic models (DDPM). Then it uses existing state-of-the-art model poisoning attacks to fabricate poisoned model updates for both compromised and fake clients to share with the server.

2.1 Background

2.1.1 Diffusion models

The *denoising diffusion probabilistic model* (DDPM) [85, 135, 99, 43, 93, 154, 40, 142, 146, 84, 158, 88] is a generative model that aims to learn the underlying structure of a complex data distribution from a small number of noisy observations. DDPM is based on the idea of diffusion, which is a process of iteratively exchanging information between the data points in order to reveal their underlying structure.

To learn the structure of the data distribution, DDPM uses a diffusion process to transform the given input data into a latent representation. This latent representation is obtained through a series of diffusive steps, which are defined by a diffusion operator. At each diffusive step, the data points are transformed by exchanging information with their neighbors in the data space. This process is repeated until the latent representation converges to a stable state, which captures the underlying structure of the data. Once the latent representation is obtained, it can be used to generate new samples that are similar to the original input data.

One key advantage of DDPM is that it is able to learn the structure of the data distribution from a small number of observations, even in the presence of noise. This makes it particularly useful for applications where the data is limited or noisy, such as in the case of compromised clients in federated learning. By using DDPM to generate new samples from a small number of compromised clients, an adversary is able to craft a malicious update for FL poisoning that is representative of the data distribution of the benign clients.

2.2 Types of Byzantine-robust aggregation rules

In order to make FL robust against malicious clients, the literature has designed various *robust aggregation rules (AGR)* [27, 128, 169, 132, 38], which aim to remove or attenuate the updates that are more likely to be malicious according to some criterion.

The existing AGRs for federated learning can be categorized into three categories: non-robust AGRs, AGRs agnostic to poisoning attacks, and AGRs that adapt to or are aware of the poisoning attacks in FL ecosystem.

Overall, the choice of aggregation rule in federated learning depends on the specific requirements of the system and the level of protection against model poisoning attacks that is desired. non-robust aggregation rules are simple and easy to implement but may not provide sufficient protection against attacks, while agnostic and adaptive aggregation rules offer greater protection but may be more complex to implement and may require additional information about the number of malicious updates.

2.2.1 Non-robust AGR

Non-robust aggregation rules, such as federated averaging (FedAvg) [103, 126], do not consider the presence of malicious clients in the federated learning ecosystem. Therefore, such AGRs simply aggregate the model updates received from all clients by computing a non-robust function of the updates. While these approaches are generally simpler and easy to implement, they are vulnerable to model and data poisoning attacks [149, 148, 63, 132]. Examples of such non-robust AGRs include FedAvg [103, 126], SCAFFOLD [92], and MIME [91].

2.2.1.1 FedAVG

In non-adversarial FL settings, i.e., without any malicious clients, the dimension-wise Average (FedAvg) [103, 126] is an effective AGR. In fact, due to its efficiency, Average is the only AGR implemented by FL applications in practice [123, 140]. However, even if there is a single malicious client, it can destroy the global model by sharing very large local model updates [27].

2.2.2 Robust AGRs agnostic to FL poisoning

Robust AGRs, such as Median and Norm-Bounding, are robust in that they aim to reduce the impact of malicious clients' updates. But, they are *agnostic* in that they do not have any knowledge of the specifics of the attacks, e.g., they do not know the number of malicious updates in each round. These rules use techniques from robust statistics, such as outlier removal or clipping the norms of updates, to exclude or mitigate the impact of malicious updates during the aggregation process. While this can provide some protection against model poisoning attacks, it may not be sufficient if the number of malicious updates is large or if the malicious updates are able to evade detection.

2.2.2.1 Median

Median is a well-known robust statistic that is less sensitive to outlier values. To compute the coordinate-wise Median [169] of updates, for each of the coordinates of the update, we compute the median of all values from all client updates. Median AGR is particularly useful in situations where there may be a small number of malicious or outlier clients that provide model updates that are significantly different from the others. By using Median, these outlier model updates will not have as much influence on the global model as they would with non-robust AGRs, such as FedAvg.

2.2.2.2 Norm-Bounding

This AGR [157] bounds the L2 norm of all submitted client updates to a fixed threshold τ, with the intuition that the effective poisoned updates should have high norms. For a threshold τ and an update ∇, if the norm, $||\nabla||_2 > \tau$, ∇ is scaled by $\frac{\tau}{||\nabla||_2}$, otherwise, the update is not changed. The final aggregate is an average of all the updates, scaled or otherwise.

2.2.3 Robust AGRs that adapt to FL poisoning

Adaptive aggregation rules have the advantage of knowing the number of malicious updates in each round for aggregation. These rules use this information to adapt their aggregation process in order to mitigate the impact of malicious updates on the final model. While this can be an effective way to protect against model poisoning attacks, it requires a way to accurately estimate the number of malicious updates, which can be a challenging task.

2.2.3.1 Multi-Krum

Blanchard et al. [27] proposed Multi-Krum AGR as a modification to their own Krum AGR. Multi-Krum selects an update using Krum and adds it to a selection set, S. Multi-Krum repeats this for the remaining updates (which remain after removing the update that Krum selects) until S has c updates such that $n - c > 2m + 2$, where n is the number of selected clients and m is the number of compromised clients in a given round. Finally, Multi-Krum averages the updates in S.

2.2.3.2 Trimmed-Mean

Yin et al. [169] proposed Trimmed-Mean that aggregates each dimension of input updates separately. It sorts the values of the j^{th}-dimension of all updates. Then it removes m (i.e., the number of compromised clients) of the largest and smallest values of that dimension, and computes the average of the rest of the values as its aggregate for the dimension j.

2.3 Types of poisoning adversaries

A poisoning attack is either *data* or *model* poisoning attack: in data poisoning, adversary can poison only the data on malicious client device, while in model poisoning, the adversary can directly manipulate/poison the model updates of the malicious clients. In this work, we focus on model poisoning as it is strictly stronger than data

poisoning [149, 148]; hence here onward, poisoning in any context refers to model poisoning, unless stated otherwise.

In this section, we provide the attack algorithms for the spectrum of adversaries who mount attacks of varying costs and impacts. The spectrum consists of adversaries who inject cheap, fake clients with no knowledge about the benign data distribution to powerful adversaries who can compromise actual clients in FL settings and use their data for crafting malicious updates. We also detail our novel, *hybrid attack* that lies in the middle of this spectrum.

2.3.1 Adversary with fake clients

In federated learning (FL) systems, an attacker can inject fake clients in order to send arbitrary fake local model updates to the cloud server. This type of attack is more affordable and easier to perform than compromising genuine clients, as the attacker does not need to bypass anti-malware software or evade anomaly detection on the clients' devices. Instead, the attacker can emulate fake clients using open-source projects or free software such as android emulators, which can be run on a single machine to emulate multiple instances, i.e., multiple FL clients, significantly reducing the attack cost. Fake clients also offer the advantage of being fully controlled by the attacker, as android emulators can grant root access to the devices. These factors make model poisoning attacks using fake clients a realistic threat in FL systems.

Cao et. al proposed MPAF [35], a method of attacking FL systems through the injection of fake clients. In MPAF, the attacker selects a randomly initialized model as the base model (θ'), whose test accuracy is near random guessing, and crafts fake local model updates to force the global model to mimic the base model. This is done by subtracting the current global model parameters (θ^t for FL round t) from the base model parameters and scaling up the fake local model updates by a factor λ to amplify their impact. Equation 2.1 shows the malicious updates of the fake clients.

$$\theta^t_{m\in[M]} = \lambda(\theta' - \theta^t) \tag{2.1}$$

where $\theta_{m\in[M]}$ are the malicious model updates for M injected fake clients, and θ' is the randomly initialized base model.

To perform MPAF, the attacker must have minimum knowledge of the FL system, meaning that they only have access to the global models received during training. Despite this limited information, MPAF is able to effectively manipulate the global model by driving it towards the base model in each FL round. This is done through the calculation of fake local model updates ($\theta_{m\in[M]}$), which are then aggregated by the cloud server along with genuine local model updates from genuine clients. The attacker can choose a large λ to ensure that the attack is effective even after aggregation.

In this work, we refer to this attack as the Fake attack. This attack is characterized by the minimal knowledge and ability required on the part of the adversary who controls the fake clients. Specifically, the Fake attack is the simplest attack of this kind in FL and represents one end of the spectrum of attacks based on the attack impact and cost.

2.3.2 Adversary with compromised clients

To evaluate robustness of various FL algorithms, we use state-of-the-art model poisoning attacks from [148]. The attack proposes a general FL poisoning framework and then tailors it to specific FL settings. It first computes an average $\theta^{b,t} = f_{\text{avg}}(\theta^t_{c\in[C]})$ of the benign updates, $\theta^t_{c\in[C]}$, available to the adversary in FL round t. Then it perturbs $\theta^{b,t}$ in a *dynamic, data-dependent malicious direction* ω to compute the final poisoned update $\theta^{t,m}_{c\in[C]} = \theta^{b,t} + \gamma\omega$. The attack, called *DYN-OPT*, finds the largest γ that successfully circumvents the target AGR. DYN-OPT is much stronger than its predecessors, because it finds the largest γ and uses a dataset tailored ω. Below, we

detail DYN-OPT attacks against the AGRs from Section 2.2 that we consider in this work.

2.3.2.1 FedAVG

DYN-OPT attack against FedAVG is quite straightforward and uses a random direction ω and a very large γ value to compute poisoned update $\theta^{t,m}_{c\in[C]}$.

2.3.2.2 Mutli-Krum

As described in Section 2.2.3.1, Multi-Krum uses Krum iteratively to construct a selection set S and computes the average of the updates in the selection set as its aggregate. Hence, DYN-OPT aims to maximize the perturbation $\gamma\omega$ used to compute the poisoned update $\theta^{t,m}_{c\in[C]}$, while ensuring that Multi-Krum selects all of its poisoned updates in S. Note that this strategy minimizes the number of benign updates in S and maximizing $\gamma\omega$ increases the poisoning impact of malicious updates on the final aggregate. The optimization problem we solve to mount DYN-OPT on Multi-Krum is given in (2.2).

$$\begin{aligned} \underset{\gamma}{\text{argmax}} \quad & |\{\theta^{t,m}_{c\in[C]} \in f_{\text{mkrum}}\left(\theta^{t,m}_{c\in[C]} \cup \theta^{t}_{i\in[C+1,n]}\right)\}| \\ \text{s.t.} \quad & \theta^{t,m}_{c\in[C]} = \theta^{b,t} + \gamma\omega \end{aligned} \tag{2.2}$$

2.3.2.3 Trimmed-Mean and Median

For Trimmed-Mean and Median AGRs, DYN-OPT solves the optimization given in (2.3). Following [148], we fix the perturbation ω and keep all the poisoned updates the same. The objective here is to maximize the L_2-norm of the distance between the reference benign update $\theta^{b,t}$ and the aggregate, $f_{\text{agr}}(.)$, computed using $f_{\text{agr}} \in \{f_{\text{trmean}}, f_{\text{median}}\}$ on the set of benign and malicious updates.

$$\underset{\gamma}{\text{argmax}} \ \| \theta^{b,t} - f_{\text{agr}} \left(\theta^{t,m}_{c \in [C]} \cup \theta^{t}_{i \in [C+1,n]} \right) \|_2 \tag{2.3}$$

$$\text{s.t.} \quad \theta^{t,m}_{c \in [C]} = \theta^{b,t} + \gamma \omega$$

2.3.2.4 Norm-Bounding

We formulate DYN-OPT attack against Norm-Bounding AGR using the original framework proposed in [148]. More specifically, to circumvent Norm-Bounding, the norm of the poisoned update should be less than the threshold norm, τ, used by Norm-Bounding AGR. Hence, to compute poisoned update $\theta^{t,m}_{c \in [C]}$ using DYN-OPT, we can scale the norm of the original poisoned update, $\theta^{b,t} + \gamma\omega$, to τ. The final poisoned update would be $\theta^{t,m}_{c \in [C]} = \text{Scale}(\theta^{b,t} + \gamma\omega, \tau)$, where $\text{Scale}(u, \tau) = u \cdot \min(1, \frac{\tau}{\|u\|_2})$.

2.3.3 Our hybrid adversary model

Compromising real clients in FL in order to launch a model poisoning attack can be a challenging task for an attacker. This is because genuine clients participating in FL are typically owned and controlled by different entities (e.g., individual users in cross-device FL and hospitals in cross-silo FL), and the attacker should get access to and take control of these clients in order to manipulate the updates they send to the server.

One way an attacker might try to do this is by using malware or phishing attacks to compromise the clients. However, successfully executing these types of attacks requires a certain level of skill and resources, and the attacker would need to be able to bypass any security measures that the clients have in place. Additionally, the cost of compromising a large number of genuine clients can be high, as the attacker would need to pay for access to undetected zombie devices or other resources. This may make it infeasible for the attacker to compromise a large fraction of genuine clients, which is typically necessary for a model poisoning attack to be successful.

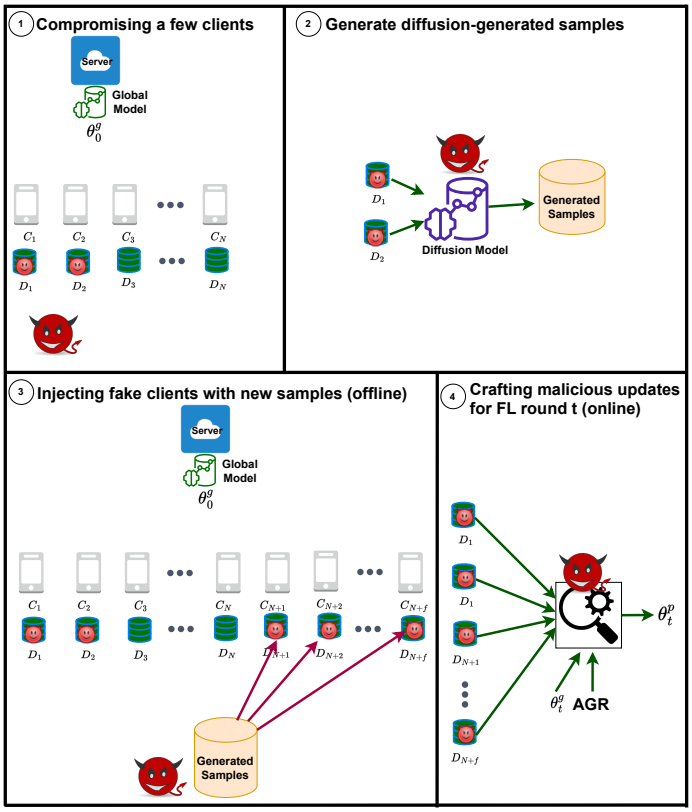


Figure 2.2. Our novel hybrid attack pipeline: The adversary of hybrid attack lies in the middle of the spectrum of FL poisoning adversaries. The hybrid attack adversary compromises a few real FL clients, trains a denoising diffusion probability model (DDPM) on their real data, and generates new synthetic data to solve an optimization to generate malicious updates to mount strong model poisoning attacks against the target robust aggregation rules. Finally, the adversary shares the malicious update with the FL server via the compromised clients as well as (cheap to inject) fake clients.

Another factor that makes it difficult to compromise real clients in FL is the decentralized nature of the system. In FL, the clients are typically distributed across a wide geographical area and may have different levels of security and defenses in place. This can make it difficult for the attacker to gain access to and take control of a large number of clients simultaneously.

Overall, the combination of the technical challenges and the high cost of compromising genuine clients in FL makes it a difficult task for an attacker to launch a successful model poisoning attack using just the compromised clients.

Instead, we propose to use both fake and compromised clients to mount a hybrid attack. Figure 2.2 demonstrates the pipeline of our hybrid attack: The hybrid adversary first compromises a few real clients, and then uses their data to generate synthetic data by using a DDPM (Section 2.1.1). Next, the adversary uses this synthetic data to emulate FL clients and uses the model poisoning attacks (Section 2.3.2) to craft strong malicious updates. The injected fake clients and compromised clients submit the generated malicious update if the server selects them in that FL round for their local updates.

2.3.4 Comparing the costs of different attacks

In this section, we discuss the cost of the three types of attacks discussed above: fake, hybrid, and compromised. We assume that the cost of compromising a client is c and cost of creating a fake client is f; depending on the scenario, c and f can vary widely, but generally the cost of a fake client is much lower than that of a compromised client, i.e., $f \ll c$. Furthermore, we assume α_f fake clients in the fake attack, β_c compromised clients in the compromised attack, and α_h fake and β_h compromised clients in the hybrid attack.

If the number of malicious clients in the three attacks are the same, i.e., $\alpha_f = \alpha_h + \beta_c = \beta_c$, the cost of each of the attacks is as follows: $f \cdot \alpha_f$ for the fake attack, $f \cdot \alpha_h + c \cdot \beta_h$ for the hybrid attack, and $c \cdot \beta_c$ for the compromised attack. Next, note that in our hybrid attack, we use very few compromised clients to launch a very large number of fake clients, i.e., $\alpha_h \gg \beta_h$, which also implies that the number of fake clients in our hybrid attack is very close to that in fake attack, i.e., $\alpha_h \approx \alpha_f$. Hence, the order of the cost of the three attacks is: $\text{cost}_f < \text{cost}_h \ll \text{cost}_c$, with cost_f and cost_h being very close.

Let's consider a concrete scenario involving IoT devices, e.g., CCTV traffic cameras or WiFi routers. The goal of the adversary is to mount a model poisoning attack

against an IoT application, e.g., predicting traffic at a certain location. The application stores and uses images from traffic cameras, and trains a global image classification model using FL. With a high probability, these IoT devices are also a part of some botnet, and the cost of owning such zombie devices in a botnet can be as low as \$1. However, all the IoT devices need not have the target application, e.g., many CCTV cameras may not have required software/hardware updates. For concreteness, consider that 1% of the devices have the target application. Furthermore, note that generally the botnet owners do not know what all applications are running on the zombie devices.

Hence, in case of the compromised attack requiring m malicious clients, where the zombie IoT devices must have the application, adversary will have to buy $100m$ devices to ensure that m of them have the target application and discard $99m$ devices. While, in the case of our hybrid attack, the adversary just needs to ensure that $m' \ll m$ devices have the application (and therefore, required data) and should buy $100m'$ devices. Then they can install the target application on $m - m'$ devices and populate them with synthetic data. In the case of fake attack, adversary simply has to buy m devices. If the cost of buying a zombie device is c, the costs of compromised, hybrid and fake attacks are $100mc \gg 100m'c > mc$; the first inequality holds because $m \gg m'$.

2.4 Experiment setup

In this section, we first introduce the evaluation datasets, the model architectures, and the hyperparameter settings. Next, we define our evaluation metrics, and finally, we explain how we generate new data samples using DDPM.

2.4.1 Datasets and hyperparameters

In this work, we conduct experiments on two datasets, CIFAR10 [104] and FEMNIST [33, 46], in order to evaluate the performance of different Byzantine robust aggregation under different threat models.

CIFAR10 dataset is a widely used image classification dataset consisting of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. For this dataset, we use VGG9 architecture. For local training in each FL round, each client uses 5 epochs. Each client uses SGD with learning rate of 0.01, momentum of 0.9, weight decay of 1e-4, and batch size of 8.

FEMNIST is a character recognition classification task with 3,400 clients, 62 classes (52 for upper and lower case letters and 10 for digits), and 671,585 gray-scale images. Each client has data of their own handwritten digits or letters. For this dataset, we use LeNet architecture. For local training in each FL round, each client uses 2 epochs. Each client uses SGD with learning rate of 0.01, momentum of 0.9, weight decay of 1e-4, and batch size of 10.

Data distribution: Most real-world FL settings have heterogeneous client data, hence following previous works [145, 86], we distribute CIFAR10 datasets among 1,000 clients in non-iid fashion using *Dirichlet* distribution with parameter $\beta = 0.5$. Note that, increasing β results in more iid datasets. FEMNIST is naturally distributed non-iid among 3,400 clients.

2.4.2 Evaluation metric

We run all the experiments for 2000 global rounds of FL for CIFAR10, and 1000 global rounds for FEMNIST, while selecting 25 clients in each round randomly. At the end of each FL round, we calculate the test accuracy of the global model on the test data, and update the maximum test accuracy. We run each experiment with

5 different random seeds, and we report the median and standard deviation of the maximum test accuracies in our experiments.

Attack impact metric (I_θ): We define attack impact, $I_\theta = A_\theta - A_\theta^M$, as the reduction of the accuracy of the global model when the attack is launched. A_θ denotes the maximum accuracy that the global model achieves overall FL training rounds without the presence of any malicious clients. A_θ^M for an attack shows the maximum accuracy of the model under a given attack. In our Tables, we report both the maximum test accuracies and Attack Impacts.

2.4.3 Generating synthetic data using DDPM

In Section 2.3.3, we explained the pipeline of our hybrid attack, which takes control of a few real clients and generates new synthetic data. In this section, we explain the details of this process for images of CIFAR10 and FEMNIST. To generate new samples, we use the following steps (similar to steps provided in Figure 2.2):

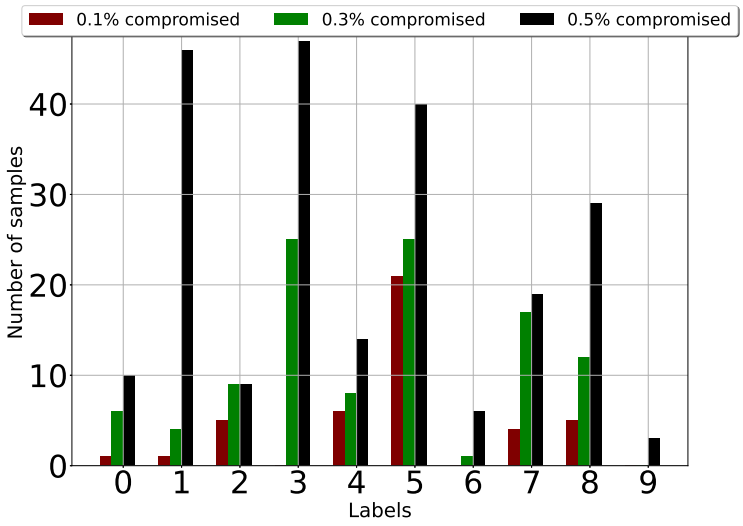


Figure 2.3. Number of samples for each label when the attacker compromised 0.1% (1 client), 0.3% (3 clients), and 0.5% (5 clients) in our data distribution (fixed through all the experiments) for learning CIFAR10 distributed over 1000 clients.

Collecting the data of compromised clients. We collect all the data samples of 0.1%, 0.3% and 0.5% of first clients in both CIFAR10 and FEMNIST learning. For CIFAR10, we distribute the data in a non-iid fashion using Dirichlet distribution with parameter $\beta = 0.5$. We saved the data assignment of the dataset and used this fixed distribution throughout our experiments. For CIFAR10, we collect the data samples of the first 1 (0.1%), 3 (0.3%), and 5 (0.5%) of the clients. Figure 2.3 shows the number of samples for each label (label 0 represents airplane images, label 1 represents car images, etc.) for our data collection. As we can see from this figure, when the attacker has only compromised 0.1% of clients, it does not have access to any data samples of labels 3, 6, and 9. This means it cannot produce any new samples for these labels. For compromising 0.3%, the adversary does not have access to any samples from label 9. For FEMNIST, we also used the same generated data assignment (produce non-iid), and we collected the data samples of the first 4 (0.1%), 7 (0.3%), and 11 (0.5%) of the clients.

Generating new samples using DDPM We use the code provided in [5] to generate new samples for the hybrid attacks. This code implemented the denoising diffusion probabilistic model (DDPM) [85] in PyTorch. It is a transcribed code from the official Tensorflow version [4]. It uses denoising score matching to estimate the gradient of the data distribution, followed by Langevin sampling to sample from the true distribution. After collecting the data samples of compromised clients, we ran the DDPM on these images for each label separately to generate new samples. To train the diffusion model, we used a batch size of 8, learning rate of 0.00008, and 250 sampling size. To generate samples for CIFAR10, we used 2000 diffusion steps, and for FEMNIST we used 1000 diffusion steps.

Figure 2.4 shows some DDPM-generated samples when the adversary has compromised 1 (0.1%), 3 (0.3%), and 5 (0.5%) of the clients in learning of CIFAR10 distributed over 1000 clients. Figure 2.3 shows the number of samples for each label.

(a) 0.1% (1 client) compromised

(b) 0.3% (3 clients) compromised

(c) 0.5% (5 clients) compromised

Figure 2.4. Airplanes generated by DDPM using different percentages of compromised client's data in our hybrid attack.

From this Figure, we can see the adversary has access to 1, 6, and 10 images of airplanes by compromising 0.1%, 0.3%, and 0.5% of the clients, respectively. In Figure 2.4(a), we can see that the DDPM model memorized the only image it has, and it just tried to add randomness to it because it has access to only one image of an airplane. Moreover, in Figure 2.4(b) and Figure 2.4(c), we can see that the model can generate better samples as it has access to more images from the true distribution.

Data assignment for the injected fake clients. In all the hybrid attacks experiments, we first create a large dataset of all synthetic images from all the labels. We create this dataset by generating 5 samples per label multiplied by the number of injected fake clients. Then we distributed this dataset over the fake clients in a non-iid fashion using Dirichlet distribution with parameter $\beta = 0.5$ for both CIFAR10 and FEMNIST experiments. Next, for launching the model poisoning attacks provided in Section 2.3.2, the adversary chooses 25 random fake clients for its optimization and creates its malicious updates. This process happens in each FL round based on the global parameters θ^t.

2.5 Empirical results

In this section, we conduct experiments to evaluate the performance of different Byzantine robust aggregation rules under different adversaries, using the FEMNIST [33, 46] and CIFAR10 [104] datasets. We consider a range of malicious client percentages, including 5%, 10%, 20%, and 30%, and report the maximum test accuracy and the impact of various attacks on the global model. For each attack, we also report the number of benign, compromised, and injected fake clients present in the FL training process.

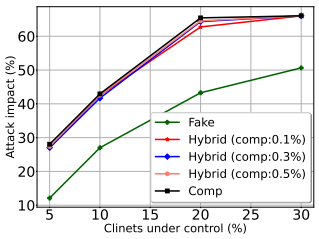
We consider five different attack scenarios, ranging from injecting fake clients with no knowledge of the true data distribution to a scenario where the adversary can compromise benign clients and use their data to craft malicious updates. Additionally, we propose and evaluate three types of hybrid attacks, where the adversary first compromises a small number of real clients and then uses their data to generate synthetic samples using a DDPM, followed by injecting fake clients with the new data samples. We explore the impact of different numbers of compromised clients in these hybrid attacks, specifically 0.5% (5 clients), 0.3% (3 clients), and 0.1% (1 client) in CIFAR10 experiments and 0.5% (17 clients), 0.3% (11 clients), and 0.1% (4 clients) in FEMNIST experiments. We rank the attacks in terms of their impact on the global model accuracy, to better illustrate the spectrum of attacks.

It is worth noting that we omit the results of the standard aggregation rule, FedAvg, as it is known to be vulnerable to even a single malicious client [27] and can result in the global test accuracy approaching random guessing.

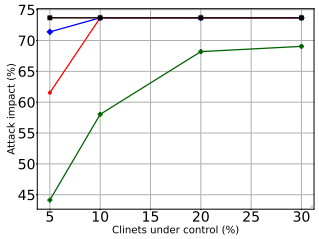
2.5.1 Attacking agnostic robust AGRs

In this section, we evaluate the performance of agnostic robust AGRs, specifically Median and Norm-Bounding, under different threat models. We specifically

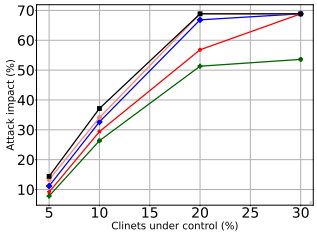
focus on a spectrum of adversaries who control different percentages of malicious clients.



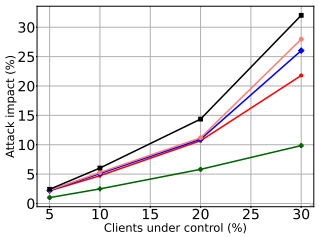
(a) CIFAR10 + Median (No attack acc=76.05%)



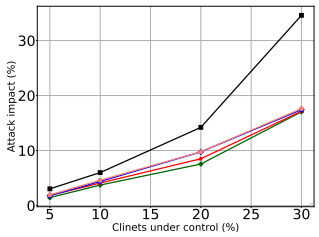
(b) CIFAR10 + NB ($\tau = 2.0$) (No attack acc=83.68%)



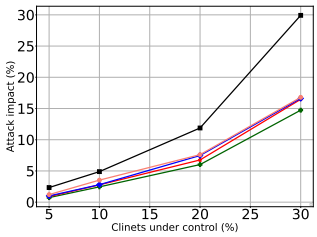
(c) CIFAR10 + NB $\tau = 0.5$ (No attack acc=78.86%)



(d) FEMNIST + Median (No attack acc=84.29%)



(e) FEMNIST + NB $\tau = 2.0$ (No attack acc=87.49%)



(f) FEMNIST + NB $\tau = 0.5$ (No attack acc=86.35%)

Figure 2.5. Attack impact (I_θ) of the Norm-Bounding and Median aggregation rules in the presence of different adversaries. For hybrid attacks, we explore the impact of different numbers of compromised clients, specifically 0.5% (5 clients), 0.3% (3 clients), and 0.1% (1 client) in CIFAR10 experiments and 0.5% (17 clients), 0.3% (11 clients), and 0.1% (4 clients) in FEMNIST experiments.

Median AGR. We present the results of our experiments when the server applies Median as the aggregation rule in Figure 2.5 (a) and (d) for CIFAR10 and FEMNIST experiments. Our results demonstrate that the most powerful adversary, who has compromised real clients, has the greatest impact on the global model. For example, on CIFAR10 with Median as the AGR, an attack launched by 10% (20%) of malicious clients reduces the model's accuracy to 33.10% (10.61%). This implies that the attacker first compromised 100 (200) clients out of the total clients participating in the FL and launched the attack discussed in Section 2.3.2 to craft its malicious update.

On the other hand, fake clients, who do not have any knowledge about the benign clients' data distribution, have the least impact on the global model. For example, on CIFAR10 with Median as the AGR, an attack launched by 10% (20%) of malicious clients reduces the accuracy of the global model to 49.04% (32.78%). To achieve this, the adversary needs to inject 112 (251) fake clients into the training of the FL.

Hybrid attacks, which lie in the middle of the spectrum, show that if the hybrid adversary has access to more data (more compromised clients), it can impose more significant damage to the global model's accuracy. For example, on CIFAR10 with Median as the AGR, a hybrid attack with 20% malicious clients, where the adversary has compromised 1, 3, and 5 clients and generated new instances and injected 249, 247, and 244 new fake clients can decrease the accuracy of the FL model to 13.29%, 11.71%, and 11.49% respectively. We make similar observations for the FEMNIST dataset as well.

Norm-Bounding AGR. We report the experimental results of our experiments when the server applies Norm-Bounding with a threshold τ as the aggregation rule in Figure 2.5 (b), (c), (e), and (f) for CIFAR10 and FEMNIST datasets with two thresholds $\tau = 0.5$ and $\tau = 2.0$. Our results show that the Norm-Bounding aggregation rule has similar impacts on the global model's accuracy as the Median AGR, when faced with different types of attacks. For example, on CIFAR10 with $\tau = 0.5$, when the adversary controls 10% of clients, the fake adversary can inject 112 fake clients and reduce the accuracy to 52.52%; the hybrid attack who compromised 1 client and injected 110 clients reduces the accuracy to 49.46%; the hybrid attacker who compromised 3 clients and injected 108 fake clients reduces the accuracy to 46.22%; the hybrid attacker who compromised 5 clients and injected 106 clients reduces the accuracy to 44.79%; and at the end of the spectrum, a powerful adversary who compromised 100 clients can

reduce the accuracy to 41.73%.

Larger upper bounds in Norm-Bounding results in more damage to the global model. In our experiments, we consider two thresholds for Norm-Bounding $\tau = 0.5$ and $\tau = 2.0$. Our results show that for a larger threshold bound (τ), the adversary has a larger space to craft its malicious updates and have a more significant impact on the FL global model. For instance, on FEMNIST, the compromising adversary with 30% malicious ratio causes the accuracy dropped by 34.58% when $\tau = 2.0$ while the accuracy drop for the same setting and $\tau = 0.5$ is about 29.92%.

Therefore, with larger norm thresholds for the Norm-Bounding aggregation rule, the attackers have more impact on the global model. Alternatively, If the server wants to use a smaller threshold, then the model will result in lower accuracy when there is no malicious client. For instance, on CIFAR10, with no malicious clients, Norm-Bounding with threshold $\tau = 0.5$ results in 78.86% while $\tau = 2.0$ results in 83.68%; 10% compromised clients will result in losing of 37.13% and 73.68% for $\tau = 0.5$ and $\tau = 2.0$ respectively. Therefore, there is a trade-off for choosing a proper threshold for bounding the local updates based on the assumption of the number of malicious clients in FL training.

Why can fake clients cause a significant attack impact for Norm-Bounding AGR? Figure 2.6 shows the L2 norm of the updates (for malicious and benign updates for 10% of malicious ratio in fake attack) before and after bounding the updates to $\tau = 0.5$ for learning CIFAR10 throughout 2000 FL rounds. From this figure, we can see that when the global model starts to converge, the L2 norm of the local updates from benign updates becomes smaller than the threshold. For the updates that have norms smaller than the threshold, no change will be applied to them. However, on the other hand, the malicious updates are always greater than the threshold, so they are scaled down to have an L2 norm of τ. In this figure, we can see

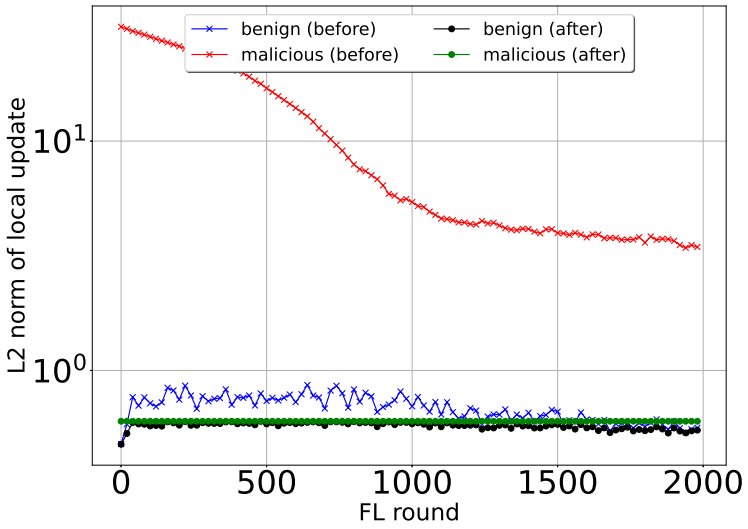


Figure 2.6. Local update norms throughout the FL training on CIFAR10 with 1000 benign clients and 112 fake clients (i.e., the adversary controls 10% of total clients). In this figure, we can see that after FL round 1500, the malicious updates have a more considerable impact on the aggregation compared to benign updates because they have larger updates after norm bounding.

that after FL round 1500, the malicious updates have a more considerable impact on the aggregation because they have larger updates.

2.5.2 Attacking adaptive robust AGRs

In this section, we conduct experiments to evaluate the robustness of adaptive Byzantine aggregation rules, specifically Trimmed-Mean [169] and Multi-Krum [27], against a spectrum of adversaries who control varying percentages of malicious clients. In adaptive aggregation rules, we assume that the server has knowledge of the exact percentage of malicious clients in each FL round.

We report the performance of the Trimmed-Mean aggregation rule against different attacks in Table 2.1 and Table 2.3 for FL models trained on the CIFAR10 and FEMNIST datasets, respectively, in the presence of 5%, 10%, 20%, and 30% of malicious clients. Similarly, Table 2.2 and Table 2.4 show the attack impacts of

Table 2.1. Attack impact (I_θ) and maximum test accuracy (A_θ^M) of the Trimmed-Mean for training on CIFAR10 distributed over 1000 initial clients in the presence of different adversaries. We specify the number of benign, compromised, and injected fake clients present for each attack. The malicious rate column shows the ratio of the clients (both compromised and fake) under the control of the adversary to the total FL clients participating in FL training. We rank the attacks in terms of their impact on the global model accuracy, to better illustrate the spectrum of attacks.

AGR	Attack Type	Mal Rate	Num of Benign Clients	Num of Compromised Clients	Num of Injected Fake Clients	Accuracy (A_θ^M)	Attack Impact (I_θ)
Trimmed-Mean (No attack acc = 83.66%)	Fake	5%	1000	0	53	59.95 (± 0.617)	23.71 (± 0.617) 5
		10%	1000	0	112	43.88 (± 0.334)	39.78 (± 0.334) 5
		20%	1000	0	251	32.49 (± 0.451)	51.17 (± 0.451) 5
		30%	1000	0	429	25.56 (± 0.238)	58.10 (± 0.238) 5
	Hybrid comp: 0.1%	5%	999	1	52	50.19 (± 2.791)	33.47 (± 2.791) 4
		10%	999	1	110	29.42 (± 1.481)	54.24 (± 1.481) 4
		20%	999	1	249	20.61 (± 5.277)	63.05 (± 5.277) 4
		30%	999	1	428	10.00 (± 1.188)	73.66 (± 1.188) 4
	Hybrid comp: 0.3%	5%	997	3	50	47.78 (± 0.928)	35.88 (± 0.928) 3
		10%	997	3	108	28.56 (± 1.071)	55.10 (± 1.071) 3
		20%	997	3	247	20.50 (± 5.415)	63.16 (± 5.415) 3
		30%	997	3	425	10.01 (± 0.209)	73.65 (± 0.209) 3
	Hybrid comp: 0.5%	5%	995	5	48	41.90 (± 3.438)	41.76 (± 3.438) 1
		10%	995	5	106	27.89 (± 0.909)	55.77 (± 0.909) 2
		20%	995	5	244	20.31 (± 5.151)	63.35 (± 5.151) 2
		30%	995	5	422	10.00 (± 0.180)	73.66 (± 0.180) 2
	Comp	5%	950	50	0	44.25 (± 1.195)	39.41 (± 1.195) 2
		10%	900	100	0	27.33 (± 0.346)	55.83 (± 0.346) 1
		20%	800	200	0	10.00 (± 4.130)	73.66 (± 4.130) 1
		30%	700	300	0	10.00 (± 0.000))	73.66 (± 0.000) 1

different attacks when the server uses Multi-Krum as the aggregation rule for the CIFAR10 and FEMNIST datasets, respectively.

Our results indicate that adversaries who can compromise clients and use their data for attacks have the most significant impact on FL global models. For instance, on the CIFAR10 dataset, an adversary who has compromised 10% (20%) of clients reduces the accuracy of FL by 55.83% (73.66%) and 49.29% (60.37%) with Trimmed-Mean and Multi-Krum, respectively. On the other hand, adversaries who can only inject fake clients into the FL training with no knowledge of the true data distribution have the lowest impact on global model accuracy. For instance, on the CIFAR10 dataset, an

Table 2.2. Attack impact (I_θ) and maximum test accuracy (A_θ^M) of the Multi-Krum for training on CIFAR10 distributed over 1000 initial clients in the presence of different adversaries.

AGR	Attack Type	Mal Rate	Num of Benign Clients	Num of Compromised Clients	Num of Injected Fake Clients	Accuracy (A_θ^M)	Attack Impact (I_θ)
Multi-Krum (No attack acc = 83.44%)	Fake	5%	1000	0	53	82.70 (± 0.291)	0.74 (± 0.291) 5
		10%	1000	0	112	82.12 (± 0.227)	1.32 (± 0.227) 5
		20%	1000	0	251	79.89 (± 0.226)	3.55 (± 0.226) 5
		30%	1000	0	429	75.29 (± 0.256)	8.15 (± 0.256) 5
	Hybrid comp: 0.1%	5%	999	1	52	70.12 (± 0.895)	13.32 (± 0.895) 4
		10%	999	1	110	48.24 (± 2.371)	35.20 (± 2.371) 4
		20%	999	1	249	24.71 (± 0.257)	58.73 (± 0.257) 4
		30%	999	1	428	20.22 (± 0.539)	63.22 (± 0.539) 4
	Hybrid comp: 0.3%	5%	997	3	50	62.65 (± 0.725)	20.79 (± 0.725) 3
		10%	997	3	108	36.70 (± 2.188)	46.74 (± 2.188) 3
		20%	997	3	247	23.79 (± 1.788)	59.65 (± 1.788) 3
		30%	997	3	425	19.90 (± 2.234)	63.54 (± 2.234) 3
	Hybrid comp: 0.5%	5%	995	5	48	62.47 (± 0.914)	20.97 (± 0.914) 2
		10%	995	5	106	35.65 (± 0.956)	47.79 (± 0.956) 2
		20%	995	5	244	23.10 (± 1.433)	60.34 (± 1.433) 2
		30%	995	5	422	19.86 (± 0.619)	63.58 (± 0.619) 2
	Comp	5%	950	50	0	62.04 (± 1.307)	21.40 (± 1.307) 1
		10%	900	100	0	34.15 (± 0.660)	49.29 (± 0.660) 1
		20%	800	200	0	23.07 (± 0.528)	60.37 (± 0.528) 1
		30%	700	300	0	19.31 (± 0.786)	64.13 (± 0.786) 1

adversary who can inject 10% (20%) of clients reduces the accuracy of FL by 39.78% (51.17%) and 1.32% (3.55%) with Trimmed-Mean and Multi-Krum, respectively.

Our experiments also show that the hybrid attack, which compromises only a few clients and use their data to produce more data samples for the fake clients, lies in the middle of the spectrum. The more clients are compromised, the more damage is done to the global accuracy. For instance, on the CIFARA10 training, a hybrid attacker who compromised 1 client, i.e., 0.1% of total clients, and can inject 110 clients (in total 10% malicious ratio) can reduce the accuracy of the FL model by 54.24% and 35.2% for Trimmed-Mean and Multi-Krum respectively. While if the hybrid attacker compromised more clients (5 clients) and injected 106 clients (in total 10% malicious ratio), it can reduce the FL global accuracy by 55.77% and 47.79% for Trimmed-Mean and Multi-Krum, respectively.

Table 2.3. Attack impact (I_θ) and maximum test accuracy (A_θ^M) of the Trimmed-Mean for training on FEMNIST distributed over 3400 initial clients in the presence of different adversaries.

AGR	Attack Type	Mal Rate	Num of Benign Clients	Num of Compromised Clients	Num of Injected Fake Clients	Accuracy (A_θ^M)	Attack Impact (I_θ)
Trimmed-Mean (No attack acc = 87.52%)	Fake	5%	3400	0	179	84.90 (± 0.108)	2.62 (± 0.108) 5
		10%	3400	0	378	82.64 (± 0.135)	4.88 (± 0.135) 5
		20%	3400	0	850	78.04 (± 0.198)	9.48 (± 0.198) 5
		30%	3400	0	1458	73.11 (± 0.384)	14.41 (± 0.384) 5
	Hybrid comp: 0.1%	5%	3396	4	175	84.04 (± 0.223)	3.48 (± 0.223) 4
		10%	3396	4	374	80.44 (± 0.672)	7.08 (± 0.672) 4
		20%	3396	4	845	72.09 (± 1.114)	15.43 (± 1.114) 4
		30%	3396	4	1452	58.28 (± 0.699)	29.24 (± 0.699) 4
	Hybrid comp: 0.3%	5%	3389	11	168	83.95 (± 0.151)	3.57 (± 0.151) 3
		10%	3389	11	366	79.38 (± 0.313)	8.14 (± 0.313) 2
		20%	3389	11	837	70.48 (± 0.815)	17.07 (± 0.815) 3
		30%	3389	11	1442	57.15 (± 1.953)	30.37 (± 1.953) 3
	Hybrid comp: 0.5%	5%	3383	17	162	83.73 (± 0.248)	3.79 (± 0.248) 2
		10%	3383	17	359	79.75 (± 0.659)	7.77 (± 0.659) 3
		20%	3383	17	829	70.33 (± 2.009)	17.19 (± 2.009) 2
		30%	3383	17	1433	54.20 (± 2.420)	33.32 (± 2.420) 2
	Comp	5%	3230	170	0	83.51 (± 0.183)	4.01 (± 0.183) 1
		10%	3060	340	0	78.71 (± 0.498)	8.81 (± 0.498) 1
		20%	2720	680	0	68.13 (± 2.040)	19.39 (± 2.040) 1
		30%	2380	1020	0	40.35 (± 2.275)	47.17 (± 2.275) 1

Additionally, we also noticed that the Trimmed-Mean and Norm-Bounding (with $\tau = 0.5$) are more vulnerable to injected fake clients with no knowledge about the true distribution of the training datasets. On the other hand, Multi-Krum can easily detect them and exclude them from aggregation. For instance, on CIFAR10, 10% of injected fake clients can reduce the accuracy of the model by 26.34% and 39.78% with Norm-Bounding and Trimmed-Mean as the aggregation rule, respectively. On the other hand, Multi-Krum only loses 1.32% with the presence of this number of injected fake clients.

2.5.3 Data poisoning with label flipped data samples

In this study, we evaluate the effectiveness of a data poisoning attack in which the adversary only injects poisoned data (by label flipping) into the benign client's datasets. We used the FedAvg algorithm and reported the results in terms of the

Table 2.4. Attack impact (I_θ) and maximum test accuracy (A_θ^M) of the Multi-Krum for training on FEMNIST distributed over 3400 initial clients in the presence of different adversaries.

AGR	Attack Type	Mal Rate	Num of Benign Clients	Num of Compromised Clients	Num of Injected Fake Clients	Accuracy (A_θ^M)	Attack Impact (I_θ)
Multi-Krum (No attack acc = 87.45%)	Fake	5%	3400	0	179	87.25 (± 0.064)	0.20 (± 0.064) 5
		10%	3400	0	378	87.11 (± 0.066)	0.34 (± 0.066) 5
		20%	3400	0	850	86.58 (± 0.178)	0.87 (± 0.178) 5
		30%	3400	0	1458	85.60 (± 0.174)	1.85 (± 0.174) 5
	Hybrid comp: 0.1%	5%	3396	4	175	86.02 (± 0.176)	1.43 (± 0.176) 3
		10%	3396	4	374	82.82 (± 0.352)	4.63 (± 0.352) 4
		20%	3396	4	845	75.88 (± 0.635)	11.57 (± 0.635) 4
		30%	3396	4	1452	62.23 (± 1.825)	25.22 (± 1.825) 3
	Hybrid comp: 0.3%	5%	3389	11	168	86.26 (± 0.106)	1.19 (± 0.106) 4
		10%	3389	11	366	81.58 (± 0.223)	5.87 (± 0.223) 1
		20%	3389	11	837	73.97 (± 0.582)	13.48 (± 0.582) 3
		30%	3389	11	1442	62.35 (± 0.859)	25.10 (± 0.859) 4
	Hybrid comp: 0.5%	5%	3383	17	162	85.87 (± 0.126)	1.98 (± 0.126) 2
		10%	3383	17	359	82.03 (± 0.376)	5.42 (± 0.376) 3
		20%	3383	17	829	71.71 (± 2.148)	15.74 (± 2.148) 2
		30%	3383	17	1433	61.94 (± 1.990)	25.51 (± 1.990) 2
	Comp	5%	3230	170	0	85.46 (± 0.113)	1.99 (± 0.113) 1
		10%	3060	340	0	81.73 (± 0.390)	5.72 (± 0.390) 2
		20%	2720	680	0	69.39 (± 1.597)	18.06(± 1.597) 1
		30%	2380	1020	0	47.83 (± 10.627)	39.62 (± 10.627) 1

maximum accuracy (A_θ^M) and attack impact (I_θ) in Table 2.5. The table shows the performance of the FedAvg algorithm in the presence of different percentages of poisoned datasets among the FL clients.

Table 2.5. Data Poisoning attack against FL learning on FEMNIST. The new samples with flipped labels are generated using DDPM and the 0.5% compromised clients' data.

AGR	Mal Rate	Num of Benign Clients	Num of Compromised Clients	Accuracy (A_θ^M)	Attack Impact (I_θ)
Avg (acc = 87.51%)	5%	3213	170	87.37 (± 0.100)	0.14 (± 0.100)
	10%	3044	339	87.15 (± 0.100)	0.36 (± 0.100)
	20%	2706	677	87.02 (± 0.096)	0.49 (± 0.096)
	30%	2368	1015	86.83 (± 0.103)	0.68 (± 0.103)

For these experiments, we used a dataset of 3400 clients in FEMNIST learning, and the attacker generated poisoned data samples from the datasets of 17 compromised

clients (equivalent to 0.5% of the total clients). These 17 compromised clients were excluded from the FL learning process. The label flipping attack [149] used in this experiment involved changing each data sample label from ℓ to $L-1-\ell$, where ℓ is the true label and L is the number of classes.

Our results indicate that injecting label-flipped data samples generated by the DDPM has a smaller impact on the global model's accuracy compared to model poisoning attacks. For example, on the FEMNIST dataset, when the attacker injects synthetic data samples to 20% of the benign clients, using data samples from 17 compromised clients, it reduces the global accuracy by only 0.49%.

2.6 Conclusions

In conclusion, we present a comprehensive study of the poisoning threats to FL by considering a spectrum of adversaries and robust AGRs. We discussed the importance of considering a spectrum of adversarial models, rather than focusing on the extremes, as it provides a more realistic understanding of the poisoning threat to various types of FL applications. We identify a hybrid threat model where an adversary first compromises a few real clients and use their data to generate more data samples for the fake clients to mount a large-scale attack. For such a hybrid threat, we propose a novel hybrid attack that leverages the denoising diffusion probabilistic model (DDPM) to generate new samples from a small number of compromised clients.

Overall, this work highlights the need for FL practitioners to consider a spectrum of adversaries and defending servers to truly understand the poisoning threat to various types of FL applications. Our proposed hybrid attack and the evaluation methodology provide valuable insights for FL practitioners to design and evaluate FL systems under realistic threat models.

CHAPTER 3

ROBUST FEDERATED LEARNING VIA LEARNING ON PARAMETER RANKS

In Chapter 2, a thorough investigation of untargeted poisoning attacks within the context of Federated Learning (FL) was conducted. Our findings demonstrated that the most potent attack in FL arises when genuine clients are compromised, leading to the creation of malicious updates based on their data. In this section, we introduce a novel federated learning algorithm designed to mitigate the impact of such attacks on FL systems.

High-level intuition of FL untargeted poisoning: Figure 3.1 shows how the poisoning adversary searches for malicious updates in the space of possible updates to maximize the distance between benign and malicious aggregates. When the server's AGR is not robust, e.g., dimension-wise average [126], there is no limitation on the adversary's choices, so they can maximize their goal using a malicious update arbitrarily far from benign updates; (Figure 3.1-a)). Therefore, even a single malicious client can jeopardize the accuracy of the global model trained using FedAvg [27]. Current robust AGRs, such as Multi-krum [27] or Trimmed-mean [169] limit the space of acceptable updates, i.e., the safe zone shown in Figure 3.1-b). These robust AGRs only consider the updates that are in the safe zone and thereby reduce the adversary's choices.

In this chapter, we present Federated Rank Learning (FRL), a novel FL algorithm that concurrently achieves the two goals of robustness against poisoning attacks and communication efficiency. FRL uses a novel learning paradigm called *supermasks*

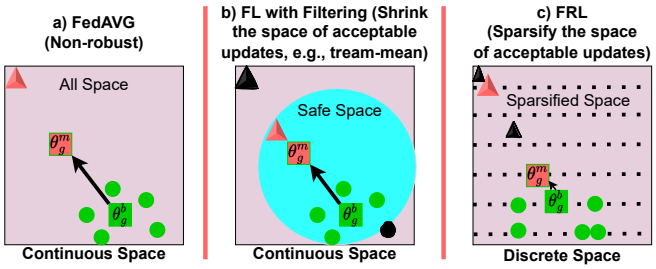


Figure 3.1. The space of client updates. Green circles represent benign updates and red triangles represent malicious updates. To defend against poisoning, existing robust AGRs filter the updates by creating a safe space (continuous $\in \mathbb{R}^d$). On the other hand, FRL limits the choices of clients by enforcing a discrete space of updates (a permutation of integers $\in [1, d]$). θ_g^b (green square) demonstrates the aggregated model for benign users, and θ_g^m (red square) demonstrates the aggregated model considering malicious updates. Black objects are updates that are ruled out by the server.

training [176, 143] to create edge rankings, which, as we will show, allows FRL to reduce communication costs while achieving significantly stronger robustness. Specifically, in FRL, clients collaborate to find a *subnetwork* within a *randomly initialized* neural network which we call the *supernetwork* (this is in contrast to conventional FL where clients collaborate to *train* a neural network). The goal of training in FRL is to collaboratively rank the supernetwork's edges based on the importance of each edge and find a *global ranking*. The global ranking can be converted to a supermask, which is a binary mask of 1's and 0's, that is superimposed on the random neural network (the supernetwork) to obtain the final subnetwork. For example, in our experiments, the final subnetwork is constructed using the top 50% of all edges. The subnetwork is then used for downstream tasks, e.g., image classification, hence it is equivalent to the global model in conventional FL. Note that in entire FRL training, weights of the supernetwork *do not* change.

More specifically, each FRL client computes the importance of the edges of the supernetwork based on their local data. The importance of the edges is represented as a ranking vector. Each FRL client will use the *edge popup* algorithm [143] and their data to compute their local rankings (the edge popup algorithm aims at learning

which edges in a supernetwork are more important over the other edges by minimizing the loss of the subnetwork on their local data). Each client then will send their local edge ranking to the server. Finally, the FRL server uses a novel *voting* mechanism to aggregate client rankings into a global ranking vector, which represents which edges of the random neural network (the supernetwork) will form the global subnetwork.

Continuous versus discrete space of updates: Figure 3.1-c) shows how our proposed defense (FRL, which is introduced next) limits the poisoning adversary's choices of malicious updates by making the space of acceptable updates *discrete*. To the best of our knowledge, most of previous Byzantine robust FL algorithms use a *continuous* space of updates ($\in \mathbb{R}^d$), as their frameworks are built on exchanging trained (32-bit) weight parameters. On the other hand, in our approach, the clients send their updates in the form of *edge rankings*, i.e., a permutation of integers $\in [1, d]$ where d is the size of the network layer; more useful edges have higher ranks. In Figure 3.1-c), the black dots show the discrete space of acceptable client updates. For example, a network with 4 edges can have 4! possible permutations of edge rankings starting from [1,2,3,4] to [4,3,2,1]. On the other hand, in FL algorithms with a continuous space of updates (with or without a safe zone), the adversary's choices are 4 weight parameters (each of 32 bits). Note that, sparsification of the space of acceptable updates is different from sparsification of model updates used in compression methods, e.g., TopK [13]), RandomK [156] and Sketched-SGD [87]. In these methods, the FL client sends only a fraction of model updates instead of all of them, but each parameter still has a continuous space.

3.1 Related works

Supermask learning: Modern neural networks have a very large number of parameters. These networks are generally overparameterized [50, 53, 109, 110], i.e., they have more parameters than they need to perform a particular task, e.g., classification.

The *lottery ticket hypothesis* [65] states that a fully-trained neural network, i.e., *supernetwork*, contains sparse *subnetworks*, i.e., subsets of all neurons in supernetwork, which can be trained from scratch (i.e., by training same initialized weights of the subnetwork) and achieve performances close to the fully trained supernetwork. The lottery ticket hypothesis allows for massive reductions in the sizes of neural networks. Ramanujan et al. [143] offer a complementary conjecture that an overparameterized neural network with randomly initialized weights contains subnetworks which perform as good as the fully trained network.

Communication cost of FL: In many real-world FL applications, it is essential to minimize the communication between FL server and clients. Especially in cross-device FL, the clients (e.g., mobile phones and wearable devices) have limited resources and communication can be a major bottleneck. There are two major types of communication reduction methods: (1) *Quantization* methods reduce the resolution of (i.e., number of bits used to represent) each dimension of a client update. For instance, SignSGD [22] uses the sign (1 bit) of each dimension of model updates. (2) *Sparsification* methods propose to use only a subset of all the update dimensions. For instance, in TopK [13], only the largest K% update dimensions are sent to the server in each FL round. We note that, communication reduction methods primarily focus on and succeed at reducing upload communication (client $\rightarrow$ server), but they use the entire model in download communication (server $\rightarrow$ client).

3.2 Preliminaries

3.2.1 Edge-popup algorithm

The edge-popup (EP) algorithm [143] is an optimization to find supermasks within a large, randomly initialized neural network, i.e., called *supernetwork*, with performances close to the fully trained supernetwork. EP algorithm does not train the weights (θ^w) of the network, instead only decides the set of edges to keep and removes (pops) the

rest of the edges. Specifically, EP algorithm assigns a positive score to each of the edges in the supernetwork (θ^s). On forward pass, it selects top k% edges with highest scores, where k is the percentage of the total number of edges in the supernetwork that will remain in the final subnetwork. On the backward pass, it updates the scores with the straight-through gradient estimator [21].

Algorithm 1 presents EP algorithm. Suppose in a fully connected neural network, there are L layers and layer $\ell \in [1, L]$ has n_ℓ neurons, denoted by $V^\ell = \{V_1^\ell, ..., V_{n_\ell}^\ell\}$. If I_v and Z_v denote the input and output for neuron v respectively, then the input of the node v is the weighted sum of all nodes in previous layer, i.e., $I_v = \sum_{u \in V^{\ell-1}} W_{uv} Z_u$. Here, W_{uv} is the weight of the edge connecting u to v. Edge-popup algorithm tries to find subnetwork E, so the input for neuron v would be: $I_v = \sum_{(u,v) \in E} W_{uv} Z_u$.

Updating scores. Consider an edge E_{uv} that connects two neurons u and v, W_{uv} be the weight of E_{uv}, and s_{uv} be the score assigned to the edge E_{uv} by Edge-popup algorithm. Then the edge-popup algorithm removes edge E_{uv} from the supermask if its score s_{uv} is not high enough. Each iteration of supermask training updates the scores of all edges such that, if having an edge E_{uv} in subnetwork reduces loss (e.g., cross-entropy loss) over training data, the score s_{uv} increases.

Algorithm 1 Edge-popup (EP) algorithm

Input: number of local epochs E, training data D, initial weights θ^w and scores θ^s, subnetwork size k%, learning rate η
for $e \in [E]$ **do**
  $\mathcal{B} \leftarrow$ Split D in B batches
  for batch $b \in [B]$ **do**
    EP FORWARD $(\theta^w, \theta^s, k, b)$
    $\theta^s = \theta^s - \eta \nabla \ell(\theta^s; b)$
  end for
end for
return θ^s
function EP FORWARD$(\theta^w, \theta^s, k, b)$
  $\mathbf{m} \leftarrow \text{sort}(\theta^s)$
  $t \leftarrow \text{int}((1-k) * \text{len}(\mathbf{m}))$
  $\theta^p = \theta^w \odot \mathbf{m}$, where $\mathbf{m}[:t] = 0;\ \mathbf{m}[t:] = 1$
  return $\theta^p(b)$
end function

The algorithm selects top k% edges (i.e., finds a subnetwork with sparsity of k%) with highest scores, so I_v reduces to $I_v = \sum_{u \in V^{\ell-1}} W_{uv} Z_u h(s_{uv})$ where $h(.)$ returns 1 if the edge exists in top-k% highest score edges and 0 otherwise. Because of existence of $h(.)$, which is not differentiable, it is impossible to compute the gradient of loss with respect to s_{uv}. Recall that, the Edge-popup algorithm use straight-through gradient estimator [21] to compute gradients. In this approach, $h(.)$ will be treated as the identity in the backward pass meaning that the upstream gradient (i.e., $\frac{\partial L}{\partial I_v}$) goes straight-through $h(.)$. Now using chain rule, we can derive $\frac{\partial L}{\partial I_v} \frac{\partial I_v}{\partial s_{uv}} = \frac{\partial L}{\partial I_v} W_{uv} Z_u$ where L is the loss to minimize. Then we can SGD with step size η to update scores as $s_{uv} \leftarrow s_{uv} - \eta \frac{\partial L}{\partial I_v} Z_u W_{uv}$.

3.3 Our proposal: Federated Rank Learning

In this section, we provide the design of our *federated rank learning* (FRL) algorithm (Algorithm 2). FRL clients collaborate (without sharing their data) to *find a subnetwork* within a *randomly initialized, untrained supernetwork*, with scores θ^s and weights θ^w. In each round, FRL first finds a unanimous (global) ranking of the supernetwork edges and then uses the subnetwork of the top ranked edges as the global model.

The objective of FRL is to find a global ranking R_g and convert it to a global binary mask, $\mathbf{m}$, such that resulting subnetwork, $\theta^w \odot \mathbf{m}$, minimizes the average loss of all clients. FRL optimization can be formalized as follows:

$$\min_{R_g} F(\theta^w, R_g) = \min_{R_g} \sum_{i=1}^{N} \lambda_i L_i(\theta^w \odot \mathbf{m}) \tag{3.1}$$

$$\text{s.t. } \mathbf{m}[R_g < k] = 0 \text{ and } \mathbf{m}[R_g \geq k] = 1$$

where N is the total number of FRL clients, L_i is the loss function for the i^{th} client, λ_i is the importance, e.g., weight, of the i^{th} client; we use $\lambda_i = \frac{1}{N}$, i.e., all clients

Algorithm 2 Federated Ranking Learning (FRL)

Input: number of rounds T, number of local epochs E, number of users per round n, seed SEED, learning rate η, subnetwork size $k\%$
Server: Initialization
$\theta^s, \theta^w \leftarrow$ Initialize random scores and weights using SEED
$R_g^1 \leftarrow \text{ARGSORT}(\theta^s)$ ▷ Sort the initial scores and obtain initial rankings
for $t \in [1, T]$ **do**
  $U \leftarrow$ set of n randomly selected clients out of N total clients
  for u in U **do**
    Clients: Calculating the ranks
    $\theta^s, \theta^w \leftarrow$ Initialize scores and weights using SEED
    $\theta^s[R_g^t] \leftarrow \text{SORT}(\theta^s)$ ▷ sort the scores based on the global ranking
    $S \leftarrow \text{Edge-PopUp}(E, D_u^{tr}, \theta^w, \theta^s, k, \eta)$ ▷ Client u uses Algorithm1 to train a supermask on its local training data
    $R_u^t \leftarrow \text{ARGSORT}(S)$ ▷ Ranking of the client
  end for
  Server: Majority Vote
  $R_g^{t+1} \leftarrow \text{VOTE}(R_{\{u \in U\}}^t)$ ▷ Majority vote aggregation
end for
function $\text{VOTE}(R_{\{u \in U\}})$:
  $V \leftarrow \text{SUM}(\text{ARGSORT}(R_{\{u \in U\}}))$, $A \leftarrow \text{SUM}(V)$
  return $\text{ARGSORT}(A)$
end function

have the same weight. $\mathbf{m}$ is the final binary mask, where edges with top k ranks (layer-wise) get '1' while others get '0'. We use $\mathbf{m}$ to compute final global model by superimposing $\mathbf{m}$ on θ, i.e., the we use the subnetwork $\theta \odot \mathbf{m}$ as the final global model. In Section 3.6, we show that probability of encountering a disconnected subnetwork is very negligibly small, and also demonstrate what happens if the subnetwork becomes disconnected for a small sparsity k. In Figure 3.2, we demonstrate a single FRL round using a supernetwork with six edges $e_{i \in [0,5]}$ and three clients $C_{j \in [1,3]}$ who aim to find a subnetwork of size k=50% of the original supernetwork. In Section 3.3.4, we show how FRL minimizes its objective and is independent of the downstream task.

3.3.1 Server: Initialization (only for round $t = 1$)

In the first round, the FRL server chooses a random seed SEED to generate initial random weights θ^w and scores θ^s for the global supernetwork θ; note that, θ^w, θ^s, and SEED remain constant during the entire FRL training. Next, the FRL server shares

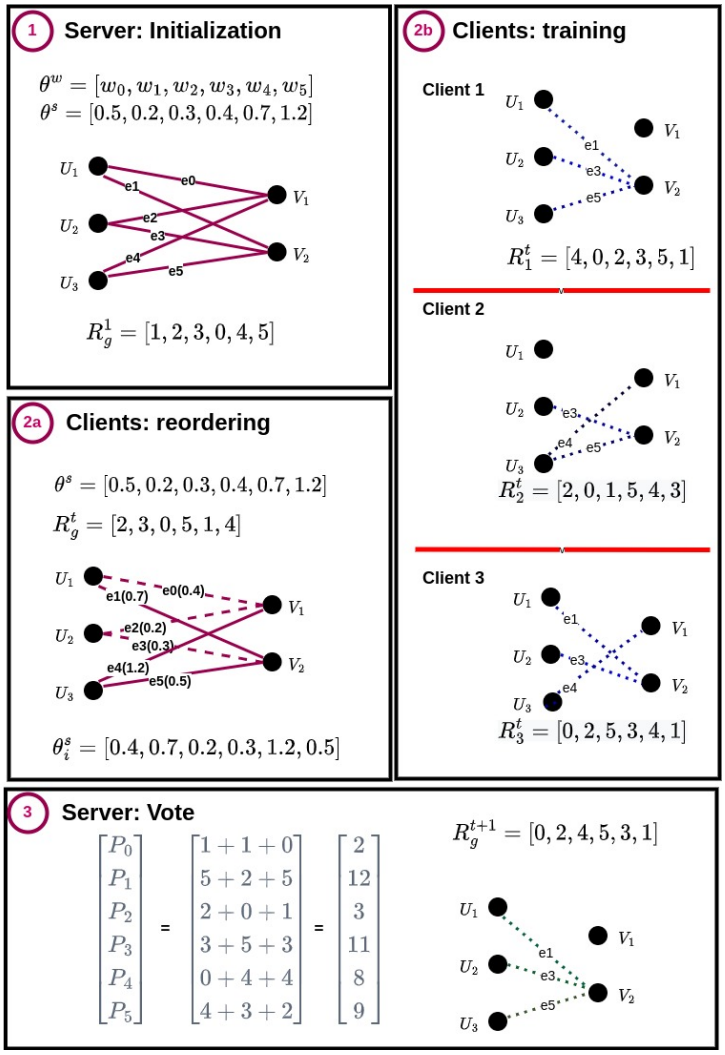


Figure 3.2. A single FRL round with three clients and supernetwork of 6 edges.

SEED with FRL clients, who can then locally reconstruct the initial weights θ^w and scores θ^s using SEED. Figure 3.2-① depicts this step.

Recall that, the goal of FRL training is to find the most important edges in θ^w without changing the weights. Unless specified otherwise, both server and clients use the Signed Kaiming Constant algorithm [143] to generate random weights and the Kaiming Uniform algorithm [80] to generate random scores. However, in Section 3.8.7.2, we also explore the impacts of different weight initialization algorithms on the performance of FRL. We use the same seed to initialize weights and scores.

At the beginning, the FRL server finds the global rankings of the initial random scores (Algorithm 2 line 4), i.e., $R_g^1 = \textsc{ArgSort}(\theta^s)$. We define *rankings of a vector* as the indices of elements of vector when the vector is sorted from low to high, which is computed using ArgSort function.

3.3.2 Clients: Calculating the ranks (for each round t)

In the t^{th} round, FRL server randomly selects n clients among total N clients, and shares the global rankings R_g^t with them. Each of the selected n clients locally reconstructs the weights θ^w's and scores θ^s's using SEED (Algorithm 2 line 9). Then, each FRL client reorders the random scores based on the global rankings, R_g^t (Algorithm 2 line 10); we depict this in Figure 3.2-(2a).

Next, each of the n clients uses reordered θ^s and finds a subnetwork within θ^w using Algorithm 1; to find a subnetwork, they use their local data and E local epochs (Algorithm 2 line 11). Note that, each iteration of Algorithm 1 updates the scores S starting from θ^s. Then client u computes their local rankings R_u^t using the final updated scores (S) and ArgSort(.), and sends R_u^t to the server. Figure 3.2-(2a) shows how each of the selected n clients reorders the random scores using global rankings. For instance, the initial global rankings for this round are $R_g^t = [2, 3, 0, 5, 1, 4]$, meaning that edge e_4 should get the highest score ($s_4 = 1.2$), and edge e_2 should get the lowest score ($s_2 = 0.2$).

Figure 3.2-(2b) shows, for each client, the scores and rankings they obtained after finding their local subnetwork. For example, rankings of client C_1 are $R_1^t = [4, 0, 2, 3, 5, 1]$, i.e., e_4 is the least important and e_1 is the most important edge for C_1. Considering desired subnetwork size to be 50%, C_1 uses edges {3,5,1} in their final subnetwork.

3.3.3 Server: Majority vote (for each round t)

The server receives all the local rankings of the selected n clients, i.e., $R^t_{\{u \in U\}}$. Then, it performs a majority vote over all the local rankings using VOTE(.) function. Note that, for client u, the index i represents the importance of the edge $R^t_u[i]$ for C_u. For instance, in Figure 3.2-(2b), rankings of C_1 are $R^t_1 = [4, 0, 2, 3, 5, 1]$, hence the edge e_4 at index=0 is the least important edge for C_1, while the edge e_1 at index=5 is the most important edge. Consequently, VOTE(.) function assigns reputation=0 to edge e_4, reputation=1 to e_0, reputation=2 to e_2, and so on. In other words, for rankings R^t_u of C_u and edge e_i, VOTE(.) computes the reputation of e_i as its index in R^t_u. Finally, VOTE(.) computes the total reputation of e_i as the sum of reputations from each of the local rankings. In Figure 3.2-(2b), reputations of e_0 are 1 in R^t_1, 1 in R^t_2, and 0 in R^t_3, hence, the total reputation of e_0 is 2. We depict this in Figure 3.2-③; here, the final total reputations for edges $e_{\{i \in [0,5]\}}$ are $A = [2, 12, 3, 11, 8, 9]$. Finally, the server computes global rankings R^{t+1}_g to use for round $t + 1$ by sorting the final total reputations of all edges, i.e., $R^{t+1}_g = \text{ARGSORT}(A)$.

Note that, *all FRL operations that involve sorting, reordering, and voting are performed in a layer-wise manner*. For instance, the server computes global rankings R^t_g in round t for each layer separately, and consequently, the clients selected in round t reorder their local randomly generated scores θ^s for each layer separately.

3.3.4 Additional details of FRL's optimization

Ramanujan et al. [143] proved that when edge (a, b) replaces (c, b) in layer ℓ and the rest of the subnetwork remains fixed then the loss of the supermask learning decreases (provided the loss is sufficiently smooth). Motivated by their proof, we show that when these two edges are swapped in FRL, the loss decreases for FRL optimization too.

Theorem 1: when edge (a, b) replaces (c, b) in layer ℓ and the rest of the subnetwork remains fixed then the loss of the FRL optimization will decrease (provided the loss is sufficiently smooth).

proof. Recall the optimization problem of FRL is as follow:

$$\min_{R_g} F(\theta^w, R_g) = \min_{R_g} \sum_{i=1}^{N} \lambda_i L_i(\theta^w \odot \mathbf{m}) \tag{3.2}$$
$$\text{s.t. } \mathbf{m}[R_g < k] = 0 \text{ and } \mathbf{m}[R_g \geq k] = 1$$

where λ_i shows the importance of the i^{th} client in empirical risk minimization which $\lambda_i = \frac{1}{N}$ gives same importance to all the participating clients. $\mathbf{m}$ is the final mask that contains the edges of top k ranks, and L_i is the loss function for the ith client. $\theta^w \odot \mathbf{m}$ shows the subnetwork inside the random θ^w that all clients unanimously vote for. In this optimization, the FRL clients try to minimize F by finding the best global ranking R_g.

We now wish to show $F(\theta^w, R_g^{t+1}) < F(\theta^w, R_g^t)$ when in FRL round $t+1$, the edge (a, b) replaces (c, b) in layer ℓ and the rest of the subnetwork remains fixed. Suppose global rank of edge (a, b) was $R_g^t[(a, b)]$ and global rank of edge (c, b) was $R_g^t[(c, b)]$ in round t, so we have:

$$R_g^t[(a, b)] < R_g^t[(c, b)] \tag{3.3}$$
$$R_g^{t+1}[(a, b)] > R_g^{t+1}[(c, b)] \tag{3.4}$$

where the order of all the remaining global ranks remains fixed, and only these two edges are swapped in global ranking. Now let $s_{ab}^{t,i}$ shows the score of weight w_{ab} in round t and i^{th} client and $s_{ab}^{t+1,i}$ shows the updated score of it after local training. As in our majority vote, we are calculating the sum of the reputation of edges we will have:

$$\sum_{i=1}^{N} s_{ab}^{t,i} < \sum_{i=1}^{N} s_{cb}^{t,i} \tag{3.5}$$

$$\sum_{i=1}^{N} s_{ab}^{t+1,i} > \sum_{i=1}^{N} s_{cb}^{t+1,i} \tag{3.6}$$

We also know that Edge-popup algorithm updates the scores in the i^{th} client as follow:

$$s_{ab}^{t+1,i} = s_{ab}^{t,i} - \eta \frac{\partial L}{\partial I_a} Z_a W_{ab} \tag{3.7}$$

Based on (3.5), and (3.6), we can say:

$$\sum_{i=1}^{N} s_{ab}^{t,i} - \sum_{i=1}^{N} s_{cb}^{t,i} < \sum_{i=1}^{N} s_{ab}^{t+1,i} - \sum_{i=1}^{N} s_{cb}^{t+1,i} \tag{3.8}$$

And based on (3.7), we also know that:

$$\sum_{i=1}^{N} \left(s_{ab}^{t+1,i} - s_{ab}^{t,i}\right) = \sum_{i=1}^{N} \left(-\eta \frac{\partial L^i}{\partial I_a^i} Z_a^i W_{ab}\right) \tag{3.9}$$

$$\sum_{i=1}^{N} \left(s_{cb}^{t+1,i} - s_{cb}^{t,i}\right) = \sum_{i=1}^{N} \left(-\eta \frac{\partial L^i}{\partial I_c^i} Z_c^i W_{cb}\right) \tag{3.10}$$

Based on (3.8), (3.9) and (3.10), we can say:

$$\sum_{i=1}^{N} \left(\frac{\partial L^i}{\partial I_c^i} Z_c^i W_{cb}\right) > \sum_{i=1}^{N} \left(\frac{\partial L^i}{\partial I_a^i} Z_a^i W_{ab}\right) \tag{3.11}$$

So based on (3.11), and what [143] proved for each supermask training we can show (3.12). We assume that loss is smooth and the input to the nodes that their edges are swapped are close before and after the swap.

$$\sum_{i=1}^{N} \left(L_i(\theta^w \odot \mathbf{m^{t+1}})\right) < \sum_{i=1}^{N} \left(L_i(\theta^w \odot \mathbf{m}^t)\right) \tag{3.12}$$

that means:

$$F(\theta^w, R_g^{t+1}) < F(\theta^w, R_g^t) \tag{3.13}$$

3.4 Robustness of FRL to poisoning

FRL and FL are distributed learning algorithms with mutually untrusting clients. Hence, a *poisoning adversary* may own or compromise some of FRL (FL) clients, called *malicious clients*, and mount a *targeted* or *untargeted* poisoning attack.

As discussed in Chapter 2, we mainly focus on the more severe untargeted attacks and show that FRL is significantly more robust by design to such poisoning attacks. However, for completeness we also evaluate robustness of FRL against targeted attacks in Section 3.8.5.

Intuition behind robustness of FRL: Existing FL algorithms, including robust algorithms, are shown to be vulnerable to various poisoning attacks [149]. One of the key reasons behind the susceptibility of existing algorithms is that their model updates can have a large continuous space of values. For instance, to manipulate vanilla FedAvg, malicious clients send very large updates [27], and to manipulate Multi-krum and Trimmed-mean, [63, 148] propose to perturb a benign update in a specific malicious direction. On the other hand, in FRL, clients must send a permutation of indices $\in [1, n_\ell]$ for each layer. Hence, FRL significantly reduces the space of the possible malicious updates that an adversary can craft. Majority voting in FRL further reduces the chances of successful attack. Intuitively, this makes FRL design robust to poisoning attacks. Below, we make this intuition more concrete.

The worst-case untargeted poisoning attack on FRL: Here, the poisoning adversary aims to reduce the accuracy of the final global FRL subnetwork on most test

inputs. To achieve this, the adversary should replace the high ranked edges with low ranked edges in the final subnetwork. For the worst-case analysis of FRL, we assume a very strong adversary (i.e., threat model): 1) each of the malicious clients has some data from benign distribution; 2) malicious clients know the entire FRL algorithm and its parameters; 3) malicious clients can collude. Under this threat model we design a worst case attack on FRL (Algorithm 3), which executes as follows: First, malicious clients compute rankings on their benign data and use VOTE(.) algorithm to compute an aggregate rankings. Finally, each of the malicious clients uses the reverse of the aggregate rankings to share with the FRL server in given round. The adversary should invert the rankings layer-wise as the FRL server will aggregate the local rankings per layer too, and it is not possible to mount a model-wise attack.

Algorithm 3 FRL Poisoning

Input: number of malicious clients M, number of malicious local epochs E', seed SEED, global ranking R_g^t, learning rate η, subnetwork size $k\%$
function MALICIOUSUPDATE(M, SEED, R_g^t, E', η, k):
  for $mu \in [M]$ **do**
    Malicious Client Executes:
    $\theta^s, \theta^w \leftarrow$ Initialize random scores and weights using SEED
    $\theta^s[R_g^t] \leftarrow \text{SORT}(\theta^s)$
    $S \leftarrow$ Edge-PopUp$(E', D_u^{tr}, \theta^w, \theta^s, k, \eta)$
    $R_{mu}^t \leftarrow \text{ARGSORT}(S)$ ▷ Ranking of the malicious client
  end for
  Aggregation:
  $R_m^t \leftarrow \text{VOTE}(R_{\{mu \in [M]\}}^t)$ ▷ Majority vote aggregation
  return REVERSE(R_m^t) ▷ reverse the ranking
end function

Now we justify why the attack in Algorithm 3 is the worst case attack on FRL for the strong threat model. Note that, FRL aggregation, i.e., VOTE(.), computes the reputations using clients' rankings and sums the reputations of each network edge. Therefore, the strongest poisoning attack would want to reduce the reputation of good edges. This can be achieved following the aforementioned procedure of Algorithm 3 to reverse the rankings computed using benign data.

3.4.1 Theoretical analysis of FRL's robustness

In this section, we prove an upper bound on the failure probability of robustness of FRL, i.e., the probability that a good edge will be removed from the final subnetwork when malicious clients mount the worst case attack.

Following the work of [22], we make two assumptions in order to facilitate a concrete robustness analysis of FRL: a) each malicious client has access only to its own data, and b) we consider a simpler VOTE(.) function, where the FRL server puts an edge e_i in the final subnetwork if more than half of the clients have e_i (a good edge) in their local subnetworks. In other words, the rankings that each client sends to the server is just a bit mask showing that each edge should or should not be in the final subnetwork. The server makes a majority vote on the bit masks, and if an edge has more than half votes, it will be in the global subnetwork. Our VOTE(.) mechanism has more strict robustness criterion, as it uses more nuanced reputations of edges instead of bit masks. Hence, the upper bound on failure probability in this section also applies to the FRL VOTE(.) function.

Assume that edge e_i is a good edge, i.e., having e_i in the final subnetwork improves the performance of the final subnetwork. Let Z be the random variable that represents the number of clients who vote for the edge e_i to be in the final subnetwork, i.e., the number of clients whose local subnetwork of size $k\%$ of the entire supernetwork (Algorithm 2 line 11) contains e_i. Therefore, $Z \in [0, n]$ where n is the number of clients being processed in a given FRL round.

Let G and B be the random variable that represent the number of benign and malicious clients that vote for edge e_i, respectively; the malicious clients inadvertently exclude the good edge e_i in their local subnetwork based on their benign training data.

There are total of αn malicious clients, where α is the fraction of malicious clients that B of them decides that e_i is a bad edge and should not be removed. Each of the malicious clients computes the subnetwork on its own benign training data, so

B of them do not conclude that e_i is a good edge. Hence, $Z = G + B$. We can say that G and B have binomial distribution , i.e., $G \sim$ binomial($[(1-\alpha)n, p]$ and $B \sim$ binomial($[\alpha n, 1-p]$ where p is the probability that a benign client has this edge in their local subnetwork and α is the fraction of malicious clients. Note that the probability that our voting in simplified FRL fails is $P[\text{failure}] = P[Z <= \frac{n}{2}]$, i.e., when more than half of the clients vote against e_i, i.e., they do not include e_i in their local subnetworks. We can find the mean and variance of Z as follows:

$$E[Z] = (1-\alpha)np + \alpha n(1-p) \tag{3.14}$$

$$Var[Z] = (1-\alpha)np(1-p) + \alpha np(1-p) = np(1-p) \tag{3.15}$$

[34] provides an inequality where for a random variable X with mean μ and variance σ^2 we have $P[\mu - X >= \lambda] <= \frac{1}{1+\frac{\lambda^2}{\sigma^2}}$. Using this inequality, we can write:

$$P[Z <= \frac{n}{2}] = P[E[Z] - Z >= E[Z] - n/2] <= \frac{1}{1 + \frac{(E[z]-n/2)^2}{var[Z]}} \tag{3.16}$$

because $1 + x^2 >= 2x$, we have:

$$\begin{aligned} P[Z <= \frac{n}{2}] &<= 1/2\sqrt{\frac{Var[Z]}{(E[Z]-n/2)^2}} \\ &= 1/2\sqrt{\frac{np(1-p)}{(np - \alpha np + \alpha n - \alpha np - n/2)^2}} \\ &= 1/2\sqrt{\frac{np(1-p)}{(n(p + \alpha(1-2p) - 1/2))^2}} \end{aligned} \tag{3.17}$$

What this means is that the probability that the simplified VOTE(.) fails is upper bounded as in (3.17). We show the effect of the different values of α and p in Figure 3.3. We can see from Figure 3.3, if the benign clients can train better supermasks (better chance that a good edge ended in their subnetwork), the probability

that the attackers succeed is lower (more robustness). VOTE(.) in FRL (Section 3.3.3) is more sophisticated and puts more constraints on the malicious clients, hence the about upper bound also applies to FRL.

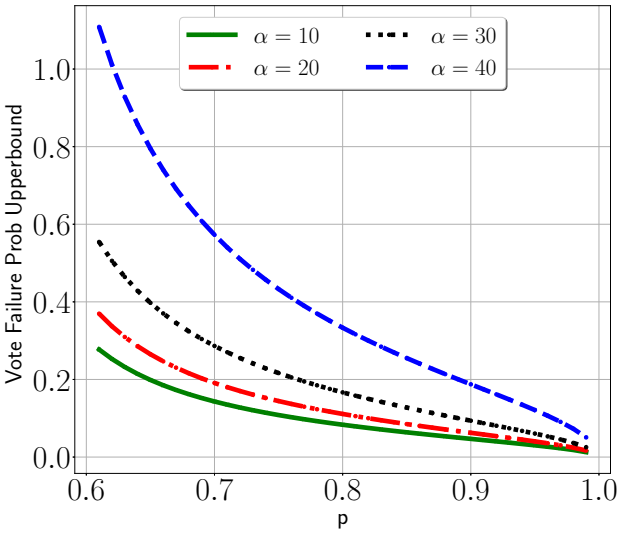


Figure 3.3. Upper bound on the failure probability of VOTE(.) function in FRL. α is the percentages of malicious clients and p is the probability that a benign client puts a good edge in its top k ranks.

3.5 Communication efficiency of FRL

In FL, and especially in the cross-device setting, clients have limited communication bandwidth. Hence, FL algorithms must be communication efficient. We discuss here the communication cost of FRL algorithm. In the first round, the FRL server only sends one seed of 32 bits to all the FRL clients, so they can construct the random weights (θ^w) and scores (θ^s). In a round t, each of selected FRL clients receives the global rankings R_g^t and sends back their local rankings R_u^t. The rankings are a permutation of the indices of the edges in each layer, i.e., of $[0, n_\ell - 1] \forall \ell \in [L]$ where L is the number of layers and n_ℓ is the number of parameters in ℓth layer.

We use the naive approach to communicate layer-wise rankings, where each FRL client exchanges a total of $\sum_{\ell \in [L]} n_\ell \times \log(n_\ell)$ bits. Because, for the layer ℓ, the client receives and sends n_ℓ ranks where each one is encoded with $\log(n_\ell)$ bits. On the other hand, a client exchanges $\sum_{\ell \in [L]} n_\ell \times 32$ bits in FedAvg, when 32 bits are used to represent each of n_ℓ weights in layer ℓ. In Section 3.8.2, we measure the performance and communication cost of FRL with other existing FL compressors SignSGD [22] and TopK [11, 13].

Sparse-FRL: Here, we propose Sparse-FRL, a simple extension of FRL to further reduce the communication cost. In Sparse-FRL, a client sends only the most important ranks of their local rankings to the server for aggregation. For instance, in Figure 3.2, client C_1 sends $R_1^t = [4, 0, 2, 3, 5, 1]$ in case of FRL. But in sparse-FRL, with sparsity set to 50%, client C_1 sends just the top 3 rankings, i.e., sends $R_1'^t = [3, 5, 1]$. For each client, the sparse-FRL server assumes 0 reputation for all of the edges not included in the client's rankings, i.e., in Figure 3.2, sparse-FRL server will assign reputation=0 for edges e_4, e_0, and e_2. Then the server uses VOTE(.) to compute total reputations of all edges and then sort them to obtain the final aggregate global rankings, i.e., R_g^{t+1}, to use for subsequent rounds. We observe in out experiments, that sparse-FRL performs very close to FRL, even with sparsity as low as 10%, while also significantly reducing the communication cost.

Lower-bound of communication cost of FRL: Since the FRL clients send and receive layer-wise rankings of indices, i.e., integers $\in [0, n_\ell - 1]$, for layer ℓ, there are $n_\ell!$ possible permutations for layer $\ell \in [L]$. If we use the best possible compression method in FRL, an FRL client needs to send and receive $\sum_{\ell \in [L]} \log(n_\ell!)$ bits. Therefore, the download and upload bandwidth for each FRL client would be $\sum_{\ell \in [L]} \log\left(n_\ell * (n_\ell - 1) * ... * 2 * 1\right) = \sum_{\ell \in [L]} \sum_{i=1}^{n_\ell} log(i)$ bits. Please note that in our experiment, FRL clients send and receive the rankings without any further

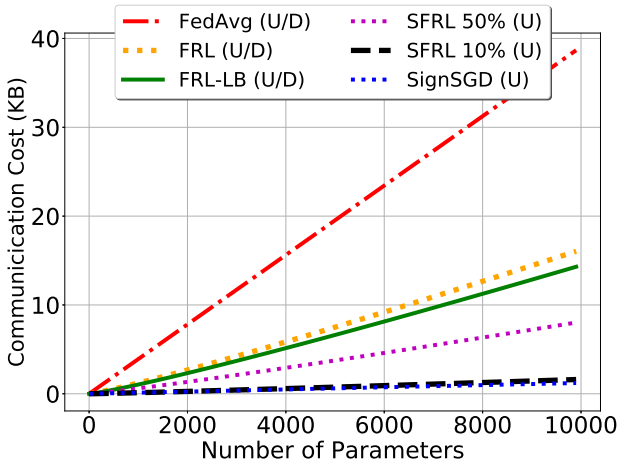


Figure 3.4. Upload (U) and download (D) Communication cost analysis. The download cost (D) of all SFRLs would be the same as FRL. Download communication cost of SignSGD would be the same as FedAvg too.

compression, and $\sum_{\ell \in [L]} \sum_{i=1}^{n_\ell} log(i)$ just shows a lower-bound of communication cost of FRL.

In Figure 3.4, we compare the upload and download communication costs of one client per FL round for FedAvg, SignSGD, and different variants of FRL for different number of parameters. U and D are showing upload and download communication cost. Please note that the download communication cost of all SFRLs would be the same as FRL, and download communication cost of SignSGD would be similar to FedAvg too. If we use a compression method to compress the local rankings (for upload) and global rankings (for download), we can improve communication cost of FRL to its lower-bound (FRL-LB in Figure 3.4). In this Figure, we can see that SFRL can provide competitive upload communication cost and lower download communication cost compared to SignSGD when the clients are sending only 10% of top local rankings (SFRL 10%).

3.6 Subnetwork connectivity in FRL

In FRL, clients collaborate to find a subnetwork within a randomly initialized neural network called the supernetwork. The goal of training in FRL is to collaboratively rank the supernetwork's edges based on the importance of each edge and find a *global ranking*. The global ranking can be converted to a supermask, a binary mask of 1's and 0's, superimposed on the random neural network (the supernetwork) to obtain the final subnetwork.

In this section, we try to answer the following key questions: **(a)** *What is the probability of a disconnected subnetwork?* **(b)** *What does happen if the subnetwork becomes disconnected?* **(c)** *Is FRL robust against an adversary who wants to make the subnetwork disconnected?*

To answer the above questions, we first quantify the cases when the subnetwork is disconnected. Next, we calculate the probability of a disconnected subnetwork. Then, we show the implication of a disconnected subnetwork and how Edge-Popup handles a disconnected subnetwork in its supermask training. Finally, we argue that the untargeted attack against FRL discussed in Section 3.4 is a weak version of an attack where the adversary wants to make the subnetwork disconnected. In this section, we provide examples of fully connected layer networks, but it is straightforward to extend them to convolution layer networks (see Appendix B.2 of [143]).

3.6.1 Quantifying the number of possibilities that the subnetwork becomes disconnected

Consider a fully connected (FC) neural network with two layers $\{$ FC(ℓ_1, ℓ_2), FC(ℓ_2, ℓ_3) $\}$, where there exist $\ell_1 \times \ell_2$ and $\ell_2 \times \ell_3$ edges in the first and second layers, respectively. When we are looking for a subnetwork with per-layer sparsity of k $(0 < k < 1)$ as in FRL, the subnetwork consists of $\ell_1 \times \ell_2 \times k$ of total $\ell_1 \times \ell_2$ edges

of layer one, and $\ell_2 \times \ell_3 \times k$ of total $\ell_2 \times \ell_3$ edges of layer two. Therefore, the total number of possible subnetworks in this supernetwork would be:

$$\binom{\ell_1\ell_2}{\ell_1\ell_2 k} \times \binom{\ell_2\ell_3}{\ell_2\ell_3 k} \tag{3.18}$$

Suppose that V_I represents the subset of neurons in the middle neuron layer (with ℓ_2 neurons) that have an incoming edge, and V_O represents the subset of the neurons in the middle layer with an outgoing edge. For a subnetwork to be disconnected, these two sets should be disjoint ($V_I \cap V_O = \emptyset$). V_I has $\ell_2 \times k$ neurons at minimum, meaning $\ell_2 \times k$ neurons cover $\ell_2 \times \ell_1 \times k$ edges by receiving maximum possible of edges for each neuron. On the other hand V_I can have $\min(\ell_2, \ell_1 \times \ell_2 \times k)$ neurons at maximum by just assigning each edge to each neuron. Therefore number of neurons in V_I is limited by $\ell_2 \times k \leq |V_I| \leq \min(\ell_2, \ell_1 \times \ell_2 \times k)$. If we want to calculate the number of possible disconnected subnetworks, we should **(a)** first calculate the number of edges from the first layer and assign them to V_I, **(b)** find the number of free neurons that do not receive any edges (i.e., $\ell_2 - |V_1|$), and **(c)** calculate the number of possible edges from these free neurons to the third layer as follows:

$$\sum_{\ell_2 k \leq v \leq \min(\ell_2, \ell_1\ell_2 k)} \binom{\ell_1}{\ell_1 k} \times \binom{\ell_2}{v} \times \binom{\ell_2 - v}{\ell_2 k} \times \binom{\ell_3}{\ell_3 k} \tag{3.19}$$

where we calculate the number of possible incoming edges to the hidden layer (with ℓ_2 neurons) and possible outgoing edges from this layer without any intersection of these neurons. Moreover, we can calculate the probability of a disconnected subnetwork in $\{\text{FC}(\ell_1, \ell_2), \text{FC}(\ell_2, \ell_3)\}$ with sparsity k ($0 < k < 1$) as:

$$\frac{\sum_{\ell_2 k \leq v \leq \min(\ell_2, \ell_1\ell_2 k)} \binom{\ell_1}{\ell_1 k} \times \binom{\ell_2}{v} \times \binom{\ell_2 - v}{\ell_2 k} \times \binom{\ell_3}{\ell_3 k}}{\binom{\ell_1\ell_2}{\ell_1\ell_2 k} \times \binom{\ell_2\ell_3}{\ell_2\ell_3 k}} \tag{3.20}$$

This probability is very low. For example, for a network of $\{\text{FC}(32, 128), \text{FC}(128, 10)\}$, there is a probability of 10^{-699} a disconnected subnetwork is initialized with $k = 0.1$,

or with probability of 10^{-1353} when $k = 0.3$. The probability becomes zero for value $k > 0.5$ as it is impossible to find enough free neurons while assigning all the incoming edges to the rest of them. It represents the worst-case scenario for a disconnected subnetwork, where all the edges of the first layer should go to neurons that do not have any outgoing edge in the second layer. If $k > 0.5$, then at least one neuron in the middle has one incoming edge and one outgoing edge, which makes the subnetwork connected. In all of our experiments, unless specified, we used $k = 0.5$ for the sparsity of the subnetwork, which shows the probability of a disconnected subnetwork is close to zero.

3.6.2 What does happen if the subnetwork becomes disconnected?

Each FRL client uses edge-popup (EP) algorithm (Algorithm 1) to train the scores of its local model and find a local subnetwork. After local training is finished, the FRL client ranks the edges from least to most important, and submits its local ranking to the FRL server for aggregation. Specifically, EP algorithm assigns a positive score to each of the edges in the supernetwork. On forward pass, it selects the top k% edges with highest scores, where k is the percentage of the total number of edges in the supernetwork that will remain in the final subnetwork. On the backward pass, it updates the scores with the straight-through gradient estimator [21].

It is possible that the subnetwork of top k% scores may become disconnected (e.g., for very small k) with a very low probability. Even in cases where the forward pass is entirely disconnected, non-zero gradients are still propagated to all edges of the network. This is because EP uses the straight-through gradient estimator in the backward pass, so it behaves as though the network is fully-connected during the gradient update step. This allows for edges to "come back" even after they have been pruned.

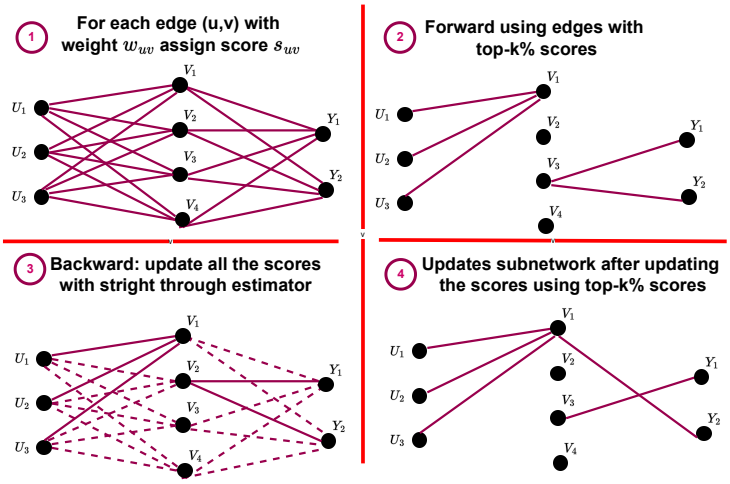


Figure 3.5. Edge-popup (EP) training steps.

Example: Figure 3.5-① depicts an example of a two-layer fully connected neural network. In this network, we have two layers {FC(3,4) and FC(4,2)}. Equations (3.21) and (3.22) show the weights (generated by Singed Kaiming Constant [143]) and scores of the edges for this network. Equation (3.23) shows the rankings of the edges for each layer for this network.

$$\theta^{w_1} = \begin{bmatrix} -0.8165 & -0.8165 & 0.8165 \\ -0.8165 & -0.8165 & -0.8165 \\ +0.8165 & +0.8165 & +0.8165 \\ +0.8165 & +0.8165 & -0.8165 \end{bmatrix}, \theta^{w_2} = \begin{bmatrix} -0.7071 & +0.7071 & +0.7071 & -0.7071 \\ -0.7071 & -0.7071 & -0.7071 & -0.7071 \end{bmatrix} \tag{3.21}$$

$$\theta^{s_1} = \begin{bmatrix} 0.5017 & 0.4830 & 0.4821 \\ 0.3313 & 0.2739 & 0.2558 \\ 0.2552 & 0.1815 & 0.1781 \\ 0.1715 & 0.1356 & 0.0554 \end{bmatrix}, \theta^{s_2} = \begin{bmatrix} 0.4216 & 0.4216 & 0.4494 & 0.2490 \\ 0.2282 & 0.2159 & 0.4512 & 0.0617 \end{bmatrix} \tag{3.22}$$

$$R(\theta^{s_1}) = [11, 10, 0, 9, 8, 7, 6, 5, 5, 3, 2, 1, 0], \quad R(\theta^{s_2}) = [7, 5, 4, 3, 1, 0, 2, 6] \tag{3.23}$$

Figure 3.5-② shows the forward pass in EP when we want to have a subnetwork with 25% of the original supernetwork size (i.e., $k = 0.25$). As we can see from this figure, the subnetwork is disconnected since all the edges in the first layer are going to neuron V_1, and there is no output edge from this neuron. Suppose we input $I = [1.0, 1.0, 1.0]$ to this network, then the output of the first layer would be $[-0.8165, 0.0, 0.0, 0.0]$ and the output of the second layer would be $[0, 0]$. For this example the cross-entropy loss would be 0.6931 for given true label of [0]. Figure 3.5-③ shows the backward pass of EP where the EP updates all the scores with the straight-through estimator, not only the subnetwork edges. Equation 3.24 shows the non-zero gradients of the scores in the backward pass of EP.

$$\frac{\partial L}{\partial \theta^{s_1}} = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 \\ -0.5774 & -0.5774 & -0.5774 \\ 0.0000 & 0.0000 & 0.0000 \end{bmatrix}, \frac{\partial L}{\partial \theta^{s_2}} = \begin{bmatrix} -0.2887 & 0.0000 & 0.0000 & 0.0000 \\ 0.2887 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix} \tag{3.24}$$

From Equation 3.24, we can see the EP Algorithm in backward pass brings the helpful edges back in the subnetwork, e.g., edge $V_1 - Y_2$ in this example, which makes this subnetwork connected. Figure 3.5-④ depicts the subnetwork after updating the scores in one round of EP, which shows the subnetwork connected after one round of applying EP Algorithm.

In the above example, we show that even if the subnetwork of one FRL round becomes disconnected, the FRL clients can train their rankings by using EP. However,

receiving global rankings of a disconnected network is very unlikely in FRL as we use $k = 0.50$ in most of our experiments, i.e., the subnetworks are half the size of the original supernetwork.

3.6.3 FRL against an adversary who wants to make the subnetwork disconnected

In the above sections, we explain that if $k > 0.5$, it is impossible that the output subnetwork of FRL becomes disconnected. Nevertheless, the subnetwork may become disconnected if we choose a smaller sparsity (k) with a very low probability (Section 3.6.1). One untargeted attack against FRL can be an adversary that aims to make the global subnetwork disconnected by crafting its malicious rankings in a particular way that only edges for neurons that have no incoming edges remain in the top $k\%$ edges with the highest scores. In each FRL round, each benign FRL client trains its local ranking on its benign local data by utilizing the Edge-Popup algorithm (Algorithm 1) starting from the received global ranking. This attack is ineffective in the middle of FRL training because if the attacker is successful in poisoning the global ranking and makes its output subnetwork disconnected, the benign clients in the next round produce their local rankings by finding a connected local subnetwork. We show this process in Section 3.6.2, where if the start point is a disconnected subnetwork, the Edge-Popup algorithm can easily bring the good edges back to the subnetwork and makes it connected. At the end of the FRL training, the local rankings become very close to each other (their distance becomes very small), where malicious clients cannot make the subnetwork disconnected as the majority vote of FRL gives each local ranking one equal influence, and the benign local rankings are in the majority, so the adversary cannot make the final global subnetwork disconnected.

The above attack is much weaker compared to the worst-case untargeted poisoning attack on FRL discussed in Section 3.4. In our attack (Section 3.4), the malicious

clients compute rankings on their benign data and use VOTE(.) algorithm to compute an aggregate ranking. Next, each malicious client uses the reverse of the aggregate rankings to share with the FRL server in a given round. By reversing, the adversary wants to create more damage than disconnecting the subnetwork because even if the adversary becomes successful in making the global subnetwork disconnected in one round, the next round, it is easy for the benign clients to make it connected again by applying EP on their benign data.

3.7 Experimental setup

3.7.1 Datasets and their distribution

We use MNIST, CIFAR10, and FEMNIST datasets. Most real-world FL settings have heterogeneous client data, hence following previous works [145, 86], we distribute MNIST and CIFAR10 datasets among 1,000 clients in non-iid fashion using *Dirichlet* distribution with parameter $\beta = 1$. Note that, increasing β results in more iid datasets. FEMNIST is naturally distributed non-iid among 3,400 clients. We further split the datasets of each client into training (80%) and test (20%).

We run all the experiments for 2000 global rounds of FRL and FL, while selecting 25 clients in each round. At the end of the training, we calculate the test accuracy of all the clients on the final global model, and we report the mean and standard deviation of all clients' test accuracies in our experiments.

We also evaluate the utility of FRL on two significantly more complicated tasks: image classification on Tiny-ImageNet [3] (Section 3.8.7.4) and text classification on IMDB reviews [124] (Section 3.8.4).

3.7.2 Hyperparameters

We use the following hyperparameters for each dataset:

MNIST is a 10-class class-balanced classification task with 70,000 gray-scale images, each of size 28×28. We experiment with LeNet architecture given in Table 3.1. For local training in each FRL/FL round, each client uses 2 epochs. For training weights (experiments with FedAvg, SignSGD, TopK), we use SGD optimizer with learning rate of 0.01, momentum of 0.9, weight decay of 1e-4, and batch size 8. For training ranks (experiments with FRL), we use SGD with learning rate of 0.4, momentum 0.9, weight decay 1e-4, and batch size 8.

CIFAR10 [104] is a 10-class classification task with 60,000 RGB images (50,000 for training and 10,000 for testing), each of size 32×32. We experiment with a VGG-like architecture given in Table 3.1, which is identical to what [143] used. For local training in each FRL/FL round, each client uses 5 epochs. For training weights (experiments with FedAvg, SignSGD, TopK), we use SGD with learning rate of 0.1, momentum of 0.9, weight decay of 1e-4, and batch size of 8. For training ranks (experiments with FRL), we optimize SGD with learning rate of 0.4, momentum of 0.9, weight decay of 1e-4, and batch size of 8.

FEMNIST [33, 46] is a character recognition classification task with 3,400 clients, 62 classes (52 for upper and lower case letters and 10 for digits), and 671,585 gray-scale images. Each client has data of their own handwritten digits or letters. We use LeNet architecture given in Table 3.1. For local training in each FRL/FL round, each client uses 2 epochs. For training weights (experiments with FedAvg, SignSGD, TopK), we use SGD with learning rate of 0.15, momentum of 0.9, weight decay of 1e-4, and batch size of 10. For training ranks (experiments with FRL), we optimize SGD with learning rate of 0.2, momentum of 0.9, weight decay of 1e-4, and batch size of 10.

***Tiny-ImageNet* [3]** contains 100K, 10k, 10k training, validation and test images respectively for 200 classes downsized to 64×64 colored images, so each class has 500 training, and 50 validation and 50 test images. We distribute the training data over 1000 clients in a non-iid fashion using Dirichlet distribution with parameter $\beta = 1$.

We run FedAvg and FRL for 4000 global rounds, and in each round, the FL server selects 25 clients randomly. For local training weights (experiments with FedAvg), we use SGD optimizer with a learning rate of 0.01, momentum 0.9, weight decays of 1e-5, batch size 8, and local epochs 2. For training ranks (experiment with FRL), we use SGD with a learning rate of 0.8, momentum of 0.9, weight decay of 1-e5, batch size 8, and local epochs 1. We optimize the hyperparameters based on the validation data and report the final accuracy on the test data.

3.7.3 Model architectures

Table 3.1 shows the state-of-the-art model architectures and corresponding datasets that we use in our experiments. We also show the number of parameters in each of their layers.

3.7.4 Baseline FL algorithms

We compare the FRL with following FL baselines:

Federated averaging: In non-adversarial FL settings, i.e., without any malicious clients, the dimension-wise Average (FedAvg) [103, 126] is an effective AGR. In fact, due to its efficiency, Average is the only AGR implemented by FL applications in practice [123, 140].

SignSGD: is a quantization method used in distributed learning to compress each dimension of gradient updates into 1 bit instead of 32 or 64 bits. To achieve this, in SignSGD [22] the clients only send the sign of their gradient updates to the server, and the server runs a majority vote on them. SignSGD is designed for distributed learning where all the clients participate in each round, so all the clients are aware of the most updated weight parameters of the global model. However, SignSGD only reduces upload communication (clients→server). But, does not reduce download communication (server→clients), i.e., to achieve good performance of the global model,

Table 3.1. In our experiments, we use the following, state-of-the-art model architectures from [143, 164, 42].

Architecture	Layer Name	Number of parameters
LeNet [164] (MNIST and FEMNIST)	Conv(32)	288
	Conv(64)	18432
	FC(128)	1605632
	FC(10) or FC(62)	1280 or 7936
Conv8 [143] (CIFAR10)	Conv(64), Conv(64)	38592
	Conv(128), Conv(128)	221184
	Conv(256), Conv(256)	884736
	Conv(512), Conv(512)	3538944
	FC(256), FC(256), FC(10)	592384
BiLSTM [42] (IMDB)	Embedding (dim=300)	7469100
	Dropout (p=0.5)	-
	LSTM (hidden_dim=300, bidirectional=True, num_layers=1)	1440000
	FC(2)	1200
ResNet18 [164] (CIFAR10 and Tint-ImageNet)	Conv(64), [Conv(64), Conv(64)] × 2	149184
	[Conv(128), Conv(128)] × 2	524288
	[Conv(256), Conv(256)] × 2	2097152
	[Conv(512), Conv(512)] × 2	8388608
	Avg-Pool	-
	FC(10) or FC(200)	5120 or 102400
ResNet34 [164] (CIFAR10 and Tint-ImageNet)	Conv(64), [Conv(64), Conv(64)] × 3	222912
	[Conv(128), Conv(128)] × 4	1114112
	[Conv(256), Conv(256)] × 6	6815744
	[Conv(512), Conv(512)] × 3	13107200
	Avg-Pool	-
	FC(10) or FC(200)	5120 or 102400

the server sends all the weight parameters (each of 32 bits) to the newly selected clients in each round. Hence, SignSGD is as inefficient as FedAvg in download communication.

TopK: is a sparsification method used in distributed learning that transmits only a few dimensions of each model update to the server. In TopK [11, 13], the clients first sort the absolute values of their local model updates, and send the Top K% largest model update dimensions to the server for aggregation. TopK suffers from the same problem as SignSGD: for performance reasons, the server should send the entire updated model weights to the new selected clients.

Multi-krum: [27] proposed Multi-krum AGR as a modification to their own Krum AGR. Multi-krum selects an update using Krum and adds it to a selection set, S.

Multi-krum repeats this for the remaining updates (which remain after removing the update that Krum selects) until S has c updates such that $n - c > 2m + 2$, where n is the number of selected clients and m is the number of compromised clients in a given round. Finally, Multi-krum averages the updates in S.

Trimmed-mean: Yin et al. [169] proposed Trimmed-mean that aggregates each dimension of input updates separately. It sorts the values of the j^{th}-dimension of all updates. Then it removes m (i.e., the number of compromised clients) of the largest and smallest values of that dimension, and computes the average of the rest of the values as its aggregate for the dimension j.

3.8 Empirical evaluation

In this section, we extensively evaluate the utility, robustness, and communication cost of our FRL algorithm. This section is organized as follows: In Section 3.8.1 we investigate the robustness and utility of FRL, followed by Section 3.8.2 in which we discuss the communication cost of FRL. Next, in Section 3.8.3, we compare FRL utility and robustness to an extension of edge-popup to FL. Moreover, we also provide results of FRL for a NLP classification task (Section 3.8.4), FRL against backdoor attacks (Section 3.8.5), and FRL with larger number of FL clients (Section 3.8.6). We conclude with Section 3.8.7, where we provide an ablation study on FRL.

3.8.1 Analyses of robustness to poisoning

We compare FRL with state-of-the-art robust aggregation rules (AGRs): Mkrum [27], and Trimmed-mean [169]. Table 3.2 gives the performances of robust AGRs, SignSGD, and FRL with different percentages of malicious clients using attacks proposed by Shejwalkar et al. [148], Bernstein et al. [22], and Algorithm 3 respectively. Here, we make a rather impractical assumption in favor of the *previous* robust AGRs: we

assume that the server knows the exact % of malicious clients in each FL round. Note that, *FRL does not require this knowledge.*

Table 3.2. Comparing the robustness of various FL algorithms: FRL and Sparse-FRL (SFRL) (in **bold**) outperform the state-of-the-art robust AGRs and SignSGD against the strongest of untargeted poisoning attacks.

Dataset	AGR	No malicious	10% malicious	20% malicious
MNIST + LeNet 1000 clients	FedAvg	98.8 (3.2)	10.0 (10.0)	10.0 (10.0)
	Trimmed-mean	98.8 (3.2)	95.1 (7.7)	87.6 (9.5)
	Multi-krum	98.8 (3.2)	98.6 (3.3)	97.9 (4.1)
	SignSGD	97.2 (4.6)	96.6 (5.0)	96.2 (5.6)
	FRL	**98.8 (3.1)**	**98.8 (3.1)**	**98.7 (3.3)**
	SFRL Top 50%	**98.2 (3.8)**	**97.04 (4.4)**	**95.1 (7.8)**
CIFAR10 + Conv8 1000 clients	FedAvg	85.4 (11.2)	10.0 (10.1)	10.0 (10.1)
	Trimmed-mean	84.9 (11.0)	56.3 (16.0)	20.5 (13.2)
	Multi-krum	84.7 (11.3)	58.8 (15.8)	25.6 (14.4)
	SignSGD	79.1 (12.8)	39.7 (15.9)	10.0 (10.1)
	FRL	**85.3 (11.3)**	**79.0 (12.4)**	**69.5 (14.8)**
	SFRL Top 50%	**77.6 (13.0)**	**41.7 (15.4)**	**39.7 (15.2)**
FEMNIST + LeNet 3400 clients	FedAvg	85.8 (10.2)	6.3 (5.8)	6.3 (5.8)
	Trimmed-mean	85.2 (11.0)	72.7 (15.7)	56.2 (20.3)
	Multi-krum	85.2 (10.9)	80.9 (12.2)	23.7 (12.8)
	SignSGD	79.3 (12.4)	76.7 (13.2)	55.1 (14.9)
	FRL	**84.2 (10.7)**	**83.0 (10.9)**	**65.8 (17.8)**
	SFRL Top 50%	**75.2 (12.7)**	**70.5 (14.4)**	**60.39 (14.8)**

FRL achieves higher robustness than state-of-the-art robust AGRs: We note from Table 3.2 that, FRL is more robust to the presence of malicious clients who mount untargeted poisoning attacks, compared to Multi-Krum and Trimmed-mean, when percentages of malicious clients are 10% and 20%. For instance, on CIFAR10, 10% malicious clients can decrease the accuracy of FL models to 56.3% and 58.8% for Trimmed-mean and Multi-Krum respectively; 20% malicious clients can decrease the accuracy of the FL models to 20.5% and 25.6% for Trimmed-mean and Multi-Krum respectively. On the other hand, FRL performance decreases to 79.0% and 69.5% for 10% and 20% attacking ratio, respectively.

We make similar observations for MNIST and FEMNIST datasets: for FEMNIST, 10% (20%) malicious clients reduce accuracy of the global model from 85.8% to 72.7%

(56.2%) for Trimmed-Mean, and to 80.9% (23.7%) for Multi-krum, while FRL accuracy decreases to 83.0% (65.8%).

FRL is more accurate than SignSGD: First, we note that, in the absence of malicious clients, FRL is significantly more accurate than SignSGD. For instance, on CIFAR10 distributed in non-iid fashion among 1000 clients, FRL achieves 85.3% while SignSGD achieves 79.1% , or on FEMNIST, FRL achieves 84.2% while SignSGD achieves 79.3%. This is because, FRL clients send more nuanced information via rankings of their subnetworks compared to SignSGD, where clients just send the signs of their model updates.

FRL is more robust than SignSGD: Next, we note from Table 3.2 that, FRL is more robust against untargeted poisoning attacks compared to SignSGD when percentages of malicious clients are 10% and 20%. For instance, on CIFAR10, 10% (20%) malicious clients can decrease the accuracy of SignSGD model to 39.8% (10.0%). On the other hand, FRL performance decreases to 79.0% and 69.5% for 10% and 20% attacking ratio respectively. We make similar observations for MNIST and FEMNIST datasets: for FEMNIST, 10% (20%) malicious clients reduce accuracy of the global model from 85.8% to 76.7% (55.1%) for SignSGD, while FRL accuracy decreases to 83.0% (65.8%).

Sparse-FRL robustness: We evaluate robustness of SFRL Top 50% against 10% and 20% malicious clients. As we can see from Table 3.2, by sending only top half of the local rankings, the accuracy goes from 85.3% (FRL) to 77.6% (SFRL). SFRL also can provide robustness to some extend, but adversary has more influence on the global ranking since half of the rankings are missing. For instance, on CIFAR10, 10% (20%) malicious clients can decrease the accuracy of global ranking to 41.7% (39.7%) from 77.6%. Also for FEMNIST, 10% (20%) malicious clients can decrease the accuracy of global ranking to 70.5% (60.39%) from 75.2%. We can see when malicious clients' percentages are higher, SFRL can perform better compared to existing robust AGR.

FRL versus FedAvg and TopK: We omit the results of non-robust aggregations, FedAvg and TopK, because even a single malicious client [27] can jeopardize their performances.

3.8.2 Communication cost analysis

In FRL, both clients and server communicate just the edge ranks instead of weight parameters. Thus, FRL reduces both upload and download communication cost. Table 3.3 illustrates the utility, i.e., the mean and standard deviation of all clients' test accuracies and, communication cost of FRL and state-of-the-art quantization (i.e., SignSGD [22]) and sparsification (i.e., TopK [13, 11]) communication-reduction methods.

Table 3.3. Comparing the accuracy and communication cost of FedAvg, SignSGD, TopK, FRL and Sparse-FRL (SFRL) with different percentages of sparsity (in **bold**). Parentheses in the accuracy column show standard deviation of the accuracy.

Algorithm	Accuracy	Upload/Download (MB)
MNIST + LeNet + 1000 clients		
FedAvg	98.8 (3.1)	6.20/ 6.20
FRL	**98.8 (3.2)**	**4.05/ 4.05**
SFRL Top 50%	**98.2 (3.8)**	**2.03/ 4.05**
SFRL Top 10%	**89.5 (9.2)**	**0.40/ 4.05**
SignSGD	97.2 (4.6)	0.19/ 6.20
TopK 50%	98.8 (3.2)	3.29/ 6.20
TopK 10%	98.7 (3.2)	0.81/ 6.20
CIFAR10 + Conv8 + 1000 clients		
FedAvg	85.4 (11.2)	20.1/ 20.1
FRL	**85.3 (11.3)**	**13.1/ 13.1**
SFRL Top 50%	**77.6 (13.0)**	**6.5/ 13.1**
SFRL Top 10%	**27.5 (14.4)**	**1.3/ 13.1**
SignSGD	79.1 (13.6)	0.63/ 20.1
TopK 50%	82.1 (11.8)	10.69/ 20.1
TopK 10%	77.8 (13.0)	2.64/ 20.1
FEMNIST + LeNet + 3400 clients		
FedAvg	85.8 (10.2)	6.23/ 6.23
FRL	**84.2 (10.7)**	**4.06/ 4.06**
SFRL Top 50%	**75.2 (12.7)**	**2.03/ 4.06**
SFRL Top 10%	**59.2 (15.0)**	**0.40/ 4.06**
SignSGD	79.3 (12.4)	0.19/ 6.23
TopK 50%	85.7 (9.9)	3.31/ 6.23
TopK 10%	85.5 (10.0)	0.81/ 6.23

FRL versus SignSGD: SignSGD in FL reduces only the upload communication, but for efficiency reasons, the server sends all of the weight parameters (each of 32 bits) to the newly selected clients. Hence, SignSGD has very efficient upload communication, but very inefficient download communication. For instance, on CIFAR10, for both upload and download, FRL achieves 13.1MB each while SignSGD achieves 0.63MB and 20.1MB, respectively.

FRL versus TopK: We compare FRL and TopK where $K \in \{10, 50\}\%$. FRL is more accurate than Topk for MNIST and CIFAR10: on CIFAR10, FRL accuracy is 85.3%, while TopK accuracies are 82.1% and 77.8% with K=50% and K=10%, respectively. Similar to SignSGD, Topk more efficiently reduces upload communication, but has very high download communication. Therefore, the combined upload and download communication cost per client per round is 26.2MB for FRL and 30.79MB for TopK with K=50%; note that, even then TopK performs worse than FRL.

Communication cost reduction due to Sparse-FRL (SFRL): We now evaluate SFRL explained in Section 3.5. In SFRL with top 50% ranks, clients send the top 50% of their ranks to the server, which reduces the upload bandwidth consumption by half. Please note that the download cost of SFRL is the same as FRL since the FRL server should send all the global rankings to the selected clients in each round. We note from Table 3.3 that, by sending fewer ranks, SFRL reduces upload communication at a small cost of performance. For instance, on CIFAR10, SFRL with top 50% reduces the upload communication by 50% at the cost reducing accuracy from 85.4% to 77.6%.

3.8.3 Comparison with naïve extension of edge-popup algorithm to FL

As discussed in Section 3.3.2, prior works [144] on supermask training and its following works [164, 42] do not consider rankings, and instead find subnetworks in randomly initialized networks by just training the scores, in a centralized training setting. Algorithm 4 shows the naïve extension of [144] to FL, where the clients train

and exchange scores (32bits floats). Like FedAvg, these scores are from a continuous space of float numbers (as opposed to parameter ranks in FRL). At the end of each FL round, the server averages the local score updates to aggregate them and produces global scores (different from our majority vote aggregation). It is important to note that our majority vote aggregation only works on a set of ranking inputs.

Algorithm 4 Edge-popup based FL (EFL)

Input: number of rounds T, number of local epochs E, number of users in each round n, seed SEED, learning rate η, subnetwork size $k\%$
Server: Initialization
$\theta^s_0, \theta^w \leftarrow$ Initialize random scores and weights using SEED
for $t \in [1, T]$ **do**
  $U \leftarrow$ set of n randomly selected clients out of N total clients
  for u in U **do**
    Clients: Calculating the scores using EP
    $\theta^w \leftarrow$ Initialize weights using SEED
    $S^t_u \leftarrow$ Edge-PopUp$(E, D^{tr}_u, \theta^w, \theta^s_{t-1}, k, \eta)$ ▷ Client u uses Algorithm1 to train a supermask starting from global scores θ^s_{t-1}
  end for
  Server: Averaging the scores
  $\theta^s_{t+1} \leftarrow \text{AVG}(S^t_{\{u \in U\}})$
end for

Exchanging such scores is as vulnerable to poisoning as regular FL because they are not scale-free and discrete similar to rankings. Table 3.4 compares the performance of FedAvg, where the clients train weight parameters, EdgePopUPFL, using this naïve application of Edge-popup where the clients train scores, and FRL, where the clients are training rankings. If we use scores as the clients' updates, even one malicious client can generate its adversarial score update and re-scales it to cancel the effect of other benign updates. On the other hand, rankings are scale-free, and the adversary cannot increase its influence by re-scaling the rankings.

Table 3.4. Comparing the robustness of EFL, a naïve edge-popup based FL (Algorithm 4), with robustness of FRL.

Dataset **# of clients**	AGR	Percentage of Malicious Clients		
		0%	10%	20%
CIFAR10 + Conv8 **1000 clients**	FedAvg	85.4 (11.2)	10.0 (10.1)	
	EFL	84.2 (11.3)	10.0 (10.1)	
	FRL	**85.3 (11.3)**	**79.0 (12.4)**	**69.5 (14.8)**

Furthermore, as discussed in Section 3.3, FRL's performance does not just come from using the Edge-popup algorithm, but rather from introducing an efficient protocol to rank and aggregate them which enables achieving high accuracy and robustness. Overall, a major novelty of our work is that, to the best of our knowledge, FRL is the first scalable, distributed training algorithm that trains using parameters rankings to effectively defend against poisoning attacks. Below is the summary of our contributions compared to the naïve extension of [144] to FL:

- Transform the continuous, local scores into discrete and scale-free rankings (Section 3.3.2).
- Develop algorithms to aggregate local rankings to produce global rankings (Section 3.3.3).
- Map global rankings back to local scores for further local training (Section 3.3.2).

3.8.4 FRL for text classification

So far, following the majority of research on supermasks, we only focused on vision tasks. However, in this section, we demonstrate the efficacy of FRL on text classification tasks.

Previous works [66, 42] shows the effectiveness of supermask learning on a text classification task in a centralized learning paradigm. To the best of our knowledge, FRL is the first work that trains a distributed model by utilizing supermask learning to generate rankings. In this section, we show the effectiveness of FRL on a text classification task when the data is distributed over the FL clients. In particular, we train a recurrent neural network (RNN), shown in Table 3.1, on the IMDB reviews dataset [124]. IMDB reviews dataset consists of 25k and 25k reviews for each training and testing data. We distribute the training data over 25 FL clients identically and independently (iid), where each client has access to 1k reviews. We also divide the

original IMDB test data and use 5k samples for validation and 20k for test. We find the hyperparameters that perform the best on the validation data, use them to train the final FL model, and report its accuracy on the test data.

Following the setting of previous work [42] which trains supermask for a text classification task, we use a BiLSTM network with an embedding layer, a dropout layer, an LSTM layer (with bidirectional=True), and a linear layer. We import the pre-trained weights for our embedding layer from [1] and freeze them during back-propagation To train FRL, we use the SparseModule [2], which freezes the weight parameters of the LSTM layer and enables us to train a supermask on top of frozen weight parameters. Using this code [2] allows training supermask on any neural network layer and producing rankings based on the scores of the supermask.

Table 3.5. Test accuracies of FRL and FedAvg on IMDB dataset [124] with the BiLSTM from Table 3.1.

Network	# of Params	FRL	FedAvg
BiLSTM	1.44M	88.00	88.46

We train FRL and FedAvg for 300 global FL rounds. In each round, we select all the FL clients for participation (i.e., $N = n = 25$). Each client uses SGD with momentum 0.9 and weight decay 1e-5 to train locally. In FedAvg, each client trains for $E = 2$ local epochs with learning rate $\eta = 0.5$ and batch size $B = 256$. In FRL, each client trains for $E = 5$, $\eta = 1.0$, $B = 16$, and sparsity of $k = 50\%$. Table 3.5 gives the test accuracies of FRL and FedAvg of BiLSTM from on the IMDB dataset. FRL achieves similar accuracy (88.0%) as FedAvg (88.46%) on the IMDB reviews classification task.

3.8.5 FRL against targeted poisoning

So far, we evaluated the robustness of FRL against untargeted attacks. In this section, we evaluate the robustness against targeted poisoning, and specifically against

backdoor poisoning attacks. We consider state-of-the-art backdoor attacks of three types: *semantic* [15], *artificial* [165], and *edge-case*[159] backdoor attacks. In this section, we first briefly discuss the three backdoor attack types; next, we discuss our evaluation setup, and finally, we present the experiment results at the end.

3.8.5.1 Existing FL backdoor attacks

(a) Semantic backdoor (b) Artificial backdoor (c) Edge-Case backdoor

Figure 3.6. Backdoor examples for different categories of targeted poisoning attacks on CIFAR10.

Backdoor attacks [149] aim to misclassify *any* input that contains a specific signal called *backdoor trigger*, while correctly classifying inputs not containing the trigger. Based on the type of triggers, there are three kinds of backdoor attacks:

Semantic backdoor attacks [15] aim to misclassify a specific set of inputs that naturally contain a backdoor trigger. Figure 3.6(a) shows inputs from CIFAR10 data with the background of black stripes on a yellow wall as the naturally present, *semantic* trigger.

Artificial backdoor attacks [15, 165] aim to misclassify any input containing a manually added trigger. Figure 3.6(b) shows inputs with F shaped white pixels as the *artificial* trigger.

Edge-Case backdoor attacks [159] aim to misclassify inputs that are from the tail of the training data distribution. Hence, adversary can compute their model updates using mislabeled edge-case data and manipulate the global model to learn the incorrect correlation between data and labels. Figure 3.6(c) shows edge-case inputs for CIFAR10 distribution which are Southwest airplane images from Internet.

3.8.5.2 Evaluation setup

We compare the performances of the FRL and FedAvg against the above backdoor attacks. We evaluate the accuracy of the (poisoned) global model on the main (inputs without trigger) and backdoor (inputs with trigger) tasks. Following [159, 15], for all the experiments, we start from a pre-trained model with 80.0% test accuracy and train it for 1,000 more FL rounds when malicious clients are present. We use $N = 1000$ clients, distribute CIFAR10 as in Section 3.8, and randomly select $n = 25$ clients in each round. We report the final test accuracy and the average of backdoor test accuracies over the 1000 FL rounds when percentages of malicious clients is in $\{1, 2, 5, 10\}$. We assume that each malicious client has some benign data (per the distribution scheme from Section 3.8) and all the backdoored data. We use the same hyperparameters discussed in Section 3.8 for training CIFAR10 on Conv8 model.

Model Replacement in FL backdoor Attacks: FL backdoor poisoning attacks use a strategy called model replacement where the malicious client first finds a malicious update that contains the backdoor, then it scales the model parameters to cancel the contributions from the other honest clients. For example, if there are m malicious clients selected in FL round t, each malicious client u calculates its backdoored update θ_u^t, and re-scaled it to $\lambda\theta_u^t$ where $\lambda = \frac{m}{n}$ where n is the number of selected clients in each FL round. Model replacement strategy requires that the global model is close to convergence, so the malicious clients can replace the global model with their backdoored model, which performs well on the main task. Following this strategy, the

backdoor accuracy would be very high in FedAvg, even if one of the malicious clients is chosen in one round. However, in FRL, re-scaling is not possible as the clients are sending their local rankings where each ranking are a permutation of the indices of the edges in each layer, i.e., of $[0, n_\ell - 1] \forall \ell \in [L]$ where L is the number of layers and n_ℓ is the number of parameters in ℓth layer. To have a fair comparison with FedAvg, we did not use a model replacement strategy for FedAvg too.

Semantic backdoors: We choose images with vertically striped walls in the background (Figure 3.7 (a)) as the backdoors. Of these 12 images with this trigger in the CIFAR10 dataset, we use 9 of them for training the backdoors while keeping the other three for testing the backdoor accuracy. The malicious clients want the global model to predict these images as the bird (class label=2). We measure the backdoor accuracy on 1000 randomly rotated and cropped versions of the three backdoor images held out of the adversary's training.

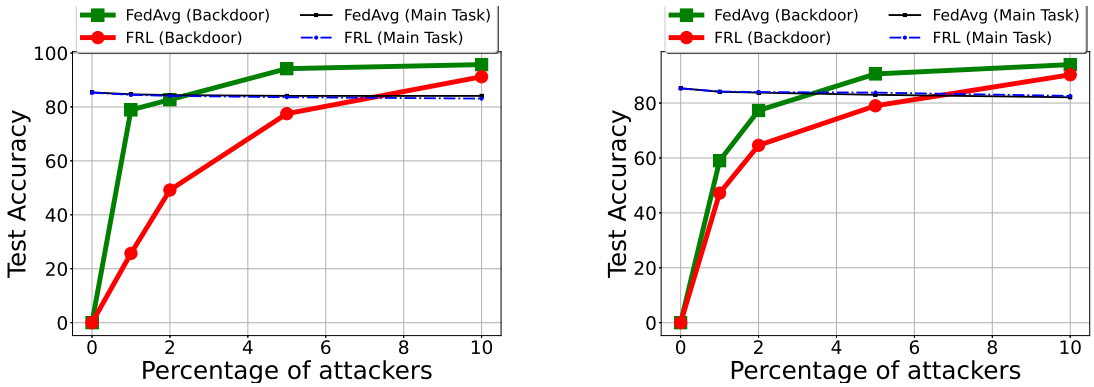
Artificial backdoors: We add a particular pixel pattern to the top left corner of the first nine (not bird) images of the CIFAR10 dataset (Figure 3.7 (b)) and change their labels to bird (label=2). To evaluate these attacks, we pick 256 random (not bird) images from CIFAR10 and add this pattern to them with the label of class bird.

Edge-Case backdoors: We collect 980 images from public web by searching for Southwest airplanes (similar to what [159] did) and resize the images to 32×32 (Figure 3.7 (c)). We set their target labels as truck (class label=9). We use 784 of these images for training and keep 196 of them for the evaluation of the backdoor.

3.8.5.3 Evaluation results

Semantic backdoor attacks: Figure 3.7 (a) shows the performance of FedAvg and FRL on the main task and the backdoor task when different percentages of malicious clients want to put a semantic backdoor in the global model. This figure shows that FRL is more robust against semantic backdoor attacks for different percentages of

malicious clients. For example, with 2% of malicious clients, training FedAvg results in 84.4% final test accuracy with 82.7% average backdoor accuracy, while training FRL results in 84.1% and 49.2% accuracy on the main task and backdoor task, respectively. The existence of a more significant number of malicious clients (e.g., 10%) results in higher backdoor accuracy for both FedAvg and FRL as the malicious clients have more influence on the global model to introduce their backdoor; with existence of 10% of malicious clients, training FedAvg and FRL achieves 95.7% and 91.2% average backdoor accuracy respectively.



(a) Semantic backdoor results

(b) Edge-Case backdoor results

Figure 3.7. FL backdoor poisoning attacks on CIFAR10 distributed over 1000 clients with Dirichlet ($\beta = 1.0$) for presence pf adversary in 1000 FL rounds.

Artificial backdoor attacks: These attacks are ineffective when the adversary cannot use model replacement strategy (i.e., cannot re-scale their parameters). In FRL, malicious clients cannot scale their updates, as they submit a local ranking (from a discrete space of updates). To be fair, we also did not use re-scaling in our experiments for FedAvg. We did not report the results for this attack, as the backdoor accuracy would be 0% for both FRL and FedAvg with no parameter re-scaling. It means that the global model always predict the right label (not the adversary target label "bird") for the test backdoor images.

Edge-Case backdoor attacks: Figure 3.7 (b) shows that FRL is more robust against Edge-case backdoor attacks for different percentages of malicious clients. For example, with 2% of malicious clients, training FedAvg results in 83.7% final test accuracy with 77.3% average backdoor accuracy, while training FRL results in 84.0% and 64.6% accuracy on the main task and backdoor task, respectively. Similar to semantic backdoors, a larger number of malicious clients (e.g., 10%) results in higher backdoor accuracy for both FedAvg and FRL; with 10% of malicious clients, training FedAvg and FRL achieves 94.0% and 90.3% average backdoor accuracy respectively.

3.8.6 FRL with larger number of clients

In the previous sections, we experiment with federated learning by distributing different datasets over 1000 clients in a non-iid fashion using Dirichlet distribution with parameter $\beta = 1$. In this section, we compare the performance and robustness of FedAvg and FRL when we have 1000, 2000, and 5000 FL clients, and we distribute the CIFAR10 dataset over them using Dirichlet distribution with the same parameter $\beta = 1$. CIFAR10 has only ten classes, and its training data contains 50000 images, so when we distribute it over these settings, each FL client gets around 50, 25, and 5 training images on average.

Table 3.6 shows the performance of FedAvg and FRL for different numbers of FL clients (N) and different percentages of malicious clients. For these experiments, we select $n = 25$ clients in each round for local training, and we follow the hyperparameters we discussed in Section 3.8 for training the models. From Table 3.6, we can see that FedAvg always is vulnerable to untargeted attacks, as the adversary has a large continuous space to craft its malicious updates, so the test accuracy of the global model would be a random guess (10%). On the other hand, FRL is robust against untargeted attack even when benign clients have access to a few training examples.

For example, training FRL with 5000 clients achieves 70.8%, 53.3%, 48.9% and 33.8% test accuracy when we 0%, 10% ,20%, 30% of clients are malicious.

Table 3.6. Comparing the robustness of FedAvg and FRL algorithms for large number of FL clients.

Dataset **# of clients**	AGR	Percentage of Malicious Clients			
		0%	10%	20%	30%
CIFAR10 + Conv8	FedAvg	85.4	10.0		
1000 clients	**FRL**	**85.3**	**79.0**	**69.5**	**40.2**
CIFAR10 + Conv8	FedAvg	79.5	10.0		
2000 clients	**FRL**	**79.7**	**75.7**	**66.9**	**38.6**
CIFAR10 + Conv8	FedAvg	68.3	10.0		
5000 clients	**FRL**	**70.7**	**53.3**	**48.9**	**33.8**

3.8.7 Ablation study

In this section, we perform an extensive ablation study to understand performances of FRL under various settings. Specifically, we evaluate the performances of FRL while varying non-iid data distributions methods (Section 3.8.7.1), weight initialization algorithms (Section 3.8.7.2), sparsity (Section 3.8.7.3), and size of supernetwork (Section 3.8.7.4). We conclude with results of FRL with different hyperparameters (Section 3.8.7.5).

3.8.7.1 FRL under different heterogeneous data distribution methods

So far, we evaluated all of our experiments when the data is distributed non-iid using Dirichlet distribution with parameter $\beta = 1$. In this method of non-iid data distribution, all clients will get at least a few samples from each data class with non-zero probabilities that Dirichlet distribution generates. However, this non-iid data distribution need not represent all the practical FL settings. In fact, there may exist non-iid distributions that make training FL models more difficult. Therefore in this section, we consider a more difficult setting where the data distribution is more non-iid.

Assigning only two classes to each client: We experiment with the more extreme non-iid distribution considered by McMahan et al. [126]. More specifically, to distribute of MNIST and CIFAR10 data among 1000 clients, we sort all the (i.e., combined train and test) MNIST and CIFAR10 data according to their classes and then we partition them into 2000 shards. Hence, each shards of training MNIST has 30 images and each CIFAR10 shard has 25 images of a single class. Then we assign two random shards to each client resulting in each client having data from at most two classes. Therefore, in CIFAR10 experiments, each client has 50 training images, and 10 test images, and in MNIST experiments, each client has 60 training images and 10 test images. We only use this assignment in Section 3.8.7.1.

Table 3.7. Comparing the performance of FRL and FedAvg in cross-device FL setting using two non-iid data distribution methods. We distribute data among 1000 clients with two methods described briefly below; please check Section 3.8.7.1 for more details.

Dataset	Type of Non-IID	Metric	Algorithm	
			FedAvg	FRL
MNIST LeNet N=1000	Dirichlet Distribution $\beta = 1$	Mean	98.8	98.8
		STD	3.1	3.1
		Min	75.0	75.0
		Max	100	100
	Randomly 2 classes assigned to each client	Mean	98.4	98.3
		STD	4.3	4.1
		Min	70.0	80.0
		Max	100	100
CIFAR10 Conv8 N=1000	Dirichlet Distribution $\beta = 1$	Mean	85.4	85.3
		STD	11.2	11.3
		Min	33.3	33.3
		Max	100	100
	Randomly 2 classes (assigned to each client)	Mean	70.6	70.9
		STD	21.9	19.2
		Min	0	10.0
		Max	100	100

Table 3.7 shows the performances of FRL and FedAvg using different methods of non-iid assignment. We distribute the data between 1000 clients using: (I) Dirichlet distribution with $\beta = 1$ similar to [145, 86] and (II) the method described above from [126]. In Table 3.7, we note that *FRL achieves similar performances as FedAvg for different heterogeneous data distribution methods.* For instance, on CIFAR10,

Table 3.8. Comparing the performance of FRL with different random weight initialization algorithms with the performance of vanilla FedAvg for cross-device setting. Using Singed Kaiming Constant (U_K) as weight initialization gives the best performance for all the datasets.

Dataset	Metric	Algorithm			
		FedAvg	FRL		
	$W_{init} \sim$	-	$\mathcal{X}_N$	$\mathcal{N}_K$	U_K
MNIST LeNet N=1000	Mean	98.8	96.6	98.7	98.8
	STD	3.1	5.2	3.2	3.1
	Min	75.0	57.1	75.0	75.0
	Max	100	100	100	100
CIFAR10 Conv8 N=1000	Mean	85.4	63.6	82.0	85.3
	STD	11.2	15.6	11.9	11.3
	Min	33.3	0	0	33.3
	Max	100	100	100	100
FEMNIST LeNet N=3400	Mean	85.8	69.2	82.9	84.2
	STD	10.2	14.2	11.1	10.7
	Min	10.0	0	14.3	7.1
	Max	100	100	100	100

FedAvg and FRL achieves similar performances of 85.4% and 85.3% respectively when data is distributed according to (I). Similarly, when data is distributed according to (II), FedAvg and FRL achieve similar performances of 70.6% and 70.9%, respectively.

We make similar observations for MNIST as well: FedAvg achieves 98.8% and 98.4% for the two methods of data distribution respectively, while FRL achieves 98.8% and 98.3% accuracy.

3.8.7.2 FRL under different weight initializations

In FRL, the weight parameters are randomly initialized at the start and remain fixed throughout the training. An appropriate initialization is instrumental to the success of FRL, since the FRL clients are sending the local rankings of these edges; more important edges get higher ranks. They generate these rankings by feeding the subnetwork of top rank edges and calculating the gradient of the loss with respect to the scores, so distribution of these random weights has a high impact on the calculated loss. We use three different distribution for initializing the weight parameters as follows:

Glorot Normal [71] where we denote by $\mathcal{X}_N$. Previous work [176] used this initialization to demonstrate that subnetworks of randomly weighted neural networks can achieve impressive performance.

Kaiming Normal [80] where we denote by $\mathcal{N}_k$ defined as $\mathcal{N}_K = \mathcal{N}\left(0, \sqrt{2/n_{\ell-1}}\right)$ where $\mathcal{N}$ shows normal distribution. n_ℓ shows the number of parameters in the ℓth layer.

Singed Kaiming Constant [143] where all the weights are a constant σ but they are assigned $\{+, -\}$ randomly. This constant, σ, is the standard deviation of Kaiming Constant. We show this initialization with U_K as we are sampling from $\{-\sigma, +\sigma\}$ where $\sigma = \left(\sqrt{2/n_{\ell-1}}\right)$.

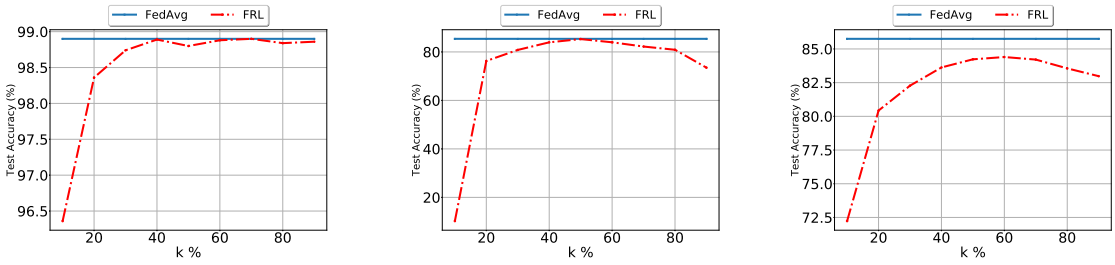
Table 3.8 shows the results of running FRL for three datasets under the three aforementioned initialization algorithms. We compare FRL with FedAvg and report the mean, standard deviation, minimum, and maximum of the accuracies for the clients' accuracies in FRL and FedAvg at the end of training. As we can see under three different random initialization, using Signed Kaiming Constant (U_K) results in better performance. We note from Table 3.8 that FRL with Signed Kaiming Constant (U_K) initialization achieves performance very close to the performance of FedAVg.

Note that, since the FRL clients update scores in each round, unlike initialization of weights, initialization of scores does not have significant impact on the final global subnetwork search. Therefore, we do not explore different randomized initialization algorithms for scores and simply use Kaiming Uniform initialization for scores.

Ramanujan et al. [143] also considered these three initialization to find the best subnetwork in centralized machine learning setting. They also showed that using Singed Kaiming Constant gives the best supermasks. Our results align with their conclusions, hence we use Singed Kaiming Constant to initialize the weights and Kaiming Uniform to initialize the scores of the global supernetwork.

3.8.7.3 FRL with varying sizes of subnetworks

In FRL, each client uses Edge-popup (Algorithm 1) and their local data to find a local ranking by finding a subnetwork within a randomly initialized global network, which we call *supernetwork*. Edge-popup algorithm uses parameter k which represents the % of all the edges in a supernetwork which will remain in the final subnetwork. For instance, $k = 50\%$ denotes that each client finds a subnetwork within a supernetwork that has half the number of edges as in the supernetwork.



(a) MNIST (b) CIFAR10 (c) FEMNIST

Figure 3.8. Comparing performance of FRL for different subnetwork sizes. k (x-axis) shows the % of weights that each client is including in its subnetwork, test accuracy (y-axis) shows the mean of accuracies for all the clients on their test data. The chosen clients in each round send all the ranks to the server. FRL with subnetworks of $\in [40\%, 70\%]$ result in better performances.

Figure 3.8 illustrates how the performance of FRL varies with the sizes of local subnetworks that the clients share with the server. In other words, when we vary the sparsity $k\%$ of edge popup algorithm during local subnetwork search $k \in [10, 20, 30, 40, 50, 60, 70, 80, 90]\%$. Interestingly we note that, FRL performs the worst when clients use all (k=100%) or none (k=0%) of the edges. This is because, it is difficult to find a subnetwork with small number of edges. While using all of the edge essentially results in using a random neural network. As we can see FRL with $k \in [40, 70]\%$, gives the best performances for all the three datasets. Hence, we set k=50% by default in our experiments.

3.8.7.4 FRL with larger networks

We already discussed the performance of the FRL for MNIST, CIFAR10, and FEMNIST datasets using moderately large neural networks with number of parameters ranging from 1.62M to 5.27M. In this section, we conduct experiments to demonstrate the effectiveness of FRL for very large networks: ResNet18 and ResNet34, with number of parameters 11.11M and 21.26M, respectively. Along with CIFAR10, we conduct experiments using a larger and significantly more challenging Tiny-ImageNet dataset. Table 3.1 shows the network architectures we use for these experiments. We have a batch normalization layer in these networks after each convolution layer. Following the setting of [143, 164, 42], we set the batch normalization to the non-affine mode, i.e., the scale and bias terms are set to $\gamma = 1$ and $\beta = 0$. We use the hyperparameters described in Section 3.7.2 for all the models.

Table 3.9. FRL with larger networks for CIFAR10 and Tiny-ImageNet distributed over 1000 FL clients.

Dataset	Network	# of Params	FRL Acc (%)	FedAvg Acc (%)
CIFAR10 $N = 1000$	Conv8	5.27M	85.3	85.4
	ResNet18	11.11M	89.3	89.9
	ResNet34	21.26M	90.2	91.4
Tiny-ImageNet $N = 1000$	ResNet18	11.26M	31.0	30.8
	ResNet34	21.36M	30.9	30.8

Tiny-ImageNet Table 3.9 shows the performance of FRL and FedAvg on Tiny-ImageNet for different network sizes, where ResNet18 and ResNet34 have 11.26 and 21.36 millions parameters respectively. From this table, we can see that FRL can achieve similar accuracy as FedAvg; for example, by using ResNet18, FRL achieves 31.0% test accuracy while FedAvg achieves 30.8% test accuracy.

CIFAR10. In Table 3.9, we show the final test accuracy of different models, Conv8 with 5.27M parameters, ResNet18 with 11.11M parameters, and ResNet34 with 21.26M parameters. We can see that FRL can achieve similar test accuracies as FedAvg for

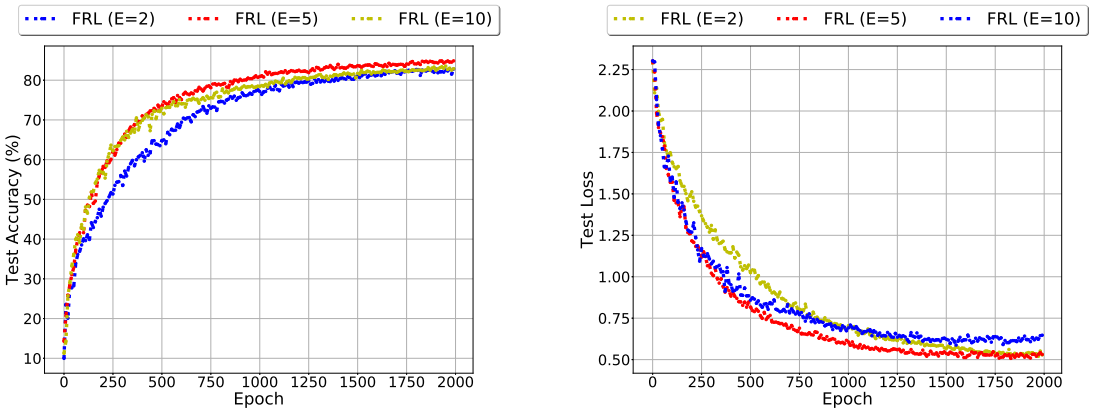
different model sizes. For example, For ResNet18, FRL achieves 89.3% test accuracy while FedAvg achieves 89.9% test accuracy.

3.8.7.5 FRL with different hyperparameters

Note that, we independently tune the hyperparameters for FRL and other baselines. More specifically, we tune batch size, local epochs, and learning rate for the non-adversarial (no malicious clients) setting and use them throughout our experiments. However, we show that FRL, unlike other robust aggregations, defends against malicious clients for a wide range of the hyperparameters: Table 3.10 shows the mean and standard deviations of accuracy of FRL and various baseline robust FL algorithms on CIFAR10 (distributed over 1000 users using Dirichlet distribution [129]) for wide ranges of the hyperparameters when there are 10% malicious clients. This experiment shows that hyperparameter tuning, a major challenge in real-world FL systems [97], is relatively easy for FRL.

Table 3.10. FRL performance is robust to a wide range of hyperparameters. FRL performs well on CIFAR10 (distributed non-iid among 1000 clients using Dirichlet distribution) even under different hyperparameters. We use the values in **bold** in our experiments. FedAvg and TopK are non-robust under any combination of hyperparameters.

Method	hyperparameter	value	Test Accuracy with 10% malicious
FRL	batch size	6	78.4 (12.6)
		8	79.0 (12.4)
		16	76.4 (13.6)
	local epochs	2	79.8 (12.2)
		5	79.0 (12.4)
		10	78.2 (12.6)
	learning rate	0.1	73.5 (13.4)
		0.2	82.4 (12.1)
		0.3	83.11 (11.8)
		0.4	**79.0 (12.4)**
		0.5	77.5 (13.1)
FedAvg	-	-	10.0 (10.1)
TopK	-	-	10.0 (10.1)
Trimmed-mean	batch size	6	55.5 (14.5)
		8	**56.3 (16.0)**
		16	37.7 (15.6)
	local epochs	2	41.0 (15.4)
		5	**56.3 (16.0)**
		10	21.0 (9.9)
	learning rate	0.01	34.0 (15.5)
		0.05	38.3 (15.3)
		0.1	**56.3 (16.0)**
		0.15	10.0 (10.0)
		0.2	10.0 (10.0)
Multi-Krum	batch size	6	19.0 (12.5)
		8	**58.8 (15.8)**
		16	36.7 (14.8)
	local epochs	2	46.1 (15.9)
		5	**58.8 (15.8)**
		10	24.3 (11.7)
	learning rate	0.01	15.3 (11.7)
		0.05	50.0 (16.2)
		0.1	**58.8 (15.8)**
		0.15	15.4 (11.9)
		0.2	10.0 (10.0)
SignSGD	batch size	6	33.1 (15.6)
		8	**39.7 (15.9)**
		16	10.2 (10.1)
	local epochs	2	10.2 (10.5)
		5	**39.7 (15.9)**
		10	41.5 (16.0)
	learning rate	0.01	44.2 (15.8)
		0.05	41.9 (15.5)
		0.1	**39.7 (15.9)**
		0.15	35.8 (15.3)
		0.2	10.2 (10.1)



(a) CIFAR10 (Test Accuracy)

(b) CIFAR10 (Test Loss)

Figure 3.9. Comparing the CIFAR10 test accuracy and losses of FRL for different number of local epochs.

Figure 3.9 shows the learning curves of FRL for different numbers of local epochs for the CIFAR10 experiment; for CIFAR10, all our experiments use 1000 clients with local data distributed in non-iid fashion using Dirichlet distribution [129]. We note that using five local epochs gives the best results. Table 3.11 shows the effect of other settings including different number of participants (n), local epochs (E), and non-iid degree (β) on performance of FRL trained on CIFAR10. The **bold** values are the once we finally use in our experiments.

Table 3.11. The effect of other settings on performance of FRL trained on CIFAR10 distributed over 1000 clients using Dirichlet distribution. The **bold** shows the value we used in our experiments.

Method	hyperparameter	value	Test Accuracy (10% malicious)
FRL	Number of participants (n)	15	84.8 (11.3)
		25	**85.3 (11.3)**
		50	84.9 (11.2)
	local epochs (E)	2	82.2 (12.0)
		5	**85.3 (11.3)**
		10	83.5 (11.9)
	Non-iid degree (β)	**1**	**85.3 (11.3)**
		10	85.6 (11.1)
		100	85.6 (10.9)

3.9 Conclusions

We designed a novel collaborative learning algorithm, called Federated Rank Learning (FRL), to address the issues of robustness to poisoning and communication efficiency in existing FL algorithms. We argue that a core reason for the susceptibility of existing FL algorithms to poisoning is the large continuous space of values in their model updates. Hence, in FRL, we use ranks of edges of a randomly initialized neural network contributed by collaborating clients to find a global ranking and then use a subnetwork based only on the top edges. Use of rankings in a fixed range restricts the space available to poisoning adversaries to craft malicious updates, and also allows FRL to reduce the communication cost.

We show, both theoretically and empirically, that ranking based collaborative learning can effectively mitigate the robustness issue as well as reduce the communication costs involved.

CHAPTER 4

HETEROGENEOUS PRIVATE INFORMATION RETRIEVAL

Another aspect of trustworthiness in distributed learning systems that we investigate is access privacy. In this chapter, we design a general approach for preserving access privacy, that one of its application could be in distributed learning systems.

Private information retrieval (PIR) is a technique to provide query privacy to users when fetching sensitive records from untrusted databases. There are two major types of PIR protocols. The first type is *computational PIR* (CPIR) [44, 107, 9, 8, 30, 32, 57, 98, 120, 155, 14] in which the security of the protocol relies on the computational difficulty of solving a mathematical problem in polynomial time by the servers, e.g., factorization of large numbers. Most of the CPIR protocols are designed to be run by a single database server, and therefore to minimize privacy leakage they perform their heavy computations on the whole database (even if a single entry has been queried). Consequently, existing CPIR protocols suffer from very high computation overheads. The second major class of PIR is *information-theoretic PIR* (ITPIR) [51, 82, 20, 45, 67, 72, 19, 54]. ITPIR protocols provide information-theoretic security, however, existing designs need to be run on more than one database servers, and they need to assume that the servers do not collude. Existing ITPIR protocols impose lower computation overheads compared to CPIR algorithms, but at the price of requiring the non-collusion assumption for the servers. Therefore, (multi-server) ITPIR protocols are best fit to scenarios involving multiple (potentially competing) data owners who collaborate to run a service privately, therefore colluding is not in

their best interest, e.g., [111, 130, 29]. Our work focuses on this class of PIR, i.e., multi-server PIR protocols.

Existing multi-server PIR protocols are homogeneous! The existing body of work on multi-server PIR considers a setting in which *the non-colluding PIR servers have similar computation and communication constraints*. We call such traditional multi-server PIR protocols *homogeneous*. Homogeneous PIR algorithms have been deployed in a wide range of homogeneous applications; this includes registering Internet domains [136], retrieving information of Tor relays [130], private media delivery [75], privacy-preserving e-commerce applications [83], private query in open-access eprint repositories [76], messaging applications [29], private online notification [141], and private file-sharing applications [111]. For instance, in PIR-Tor [130] the servers participating in the protocol are Tor directory servers with similar resources, in DP5 [29] the servers are messaging servers with similar settings, and in rPIR [111] the servers are p2p file-sharing seeds with similar resources. In all of these settings, the proposed multi-server PIR protocols *impose symmetric computation and communication loads on all of the servers* involved in the multi-server PIR protocol.

Introducing heterogeneous multi-server PIR. In this work, we introduce a new class of multi-server PIR, which we call *heterogeneous PIR (HPIR)*. An HPIR protocol is a multi-server PIR protocol with *asymmetric computation and communication constraints on its servers*, i.e., some of its servers handle higher computation/communication overheads than the others. We argue that *HPIR algorithms enable new applications for PIR, as well as improve the utility of some of the existing applications of PIR*; this is because HPIR allows the participation of low-resource entities in running private services.

Table 4.1. List of PIR notations

ℓ	Number of servers
t	Privacy threshold (max number of colluding servers)
k	Number of server's responses
$\mathbb{D}$	Database matrix
r	Number of rows in the database
s	Number of elements in each record of the database
w	Element size (bits)
N	Total size of the database (bits)

4.1 Background

Private information retrieval (PIR) is a technique to provide query privacy to users when fetching sensitive records from untrusted databases. The existing body of work on multi-server PIR considers a setting in which the non-colluding PIR servers have similar computation and communication constraints. We call such traditional multi-server PIR protocols homogeneous. In this thesis, we present a new class of multi-server PIR protocols, which we call heterogeneous PIR (HPIR). In HPIR, the computation and communication overheads imposed on the PIR servers are non-uniform, i.e., some servers handle higher computation/communication burdens than others. This enables heterogeneous PIR protocols to be suitable for a range of new PIR applications. What enables us to enforce such heterogeneity is a unique PIR-tailored secret sharing algorithm that we leverage in building our PIR protocol.

In this section, we introduce the main concepts of PIR. Table 4.1 shows the notations we use for PIR protocols.

Database as a Matrix In a PIR protocol, one or multiple servers, called *PIR servers*, host a database $\mathbb{D}$, which can be represented as an r-by-s matrix over a finite field $\mathbb{F}$. The goal of a *client* (*querier*) is to retrieve one row of $\mathbb{D}$, called a *data record*, through some interactions with the PIR servers in a way that the PIR servers do not learn which record of $\mathbb{D}$ was retrieved by the client.

$$\mathbb{D} = \begin{bmatrix} \mathbb{D}_{1,1} & \mathbb{D}_{1,2} & \dots & \mathbb{D}_{1,s} \\ \mathbb{D}_{2,1} & \mathbb{D}_{2,2} & \dots & \mathbb{D}_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{D}_{r,1} & \mathbb{D}_{r,2} & \dots & \mathbb{D}_{r,s} \end{bmatrix} \tag{4.1}$$

Non-private Information Retrieval Suppose that the client aims at retrieving the jth record of the database. She will create a unit vector $\vec{e}_j$ of size r where all the bits are set to zero except the jth position being set to one:

$$\vec{e}_j = \begin{bmatrix} 0 & 0 & \dots & 1 & \dots & 0 & 0 \end{bmatrix} \tag{4.2}$$

If the client did not care about privacy, she would send $\vec{e}_j$ to the server(s), and the server(s) could generate the client's response by multiplying the vector into the database matrix $\mathbb{D}$:

$$\vec{e}_j.\mathbb{D} = \begin{bmatrix} \mathbb{D}_{j,1} & \mathbb{D}_{j,2} & \dots & \mathbb{D}_{j,s} \end{bmatrix} \tag{4.3}$$

Private Information Retrieval A PIR technique allows the client to obtain this response without revealing $\vec{e}_j$ to the server(s). Existing PIR techniques use two main approaches to obfuscate $\vec{e}_j$: (a) Homomorphic encryption: In such protocols [155, 107, 9, 8, 14], the client encrypts $\vec{e}_j$ element by element before being sent to the servers. During the data recovery phase, the client will extract her intended record by decrypting the components of $\vec{e}_j \times \mathbb{D}$. (b) Secret Sharing: In other PIR protocols [72, 82, 81, 54, 111], the client will use secret sharing to generate different vector shares for $\vec{e}_j$, and she will send the shares to the PIR servers. Most of the existing single-server, CPIR protocols use homomorphic encryption, and most of the existing multi-server, ITPIR protocols use secret sharing.

4.2 Related Works

Information-Theoretic PIR (ITPIR) protocols ITPIR protocols require more than one server, and there is an assumption that these servers are not colluding. These protocols have two main advantages, first they are fast since they do not use complicated cryptography operations. Second, the query is information-theoretic private i.e. the adversary can not learn anything about the queries even though she has unlimited computation power. This kind of PIR requires low computational resources compared to the CPIR protocols.

Chor ITPIR Chor et al. [45] introduced a very basic ITPIR which uses exclusive OR as the main operation. One advantage of XOR is that it can cancel the effect of repeated elements, so the client makes her queries in a way that all the records have an even number of repeats, so at the end, she can cancel their effects. In this protocol the client create ℓ queries that $\ell - 1$ of them are totally random and the last one is the result of XOR of the first $\ell - 1$ vectors and $\vec{e}_j$. Then it will send them to PIR servers, each server multiplies these vectors to the database in GF(2). This dot production in GF(2) is simple XOR, which "1" in position j^{th} of $\vec{e}_j$ means XOR this record, and "0" means do not XOR this record. At the end, each server sends back the result of the XOR to the client, and the client XOR all the responses.

Although this scheme is the first ITPIR protocol, it is still widely cited and proposed for different applications. This amount of citations comes from the fact that this scheme is very fast compared to all other PIR protocols.

Robustness Most of the PIR protocols use "Honest-but-Curious" adversarial model. This model assumed that all the servers are honest, it means they always respond a correct answer, but they try to infer which record has been fetched. One of the main issues with ITPIR is that how the client should deal with servers that do not respond at all or send an incorrect answer that makes the client's result incorrect. A t-private ℓ-server PIR is a private information retrieval protocol which information-theoretically

protects the privacy of the client's queries when less than $t+1$ servers collude. Beimel et al. [20] explore the situation in which some servers cannot respond, but the client still can retrieve the data. They define a t-private k-out-of-ℓ PIR as a PIR in which the client only needs k answers out of ℓ servers to recover her record, and if up to t servers collude then the client still has privacy. They also examine what happens if v servers reply incorrect answers, and how many correct answers the client needs to recover the record successfully. They defined t-private v-byzantine-robust k-out-of-ℓ PIR as a PIR that can handle a v number of byzantine servers which sends incorrect answers to corrupt the client's result.

PIR using secret sharing [18] and [17] show that secret sharing and secret conversion can be used to construct a private information retrieval. Li et al. [111] propose four different multi-query ITPIR protocols based on ramp secret sharing schemes [106, 166], and they call them ramp secret sharing-based PIR (rPIR). [81] proposes new techniques that increase efficiency of multi-server ITPIR protocols based on ramp secret sharing. This paper shows ramp secret sharing can help in encoding the data similar to encoding the query. They encode each record of the database into multi shares of a secret, and the client can recover the record by sending multi queries for these shares.

Computational PIR (CPIR) protocols Most of CPIR protocols use a single server which is computationally bounded for retrieving data. It means the security of these protocols is based on a very difficult mathematical problem, and if the adversary finds a solution for the problem, or if she has enough time and computation resources, some data will be leaked.

Stern et al. CPIR Stern et al. [155] propose a CPIR protocol that its algorithm is based on additively homomorphic cryptosystem which has these functions: (i) *Gen*: function of generating public and private keys pk, sk and system parameters , (ii) *Enc*: Encryption function with public key ,and (iii) *Dec*: Decryption function. The

important point in this scheme is that the cryptosystem is non-deterministic i.e. the *Enc* is a randomize function that encrypts the same input to different output each time. *Dec* function will cancel the effect of the random variable that was used in *Enc* function. The most famous non-deterministic cryptosystem that is used for CPIR is Paillier [138].

XPIR [8] proposes a new CPIR based on this protocol by looking at each record of the database as a polynomial (the elements of the databases are encoded as coefficients of the polynomial). However, the bandwidth consumption is not very good, so SealPIR [14] introduce a way to compress the queries in this system.

Kushilevitz and Ostrovsky CPIR One of the merits of CPIR is that the client can run the protocol recursively to reduce the bandwidth consumption. This idea was used by Kushilevitz and Ostrovsky [107] in their scheme for improving the communication cost. First, they split the database into several virtual blocks, and each one of these blocks contains some of the real blocks, then the client sends her query for a specific virtual block, then server will calculate the result of that query on the database and will look at the result as a new database, the next query will be applied to this temporary database, and this process continues until one vector of size s will be sent to the client, and client can recover her requested index out of this result.

Aguilar-Mechor et al. CPIR Aguilar-Mechor et al. [9] propose a lattice-based PIR, and their security is based on the differential hidden lattice problem which is an NP problem. Olumofin and Goldberg [137] show that this design is an order of magnitude faster than trivial download which downloads the entire database.

4.3 Preliminaries

4.3.1 Preliminaries on secret sharing

The goal of secret sharing is to split a secret into multiple *shares* (e.g., by a trusted *dealer*) such that the secret can be reconstructed by combining some of the shares.

The dealer distributes these shares among multiple *shareholders* who participate in the protocol. A $(t+1, \ell)$ *threshold secret sharing scheme* distributes a secret among ℓ shareholders in a way that any coalition of up to t shareholders can not learn anything about the secret, while a coalition of more than t shareholders will fully reconstruct the secret. A scheme is called *multi-secret sharing* [167, 41, 139] if it shares multiple secrets in each round of the protocol.

Notations: Table 4.2 lists the notations we use for secret sharing algorithms.

Table 4.2. List of notations used in secret sharing schemes

ℓ	Number of participants
t	Privacy threshold
ρ	Large prime number
q	Number of secrets
$\mathbb{S} = \{s_1, s_2, \ldots, s_q\}$	Secrets

4.3.2 Key secret sharing designs

We introduce two secret sharing schemes that have been the bases of state-of-the-art PIR protocols. Our secret sharing algorithm, introduced later, is built upon these schemes.

Shamir Secret Sharing: Shamir's scheme [147] is a $(t+1, \ell)$-threshold scheme, in which the shares are the points of a polynomial function. Specifically, a secret s is shared as follows:

I The dealer chooses a random polynomial function $f(x) \in_r \mathbb{F}_\rho$ of degree t, where $f(0) = s$ is the secret.

II The dealer chooses ℓ x-coordinates $\{x_1, \ldots, x_\ell\}$ uniformly at random, where $x_i \in_r \mathbb{F}_\rho$ for $1 \leq i \leq \ell$.

III The dealer sends $(x_i, f(x_i))$ to the i^{th} shareholder.

Any coalition of $k > t$ shareholders can recover the secret s from their shares by using Lagrange polynomial interpolation. Therefore, given $k > t$ shares $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ the shared secret is reconstructed as

$$s = \sum_{m=1}^{k} y_m \Big(\prod_{n=1, n \neq m}^{k} x_n (x_n - x_m)^{-1} \Big) \; mod(\rho) \tag{4.4}$$

On the other hand, for a coalition of $k \leq t$ shareholders, any $s \in \mathbb{F}_\rho$ has the same probability of being the secret.

Ramp Variant of Shamir Secret Sharing: While a Shamir $(t+1, \ell)$ threshold scheme shares only one secret using a t-degree polynomial, a $(t+1, q, \ell)$-ramp secret sharing [26] uses a $(t+q-1)$-degree polynomial to share q secrets simultaneously with the privacy level of t. That is, the dealer shares q secrets $\{s_1, \ldots, s_q\}$ from $\mathbb{F}_\rho$ among ℓ participants in a way that any coalition of $t+q$ or more participants can retrieve *all* of the q secrets, but any set of participants up to t cannot learn anything about the secrets. However, for $t < k < t+q$ participants, the joint distribution of the secrets is not uniform, therefore it leaks information about the secrets. The dealer takes the following steps to share q secrets $\{s_1, \ldots, s_q\}$:

I Chooses $\{y_{q+1}, \ldots, y_{q+t}\}$ randomly from $\mathbb{F}_\rho$

II Finds a polynomial $f(x)$ with degree at most $t+q-1$ that contains the following points:

$$(1, s_1), \ldots, (q, s_q), (q+1, y_{q+1}), \ldots, (q+t, y_{q+t})$$

III Sends secret share $(x_i, f(x_i))$ to the i^{th} shareholder for $1 \leq i \leq \ell$ (x_is are random numbers from $\mathbb{F}_\rho$).

To retrieve the secrets, any $k \geq t+q$ shares can give away the secrets using Lagrange interpolation:

$$s_j = \sum_{m=1}^{k} y_m(\prod_{n=1, n\neq m}^{k} (j - x_n)(x_m - x_n)^{-1}) \; mod(\rho) \quad 1 \leq j \leq q \tag{4.5}$$

4.3.3 Key PIR Designs

Here we overview the two key multi-server PIR designs that are the most relevant to our work.

Goldberg ITPIR using Shamir Secret Sharing Goldberg's PIR [72] is an ITPIR scheme. The client uses Shamir's secret sharing [147] to split the unit vector $\vec{e}_j$ into ℓ shares (each a vector of size r), where the shares are sent to ℓ servers. Each server will send back the multiplication of its received share into the database matrix $\mathbb{D}$. Finally, the client will interpolate the query responses component-wise at $x = 0$ to extract her interested row of the database.

Henry et al. ITPIR using Ramp Variant of Shamir Secret Sharing Henry et al. [82] modify Goldberg's PIR [72] by replacing Shamir's $(t + 1, \ell)$-threshold secret sharing with a $(t + 1, q, \ell)$-ramp secret sharing [26]. This enables a client to encode q secrets in a $(t + q - 1)$ degree polynomial, as opposed to only one secret in a t degree polynomial in [72]; therefore, the protocol is able to query multiple queries from the PIR servers at any round of the PIR protocol. To query the q records, the client will receive $(t + q)$ responses from the PIR servers.

Note that existing multi-server PIR protocols can *not* be trivially extended to heterogeneous constructions. One can modify a single-query PIR design like Goldberg's PIR [72] to a heterogeneous one by sending more than one query share to some of the PIR servers; however, this will increase the bandwidth/computation overhead on some of the servers (who receive multiple shares) *without reducing the overhead on any of the PIR servers*. The goal of HPIR is to reduce the overhead on resource-constraint servers (through increasing the overhead on resourceful servers). Therefore, for our HPIR protocol, we design a *PIR-tailored multi-secret sharing algorithm* that enables us to

split a query non-uniformly between multiple PIR servers. The goal of PIR-tailored multi-secret sharing is different from general existing secret sharing schemes. The main purpose of PIR-tailored secret sharing is splitting query in a PIR design. In Section 4.5, we will discuss the main differences between a general secret sharing scheme and PIR-tailored secret sharing.

4.4 Introducing Heterogeneous PIR

There are two main classes of PIR protocols based on the number of servers that deploy the protocol: single-server PIR, and multi-server PIR. Note that this is a different classification than computational PIR (CPIR) versus information-theoretic PIR (ITPIR), but all single-server PIR protocols fall in the category of computational PIR (CPIR) [44, 107, 9, 8, 30, 32, 57, 98, 120, 155, 14], as proved by Chor et al. [45]. The security of multi-server PIR relies on assuming that the multiple PIR servers do not collude; this allows multi-server protocols to impose lower computation overheads than single-server protocols. Consequently, single-server and multi-server protocols are suited to different application scenarios.

Existing multi-server PIR constructions [82, 20, 45, 67, 72, 168, 19, 54, 51] impose *uniform* computation and communication overheads on their (non-colluding) multiple PIR servers; therefore, we call them homogenous. In this work, we introduce *heterogeneous PIR (HPIR)*,[1] which is a subclass of multi-server PIR protocols. An HPIR protocol is a multi-server PIR protocol with *asymmetric computation and communication constraints on its servers*. That is, some of its servers (called *rich servers*) handle higher computation/communication overheads than the others (called *poor servers*).

[1]Note that some previous work use HPIR to refer to hybrid PIR [24], another class of PIR.

4.4.1 Other potential applications scenarios

We believe that HPIR protocols will enable new application scenarios for multi-server PIR, as well as improve the usability of some of the known applications of PIR. To support this claim, in this section we present several potential applications scenarios for heterogeneous PIR algorithms.

Note that for some of these applications, one could instead use a single server PIR, however, existing single-server PIRs are too slow for most of the in-the-wild applications. Also, note that we only present the intuitions on why HPIR will fit these application scenarios; integrating HPIR in each of the following scenarios will require additional engineering effort (e.g., to synchronize the PIR servers), which is out of our scope.

4.4.1.1 Privacy from content delivery networks (CDN)

Content publishers increasingly use CDNs to improve the security and performance of their services. However, to do so, they have no choice but revealing their clients' communications to the CDN operators. For instance, a CDN hosting a banking service will see the private information of the bank's clients (as the bank will provide the CDN with her certificate private keys). We suggest to deploy PIR on CDN servers to enable private content retrieval by clients.

Existing single-server PIR protocols are too slow to be used in this application (and most other proposed applications of PIR); we therefore suggest a heterogeneous 2-servers PIR protocol to be used for this application. This is illustrated in Figure 4.1: the CDN edge servers act as the "rich" servers and the content publisher's origin servers act as the "poor" servers. This heterogenous setting is ideal for this application: The CDN edge servers are often much closer to the clients and are designed to be capable of handling very large traffic volumes. By contrast, content publisher servers aim to minimize their communication and computation loads (in fact, this is one of the

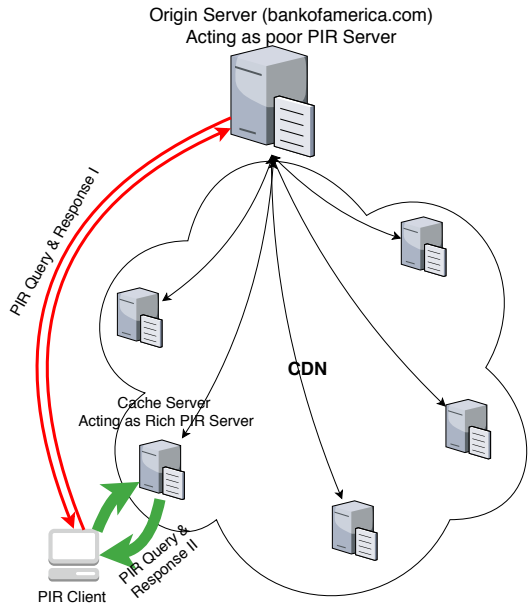


Figure 4.1. Illustrating how a heterogeneous PIR scheme can enable private content delivery by CDNs.

key reasons for using CDNs for content hosting). As mentioned before, **all existing multi-server PIR protocols are homogeneous**. This scenario demonstrates the need for heterogeneous PIR protocols that impose *non-uniform computation and communication loads* on the multiple PIR servers running the protocol.

Needless to say, we can assume that the PIR servers do not collude in this setting as collusion is against the best interest of content publishers.

4.4.1.2 Private P2P file sharing

Various popular services such as Spotify, PPTV, and BitTorrent use their clients for content distribution. Recent work [111] has suggested to use PIR to protect privacy in such services, particularly for BitTorrent and Spotify. We argue that an HPIR protocol will significantly improve the usability of PIR for such applications. This is because in these systems, the peers are located in diverse geographic locations, and have different computation and communication resources. Therefore, when a

heterogeneous PIR is deployed in this setting, a client can obtain larger traffic volumes from nearby seeding peers compared to distant peers (while protecting privacy through PIR), therefore improving the overall download experience.

4.4.1.3 Query privacy in cache networks

A multitude of next-generation network architectures like Named Data Networking (NDN) [173] cache content objects to improve the overall utility of the network. A key privacy challenge to the design of cache networks like NDN is the privacy of queries against cache routers. That is, a cache router will learn the content names requested by a client in order to be able to serve her. We propose to use PIR as a mechanism to enforce cache privacy in cache networks. Our proposal is to have the cache routers serve as PIR servers, and store named objects into a PIR database. A client interested in a particular named object will need to query the cache routers through a PIR protocol in order to preserve her query privacy. We therefore suggest multi-server PIR protocols to be used for this application. However, existing multi-server designs rely on the assumption that PIR servers should not collude. Therefore, the two (or more) cache routers queried by a client should be under different jurisdictions, i.e., run by competing Internet entities. For instance, in a 2-server setting, the first PIR server queried by the client can be the edge router of the client, and the second router can be a router in a non-peer AS or the content publisher itself (therefore non-colluding with the edge router).

As can be seen, in this setting, the edge router (e.g., the client's default gateway) can tolerate much higher bandwidth and computation burden than the distant router/publisher—in fact, the whole purpose of information centric networks is to reduce transition loads by caching content on local routers. Therefore, the deployed PIR protocol needs to be heterogenous.

4.5 Our PIR-tailored secret sharing algorithm

In this section, we define and design a PIR-tailored secret sharing scheme that we use in the design of our PIR protocol.

Why a new scheme? Similar to the state-of-the-art multi-server PIR schemes [72, 82, 111], our proposed HPIR scheme uses secret sharing to split the query vector between PIR servers. To enable heterogeneity, an HPIR protocol needs to use a *multi-secret sharing* algorithm, as introduced earlier. This will allow an HPIR protocol to split PIR computations and communications unevenly between PIR servers. Note that some prior PIR protocols by Henry et al. [82] and Li et al. [111] have used multi-secret sharing algorithms. However, a ramp secret sharing algorithm is not suitable for heterogeneous PIR: in a ramp secret sharing scheme, the number of required servers increases with the number of shared secrets; therefore, for an HPIR protocol based on a ramp secret sharing scheme, the number of PIR servers increases with the degree of heterogeneity. However, most of the practical application scenarios of HPIR (as introduced in Section 4.4) need to be deployed on two servers, as they comprise two non-colluding parties (e.g., a content publisher and a CDN provider). Therefore, *we design a PIR-tailored multi-secret sharing algorithm in which the number of shareholders does not increase with the number of shared secrets.* This allows our HPIR protocol that uses this PIR-tailored secret sharing to be run by as few as only 2 servers, regardless of the degree of heterogeneity.

What is PIR-tailored secret sharing? In A PIR-tailored secret sharing scenario, a dealer wants to share one or multiple secrets from values of $\{0, 1\}$ among ℓ shareholders. The goal of PIR-tailored secret sharing is different from the goal of general secret sharing schemes. A PIR-tailored secret sharing is not designed to enable recovering the secrets by the shareholders since all the shareholders are adversary (e.g., PIR servers), and it is only the dealer that should have access to the secrets. In a PIR-tailored secret sharing, the dealer wants to get some information from shareholders, so the dealer will

break his query into shares, and send them to the shareholders. This scheme will be used as the core of our PIR, and it does not have an application by itself. Similar to previous PIR designs based on secret sharing, the client will construct a secret sharing for each record of the database to share secrets from $\{0, 1\}$. For a secret, value 0 shows that the client is not interested in retrieving that record, and value 1 shows he wants to retrieve that record. Another main difference is that in PIR-tailored secret sharing schemes, only one of the secrets in each PIR-tailored polynomial can be 1, and the rest of them are 0.

4.5.1 The differences between secret sharing and PIR-tailored secret sharing

There are four main differences between general secret sharing schemes and PIR-tailored secret sharing:

1. In general secret sharing, a dealer shares value(s) from $\mathbb{F}_\rho$ where ρ is a prime number. However, In PIR-tailored secret sharing, the dealer is sharing value(s) from $\{0, 1\}$.

2. In a multi-secret sharing algorithm, there is no constraint for dealer for choosing the values of the secrets. However, In PIR-tailored secret sharing, only one of the secrets can be 1, and the rest should be 0 (maximum of number of secrets with value of 1 is one).

3. The x-coordinates for the points used for constructing the secrets sharing polynomial is known to public as $X = \{1, 2, \dots\}$, but in PIR-tailored secret sharing these values are secret, and only known to dealer.

4. The x-coordinates used in generating share for shareholder ith is known to that specific shareholder, i.e., ith shareholder knows $(x'_i, f(x'_i))$, but in PIR-tailored secret sharing these values (x'_i) are secret, and only known to dealer.

4.5.2 Algorithm details

Parameters Suppose the dealer plans to share q secrets, $\mathbb{S} = \{s_1, s_2, \ldots, s_q\}$, among ℓ participants, $\mathbb{M} = \{m_1, m_2, \ldots, m_\ell\}$, with a threshold $t \leq \ell$. All the secrets are from $\{0, 1\}$, and only one of the secrets could be 1. At the first step, the dealer generates q prime numbers $\mathbb{P} = \{p_1, p_2, \ldots, p_q\}$ at random. Then, the dealer calculates $n = p_1 \times p_2 \times \cdots \times p_q$. Note that our shares are about q times of prior works since all the calculations are in $mod(n)$.

Initial phase: At the start of the protocol, the dealer creates the $\mathbb{P}$ and n parameters and announces value of n publicly.

Sharing Secrets: In each round of the protocol, the dealer shares q secrets with the shareholders by taking the following steps (we first describe this for $q \leq t$, and then will discuss the modifications for $q > t$). Our scheme is based on the ramp secret sharing introduced in Section 4.3.2, so we discuss the modifications with respect to that scheme. For simplicity, we use $\mathbb{X} = \{1, \ldots, t+1\}$ as the x-coordinates of the root points, but they can be chosen randomly from $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid gcd(n, x) = 1\}$. Like other systems working based on Lagrange interpolation, we have the constraint that $gcd(x_i - x_j, n) = 1$ for $x_i, x_j \in \mathbb{X}$ and $i \neq j$.

I In contrast to the ramp scheme that uses points (i, s_i), our scheme uses the points $(i, r_i \times p_i + s_i)$ to build our secret sharing polynomial function. $r_i \in \mathbb{Z}_n^*$ are random numbers that increase the degree of freedom of the secrets. Therefore, the dealer uses Lagrange polynomial interpolation to find a polynomial $f(x)$ (with a degree of at most t) that contains these $t + 1$ points:

$$(1, (r_1 \times p_1) + s_1 \ mod(n)), \ldots, (q, (r_q \times p_q) + s_q \ mod(n)) \ldots$$

$$(q+1, r_{q+1} \ mod(n)), \ldots, (t+1, r_{t+1} \ mod(n))$$

II The dealer will send the secret share $(f(x_i))$ to the ith shareholder for $1 \leq i \leq \ell$. ($x_i \in \mathbb{X}'$s are random numbers from $\mathbb{Z}_n^*$ with Lagrange constraint $gcd(x'_i - x'_j, n) = 1$ for $x'_i, x'_j \in \mathbb{X}'$ and $i \neq j$ and a constraint that $gcd(x_i - x'_j) = 1$ for $x_i \in \mathbb{X}$ and $x'_j \in \mathbb{X}'$.

Secret reconstruction: To retrieve the secrets, any $k \geq t+1$ shares can give away the secrets using Lagrange interpolation:

1. The combiner will construct the polynomial $f(x)$ using the $k \geq t+1$ points of $(x'_j, f(x'_j))$ for $x'_j \in \mathbb{X}'$ with Lagrange polynomial interpolation.

2. The combiner uses $f(x)$ to obtain the secrets. Suppose the combiner wants to recover the ith secret, s_i. As $(i, (r_i \times p_i) + s_i \ mod(n))$ is a point of the known $f(x)$, she can extract s_i using the p_i:

$$\begin{aligned} s_i &= f(i) \ mod(p_i) \\ &= \sum_{m=1}^{k} f(x'_m)(\prod_{n=1, n\neq m}^{k} (i - x'_n)(x'_m - x'_n)^{-1}) \ mod(p_i) \quad 1 \leq j \leq q \end{aligned} \tag{4.6}$$

Extension to $q > t$: The protocol will operate with small changes. $f(x)$ will have a degree at most q. Therefore, in the first step of the sharing protocol, the dealer will use the points $(1, (r_1 \times p_1) + s_1 \ mod(n)), \ldots, (q, (r_q \times p_q) + s_q \ mod(n)), (q+1, r_{q+1} \ mod(n))$ to construct $f(x)$. The dealer releases $q - t$ random points of $f(x)$ publicly to make the scheme a $(t+1,\ell)$ threshold secret sharing algorithm. The rest of the protocol is the same.

4.5.3 Security analysis

Degree of freedom of secrets The degree of freedom of shared secrets demonstrates how many *independent variables* are used in generating the secret sharing function. A multi-secret sharing scheme is secure if the q shared secrets $\{s_1, \ldots, s_q\}$ have a degree

of freedom of q or higher when up to t shares are available. Each secret share known to the adversary reduces the degree of freedom by one, as it provides the adversary with a new equation for the main polynomial. Therefore, if there are d independent variables in $f(x)$, the degree of freedom given t secret shares will be $d - t$.

Increasing the degree of freedom by introducing new random variables Our PIR-tailored secret sharing scheme shares q secrets using a t degree polynomial. We add q random variables, r_is, to the polynomial points we share, i.e., $(1, (r_1 \times p_1) + s_1 \ mod(n)), \ldots, (q, (r_q \times p_q) + s_q \ mod(n))$. Therefore, the adversary's degree of freedom to reconstruct the secrets will be $t + q + 1$. Assuming that the adversary is provided with t shares, the degree of freedom will reduce to $q + 1$, which is still larger than the number of secrets, q.

The $q > t$ state of the PIR-tailored secret sharing is used as the core of our HPIR since we want to deploy our HPIR with minimum number of PIR servers (e.g., two servers with $t = 1$). Therefore, when $q > t$, we claim that any coalition with up to q shares can not learn *anything* about the secrets. Note that this is unlike the ramp secret sharing which leaks some information if the number of shares is between $t + 1$ and $t + q - 1$.

4.5.3.1 Security proof

Theorem 1: In our PIR-tailored secret sharing (when $q > t$), regardless of the number of secrets being shared, the participants can not learn anything about the secrets with up to q shares.

Proof To prove the security of this scheme, we should show that with less than $q + 1$ shares, there is no information about the secrets (when $q > t$). For proving security, we should prove that the adversary given q shares *can not* differentiate that a polynomial with one secret with value of 1 generates these q shares or a polynomial with all secrets with value 0. By this proof, we can show that in the PIR protocol

based on this scheme, the PIR server cannot differentiate that the PIR client wants a record ($s_i = 1$) or not ($s_i = 0$).

Suppose that the polynomial $f(x)$ is sharing one secret with value 1 and $q-1$ secrets with value 0 (with degree q), and there is another polynomial $f'(x)$ that is sharing 0 as the values of secrets with the same degree of q. We can show that both of these polynomial functions can generate the q shares $(x'_j, f(x'_j))$ the adversary has, i.e., both of them can generate the same q shares ($f'(x'_j) = f(x_j)$ for $1 \leq j \leq q$). We assume that at the worst case, the adversary knows the x-coordinates used for generating secret sharing polynomial $\mathbb{X} = \{x_1, \ldots, x_{q+1}\}$ and the x-coordinates used for generating shares ($\mathbb{X}' = \{x'_1, \ldots, x'_{q+1}\}$).
For generating the polynomial $f(x)$ which shares one secret 1, we use the following points:

$$(1, (r_1 \times p_1) + 1 \; mod(n)), \ldots, (q, (r_q \times p_q) \; mod(n)), (q+1, r_{q+1} \; mod(n))$$

For generating the polynomial $f'(x)$ which shares 0s, we use the following points:

$$(1, (r'_1 \times p_1) + 0 \; mod(n)), \ldots, (q, (r'_q \times p_q) \; mod(n)), (q+1, r'_{q+1} \; mod(n))$$

Suppose that the adversary has q shares (x'_j, y_j) for $1 \leq j \leq q$, so he has q equations based on Lagrange interpolation. First we prove that the value of the first secret (s_1) is indistinguishable, and then the same proof can be used for other secrets too. If the adversary assumes that this secret sharing is sharing value of $s_1 = 0$:

$$p_1 r_1 L_1(x'_1) + p_2 r_2 L_2(x'_1) + \cdots + p_q r_q L_q(x'_1) + r_{q+1} L_{q+1}(x'_1) = y_1 \; mod(n)$$

$$p_1 r_1 L_1(x'_2) + p_2 r_2 L_2(x'_2) + \cdots + p_q r_q L_q(x'_2) + r_{q+1} L_{q+1}(x'_2) = y_2 \; mod(n)$$

$$\vdots$$

$$p_1 r_1 L_1(x'_q) + p_2 r_2 L_2(x'_q) + \cdots + p_q r_q L_q(x'_q) + r_{q+1} L_{q+1}(x'_q) = y_q \; mod(n)$$

where $\{p_1, p_2, \ldots, p_q\}$ are q different prime numbers and $n = p_1 p_2 \ldots p_q$.
If the adversary assumes that this secret sharing is sharing value of $s_1 = 1$:

$$(p_1r_1+1)L_1(x'_1)+p_2r_2L_2(x'_1)+\cdots+p_qr_qL_q(x'_1)+r_{q+1}L_{q+1}(x'_1)=y_1\ mod(n)$$

$$(p_1r_1+1)L_1(x'_2)+p_2r_2L_2(x'_2)+\cdots+p_qr_qL_q(x'_2)+r_{q+1}L_{q+1}(x'_2)=y_2\ mod(n)$$

$$\vdots$$

$$(p_1r_1+1)L_1(x'_q)+p_2r_2L_2(x'_q)+\cdots+p_qr_qL_q(x'_q)+r_{q+1}L_{q+1}(x'_q)=y_q\ mod(n)$$

where $L_m(x'_j)$ is the Lagrange function for specific values of $\mathbb{X}=\{x_1,x_2,\ldots,x_{q+1}\}$:

$$L_m(x'_j)=\prod_{n=1,n\neq m}^{q+1}(x'_j-x_n)(x_m-x_n)^{-1})\ mod(n)$$

If we show that there is at least one solution for both of these set of equations, we can show that the adversary cannot differentiate that the secret was zero or one. So for set of unknowns $\{r_1,r_2,\ldots,r_{q+1}\}$ and $\{r'_1,r'_2,\ldots,r'_{q+1}\}$ we should show that:

$$\begin{aligned}&(p_1r_1+1)L_1(x'_1)+p_2r_2L_2(x'_1)+\cdots+p_qr_qL_q(x'_1)+r_{q+1}L_{q+1}(x'_1)=\\&p_1r'_1L_1(x'_1)+p_2r'_2L_2(x'_1)+\cdots+p_qr'_qL_q(x'_1)+r'_{q+1}L_{q+1}(x'_1)\ mod(n)\end{aligned}$$

$$\begin{aligned}&(p_1r_1+1)L_1(x'_2)+p_2r_2L_2(x'_2)+\cdots+p_qr_qL_q(x'_2)+r_{q+1}L_{q+1}(x'_2)=\\&p_1r'_1L_1(x'_2)+p_2r'_2L_2(x'_2)+\cdots+p_qr'_qL_q(x'_2)+r'_{q+1}L_{q+1}(x'_2)\ mod(n)\end{aligned}$$

$$\vdots$$

$$\begin{aligned}&(p_1r_1+1)L_1(x'_q)+p_2r_2L_2(x'_q)+\cdots+p_qr_qL_q(x'_q)+r_{q+1}L_{q+1}(x'_q)=\\&p_1r'_1L_1(x'_q)+p_2r'_2L_2(x'_q)+\cdots+p_qr'_qL_q(x'_q)+r'_{q+1}L_{q+1}(x'_q)\ mod(n)\end{aligned}$$

We can write:

$$r'_1 = r_1 + k_1$$

$$r'_2 = r_2 + k_2$$

$$\vdots$$

$$r'_q = r_q + k_q$$

$$r'_{q+1} = r_{q+1} + k_{q+1}$$

By putting the above solution in the equations we will have:

$$L_1(x'_1) = p_1 k_1 L_1(x'_1) + \cdots + p_q k_q L_q(x'_1) + k_{q+1} L_{q+1}(x'_1)\ mod(n)$$

$$L_1(x'_2) = p_1 k_1 L_1(x'_2) + \cdots + p_q k_q L_q(x'_2) + k_{q+1} L_{q+1}(x'_2)\ mod(n)$$

$$\vdots$$

$$L_1(x'_q) = p_1 k_1 L_1(x'_q) + \cdots + p_q k_q L_q(x'_q) + k_{q+1} L_{q+1}(x'_q)\ mod(n)$$

We know that if an equation has an answer in $mod(p_i)$, it will have answer in $mod(n)$ too where n is a multiple of p_i. So we can apply $mod(p_i)$ on the ith equation of above system, then we have the following equations:

$$L_1(x'_1) = p_2 k_2 L_2(x'_1) + \cdots + p_q k_q L_q(x'_1) + k_{q+1} L_{q+1}(x'_1)\ mod(p_1)$$

$$L_1(x'_2) = p_1 k_1 L_1(x'_2) + \cdots + p_q k_q L_q(x'_2) + k_{q+1} L_{q+1}(x'_2)\ mod(p_2)$$

$$\vdots$$

$$L_1(x'_q) = p_1 k_1 L_1(h x'_q) + p_2 k_2 L_2(h_q) + \cdots + k_{q+1} L_{q+1}(x'_q)\ mod(p_q)$$

Therefore, based on Multivariable Chinese Reminder Theorem (Section 4.5.3.3), since all the p_is are co-prime to each other and in each equation we have the $p_i L_i(h)$s

that are co-prime in $mod(p_j)$ where $j \neq i$ (we used $\mathbb{X}$ and $\mathbb{X}'$ in Section 4.5.2 that $gcd(x_i - x'_j, n) = 1$ for $x_i \in \mathbb{X}$ and $x'_j \in \mathbb{X}'$, there is at least one solution for $\{k_1, k_2, \ldots, k_{q+1}\}$. Therefore the adversary given q shares *cannot* differentiate what secret was shared.

4.5.3.2 Chinese remainder theorem (CRT)

Chinese Remainder Theorem is one of the most useful tools in number theory [56]. This theorem shows the existence of solution for following q equations:

$$x = a_i \mod(p_i) \quad \text{for } 1 \leq i \leq q$$

This theorem says that if p_is are co-prime to each other, then there is one and only one value for $x \ mod(n)$ where n is $\prod_{i=1}^{q} p_i$.

4.5.3.3 Multivariable chinese remainder theorem

This theorem [101] says that for a linear systems of equations $A\vec{x} = \vec{b} \ \ mod(\vec{P})$ (each equation is $a_{i,1}x_1 + \cdots + a_{i,n}x_n = b_i \ \ mod(p_i)$) has solutions for all $\vec{b}$ if the $p_i \in \vec{P}$ are co-prime to each other and there is at least one element in ith row (all rows) of the matrix A that is co-prime with p_i.

4.6 Sketch of our HPIR protocol

Earlier we motivated the need for heterogeneous multi-server PIR (HPIR). We design the first HPIR protocol; the core of our HPIR construction is the PIR-tailored multi-secret sharing algorithm introduced in Section 4.5.

The high-level idea of our HPIR Our HPIR protocol has the same high-level architecture as Henry et al.'s multi-server PIR [82]. The querying client will act as the secret sharing dealer, and the PIR servers act as the shareholders. The client will use the PIR-tailored secret sharing algorithm of Section 4.5 to split queries into shares,

which are then sent to the servers. The servers will make some computation using the query shares and will send the results back to the querying client. Finally, the client recovers her requested records by combing the responses from the PIR servers. Like previous information-theoretic PIR systems, we assume that all the PIR servers are not colluding, so they cannot reconstruct the secret sharing polynomials to recover the secrets.

Our HPIR protocol is a multi-query protocol, i.e., the client queries multiple (q) records in each round of the protocol. The client will generate a polynomial $f_i(x)$ for *each record* of the database. Each polynomial is used to share q secrets with the possible values of 0 or 1. A value of 1 means that the client is asking for the record corresponding to the index of that polynomial. To query for q records in a given round of the protocol, the client will send $q + 1$ vectors of size r elements to the PIR servers to retrieve q records of the database.

Note that the key enabler of heterogeneity in our HPIR is the PIR-tailored secret sharing scheme of Section 4.5. To enforce heterogeneity, the client simply sends a different fraction of query shares to different servers based on their bandwidth and computational capabilities. For instance, consider a three-server setting, where the three servers plan to handle 30%, 10%, and 60% of the communication/computation overheads, respectively. Then, the client will send 1/10 of her query shares to the first server, 3/10 of the shares to the second server, and the rest of them to the third server. The non-ramp property of our PIR-tailored secret sharing scheme enables us to design multi-query PIR algorithms that can operate using as few as two PIR servers.

Two Versions To better present the technical details of our HPIR protocol, we present a basic version and a complete version for our HPIR protocol. In our basic HPIR (Section 4.7), there are q prime numbers involved in generating the queries, so the communication cost will increase with the number of queries linearly; we address this in our complete version (Section 4.8) by introducing additional parameters.

4.7 Our HPIR algorithm (basic version)

In the following, we present the steps of our HPIR protocol. For clarity of presentation, we present our protocol for a 2-server PIR setting (composed of a rich server and a poor server). Please refer to Table 4.1 for the notations.

4.7.1 Client generates r polynomials

Suppose that the client wants to query q records of the database with indices $\beta = \{\beta_1, \ldots, \beta_q\}$; she takes the following steps to generate r polynomials (one for each row of the database):

1. The client will choose $q > 2$ different prime numbers $P = \{p_1, p_2, \ldots, p_q\}$ greater than 2^w, where w is the element size in the database. The client will calculate $n = p_1 \times p_2 \times \cdots \times p_q$, and will send it to the PIR servers (note that all the calculations in this protocol are in $mod(n)$).

2. The client will construct r polynomials of degree q based on our PIR-tailored secret sharing algorithm (see Section 4.5.2). In generating each of the polynomials, the client will choose $q + 1$ points with random distinct x-coordinates $\mathbb{X} = \{x_1, x_2, \ldots, x_{q+1}\}$ (used for all the polynomials) and y-coordinates given by (for $1 \leq i \leq r$):

$$y_{i,j} = f_i(x_j) = \begin{cases} (r_{i,j} \times p_j) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q \\ r_{i,j} \ mod(n) & \text{for } j = q + 1 \end{cases} \tag{4.7}$$

where $r_{i,j}$s are random numbers from $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n \mid gcd(n, x) = 1\}$, and the secrets are:

$$\delta_{ij} = \begin{cases} 1 & i = \beta_j \\ 0 & \text{o.w.} \end{cases} \tag{4.8}$$

where $\beta = \{\beta_1, \ldots, \beta_q\}$ are the indices of the data records being queried by the client.

3. Finally, after choosing these points, the client uses Lagrange interpolation to find the r polynomials of degree q that contain these points.

Constraints All the members of $\mathbb{X}$ are chosen from $\mathbb{Z}_n^*$ at random. Like other systems working based on Lagrange interpolation, we have the constraint that $gcd(x_i - x_j, n) = 1$ for $x_i, x_j \in \mathbb{X}$, $i \neq j$.

Example: Suppose we have five records in our database ($r = 5$), and the client wants to retrieve records with indices $\beta = \{1, 3, 4\}$. Each row of the following matrix shows the y-coordinates of each of the $r = 5$ polynomials ($r_{i,j} \in_r \mathbb{Z}_n^*$ for $1 \leq i \leq r, 1 \leq j \leq q+1$):

$$\mathbb{Y} = \begin{bmatrix} (p_1 \times r_{1,1}) + 1 & p_2 \times r_{1,2} & p_3 \times r_{1,3} & r_{1,4} \\ p_1 \times r_{2,1} & p_2 \times r_{2,2} & p_3 \times r_{2,3} & r_{2,4} \\ p_1 \times r_{3,1} & (p_2 \times r_{3,2}) + 1 & p_3 \times r_{3,3} & r_{3,4} \\ p_1 \times r_{4,1} & p_2 \times r_{4,2} & (p_3 \times r_{4,3}) + 1 & r_{4,4} \\ p_1 \times r_{5,1} & p_2 \times r_{5,2} & p_3 \times r_{5,3} & r_{5,4} \end{bmatrix} \tag{4.9}$$

4.7.2 Client generates queries

Using the r polynomials generated above, the client will generate secret shares for her q queries. To do so, as described in Section 4.5.2, the client will pick $(q+1)$ random x-coordinates of $\mathbb{X}' = \{x'_1, x'_2, \ldots, x'_{q+1}\}$ (different from $\mathbb{X}$, with the same constraint), which are kept secret from the server, to generate the query matrices. $\mathbb{Q}_c$ is the query matrix for the rich (resourceful) server with q rows and r columns, and $\mathbb{Q}_r$ is the query matrix for the poor (low-resource) server with one row and r columns:

$$\mathbb{Q}_c = \begin{bmatrix} \vec{F}(x'_1) \\ \vec{F}(x'_2) \\ \vdots \\ \vec{F}(x'_q) \end{bmatrix} = \begin{bmatrix} f_1(x'_1) & f_2(x'_1) & \ldots & f_r(x'_1) \\ f_1(x'_2) & f_2(x'_2) & \ldots & f_r(x'_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x'_q) & f_2(x'_q) & \ldots & f_r(x'_q) \end{bmatrix} \tag{4.10}$$

$$\mathbb{Q}_r = \begin{bmatrix} \vec{F}(x'_{q+1}) \end{bmatrix} = \begin{bmatrix} f_1(x'_{q+1}) & f_2(x'_{q+1}) & \ldots & f_r(x'_{q+1}) \end{bmatrix} \tag{4.11}$$

where $f_i()$ is the polynomial function corresponding to the record i. The client sends the query matrices $\mathbb{Q}_r$ and $\mathbb{Q}_c$ to the servers.

4.7.3 The servers respond

After receiving the query matrices, each server will calculate the multiplication of $\mathbb{R}_{c/r} = \mathbb{Q}_{c/r} \times \mathbb{D}\ mod(n)$, and it will return the results to the client. Note that the servers do not know the values of r_{ij}, $\mathbb{X}'$, $\mathbb{X}$, and the prime numbers (p_js).

4.7.4 Reconstructing the records by the client

Using the responses received from the servers, $\mathbb{R} = \mathbb{R}_c||\mathbb{R}_r$, the client will construct s polynomials $\phi_k(x)$. Each ϕ_k is a q-degree polynomial that produces the kth elements of the q queried records.

Each column k of $\mathbb{R}$ contains $q+1$ points for a polynomial $\phi_k(x)$, i.e., the points $(x'_j, \mathbb{R}_{j,k})$ for $1 \leq j \leq (q+1)$. Each $\mathbb{R}_{j,k}$ (for $1 \leq j \leq q+1$ and $1 \leq k \leq s$) is given by:

$$\begin{aligned}
\mathbb{R}_{j,k} &= \sum_{i=1}^{r} f_i(x'_j) \times D_{i,k}\ mod(n) \\
&= \sum_{i=1}^{r} (D_{i,k} \times \sum_{v=1}^{q+1} (a_{i,v} \times x_j'^{v-1}))\ mod(n) \\
&= \sum_{v=1}^{q+1} (x_j'^{v-1} \times \sum_{i=1}^{r} (a_{i,v} \times D_{i,k}))\ mod(n) \\
&= \phi_k(x'_j)\ mod(n)
\end{aligned} \tag{4.12}$$

where $\phi_k()$ is a polynomial of degree q, and $a_{i,v}$ is the v^{th} coefficient of the polynomial $f_i(x)$. ϕ_k has a degree of q, and therefore the client can derive it using Lagrange interpolation by using the $q+1$ points $(x'_j, \mathbb{R}_{j,k})$ for $1 \leq j \leq (q+1)$.

Finally, the client retrieves the responses to her q queries using the derived $\phi.(\cdot)$ polynomials by feeding the x-coordinates $\mathbb{X} = \{x_1, \ldots, x_q\}$ into $\phi_k(x_j)$. Specifically, the client derives the kth element of the jth queried record, $\mathbb{D}_{\beta_j,k}$, as:

$$D_{\beta_j,k} = \phi_k(x_j)\ mod(p_j) \text{ for } 1 \leq k \leq s, 1 \leq j \leq q \tag{4.13}$$

4.7.5 Communication overhead

Since the client uses q prime numbers (one for each query), the upload and download overheads are linear with the number of queries. To retrieve q records, the client should send $q^2 \times r \times w$ bits to the rich server, and $q \times r \times w$ bits to the poor server. The rich server will send back $q^2 \times s \times w$ bits, and the poor server will send back $q \times s \times w$ bits.

4.7.6 Security

This PIR protocol is built on our PIR-tailored secret sharing scheme of Section 4.5, so its security is based on the underlying PIR-tailored secret sharing scheme. Our HPIR protocol uses r PIR-tailored secret sharing functions (one for each database record), and retrieves q secrets at each round. To provide information-theoretic security, the set of q secrets should have a degree of freedom more than $q \times r$. In a two servers setting, the rich server will know $q \times r$ points of these PIR-tailored secret sharing schemes, so the number of independent variables in the system should be more than $2 \times q \times r$. There would be $(2q + 1) \times r + 2q + 2$ random variables inside the system: $(q + 1) \times r$ variables of r_{ij}, $q \times r$ secret values of δ_{ij}, $q + 1$ random variables of $\mathbb{X}$, and $q + 1$ random variables of $\mathbb{X}'$. If the poor server does not collude with the rich server, the degree of freedom of secrets will be more than the $q \times r$ threshold. Therefore, our HPIR is information-theoretic secure if at least one of the PIR servers does not collude with the others.

4.8 Our HPIR algorithm (complete version)

Why extending the design As mentioned above, the basic version of our HPIR protocol has high communication costs due to using q prime numbers (one for each query). Our extended protocol uses only two prime numbers $\mathbb{P} = \{p_1, p_2\}$ for query construction, therefore reduces communication costs significantly. To preserve its

information-theoretic security, we add multiple parameters to its polynomials. To do so, we make the x-coordinates of the PIR-tailored secret sharing polynomials unique for each row of the database, and unknown to the servers.

Improving efficiency by introducing unique x-coordinates for each PIR-tailored secret sharing function Recall from Section 4.7.4 that the constructed PIR-tailored secret sharing polynomials, $\phi_k(\cdot)$s, are functions of client polynomials $f_i(x)$ and database elements, i.e., $\phi_k(x)\ mod(n) = \sum_{i=1}^{r} f_i(x) \times \mathbb{D}_{i,k}\ mod(n)$ for $1 \leq k \leq s$. To be able to extract the kth element of the desired rows (β) using servers' responses, the querying client should remove the effect of the undesired records ($\mathbb{D}_{i,k}(x)$, for $i \notin \beta$) in $\phi_k(x)$. To do so, the client should construct the polynomials in a way that for any inputs sample $x \in \mathbb{X}$, the functions $f_i(x)$ output zero for $i \notin \beta$, and output a non-zero value for $i \in \beta$. Recall that each client polynomial $f_i(x)$ can be represented as:

$$\begin{aligned} f_i(x) &= \sum_{m=1}^{q+1} y_m \times \ell_m(x) \quad mod(n) \\ \ell_m(x) &= \prod_{1 \leq v \leq q+1, v \neq m} \frac{(x - x_v)}{(x_m - x_v)} \quad mod(n) \end{aligned} \tag{4.14}$$

where (x_m, y_m)'s are the points used to generate the polynomials. As can be seen from the above equation, there are two approaches for making each polynomial ($f_i(x)$) zero for $i \notin \beta$. The first approach is making y_m zero in (4.14) for input $x = x_m$, so $(x_m, 0)$ is one of the points used for interpolation of all the polynomials ($f_i(x)$) for $i \notin \beta$. This requires a fix set of x-coordinates $\mathbb{X} = \{x_1, \ldots, x_{q+1}\}$ be used across all of the r PIR-tailored secret sharing polynomials. This is what has been done by prior PIR protocols [82, 111, 72, 54].

The second approach to make $f_i(x)$ zero for $i \notin \beta$ is to make $\ell_m(x)$ in (4.14) zero by choosing x and x_v in a way that $x - x_v$ becomes zero. In $mod(p_j)$, if $p_j|(x - x_v)$ or $p_j|y_m$ (where | means division), then $f_i(x)\ mod(p_j)$ will produce zero. In this approach, the x-coordinates used for generating functions, x_ms, can be different from

the input x-coordinate, x. This approach enables us to choose different random x-coordinates for each polynomial. In our complete HPIR protocol, we combine these two approaches, i.e., we use different x-coordinates and y-coordinates for the PIR-tailored secret sharing polynomials. Combining these two approaches enables us to use just two prime numbers in constructing the PIR-tailored secret sharing polynomials, which reduces the communication overhead as explained below. This is while the basic version of our protocol (Section 4.7) needs q prime numbers. Specifically, in our basic protocol, all of the calculations are in $mod(n)$, and n is the multiplication of q prime numbers, so size of sending and receiving elements linearly depends on the number of queries. Using just two prime numbers in our PIR-tailored secret sharing constructions will keep the size of each element in query and response vectors to be fixed and independent of the number of queries. This will improve the efficiency of our complete HPIR in upload and download bandwidth consumption by sending and receiving smaller elements.

Preserving the degree of freedom of secrets using two prime numbers We add q new random variables r'_{ij} in x-coordinates of each polynomials, then the degree of freedom of the secrets will be two times of the basic version (Section 4.7). If the adversarial server removes the effect of half of the prime numbers, the secrets will still have enough degree of freedom. To do so, the client will create half of the PIR-tailored secret sharing points using p_1 (i.e., $(p_1 \times random + secret)$), and the other half using p_2. Algorithm 4.8.1 summarizes our complete HPIR protocol.

Client (querying for block numbers $\beta = \{\beta_1, \ldots, \beta_q\}$) :
P1. Choose two prime numbers $\mathbb{P} = \{p_1, p_2\}$ with more than w bits ($p_i > 2^w$).
P2. Calculate $n = p_1 \times p_2$, and release it to the servers.
P3. Choose q random distinct $\{\alpha_1, \ldots, \alpha_q\}$ from $\mathbb{Z}_n^*$.
P4. Construct r polynomials of degree q. For generating the ith function, the client will take the following steps:

P4(a). Construct $(q+1)$ x-coordinates $\mathbb{X} = \{x_{i,1}, \ldots, x_{i,q+1}\}$ as follows for $1 \leq i \leq r$ ($r'_{i,j}$ and r' are random numbers from $\mathbb{Z}_n^*$ such that $gcd(x_{i,j} - x_{i,k}, n) = 1$ for a specific i and different j and ks:

$$x_{i,j} = \begin{cases} (r'_{i,j} \times p_1) + \alpha_j \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 0 \text{ (even)} \\ (r'_{i,j} \times p_2) + \alpha_j \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 1 \text{ (odd)} \\ r' \ mod(n) & \text{for } j = q+1 \end{cases}$$

P4(b). Construct $(q+1)$ y-coordinates $\mathbb{Y} = \{y_{i,1}, \ldots, y_{i,q+1}\}$ as follows for $1 \leq i \leq r$ ($r_{i,j}$ is a random number from $\mathbb{Z}_n^*$):

$$y_{i,j} = \begin{cases} (r_{i,j} \times p_1) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 0 \text{ (even)} \\ (r_{i,j} \times p_2) + \delta_{i,j} \ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 1 \text{ (odd)} \\ r_{i,j} \ mod(n) & \text{for } j = q+1 \end{cases}$$

P4(c). Use Lagrange polynomial interpolation to find $f_i(x)$ of degree q that satisfies $(x_{i,j}, y_{i,j})$ for $1 \leq i \leq r, 1 \leq j \leq q+1$.

P5. Choose q random distinct x-coordinates $\mathbb{X}' = \{x'_1, \ldots, x'_q\}$.
P6. Send matrix $\mathbb{Q}_c$ to the rich server ($\mathbb{Q}_c[j][i] = f_i(x'_j)$ for $1 \leq i \leq r, 1 \leq j \leq q$).
P7. Send the the query matrix $\mathbb{Q}_r$ ($\mathbb{Q}_r[0][i] = f_i(r') = r_{i,j}$) for $1 \leq i \leq r, j = q+1$) to the poor server.
Each Server:
S1. Multiply the $\mathbb{Q}_c$ and $\mathbb{Q}_r$ matrices to the database matrix, and return the results ($\mathbb{R}_c = \mathbb{Q}_c * \mathbb{D}$ and $\mathbb{R}_r = \mathbb{Q}_r * \mathbb{D}$) to the client.
Client:
C1. Construct polynomials $\phi_k(x)$ (for $1 \leq k \leq s$) that satisfy $(x'_j, R_c[j][k])$ for $1 \leq j \leq q$ and $(r', \mathbb{R}_r[0][k])$ using Lagrange polynomial interpolation.
C2. extract the items of queried records (for $1 \leq j \leq q, 1 \leq k \leq s$):

$$\mathbb{D}_{\beta_j,k} = \begin{cases} \phi_k(\alpha_j) \times f_{\beta_j}(\alpha_j)^{-1} \ mod(p_1) & j\%2 == 0 \text{ (even)} \\ \phi_k(\alpha_i) \times f_{\beta_j}(\alpha_j)^{-1} \ mod(p_2) & j\%2 == 1 \text{ (odd)} \end{cases}$$

Algorithm 4.8.1. Our HPIR Protocol (Complete Version)

Correctness: In extracting the requested records (step C2 in Algorithm 4.8.1), when j is an even number ($j\%2 == 0$), the client will use p_1, otherwise (j is an odd number) she will use p_2, where j is the index of the query. Below we demonstrate the correctness of our HPIR protocol (when j is even) by showing that the client can always reconstruct her queried records using our HPIR protocol.

$$
\begin{aligned}
&\phi_k(\alpha_j)\times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = \\
&\quad (\sum_{i=1}^{r} f_i(\alpha_j) \times \mathbb{D}_{i,k}) \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = \\
&\quad (\sum_{i=1}^{r}(\sum_{m=1}^{q+1} y_{i,m} \times \prod_{v=1,v\neq m}^{q+1} \frac{\alpha_j - x_{i,v}}{x_{i,m} - x_{i,v}}) \times \mathbb{D}_{i,k}) \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = \\
&\quad f_{\beta_j}(\alpha_j) \times \mathbb{D}_{\beta_j,k} \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1) = \mathbb{D}_{\beta_j,k}
\end{aligned}
\tag{4.15}
$$

where $\mathbb{D}_{\beta_j,k}$ is the kth element of the jth query. As can be seen, the effect of undesired records ($i \notin \beta$) will be cancelled in the calculation of $\phi_k(\alpha_j) \times f_{\beta_j}(\alpha_j)^{-1}\ mod(p_1)$ since $f_i(\alpha_j)$ will produce zero for them in $mod(p_1)$.

4.8.1 Communication costs

To retrieve q records, the client should send $2 \times q \times r \times w$ bits to the rich server, and $2 \times r \times w$ to the poor server. The rich server will send back $2 \times q \times s \times w$ bits, and the poor server will send back $2 \times s \times w$ bits.

Pseudo-random number generator for coordinates We can further improve the communication overhead of the poor server by having the client use a pseudo-random number generator to generate the query vectors of the poor server. For our protocol, instead of random $r_{i,q+1}$ for $1 \leq i \leq r$, client will generate r numbers $\{g_1, g_2, \ldots, g_r\}$ in $mod(n)$ using a random seed. Now, the client in our basic HPIR will construct the polynomials as follows:

$$
y_{i,j} = \begin{cases} (r_{i,j} \times p_j) + \delta_{i,j}\ mod(n) & \text{for } 1 \leq j \leq q \\ g_i\ mod(n) & \text{for } j = q+1 \end{cases}
\tag{4.16}
$$

and in the complete version, client will construct the polynomials as follows:

$$
y_{i,j} = \begin{cases} (r_{i,j} \times p_1) + \delta_{i,j}\ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 0 \\ (r_{i,j} \times p_2) + \delta_{i,j}\ mod(n) & \text{for } 1 \leq j \leq q, j\%2 == 1 \\ g_i\ mod(n) & \text{for } j = q+1 \end{cases}
\tag{4.17}
$$

Therefore, instead of sending r elements to the poor server, she will just send the seed of the pseudo-random number generator, and the poor server can reproduce these r numbers $\{g_1, g_2, \ldots, g_r\}$ and construct the query vector. Note that using a PRNG will change our security from information-theoretic to computational security.

4.8.2 Security

Information-theoretic security If all the PIR servers do not collude, they can not learn anything about client's queries. There are $(3q+1) \times r + 2q + 1$ variables in the system that are only known to the querying client: $(q+1) \times r$ variables of r_{ij}, $q \times r$ variables of r'_{ij}, $q \times r$ secrets δ_{ij}, q variables of α_j, q variables of $\mathbb{X}'$, and one variable r'. Half of the random variables r_{ij} and r'_{ij} are obfuscated with p_1 and the other half are obfuscated with p_2. Even if the PIR server knows the values of these two prime numbers, it can only remove half of the variables by calculating the shares in $mod(p_1)$ or $mod(p_2)$. The server knows at most $q \times r$ points of the polynomials, and it can remove half of r_{ij} and r'_{ij}, which results in a more than $q \times r$ degree of freedom for the secrets.

Robustness against colluding servers If all the PIR servers collude, they can determine the client's query since they can factorize n and find the prime numbers used in the scheme. However, if we use larger prime numbers, the security of the protocol will reduce to computationally secure against all colluding servers based on the factorization problem, which is an NP problem. To compromise a client, the servers first need to factorize n. This trades off between performance and security: increasing the size of the prime numbers improves the computational security, but the matrix multiplication will take longer. Therefore, **our protocol is information-theoretically secure if up to t servers collude; with large prime numbers, our protocol is computationally secure even all the servers collude.**

Table 4.3. Communication cost comparison (bits)

PIR Protocol		Upload BW for Each Server	Download BW for Each Server	Total Upload BW	Total Download BW	Minimum Number of PIR Servers (ℓ)
Goldbergs's PIR [72]		$q \times r \times w$	$q \times s \times w$	$\ell \times q \times r \times w$	$\ell \times q \times s \times w$	$t+1$
Henry et al. PIR [82]		$r \times w$	$s \times w$	$\ell \times r \times w$	$\ell \times s \times w$	$t+q$
Homogeneous Version of Our Protocol (Complete Version) ($q+1$ servers, $t=q$)		$2 \times r \times w$	$2 \times s \times w$	$2 \times \ell \times r \times w$	$2 \times \ell \times s \times w$	$q+1$
Homogeneous Version of Our Protocol (Complete Version) (2 servers, $t=1$)		$r \times (q+1) \times w$	$s \times (q+1) \times w$	$2 \times (q+1) \times r \times w$	$2 \times (q+1) \times s \times w$	2
Heterogeneous version of Goldberg's PIR [72]	Rich Server: Poor Server:	$q \times r \times t \times w$ $q \times r \times w$	$q \times s \times t \times w$ $s \times t \times w$	$(\ell+t) \times q \times r \times w$	$(\ell+t) \times q \times s \times w$	$t+1$
Heterogeneous version of Henry et al. PIR [82]	Rich Server: Poor Server:	$r \times t \times w$ $r \times w$	$s \times t \times w$ $s \times w$	$(\ell+t) \times r \times w$	$(\ell+t) \times s \times w$	$t+q$
Our Heterogeneous Protocol (Complete Version), $t=1$	Rich Server: Poor Server:	$2 \times q \times r \times w$ $2 \times r \times w$	$2 \times q \times s \times w$ $2 \times s \times w$	$2 \times (q+1) \times r \times w$	$2 \times (q+1) \times s \times w$	2
Our Heterogeneous Protocol (Complete Version) Using PRNG, $t=1$	Rich Server: Poor Server:	$2 \times q \times r \times w$ $2 \times w$	$2 \times q \times s \times w$ $2 \times s \times w$	$(2 \times q \times r+1) \times w$	$2 \times (q+1) \times s \times w$	2

4.8.3 Overhead comparison to prior work

4.8.3.1 Communication cost

Table 4.3 compares the communication costs of our protocol (Complete Version) with different protocols and in different settings. We also compare a homogeneous version of our HPIR protocol with state-of-the-art homogenous protocols (in the homogenous version of our protocol, we send equal number of shares to the servers). We compare the protocols for the same volume of retrieved traffic. Therefore, we amplify Goldberg's [72] communications by q as it is a single-query protocol.

Also, note that our element size is two times of prior works since our calculations are in $mod(n)$, where n is a multiplication of two prime numbers. Each prime number has about w bits, so our elements are about $2 \times w$ bits. However, in prior works the calculations are in $mod(p)$, where p is a w bits prime number.

Homogeneous version of our protocol Compared to Goldberg's PIR [72], homogeneous version of our protocol is slightly higher in upload and download bandwidth consumption. Comparing to Henry et al. PIR [82], the number of exchanged elements are the same, but each element of our protocol is two times as their elements. However, since Henry et al. PIR protocol is based on ramp secret sharing, they need $t+q$ servers for their protocol. It means that by increasing the number of queries, they need more non-colluding servers, while our protocol can be run on as few as two servers.

Table 4.4. Computation cost comparison

PIR Protocol		Computation Cost for Each Server	Total Computation Cost	Minimum Number of PIR Servers (ℓ)
Goldbergs's PIR [72]		$\mathcal{O}(q \times r^2 \times s)$	$\mathcal{O}(\ell \times q \times r^2 \times s)$	$t+1$
Henry et al. PIR [82]		$\mathcal{O}(r^2 \times s)$	$\mathcal{O}(\ell \times r^2 \times s)$	$t+q$
Our Heterogeneous Protocol (Complete Version)	Rich Server: Poor Server:	$\mathcal{O}(q \times r^2 \times s)$ $\mathcal{O}(r^2 \times s)$	$\mathcal{O}((q+1) \times r^2 \times s)$	2

Heterogeneous version of our protocol We also compare the heterogeneous version of our protocol (complete version) with the heterogeneous versions of Goldberg's PIR and Henry et al. PIR. We create a heterogeneous version for these prior works by making the client send more queries to one of PIR servers (the rich server). However, to maintain the privacy level (the maximum number of colluding servers without compromising client's privacy), we need to increase the degree of the polynomials used in their schemes. This results in increasing the bandwidth/computation overhead on the rich server of these prior works without reducing the burden on the poor server. Also, we see that using a PRNG in our protocol reduces upload bandwidth with the poor server, at the cost of weakening our information-theoretic security to a computational security.

4.8.3.2 Computation Cost

Table 4.4 compares the computation cost of our protocol (Complete Version) with other protocols. These numbers are based on standard matrix multiplication, which take $O(n^3)$ operations. For large number of queries (q), using the Strassen's algorithm for matrix multiplication will further reduce the order of matrix multiplication to $O(n^{2.8})$ operations.

4.9 Implementation

Implementation Setup We have implemented our HPIR protocol in C++, wrapped in Rust. We have implemented our code to be compatible with the Percy++ PIR

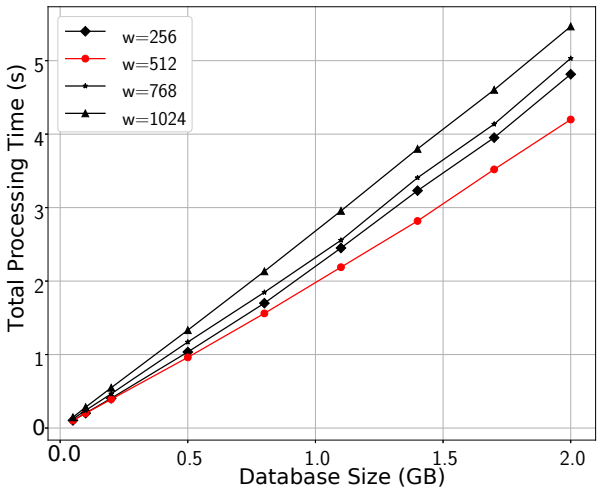
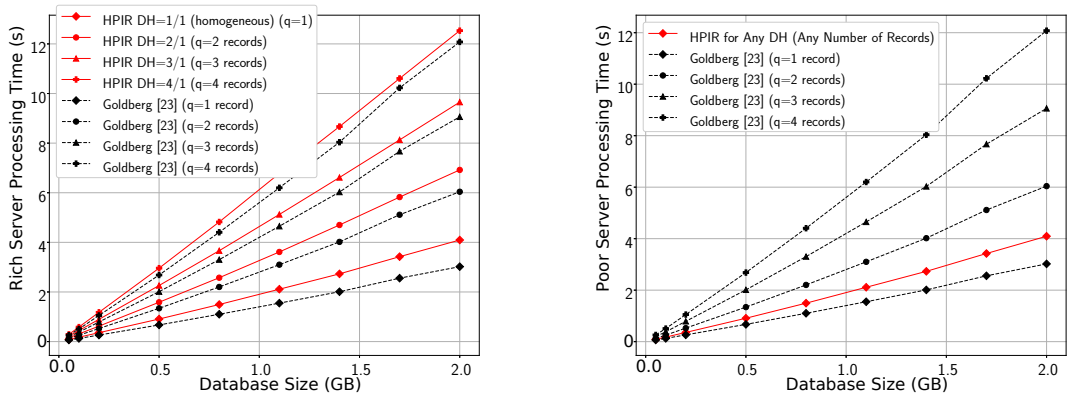


Figure 4.2. Total computation time (s) (i.e., server and client running times) vs. database sizes (GB) of protocol (complete version) in retrieving one record with different element sizes. $w = 512$ provides the least overhead.

library [73]. We use the NTL library [151] for handling big number operations similar to the Percy++ [73] PIR suite.

We measure the performance of our algorithm on a desktop computer with a quad-core i7 CPU @ 3.6 GHz and 32 GB of RAM, running Ubuntu 18.04. All of our experiments are single-threaded, though our most expensive operation, which is server computation, is highly parallelizable. In all of the experiments, we load the database into the RAM before measuring the time, so the measured times do not include the I/O times. Note that the computation time of our protocol depends mostly on the size of the database, not its dimensions, so in all of the experiments we choose $s = r = \sqrt{N/w}$, which is the communication-optimal block size derived by Goldberg et al. [72] (N is the size of database in bits and w is the element size in bits). All of our experiments are performed in a two servers scenario (that suits most of the real world applications): a rich server with high communication/computation resources and a poor server with lower resources.



(a) Server processing time of the rich server vs. Database size

(b) Server processing time of the poor server vs. Database size

Figure 4.3. Server processing time (for a degree of heterogeneity of $q/1$)

We compare our protocol with Goldberg PIR [72] design in a two servers scenario with privacy level $t = 1$. To ensure a fair comparison, we integrate Goldberg's PIR protocol from Percy++ [73] into our test framework, which is wrapped in Rust. We compare our design with this paper since Henry et al. PIR [82] with single query ($q = 1$) is a variant of Goldberg's PIR.

Degree of Heterogeneity (DH) We define a *degree of heterogeneity (DH)* parameter to represent the heterogeneity of our protocol. For a two-server setting, we define DH to be the ratio of the number of query shares the client sends to the rich PIR server divided by the number of query shares she sends to the poor PIR server. This metric represents the bandwidth/computation ratio of the PIR servers. Note that in our protocol, when retrieving q records, the maximum DH is $q/1$, as the client will have $q + 1$ secret shares to send to the servers.

Tuning the element size (w) parameter First, we measure the computation overhead performance of our protocol for different element sizes to find the optimal value of w. Figure 4.2 shows the total computation times (server and client side) for

various database sizes, and for different values of w. The plot suggests using $w = 512$ bits as the most efficient value, which we use for the rest of our experiments.

Note that a $w = 512$ bits results in n to have a size of 1024. In case of the collusion of the PIR servers, the security of the protocol will be tied to factorizing n. Therefore, increasing w will improve the security of the protocol in case of collusion at the cost of higher processing times.

Server computation overhead The major computation performed by the servers in our HPIR protocol is matrix multiplication (i.e., multiplying the query matrix, $\mathbb{Q}$, into the database matrix, $\mathbb{D}$). Figure 4.3 shows the server processing time for both the rich and poor servers for different database sizes and different number of queries (q) for a fixed $w = 512$. As can be seen, the server processing time is linear with the size of the database. The figure also compares the rich and poor server processing times with that of Goldberg's ITPIR [72] for various number of queries. As can be seen, the rich server processing time of our HPIR protocol is *very close to, but slightly larger than* Godlberg's server processing time (for the same number of records q being retrieved). On the other hand, the processing time of the poor server is *significantly smaller* than Goldberg's homogeneous server. As an example, to retrieve a 1.4 MB file (4 records) from a 2 GB database, the rich and poor HPIR servers will take 12.5 and 4.09 seconds, respectively, whereas the two servers of Goldberg will take 12.04 seconds each. That is, **our HPIR protocol significantly reduces the computation overhead on the poor server by slightly increasing the computation on the rich (resourceful) server**. We see that the computation gain of our HPIR further increases by increasing the degree of heterogeneity. As shown in below, increasing the degree of heterogeneity slightly increases the client's computation.

Client computation overhead The client computation overhead has two parts: client preparation time, which includes constructing the r polynomials and generating the query matrices, and client data extraction time, which includes the time of con-

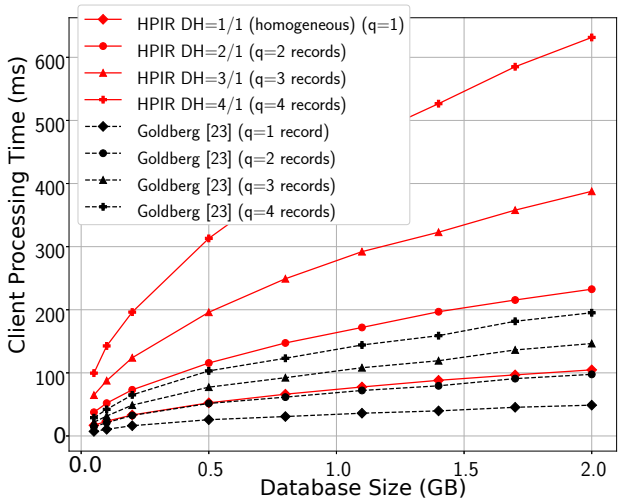
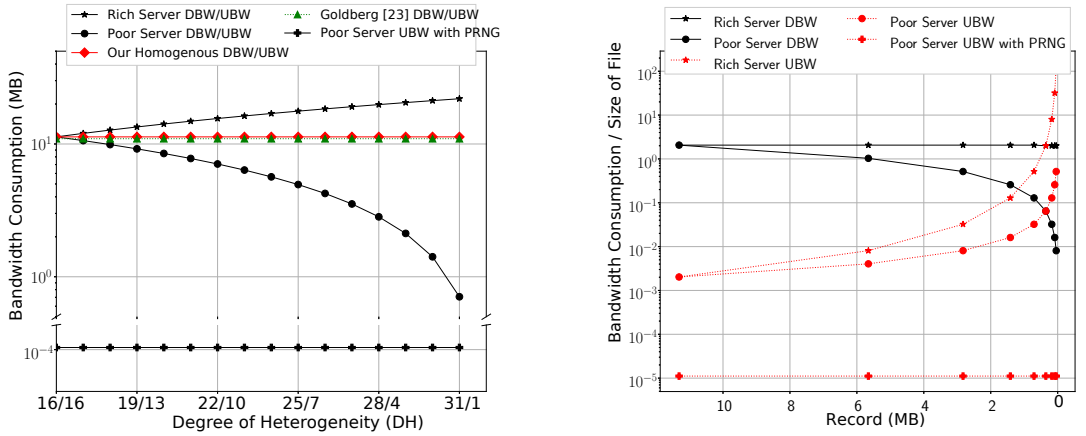


Figure 4.4. Client processing time vs. Database size

structing s functions $\phi_k(x)$ and extracting the elements of queried records. Figure 4.4 shows the client processing time of our HPIR protocol for different database sizes and different number of queries (for $w = 512$). As can be seen, client processing time has a sub-linear (square-root) relation with the database size, which is in agreement with Goldberg's results [72].

Also, as Figure 4.4 shows, our client processing time is *larger* than Goldberg's homogeneous algorithm. However, we see that even the increased client computation times are highly practical for typical clients, e.g., the computation time for $q = 4$ records in a 1.5GB database is around 500ms for HPIR, compared to 200ms for Goldberg's. Also, note that *server computation times are the practical bottleneck in PIR protocols* since they are an order of magnitude larger that client computation times (e.g., 0.2s client computation time compared to 9s server computation time in Goldberg's). Therefore, we believe that HPIR improves a client's overall experience by offloading the bulk of computations to the resourceful server, which can reduce the client's overall retrieving time.



(a) Actual overhead for $q = 31$

(b) normalized by size of the requested file

Figure 4.5. The upload and download overheads for our HPIR (complete version). We download a 10.95MB file from a 2GB database

Communication overhead Recall that, in HPIR the client can control the DH parameter by splitting the $q + 1$ query vectors non-uniformly among the PIR servers, e.g., q and 1 vectors to the rich and poor servers, respectively. Figure 4.5(a) shows the download and upload bandwidth overheads of our heterogeneous protocol for retrieving $q = 31$ records from a 2 GB database, with different degrees of heterogeneity. As can be seen, increasing the degree of heterogeneity trades off the communication overhead of the poor and rich servers. For instance, for a $DH = 16/16$ (which represents a homogeneous setting), the download/upload bandwidth of the rich and poor servers are 11.3MB each. By increasing DH to 31/1, the bandwidth of the rich and poor servers will be 21.9MB and 724KB, respectively. Therefore, we see that **HPIR reduces the communication overhead of the poor server by increasing the communication overhead on the rich server.** We also see that the homogeneous version of our HPIR protocol (i.e., for $DH = 16/16$) imposes computation overheads very close to that of Goldberg's homogeneous protocol. Finally, the figure shows that the bandwidth of our poor server when we are using a PRNG is always fixed regardless of the value of DH, since the client sends only one element (the seed of the PRNG).

Note that for the above results, we set $s = r$ [72], as described in our implementation setup section. The client can also control DH by changing the value of s (the number of elements in each database record), therefore changing the required number of queries q for a given PIR transaction. We demonstrate this in Figure 4.5(b); The figure shows the download and upload bandwidth overheads (normalized by the size of the queried file) of our heterogeneous protocol for retrieving a 10.95MB file from a 2GB database for different record sizes. We can see that there is a *trade-off between the upload bandwidth of the rich server and the download bandwidth of the poor server*; the client can adjust this by changing s.

4.10 Conclusions

We introduced a new class of multi-server PIR protocols, called heterogeneous PIR (HPIR), in which the PIR servers running the protocol undertake different computation and communication overheads. We argue that HPIR algorithms enable new applications for PIR by allowing the participation of low-resource parties in running private services, as well as improve the utility of some of the existing applications of PIR.

We design the first HPIR protocol, which is based on novel PIR-tailored secret sharing construction, and deploy an efficient implementation of it compatible with the Percy++ PIR library [73]. We extensively evaluate the performance of our implemented HPIR protocol in different settings, e.g., for different degrees of heterogeneity.

CHAPTER 5

PRIVACY ANALYSIS OF FEDERATED RANK LEARNING

In Chapter 4, we propose a heterogeneous private information retrieval mechanism to enhance privacy for FL clients against adversarial servers that aim to infer client data distribution through monitoring access patterns. In Chapter 3, we introduce the first FL framework that trains on parameter ranks. Our results demonstrate that exchanging rankings is more robust compared to existing FL designs that train on parameter weights.

This chapter focuses on membership inference attacks, which allow the adversary to determine whether a specific data sample was used in the target client's training data based on the local model of the target client throughout FL training. Previous research on privacy analysis in federated learning has mainly focused on measuring privacy leakage of weight parameters in the target model. The federated rank learning (FRL) approach introduced in Chapter 3 is the first FL that operates on parameter rankings, with weight parameters fixed and randomly initialized. However, there has been no investigation into the information leakage of local rankings trained on private data.

In this chapter, we aim to answer the following critical question: what is the privacy risk of federated rank learning for individuals whose data is used for training the global model (global ranking)? In other words, how much information leakage does the FRL algorithm present about their individual training data samples?

5.1 Background

Though highly promising, FL faces multiple challenges [90] to its practical deployment. One of these challenges is data privacy for clients' training data. The data privacy challenge emerges from the fact that raw model updates of federation clients are susceptible to privacy attacks by an adversarial server as demonstrated by several recent works [117, 119, 134, 163, 177]. In this section, we first explain membership inference attacks and then describe two approaches that can address the privacy issues of FL.

5.1.1 Membership inference attack (MIA)

Membership inference attacks in Federated Learning (FL) have received increased attention in recent years. Nasr et al. [134] were among the first to analyze such attacks, where the adversary could be either the central server or a participant in the FL framework. With passive attackers using the gradients, activation maps, prediction vectors, loss, and true label of a single instance can classify the samples into members and non-members. Zari et al. [171] proposed an efficient passive membership inference attack that used the same idea of [134], but using only the probabilities of the correct label under local models at different epochs as the feature vector. Melis et al. [127] identified membership leakage when using FL for training a word embedding function, where the set of words used in the training sentences could be inferred from the gradients being 0 for words not appearing in a training batch. This attack assumes that participants update the central server after each mini-batch rather than after each training epoch.

5.1.2 Central differential privacy in FL (CDPFL)

In CDPFL [31, 68], a trusted server first collects all the clients' raw model updates ($\theta_i \in \mathbb{R}^d$), aggregates them into the global model, and then perturbs the model with carefully calibrated noise to enforce differential privacy (DP) guarantees. The

server provides participant-level DP by the perturbation. Formally, consider *adjacent* datasets ($X, X' \in \mathbb{R}^{n \times d}$) that differ from each other by the data of one federation client. Then:

Definition 5.1.1 (Centralized Differential Privacy (CDP)) *A randomized mechanism $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ is said to be (ε, δ)-differential private if for any two adjacent datasets $X, X' \in \mathcal{X}$, and any set $Y \subseteq \mathcal{Y}$:*

$$\Pr[\mathcal{M}(X) \in Y] \leq e^{\varepsilon} \Pr[\mathcal{M}(X') \in Y] + \delta \tag{5.1}$$

where ε is the privacy budget (lower the ε, higher the privacy), and δ is the failure probability.

Algorithm 5 shows how CDPFL works which is also discussed in [31, 68, 133]. In CDPFL, the server receives model updates capped by norm C, and after averaging them, it adds i.i.d sampled noise to the parameters $\theta_g^{t+1} \leftarrow \theta_g^t + \frac{1}{n}\left(\sum_{u \in U} \theta_u^t + \mathcal{N}(0, \sigma^2 \mathbb{I})\right)$ where σ is the noise scale multiplied by the sensitivity of the local updates.

5.1.3 Local differential privacy in FL (LDPFL)

CDPFL relies on availability of a trusted server for collecting raw model updates. On the other hand, LDPFL [162, 121] does not rely on this assumption and each client perturbs its output locally using a randomizer $\mathcal{R}$. If each client perturbs its model updates locally by $\mathcal{R}$ which satisfies $(\varepsilon_\ell, \delta_\ell)$-LDP, then observing collected updates $\{\mathcal{R}(x_1), \ldots, \mathcal{R}(x_n)\}$ also implies $(\varepsilon_\ell, \delta_\ell)$-DP [59].

Definition 5.1.2 (Local Differential Privacy (LDP)) *A randomized mechanism $\mathcal{R} : \mathcal{X} \to \mathcal{Y}$ is said to be $(\varepsilon_\ell, \delta_\ell)$-locally differential private if for any two inputs $x, x' \in \mathcal{X}$ and any output $y \in \mathcal{Y}$,:*

$$\Pr[\mathcal{R}(x) = y] \leq e^{\varepsilon_\ell} \Pr[\mathcal{R}(x') = y] + \delta_\ell \tag{5.2}$$

Algorithm 5 Central Differential Privacy in FL (CDPFL)

Input: number of FL rounds T, number of local epochs E, number of all the clients N, number of selected users in each round n, total privacy budget TP, probability of subsampling clients q, learning rate η, noise scale z, bound C

Output: global model θ_g^T

$\theta_g^0 \leftarrow$ Initialize weights
Initialize MomentAccountant(ε, δ, N)
for each iteration $t \in [T]$ **do**
    $U \leftarrow$ set of n randomly selected clients out of N total clients with probability of q
    $p_t \leftarrow$ MomentAccountant.getPrivacySpent() $\triangleright$ % privacy budget spent till this round
    if $p_t > TP$ **then**
        return θ_g^T $\triangleright$ % if spent privacy budget is passed over the threshold finish FL training
    end if
    for u in U **do**
        $\theta \leftarrow \theta_g^t$
        for local eopoch $e \in [E]$ **do**
            for batch $b \in [B]$ **do**
                $\theta \leftarrow \theta - \eta \nabla L(\theta, b)$
                $\triangle \leftarrow \theta - \theta_g^t$
                $\theta \leftarrow \theta_g^t + \triangle \min\left(1, \frac{C}{||\triangle||_2}\right)$
            end for
        end for
        Client u sends $\theta_u^t = \theta - \theta_g^t$ to the server
    end for
    $\sigma \leftarrow zC/q$
    $\theta_g^{t+1} \leftarrow \theta_g^t + \frac{1}{n} \sum_{u \in U} \theta_u^t + \mathcal{N}(0, \sigma^2 \mathbb{I})$
    MomentAccountant.accumulateSpentBudget(z)
end for
return θ_g^T

In LDPFL, each client perturbs its local update (θ_i) with $(\varepsilon_\ell, \delta_\ell)$-LDP. Unfortunately, LDP hurts the utility, especially for high-dimensional vectors. Its mean estimation error is bounded by $O(\frac{\sqrt{d \log d}}{\varepsilon_\ell \sqrt{n}})$ meaning that for better utility we should increase the privacy budget or use larger number of users in each round [25].

5.2 Privacy analysis setup

In this section, we explain different settings, hyperparameters, etc, about how we measure the privacy leakage of local updates in FedAvg vs. FRL.

5.2.1 Membership inference attacks (MIA)

In order to compare the privacy leakage between Federated Averaging (FedAvg) with local weight parameters (16/32 bits float numbers) and Federated Rank Learning (FRL) with local ranks (permutation of integers), we adopt the passive whitebox access membership inference attack method from Nasr et al. [134]. This attack conducts an analysis of membership inference attacks on neural networks under federated learning with access to some training members and some training non-members (not used in training but drawn from the same distribution). The adversary can be the central server or one of the FL participants. In their work, passive attackers perform the training normally following the FL protocol. Nasr et al. showed that they can leverage information such as gradients, activation maps, prediction vectors, loss, and true label of instances obtained from the local models of each user at different epochs during training to conduct MIAs.

5.2.2 FL setting

In our federated learning experiments, we consider a scenario with 5 participating clients in each round of training ($N = n = 5$). The clients communicate with the central server after each local epoch ($E = 1$). For example, with CIFAR-100 dataset, each client has 10,000 instances, leading to 1,250 steps of SGD (with a mini-batch size of $B = 8$) before sending the updated parameters to the central server.

In our experiments, we evaluate the trustworthiness of federated learning on two widely used datasets, CIFAR10 and CIFAR100. We focus on the most challenging setting of membership inference attacks, where the client's data is drawn i.i.d from the training distribution.

Table 5.1. Dataset sizes in the experiments of FRL and FedAVG.

Dataset	FL training		Inference Attack Model			
	FL Client Training	Test	Training Members	Training Non-members	Test Members	Test Non-members
CIFAR100	10,000	10,000	5,000	5,000	5,000	5,000
CIFAR10	10,000	10,000	5,000	5,000	5,000	5,000

Table 5.2. Final test accuracies of different FL algorithms.

Dataset	FedAvg	FRL
CIFAR100	57.46	56.15
CIFAR10	82.55	80.12

To assess the membership inference attack against each participant, we create a balanced evaluation set in line with existing literature (Nasr et al. [134]; Shokri et al. [150]; Carlini et al. [37]), which consists of half members and half non-members. The data distribution for the federated learning training and the number of members and non-member samples available to the attacker is presented in Table 5.1. The non-member samples, which are not used in the local model's training, are drawn from the same distribution as the training data. We use ReNet18 and Conv8 to train CIFAR100 and CIFAR10 with 500 FL rounds in our experiments. Table 5.2 shows the final test accuracies for the FedAvg and FRL training in our experiments.

5.2.3 Evaluation metrics

In this work, we use attack accuracy as a metric to measure the privacy leakage of the data samples used in the training of local models. We also report the True Positive Rate (TPR) at a low False Positive Rate (FPR) to evaluate the performance of the membership inference attacks, following the methodology introduced by Carlini et al. [37]. A higher TPR at low FPR value indicates a successful attack that is able to generate more true positive results than false positive results.

5.3 Privacy analysis of FRL vs. FedAvg

In this study, we aim to compare the privacy leakage between the FedAvg and FRL algorithms. To achieve this, we have designed two evaluation scenarios: 1) Global

attacker: In this scenario, the attacker has access to the local updates of the target client. For example, if the central server is adversarial and attempts to infer the presence of a specific data sample in the target client's training data, or if the attacker can eavesdrop on the connection between the target client and the central server, and observe the actual local updates in each FL round. In this scenario, we can evaluate the privacy leakage of type of local updates by observing the weight parameters in FedAvg compared to the rankings in FRL. 2) Local attacker: In this scenario, the attacker has access to the global model. For example, if one of the participating clients is adversarial or if the attacker can obtain the global model. In this scenario, we can evaluate the privacy leakage of the aggregation method in addition to the type of local updates. This means we can assess the majority vote aggregation rule in FRL compared to the average method in FedAvg, as well as the type of updates.

5.3.1 Measuring privacy leakage: Local attacker

In this section, we conduct experiments to assess the privacy of FL against a local passive attacker. The attacker is limited to observing only the aggregate model parameters of each FL round and cannot access the individual model updates of the participants. Our experiments involve five participants, including the local attacker, and the objective of the attacker is to determine if a specific target input was included in the training data of any other participants. We adopt the attack model architecture used by Nasr et al. [134] to conduct these experiments.

5.3.1.1 Impact of observed epochs

We examine the impact of observing different numbers of FL rounds on the accuracy and true positive rate of membership inference attack launched by a local passive attacker. Our results are based on two datasets, CIFAR10 and CIFAR100. To reduce computational resource limitations, we only input a global model of five FL rounds to the attack network.

Table 5.3. The accuracy and TPR of the passive local attacker in the federated setting when the attacker uses various training epochs.

Dataset	Observed Epochs	Metric			
		FedAvg		FRL	
		Attack Accuracy	TPR @ FPR=1%	Attack Accuracy	TPR @ FPR=1%
CIFAR10	[10,20,30,40,50]	59.11	5.51	54.83	3.18
	[50,100,150,200,250]	68.40	18.70	59.35	6.41
	[460,470,480,490,500]	68.00	19.16	63.21	11.05
	[100,200,300,400, 500]	68.71	20.15	63.67	11.32
CIFAR100	[10,20,30,40,50]	56.67	6.34	52.51	2.16
	[50,100,150,200,250]	83.37	30.28	54.60	2.34
	[460,470,480,490,500]	90.61	41.30	56.61	4.87
	[100,200,300,400, 500]	88.76	39.01	59.07	11.16

Table 5.3 compares the results of the membership inference attack between standard FL and Federated Rank Learning (FRL). Our results show that observing the late FL rounds leaks more information compared to the initial rounds, as the models converge at the end of the training. For instance, if the attacker observes the global model of [10,20,30,40,50] FL rounds, the attack accuracy and true positive rate (for 1% false positive rate) are 56.67% and 6.34% for CIFAR100, and 59.11% and 5.51% for CIFAR10. On the other hand, observing the global ranking in FRL leads to lower attack accuracy and true positive rate, with values of 52.51% and 2.16% for CIFAR100, and 54.83% and 3.18% for CIFAR10.

Our results show that local rankings of FRL leak less information about the training data compared to weight parameters in standard FL. This holds true across all scenarios, from observing the global model of initial and small distance FL rounds to large-distance FL rounds. For instance, if the attacker observes the global model of [100,200,300,400,500] FL rounds, the attack accuracy and true positive rate (for 1% false positive rate) are 88.76% and 39.01% for CIFAR100, and 68.71% and 20.15% for CIFAR10 with standard FL. In contrast, observing the local rankings in FRL results in 59.07% attack accuracy and 11.16% true positive rate for CIFAR100, and 63.67% attack accuracy and 11.32% true positive rate for CIFAR10. These results demonstrate

the privacy benefits of using local rankings in FRL over weight parameters in standard FL.

5.3.1.2 Impact of the training size

Table 5.4 presents the results of the membership inference attack accuracy for different sizes of the attacker's training data in two datasets, CIFAR10 and CIFAR100. The attack is launched on both standard Federated Learning (FL) and Federated Rank Learning (FRL) by observing the global model of the [100,200,300,400,500] FL rounds. The attack is performed on the same test set for all the scenarios. The results show that as expected, the accuracy of the membership inference attack improves as the size of the attacker's training data increases. For instance, when the attacker has access to 2000 members and 2000 non-members, the attack accuracy and true positive rate (for 1% false positive rate) are 68.57% and 8.66% for CIFAR10 and 87.86% and 34.0% for CIFAR100 in standard FL. On the other hand, in FRL, the attacker achieves a lower success rate with 62.35% attack accuracy and 7.28% true positive rate for CIFAR10 and 58.58% attack accuracy and 5.90% true positive rate for CIFAR100. With larger training data, for example 5000 members and 5000 non-members, the attack accuracy and true positive rate (for 1% false positive rate) are 88.76% and 39.01% for CIFAR100, and 68.71% and 20.15% for CIFAR10 in standard FL.

Table 5.4. Attack accuracy and TPR for various sizes of the attacker's training dataset.

Dataset	Member Sizes	Non-member Sizes	Metric			
			FedAvg		FRL	
			Attack Accuracy	TPR @ FPR=1%	Attack Accuracy	TPR @ FPR=1%
CIFAR10	2,000	2,000	68.57	8.66	62.35	7.28
	2,000	5,000	68.98	18.30	62.94	7.86
	5,000	2,000	68.69	19.65	63.52	10.02
	5,000	5,000	68.71	20.15	63.67	11.32
CIFAR100	2,000	2,000	87.86	34.00	58.58	5.90
	2,000	5,000	88.44	35.06	58.90	8.56
	5,000	2,000	87.92	36.37	58.64	8.40
	5,000	5,000	88.76	39.01	59.07	11.16

5.3.2 Measuring privacy leakage: Global attacker

In this investigation, a global attacker has access to the parameter updates of each participant at each FL round. The attacker passively collects all updates from each participant and performs the membership inference attack against each target participant individually. Due to resource constraints, the attack only observes each target participant during five non-consecutive training epochs. Table 5.5 shows the accuracy of the attack using different sets of training epochs for the datasets CIFAR10 and CIFAR100, and we report the average of metrics for all the targets. The results indicate that using later epochs significantly increases the attack accuracy. This is due to the fact that earlier training epochs contain generic features of the dataset, which do not reveal significant membership information, while later epochs contain more membership information as the model starts to learn the outliers in such epochs [134].

Table 5.5. The accuracy and TPR of the passive global attacker in the federated setting when the attacker uses various training epochs.

Dataset	Observed Epochs	Metric			
		FedAvg		FRL	
		Attack Accuracy	TPR @ FPR=1%	Attack Accuracy	TPR @ FPR=1%
CIFAR10	[10,20,30,40,50]	65.95	11.17	59.09	5.87
	[50,100,150,200,250]	68.88	19.75	61.43	8.62
	[460,470,480,490,500]	68.75	19.85	65.46	15.44
	[100,200,300,400, 500]	69.70	20.27	64.03	16.32
CIFAR100	[10,20,30,40,50]	72.82	20.22	56.05	4.12
	[50,100,150,200,250]	84.94	30.46	57.04	6.93
	[460,470,480,490,500]	90.76	41.92	62.69	25.68
	[100,200,300,400, 500]	88.88	39.93	60.40	15.76

Comparing the global and local attacks, the global attack has higher success as it observes the aggregate model parameters of all participants, allowing for a larger extent of membership leakage. For instance, if the local attacker observes the global model of [100,200,300,400,500] FL rounds in standard FL, the attack accuracy and true positive rate (for 1% false positive rate) are 88.76% and 39.01% for CIFAR100 and 68.71% and 20.15% for CIFAR10. Observing local rankings in FRL results in

59.07% attack accuracy and 11.16% true positive rate for CIFAR100, and 63.67% attack accuracy and 11.32% true positive rate for CIFAR10. On the other hand, the global attacker can achieve a higher success rate in both standard FL and FRL. For CIFAR10, the attacker achieves 69.70% attack accuracy and 20.27% true positive rate in standard FL and 64.03% and 16.32% in FRL. For CIFAR100, the attacker achieves 88.88% attack accuracy and 39.93% TPR in standard FL and 60.40% attack accuracy and 15.76% TPR in FRL.

5.4 Differential Privacy and FRL

In this section, we present a differential private version of federated rank learning (FRL). We begin by elucidating the concept of differential privacy in rankings and then extend it to private rank aggregation in FRL.

In the field of rank aggregation, the objective is to derive a consolidated ranking that accurately reflects the input rankings from a group of individuals. As a result, numerous rank aggregation algorithms have been proposed in the literature [79, 10, 55, 58, 108, 60]. An example of its application can be seen in recommendation systems, where rank aggregation is used to integrate users' preferences into a single ranking that aligns with those preferences.

In the context of rank aggregation, a ranking refers to an ordered list of elements from a universe U of elements. The ranking can be represented as $\tau = \{x_1, x_2, ..., x_m\}$ where each $x_i \in U$ and m is the number of elements in the universe. The position of an element x in a ranking τ can be denoted by $\tau(x)$. Rankings are typically sorted in ascending or descending order based on the level of preference for each element. A dataset of rankings, $T = \{\tau_1, \tau_2, ..., \tau_n\}$, is created when n users provide their rankings. The objective of rank aggregation is to find a single, representative ranking that reflects the rankings within the dataset T.

5.4.1 Sensitivity of local rankings

In the context of Differential Privacy, the amount of noise required to ensure privacy is proportional to the sensitivity of the outputs. The sensitivity of the outputs measures the maximum possible change in the result that could occur due to the addition or removal of a single record.

Definition 5.4.1 *The quality of an aggregate ranking is typically measured by* Kendall tau *distance. This metric measures the number of pairwise disagreements between two permutations. The Kendall distance of two rankings R_1 and R_2 is:*

$$\boldsymbol{K}(R_1, R_2) = |\{(i,j) \ : \ R_1(i) < R_1(j) \text{ and } R_2(i) > R_2(j)\}| \tag{5.3}$$

The maximum possible Kendall tau distance occurs when R_2 is the reverse rankings of R_1; In this case $\mathbf{K}(R_1, R_2) = \frac{d(d-1)}{2}$ where d is the size of rankings.

In Figure 5.1, we evaluate the Kendall tau distance between the rankings of each client, for before and after local updates, across multiple FL rounds. The experiment features 5 clients and focuses on the last layer ranking (i.e., a fully connected layer with 2560 edges). The distance is normalized by dividing it by $\frac{2560\times2559}{2}$, the maximum possible value. The blue line in the figure represents the median distances of the clients at each FL round. The results show that initially, the difference between the global ranking and the local ranking after the update is substantial, with an average of 34065 disagreements (0.0104 as normalized) in the first rounds. However, as the FL training progresses, this difference decreases, reaching an average of 6878 disagreements (0.0021 as normalized) at the end. This reduction in disagreements and sensitivity of the updates highlights the convergence of the model rankings throughout FL training.

The concept of Differential Privacy provides a framework for quantifying and controlling the privacy loss incurred by individuals. It asserts that, given a privacy parameter ϵ, the output of an algorithm that processes sensitive information will

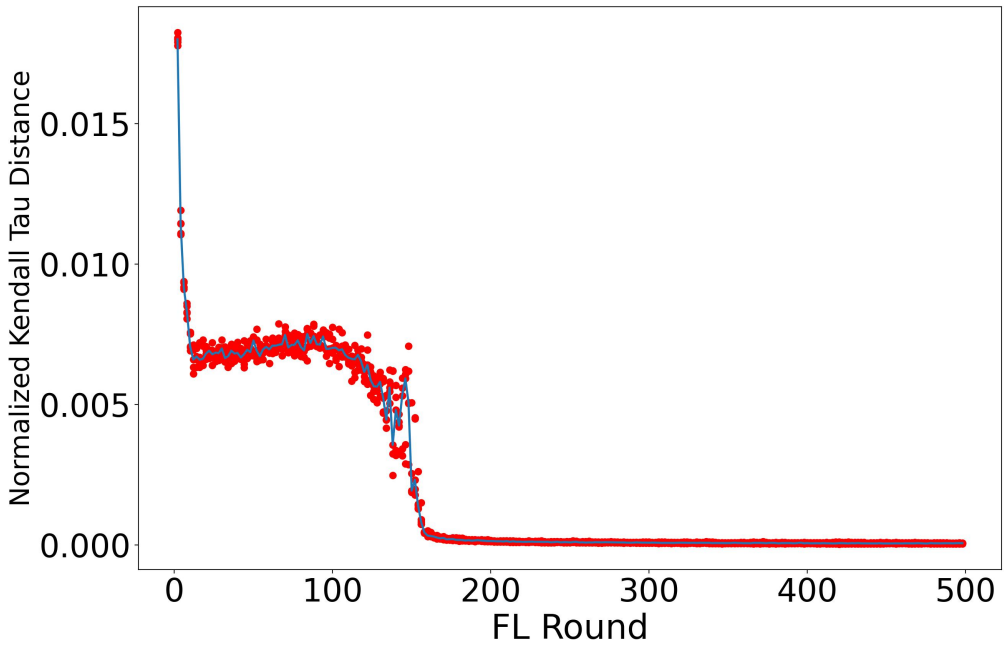


Figure 5.1. Normalized Kendall tau distance of rankings in federated rank learning (FRL) for the last layer with 2560 ranks over multiple FL rounds.

be approximately the same, regardless of whether or not the information of a single individual is included. This property is formalized as a characteristic of an algorithm that operates on a database, where the database D consists of records that represent the sensitive information of individual subjects. Two databases, D and D', are considered neighbors if they differ by only one tuple. We define two sets of rankings, $T = \{\tau_1, \tau_2, ..., \tau_n\}$ and $T' = \{\tau'_1, \tau'_2, ..., \tau'_n\}$, to be neighbors if they differ only in one ranking.

5.4.2 Borda Count Aggregation

The rank aggregation method used in each round of Federated Rank Learning (FRL) is the Borda count method [60]. The Borda count method was originally introduced as a single-winner selection technique, where each item is assigned a certain number of points based on its position in a given ranking. For instance, in a universe of

five elements, an element will receive 0 points every time it is ranked first, 1 point for second place, and so on. The element with the lowest number of points is determined as the winner.

Definition 5.4.2 *(Borda Scores) Given a ranking database* $T = \{\tau_1, \tau_2, ..., \tau_n\}$ *over a universe* $U = \{x_1, \ldots, x_d\}$*, the Borda score of element* x_i *is defined as* $\text{BSCORE}(x_i) = \sum_{j=1}^{n} \tau_j(x_i)$.

The Borda scores can be utilized as a rank aggregation method by sorting the elements according to their Borda scores. The Borda score aggregator is computationally efficient and, when ties are resolved arbitrarily, has been shown to be a 5-approximation of the optimal rank aggregation [79, 12].

5.4.3 Private Borda Count Aggregation

Differential privacy provides a strong, proven guarantee of privacy, ensuring that the result of any analysis on a database is not significantly affected by the presence or absence of a single individual's data. Rankings pose a challenge when working with differential privacy due to their high dimensionality (the number of ranked items) and the limitations in analyzing rankings and sets of rankings.

Similar to [79], we use Borda scores as the foundation for a differentially private ranking method by first privately computing an estimate of the Borda score for each candidate using the Laplace mechanism. Then, the noisy scores are sorted to obtain an aggregate ranking.

Definition 5.4.3 *(Private Borda Scores) Given a ranking database* $T = \{\tau_1, \tau_2, ..., \tau_n\}$ *over a universe* $U = \{x_1, \ldots, x_d\}$ *and privacy budget* ε*, the P-BORDA algorithm returns* $[b_1, \ldots, b_d]$ *where* $b_i \in \mathbb{R}$ *represents a noisy estimate of the Borda score of element* i*. For each* $i \in [1, d], b_i = bscore(x_i) + Z$*, where* $Z \sim Laplace\left(\frac{d(d-1)}{2\varepsilon}\right)$.

5.4.4 Differential Private FRL (DP-FRL)

In Federated Rank Learning (FRL), we incorporate the Private Borda count method (P-BORDA [79]). We apply differential privacy to ranking contexts by treating an individual's information as their complete ranking of all elements in U, thus providing robust privacy protection for all preference information they may disclose by participating in the database. Alternative approaches, which might only protect parts of an individual's ranking or just pairwise comparison information, would be less secure: for instance, they might protect an individual's top preferences but expose preferences about lower-ranked elements. Consequently, we consider two rank databases T and T' as neighboring if they differ by the presence or absence of precisely one ranking.

We employ the Laplace mechanism to introduce noise to the scores, which are then sorted to establish the global rankings. Nonetheless, due to the high dimensionality of the rankings in FRL, their sensitivity is also considerable, resulting in a substantial amount of noise being added to the rankings. Consequently, DP-FRL might not converge owing to the excessive noise introduced into the Borda scores.

5.5 FRL against FL with differential privacy

In this section, we evaluate the privacy of FRL in comparison to FL algorithms that aim to preserve privacy. Table 5.6 presents a comparison of utility and privacy leakage for standard FL (FedAvg), FedAvg with norm bounding at the server, FL with local differential privacy (LDPFL [133, 121, 162]), and centralized differential privacy FL (CDPFL [31, 68, 133]) when learning CIFAR100 using the ResNet18 model architecture. For LDPFL and CDPFL, we employ the code provided in [133]. A threshold bound of $C = 1.0$ is utilized to clip the updates in these FL algorithms, and the momentum accountant [6] is used to monitor the spent privacy loss.

Table 5.6. Comparison of utility and privacy leakage for Federated Learning algorithms - FedAvg, FedAvg with Norm-bounding, LDPFL, CDPFL, and FRL, on CIFAR100 with ResNet18 model architecture.

FL type	Metric		
	Global model Accuracy	Membership Inference Attack	
		Attack Accuracy	TPR @ FPR=1%
FedAvg	57.46	88.76	39.01
Norm-bound ($C = 1.0$)	47.51	52.43	5.92
LDPFL ($C = 1.0, \varepsilon = 9.8, \delta = 10^{-5}$)	41.23	51.20	1.84
CDPFL ($C = 1.0, \varepsilon = 5.0, \delta = 10^{-5}$)	43.90	53.39	2.32
FRL	56.15	59.07	11.16

From Table 5.6, we observe that FedAvg offers the best utility (i.e., accuracy of 57.46%) but suffers from the highest privacy leakage, allowing a local attacker to achieve an 88.76% accuracy and a true positive rate of 39.01% when the false positive rate is 1%. This is due to the lack of defense mechanisms in this FL algorithm. By implementing norm bounding, CDPFL, and LPDFL, we can enhance privacy at the cost of reduced accuracy. For instance, CDPFL lowers the model accuracy to 43.90% while improving privacy, with the attack accuracy reduced to 53.39% and a true positive rate of only 2.32% when the FPR is 1%. FRL strikes a balance by achieving a respectable test accuracy of 56.15% (close to FedAvg) while limiting membership inference attack accuracy to 59.07% (similar to privacy-preserving FL algorithms). Privacy attack on FRL also results in a TPR of 11.16% when the FPR is 1%.

5.6 Conclusion

In this chapter, we explore the privacy leakage of ranking updates in Federated Rank Learning (FRL). We employ the membership inference attack developed by Nasr et al. [134] to assess the privacy leakage of local updates and the aggregation rule in FRL. Additionally, we compare the privacy leakage of FRL to that of standard FL, demonstrating that FRL protects the privacy of private training data from both local attackers (who observe the aggregated global model) and global attackers (who

observe local ranking updates). This enhanced privacy comes at the expense of a slight decrease in global model accuracy. We evaluate the privacy leakage using two metrics of attack accuracy and report the true positive rate for a low false positive rate (i.e., 1%).

We also evaluate the sensitivity of local rankings in FRL using the Kendall Tau distance. Moreover, we propose a differentially private aggregation rule for FRL based on adding Laplace noise to the outcome of the Borda count aggregation rule. Given the large size of rankings in FRL, the sensitivity of the rankings is also substantial, resulting in significant noise being added to the outcome of the aggregated rankings. We observe that FRL with differential privacy fails to converge due to the magnitude of noise added to the final rankings in each FL round.

CHAPTER 6

FAIR FEDERATED LEARNING BY TRAINING ON RANKS

In Federated Learning (FL), the performance of the global model varies across the clients due to heterogeneity in the data that each client owns. This concern is called *representation disparity* [78] and results in unfair performance gaps for the participating clients. That is, although the accuracy may be high on average, some tail user whose data distribution differs from the majority of the clients is likely to receive a much lower performance compared to the average.

In this chapter, we look at FL fairness with two different lenses: **a)** *Equality*: whose goal is providing similar performances for all individual clients; **b)** *Equity*: whose goal is providing similar performances across all groups of clients (i.e., groups of majority and minority), where a group is defined as a set of clients with similar data distributions. The key question we try to answer is: *Can we design an efficient federated learning algorithm that achieves both equality and equity concurrently?*

Due to the heterogeneity in clients' data distributions, one single model cannot represent all the clients equally. There is a *trade-off* between training one global model and multiple global models; if we train one global model, all the clients can utilize each other's knowledge; however, it will be biased towards those that have the majority of the population. On the other hand, if we train multiple models (e.g., as in IFCA [70], HypCluster [125] and MOCHA [153]), we improve fairness, but each global model will lose the knowledge from excluded clients. To get the best of both worlds, we present **E**qual and **E**quitable **F**ederated **L**earning (E2FL), a novel FL algorithm to

achieve both equality and equity. In E2FL, we train multiple global models, but in each round, we combine all of the models into one global model to take advantage of the knowledge of all client groups.

The key insight used in E2FL is converting the problem of model weight optimization (in standard FL) to the problem of ranking model edges (Chapter 3). Therefore, in each round of E2FL training, the clients and the server exchange rankings for the edges of a randomly initialized neural network (called *supernetwork*), as opposed to exchanging parameter gradients. More specifically, each client computes the importance of the edges within the supernetwork on their local data, represented by a ranking vector. Next, E2FL server uses a majority voting mechanism to aggregate the collected local rankings into multiple global rankings based on the index of group they belong to. Finally, the E2FL server aggregates all the group rankings into one global ranking for next round of training. Applying the majority vote on the group rankings instead of all the local rankings helps E2FL enforce equity because each group has an equal vote to influence the global model. To provide equality in E2FL, if a client wants to use the model in a downstream task, they use their own group global ranking, instead of the global ranking, which is a better representation model for the client and its groupmates.

Our ranking-based FL training enables attractive fairness properties, as shown through our experiments, which is intuitively due to the following reason: In rank-based federated learning, each client computes a local ranking (i.e., a permutation of integers $\in [1, d]$ where d is the layer size), so each local ranking has a fixed norm (i.e., $\sqrt{1^2 + 2^2 + ... + d^2}$). This fixed norm of local updates makes the rank aggregation more fair as each local ranking has the same impact on the aggregated global ranking. On the other hand, in standard FL, when the server aggregates the local model updates into the global model, each local update has a different impact on the global model (because of their different l_2 norms). For example, in FedAvg, the server averages the

parameter updates for the d dimensions, therefore a large parameter update has more influence on the final average compared to a small parameter update.

E2FL when the group IDs are unknown. In many applications, clients may be unaware of their protected attributes (i.e., the group they belong to). We propose one approach on server-side and three approaches on client-side for inferring group IDs. To infer the group IDs on the server-side, we propose to use a rank clustering approach to cluster clients into groups. Moreover, a client can also infer its group ID by picking the right group based on their local training data. Using rankings allows us to exchange only the binary masks produced by each group ranking which lowers the communication cost compared to prior works. Each client can pick the right binary mask based on three approaches. First, each client can pick the binary mask that produces the smallest loss. Binary masks also enable the clients to find their matching group by a new novel idea from [164], where clients can infer the group ID using gradient based optimization to find a linear superposition of learned masks which minimizes the output entropy. We propose two variants of this approach, one based on a binary search and the other using OneShot optimization.

6.1 Fairness using two lenses: Equity and Equality

Fairness in FL can be evaluated from two main perspectives: a) *Equality*, which is fairness between individuals, and b) *equity*, which is fairness between groups. A group is a set of individuals with the same protected attribute. The protected attribute may be known to the clients, e.g., race, gender, or age. Alternatively, clients may be unaware of their particular group, e.g., handwriting style (as this needs clustering clients into groups by someone who has samples from all clients).

Figure 6.1 shows an example of two FL systems where six clients want to learn a global model for prediction of handwritten digits. These clients have three handwriting styles: **(A)** normal handwriting style, **(B)** a little bit rotated handwriting, and **(C)**

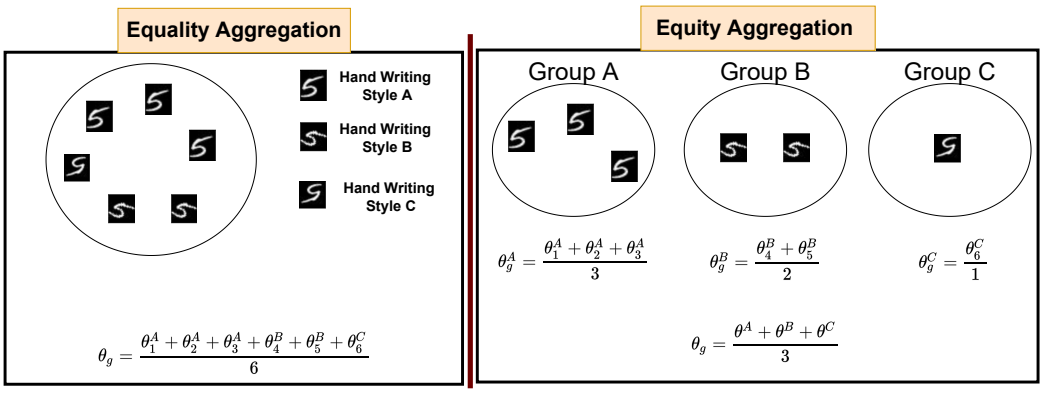


Figure 6.1. An example showing two different FL systems with two goals: equality (on left) and equity (on right).

180 degree rotated handwriting (upside-down). We consider each model update (θ_u^q for client u in group q) has the same effect on updating the global model, so each client update is like a vote. In this example, group A has the majority of the voters, and groups B and C are minorities. The left part of the figure shows an FL in which the goal is providing equality, so we give each client the same chance (one vote) to change the final model by an aggregation such as averaging (e.g., what we have in FedAvg). In this setting, the majority group with a higher population (group A) has more influence on the final vote. On the other hand, the right part shows an FL in which the goal is to provide equity. In this setting, first, we aggregate the votes inside each group to find the group votes ($\theta_g^A, \theta_g^B, \theta_g^C$), and then aggregate the group votes to produce the final model. In this setting, each client has the same chance (one vote) to influence its own group vote, and finally, each group of voters has the same chance (one vote) to influence the final vote. We define two aspects of fairness in FL as follows:

Definition 1 (Equality: User-level fairness): Trained global model θ is more *equalized* when its performance is more uniform across the individual clients participating in FL, i.e., when $\text{STD}\{F_u(\theta)\}_{u\in[N]}$ is smaller where STD{.} is the standard deviation, and $F_u(.)$ denotes the local objective function of client u from N clients.

Existing fair federated learning literature [116, 113, 153, 78, 174, 131, 170] use this definition in their designs.

Definition 2 (Equity: Group-level fairness): Trained global model θ is more *equitable* when its performance is more uniform across the groups, i.e., when $\text{STD}\{\text{Avg}\{F_u(\theta)\}_{u\in[q]}\}_{q\in[Q]}$ is smaller where $\text{AVG}\{\}_{u\in[q]}$ denotes the average of performances for all the individual clients in the qth group, and there are Q total groups.

In E2FL, our goal is to provide both equity and equality. To provide equity, an individual client has one vote in their group, and each group has one vote among all the groups. To provide equality, we allow the clients in each group use their group model, which represents their training data better.

6.2 Our design: Equal and Equitable Federated Learning

This section provides the design of our Equal and Equitable Federated Learning (E2FL) algorithm. We first describe how E2FL provides equity and then discuss how it provides equality. The intuition behind E2FL is to train multiple global models and first perform a majority vote among the clients' model updates in each group and then another majority vote among the group models to find the global model. Algorithm 6 describes E2FL training.

The critical insight used in E2FL is converting the problem of model weight optimization (in standard FL) to the problem of ranking model edges (FRL in Chapter 3). In E2FL, each local update (one vote) is a ranking, i.e., a permutation of integers $\in [1, d]$ where d is the size of the network layer. We use rankings because of their intrinsic fairness feature: In rank aggregation, each local ranking has the same impact on the aggregated global ranking. In rank-based FL, all the local rankings are bounded to be a permutation of unique integers $\in [1, d]$. For example, for a network layer with $d = 3$ parameters, there are only 3! possible permutations for local ranking $([1, 2, 3], \ldots, [3, 2, 1])$. However, in existing standard FL designs, the local updates

Algorithm 6 Equal and Equitable Federated Learning (E2FL) Algorithm.

Input: number of FL rounds T, number of local epochs E, number of selected users in each round n, number of groups Q, seed SEED, learning rate η, subnetwork size $k\%$
$\theta^s, \theta^w \leftarrow$ Initialize random scores and weights of global model θ using SEED
$R_g^1 \leftarrow \text{ArgSort}(\theta^s)$ ▷ Sort the initial scores and obtain initial global rankings
for $t \in [1, T]$ **do**
    $U \leftarrow$ set of n randomly selected clients out of N total clients
    for u in U **do**
        $\theta^s, \theta^w \leftarrow$ Initialize scores and weights using SEED
        If q (group ID) is not known, use Algorithms 7 and 8 for ID inference (Section 6.3)
        $\theta^s[R_g^t] \leftarrow \text{SORT}(\theta^s)$ ▷ Reorder the scores based on the global ranking
        $\theta_u^s \leftarrow \text{Edge-PopUp}(E, D_u^{tr}, \theta^w, \theta^s, k, \eta)$ ▷ Train local scores on the local training data
        $R_{u,q}^t \leftarrow \text{ArgSort}(\theta_u^s)$ ▷ Ranking of the client u with estimated group ID: q
        return $R_{u,q}^t$
    end for
    $R_{g,q\in[Q]}^{t+1} \leftarrow \text{Vote}(R_{u\in U,q\in[Q]}^t)$ ▷ Majority vote aggregation inside each group
    $R_g^{t+1} \leftarrow \text{Vote}(R_{g,q\in[Q]}^{t+1})$ ▷ Majority vote aggregation among all the groups
end for
function Vote $(R_{\{u\in U\}})$
    $V \leftarrow \text{ArgSort}(R_{\{u\in U\}})$ ▷ Reputation of each edge in each local ranking
    $A \leftarrow \text{Sum}(V)$ ▷ Sum the reputations
    return $\text{ArgSort}(A)$ ▷ Order of the reputations
end function

($\in \mathbb{R}^d$) have different impacts on the aggregated global model because the direction and magnitude of each parameter update is not bounded to other parameters.

In E2FL, different FL clients gather together to learn a global model, but each one belongs to a different group (which could be considered known or unknown). In this section, we assume the clients know their group IDs, and in Section 6.3, we explain how the clients can infer their group IDs using the features of rankings when the groups are unknown. In E2FL, the server trains multiple global rankings, each belonging to a different group. These global group rankings show different orders of importance of the same supernetwork for different groups from least to most important edges. Each client participates in the training of their group model by sending the local ranking they have. For aggregation, the server performs a majority vote among the local rankings (local votes) in each group. It then performs another majority vote among global group rankings (group votes) to find the global model for the next round (i.e., a global ranking that clients will start their training for the next E2FL round).

Edge-PopUp algorithm: The edge-popup (EP) algorithm [143] is an optimization method to find supermasks within a large, randomly initialized neural network, i.e., a supernetwork, with performances close to the fully trained supernetwork. EP algorithm does not train the weights of the network; instead only decides the set of edges to keep and removes the rest of the edges (i.e., pop). Specifically, the EP algorithm assigns a positive score (θ^s) to each of the edges in the supernetwork and updates it. In E2FL, each client learns its local scores θ^s_u by using EP on its local data for E local epochs starting from the global scores θ^s. On the forward pass, it selects the top $k\%$ edges with the highest scores, where k is the percentage of the total number of edges in the supernetwork that will remain in the final subnetwork. On the backward pass, it updates the scores with the straight-through gradient estimator [21].

6.2.1 E2FL: Design

This section explains the different steps of one round of E2FL training. We detail a round of E2FL training and depict it in Figure 6.2, where we use a supernetwork with six edges $e_{i\in[0,5]}$ to demonstrate a single E2FL round and consider six clients $C_{j\in[1,6]}$ from three groups (handwriting style A, B, C) who aim to find a subnetwork of size k=50% of the original supernetwork.

6.2.1.1 Server: Initialization phase (only for round $t = 1$)

In the first round, the E2FL server chooses a random seed Seed to generate initial random weights θ^w and scores θ^s for the global supernetwork θ; note that, θ^w, θ^s, and Seed remain constant during the entire E2FL training. Next, the E2FL server shares Seed with E2FL clients, who can then locally reconstruct the initial weights θ^w and scores θ^s using Seed. Figure 6.2-① depicts this step. Recall that the goal of E2FL training is to find the most important edges in θ^w without changing the weights. At the beginning, the E2FL server finds the global rankings of the initial random scores, i.e., $R^1_g = \textsc{ArgSort}(\theta^s)$. We define *rankings of a vector* as the indices

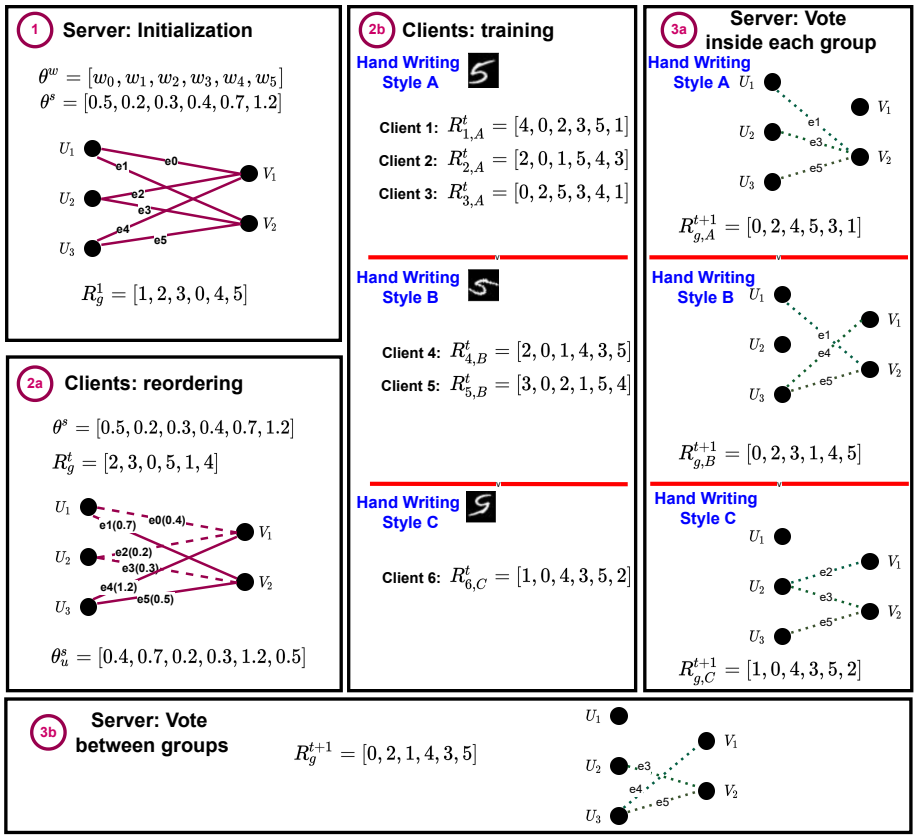


Figure 6.2. A single E2FL round with six clients from three groups and a network of 6 edges. Note that all the operations in E2FL training are performed in a layer-wise manner.

of elements of a vector when the vector is sorted from low to high, which is computed using ArgSort function.

6.2.1.2 Clients: Calculating the ranks (for each round t)

In the t^{th} round, E2FL server shares the global rankings R_g^t with the clients. Each client locally reconstructs the weights θ^w's and scores θ^s's using SEED. Then, each E2FL client reorders the random scores based on the global rankings, R_g^t. We depict this in Figure 6.2-(2a). For instance, the initial global rankings for this round are $R_g^t = [2, 3, 0, 5, 1, 4]$, meaning that edge e_4 should get the highest score ($s_4 = 1.2$), and edge e_2 should get the lowest score ($s_2 = 0.2$).

Next, each client uses reordered θ_u^s and finds a subnetwork within θ^w using edge-popup algorithm [143]; to find a subnetwork, they use their local data and E local epochs. Note that each iteration of the edge-popup algorithm updates the scores θ_u^s. Then client u computes their local rankings R_u^t using the final updated scores and ARGSORT(.), and sends $R_{u,q}^t$ to the server where q is the group identifier. We will explain the group inference methods we propose in Section 6.3. Figure 6.2-(2b) shows, for each client, the local rankings they obtained after finding their local subnetwork. For example, rankings of client C_1 are $R_{1,A}^t = [4, 0, 2, 3, 5, 1]$, i.e., e_4 is the least important and e_1 is the most important edge for C_1. Considering desired subnetwork size to be 50%, C_1 uses edges {3,5,1} in their final subnetwork in this round.

6.2.1.3 Server: Majority Vote (for each round t)

The server receives all the local rankings of the clients, i.e., $\{R_{1,A}^t, R_{2,A}^t, R_{3,A}^t, R_{4,B}^t, R_{5,B}^t, R_{6,C}^t\}$. Then, it performs a majority vote over all the local rankings inside each group, i.e., $\{A, B, C\}$. We depict this in Figure 6.2-(3a). Note that, for group q, the index i in $R_{g,q}^{t+1}$ represents the importance of the edge ith for clients in group q. For instance, in Figure 6.2-(3a), rankings of A are $R_{g,A}^t = [0, 2, 4, 5, 3, 1]$ and rankings of B are $R_{g,B}^t = [0, 2, 3, 1, 4, 5]$, hence the edge e_1 is the most important edge for group A, while the edge e_5 is the most important edge for group B. Next, the server performs a majority vote over all the group rankings of different groups $\{R_{g,A}^{t+1}, R_{g,B}^{t+1}, R_{g,C}^{t+1}\}$ to find the global ranking R_g^{t+1}. We depict this in Figure 6.2-(3b).

E2FL provides both equity and equality. Equity is not the main goal of existing distributed learning systems because it can hurt the motivation of the majority of clients to participate in the FL. If we have a learning algorithm that provides equity, it has this constraint to allow the same contribution from all groups (i.e., majorities and minorities). This comes with the price of reducing the performance of the majorities, which can demotivate them to participate in learning a model.

E2FL provides both equity and equality. In this algorithm, at the final round of the learning, each group uses its own global rankings instead of using the global ranking. The global rankings can provide better performances to the majority groups as they have access to more training data. For example, a client of handwriting style A will use $f(x, \theta^w \bigodot M^t_{g,A})$ in their downstream classification task, where $M^t_{g,A}$ is the learned binary mask for group A at FL round t, and θ^w is the random weights (initialized randomly and kept fixed), and x is the test input. Note that in E2FL and its variants, $M^t_{g,q}$ is the supermask trained for group q where for top k% of the top rankings of group ranking $R^t_{g,q}$, we put 1's, and we set other masks to 0's.

6.3 E2FL when group IDs are unknown

In the previous section, we assumed that the clients know their group IDs. In this section, we explain the approaches the server or a client can utilize to estimate the group IDs when the groups are unknown. In this setting, there are federated clients that have a small amount of data with no known protected attribute. For instance, people with their own style of handwriting want to learn a global model by learning a local model on the images of their handwriting. The clients have no knowledge about their style as it is not something to be identified. It is not even possible to announce that clients with similar styles collaborate with each other. This type of scenario usually happens in cross-device settings, where each user has a small dataset, and there are many clients in the system. In this section, we propose group inference approaches both for server and client side. For the server-side approach, the server clusters the local rankings into different groups and assigns a group ID to each client. For client-side approaches, each client should estimate its own group ID using the binary masks learned so far (Algorithm 6 line 8).

Communication cost of E2FL when group IDs are unknown: Note that for finding the best group ID at the client-side, there is no need to send all the group

rankings (e.g., $\{R_{g,A}^{t+1}, R_{g,B}^{t+1}, R_{g,C}^{t+1}\}$ in our example) to the clients. As we mentioned before, in E2FL, each ranking (local or global) can be converted to a binary mask of '0's and '1's that is superimposed on the random weights (i.e., supermask). Thus, the server only broadcasts the binary masks of groups (e.g., $\{M_{g,A}^{t+1}, M_{g,B}^{t+1}, M_{g,C}^{t+1}\}$ in our example) to the clients so they can estimate their group ID where they belong to.

6.3.1 Server-side: Rank clustering

Working with rankings enables us to design an efficient algorithm to cluster the local rankings. Clustering rankings is more efficient than clustering the model weight updates in standard FL. The main reason is that rankings are from a *discrete* space ($\in perm([1, d])$, all the possible permutation of integers $\in [1, d]$ where d is the layer size) while model updates in standard FL are from a *continuous* space ($\in \mathbb{R}^d$). In this approach, all the clients should learn a local rankings on their local data at the beginning of the learning, and send it to the server. Then the server clusters the rankings into Q clusters to find the group ID of each client. This is just one-time clustering, and throughout the E2FL learning, the server selects the local rankings for different group rank aggregation (majority voting) based on their group IDs (estimated with this approach).

Algorithm 7 Identity Inference with Rank Clustering

1: **Input:** number of clients N, Local rankings $R_{\{i\in[N]\}}$, number of clusters Q, number of iterations T
2: **function** RankClustering ($R_{\{i\in[N]\}}$)
3: Centroids ← pick Q random rankings from $R_{\{i\in[N]\}}$
4: $r \leftarrow 0$ ▷ iteration counter
5: **while** $r < T$ **do**
6: ClustersRanking ← [[] for $q \in [Q]$)]
7: **for** $u \in [N]$ **do**
8: $q \leftarrow$ getClosestCluster(Centroids, R_i)
9: ClustersRanking[q].append(u)
10: **end for**
11: **for** $q \in [Q]$ **do**
12: Cnetroids[q] ← Vote (ClustersRanking[q])
13: **end for**
14: $r+ = 1$
15: **end while**
16: **return** ClustersRanking
17: **end function**

Algorithm 7 shows how the server can cluster the local rankings of N clients into Q groups. We adapt K-means clustering to cluster the rankings. In this algorithm, at the first step (Algorithm 7 line 3), the E2FL server chooses Q random rankings as the initial Q clusters, called centroids. Then it assigns the cluster ID of the closest centroid for all the N local rankings (Algorithm 7 line 7-10). To determine the distance of two rankings, we use Spearman rank distance in which the distance between two rankings of R_1 and R_2 is $D(R_1, R_2) = \sum_{\ell\in[L]} \sum_{i\in[n_\ell]} |R_1[i] - R_2[i]|$ where $R_1[i]$ shows the rank of parameter ith in ranking R_1, L is the number of layers in the network, and n_ℓ shows the number of parameters in layer ℓ. In the next step (Algorithm 7 line 11-13), the server updates the centroid of each cluster by applying the majority vote on the rankings inside each cluster. It repeats this process for T iterations to find the final Q rankings groups.

6.3.2 Client-side: Lowest loss

In this approach, each client estimates its group ID by choosing the group that its binary mask that produces the lowest loss. Thus, in each E2FL round, the server broadcasts all the binary masks related to existing groups ($M^t_{g,q\in[Q]}$), and each selected

client calculates the loss for each binary mask on its training data. Algorithm 8 line 2-4 shows this approach. This approach was used by IFCA [70], where in their algorithm, in each training round, the server broadcasts all the model parameters to clients, and then they can find the lowest loss group. Note that our E2FL, compared to IFCA, needs $\times 32(\times 64)$ less download bandwidth in each round because it is working on the binary masks.

Algorithm 8 Identity Inference at Client Side

Input: training data D_u^{tr}, random weights θ^w, number of groups Q, group binary masks $M^t_{g,q\in[Q]}$, loss function $loss(.)$
function LOWESTLOSS $(\theta^w, Q, M^t_{g,q\in[Q]}, D_u^{tr})$
    return $\text{argmin}_{q\in[Q]}\, loss\left(D_u^{tr}, \theta^w \bigodot M^t_{g,q\in[Q]}\right)$
end function
function ONESHOT $(\theta^w, Q, M^t_{g,q\in[Q]}, D_u^{tr})$
    $\alpha \leftarrow [\frac{1}{Q}, \frac{1}{Q}, ..., \frac{1}{Q}]$
    $p(\alpha) \leftarrow f\left(D_u^{tr}, \theta^w \bigodot (\sum_{q=1}^{Q} \alpha_q M^t_{g,q})\right)$
    return $\text{argmax}_{q\in[Q]}\left(-\frac{\partial H(p(\alpha)))}{\partial \alpha_q}\right)$
end function
function BINARY $(\theta^w, Q, M^t_{g,q\in[Q]}, D_u^{tr})$
    $\alpha \leftarrow [\frac{1}{Q}, \frac{1}{Q}, ..., \frac{1}{Q}]$
    while $||\alpha||_0 > 1$ **do**
        $p \leftarrow f\left(D_u^{tr}, \theta^w \bigodot (\sum_{q=1}^{Q} \alpha_q M^t_{g,q})\right)$
        $g \leftarrow \nabla_\alpha H(p)$
        for $q \in [Q]$ **do**
            if $g_q \leq \text{median}(g)$ **then**
                $\alpha_q \leftarrow 0$
            end if
        end for
        $\alpha \leftarrow \alpha / ||\alpha||_1$
    end while
    return $\text{argmax}_{q\in[Q]}\,(\alpha_q)$
end function

6.3.3 Client-side: Entropy of the output

Binary masks enable us to utilize other approaches for group inference. In these approaches [164], the client can infer the group ID using gradient-based optimization to find a linear superposition of learned binary masks which minimizes the output entropy. [164] proposed these solutions to learn multiple tasks without catastrophic forgetting in continual learning. In these solutions, each client infers the group ID by

choosing the most confident binary mask that produces more stable results. There are two variants of this approach as follows:

OneShot inference: Algorithm 8 line 5-9 shows this approach. At E2FL training round t, the server broadcasts Q leaned binary masks $M^t_{g,q\in[Q]}$ to the selected clients. Next, each client assigns a confidence coefficient (α_q) to each binary mask. Each α_q represents how much the client is confident that qth binary mask is its match. Then it calculates the output of the model as the weighted superposition of these masks (i.e., $p(\alpha) = f\left(x, \theta^w \bigodot(\sum_{q=1}^{Q} \alpha_q M^t_{g,q})\right)$). The α is initialized in a way that all the masks have equal chance ($\alpha_0[q] = \frac{1}{Q}$ for $q \in [Q]$). Now, Each client tries to find the perfect α_q that minimizes the entropy of the outputs $H(p(\alpha))$ by applying gradient descent with respect to α just for one round, i.e., $\alpha \leftarrow \alpha - \beta \nabla_\alpha H(p(\alpha))$. In this approach, the client chooses the group ID q that changing its confidence level (α_q) has the most impact on the entropy of the output of mixed models, i.e., $\arg\max_q \left(-\frac{\partial H(p(\alpha)))}{\partial \alpha_q}\right)$.

Binary Search: Algorithm 8 line 10-23 shows this approach. In this approach, the client utilizes binary search to find the best group ID by removing half of the candidates at each step until one α_q remains nonzero, which indicates the best candidate. The client calculates the $p(\alpha)$ for α, then it applies the gradient descent with respect to α on entropy of $p(\alpha)$. Next, the client eliminates half of the coefficients where they produce gradients less than the median of the gradients.

Time complexity comparison: The lowest loss approach needs to have $\mathcal{O}(Q)$ forward passes to find the binary mask that produces the lowest loss for Q groups. Binary search over the entropy needs to have $\mathcal{O}(\log(Q))$ forward and backward passes. Finally, the OneShot inference only requires $\mathcal{O}(1)$ forward and backward passes, making the estimation process very fast.

6.4 Experiments

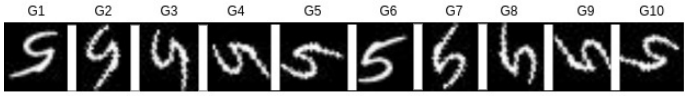
In this section, we examine the performance, fairness, and communication cost of the proposed E2FL algorithm in two different settings with known and unknown group IDs. To assess equity, we first calculate the average final accuracies of the clients in each group; then, we report the mean and variance of accuracies of groups' accuracies (see Section 6.1). For equality, we measure these metrics on the final accuracies of all the FL clients. We use three benchmark datasets widely used in prior works on federated learning applications. We run all the experiments for five runs with different seeds and report their average. We also report results of a new FL algorithm, IFCA_Avg, where similar to IFCA [70] and HypCluster [125], trains multiple separated group models and combines the models in each round by averaging the weights (as opposed to rankings). Therefore, in each FL round, the IFCA_Avg clients need to download the Q models (consisting of 32/64 bits parameters) and the average of them.

It is worth noting that the use of Personalized FL (PFL) algorithms to reduce the variance of clients' performances by personalizing the global model for each client is orthogonal to our approach in E2FL. The various personalization techniques employed in PFL, such as fine-tuning [170, 125, 161, 175], mixing global and local models [153, 113, 77, 52], re-weighting [174, 112, 131], meta-data learning [39, 89, 64, 62], and representation learning [118, 47], could lead to an increase in performance and further reduction of the variance of the clients' performances. However, exploring the utility and performance of PFL in combination with our approach is outside the scope of this work and is deferred to future research.

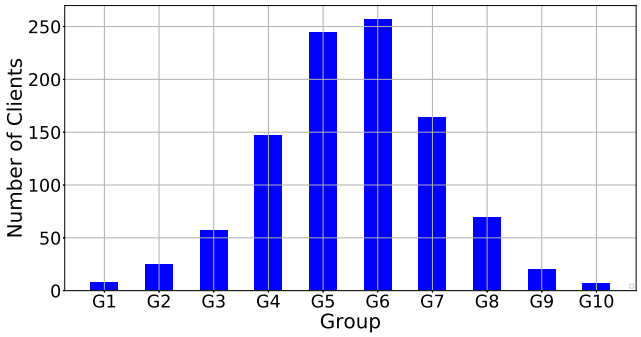
6.4.1 Equality vs Equity via E2FL

FairMNISTRotate: In this work, we present a novel dataset for evaluating fairness in FL applications. Our goal is to measure both equity and equality by conducting

experiments using this new dataset. To create this dataset, we utilize a widely adopted method in the continual research community [74, 100, 122] by manipulating the MNIST dataset. Specifically, we rotate the images in MNIST to create ten different data distributions, each with a different number of clients.



(a) Data sample from each group



(b) Number of clients in each group

Figure 6.3. FairMNISTRotate: a new dataset to investigate equality and equity in FL application.

The dataset is comprised of 1000 clients in total, with each client receiving 200 training and 50 test samples based on their group number. The distribution of clients across groups is diverse, with some groups having a large number of clients (e.g., G6 with 257 clients) and others having a small number of clients (e.g., G1 with 8 clients). Figures 6.3(a) and 6.3(b) show sample images and the number of clients in each group, respectively. In this dataset, G1 and G10 are considered minority groups, while G5 and G6 are considered majority groups. We evaluate the performance of our proposed E2FL algorithm, along with other FL algorithms, in terms of utility, equity, equality, and communication cost on the FairMNISTRotate dataset and present the results in Table 6.1.

Table 6.1. Comparison of equality, equity, and communication cost among various variants of FLs on FairMNISTRotate with 1000 clients. The algorithms are ranked based on the respective metric in each column.

Approach	Metric					
	Group-level Fairness (Equity)		User-level Fairness (Equality)		Communication Cost	
	Average	Variance	Average	Variance	Up (MB)	Down (MB)
Local training	84.78	0.11	85.03	3.44	0	0
FedAvg	94.20 (5)	17.39 (4)	97.85 (4)	4.46 (4)	6.20 (2)	6.20 (3)
FRL	93.41 (6)	41.73 (6)	97.27 (6)	5.61 (6)	4.05 (1)	4.05 (1)
IFCA	97.74 (1)	2.40 (2)	98.75 (1)	0.41 (2)	6.20 (2)	62.01 (4)
IFCA_Avg	95.39 (4)	14.06 (3)	97.62 (5)	5.02 (5)	6.20 (2)	68.21 (5)
q-FFL	95.41 (3)	39.12 (5)	98.52 (2)	2.76 (3)	6.20 (2)	6.20 (3)
Our E2FL	96.81 (2)	1.41 (1)	98.03 (3)	0.28 (1)	4.05 (1)	5.99 (2)

Our experimental results on the FairMNISTRotate dataset demonstrate the following findings: (1) **All groups, including minority and majority groups, benefit from participating in FL**. Without access to other clients' knowledge, local models perform poorly. (2) **FedAvg [103, 126] and FRL (Chapter 3) prioritizes majority groups**, with clients from these groups having a higher chance of being selected in each round and thus having a larger impact on the global model. Although FedAvg achieves 97.85% mean test accuracy for individual clients, the mean accuracy for groups is only 94.20%, indicating a focus on user-level fairness rather than group-based fairness (equity). (3) **q-FFL [116] improves equality, but worsens equity**. This user-level fairness framework improves fairness for the majority groups, but at the expense of the minority groups. q-FFL reduces the accuracy variance for all clients by 38% but increases the variance between groups by 125% compared to FedAvg, at a communication cost of 12.4 MB per client. (4) **Training multiple FLs (i.e., IFCA [70] and HypCluster [125]) is not optimal for minorities** (e.g., G1, G2, and G10), as these groups are unable to benefit from shared knowledge. Additionally, communication cost is very high for IFCA compared to E2FL. Specifically, IFCA reduces the variance for groups and clients by 86% and 91% compared to FedAvg, respectively, but at a higher communication cost of 68.21 MB per client. IFCA is also unable to benefit from our entropy-based approaches, which are specifically designed

for binary masks. (5) **IFCA_Avg is inferior to IFCA as it reduces the utility and fairness of IFCA**. When the server aggregates the group model updates (i.e., averaging weight updates, each consisting of 32/64 bits parameters) into the global model, each group update has a different impact on the global model (because of their different l_2 norms), reducing the overall utility and fairness of the algorithm.. (6) **Our proposed algorithm, E2FL, provides both equality and equity**. Compared to FedAvg (FRL), E2FL decreases the variance of group and client accuracies by 92% (97%) and 94% (95.0%) with a communication cost of 10.04 MB per client.

6.4.2 E2FL when group IDs are unknown

In this section, we provide experimental results on the FEMNIST [33], which is a character recognition classification task distributed non-iid over 3,400 clients. At first, data distribution among all the clients seems similar as all of them classify handwritten letters or digits. However, there might be hidden groups of clients among these 3400 clients with even more similar handwriting styles. To address this, we use three group inference approaches, which are introduced in section 6.3. These include lowest loss, oneshot, and rank clustering. The experiments use two clusters ($Q = 2$) for the group inference and for IFCA and IFCA_Avg. We also report more details of 2, 3, 4, and 5 clusters for rank clustering E2FL on FEMNIST in Table 6.6. The results of the binary search in the entropy of outputs are not presented as they are similar to the oneshot inference approach.

In Table 6.2, we present a comparison of the performance and fairness of various FL algorithms on the FEMNIST dataset. The table only showcases the equality metrics at the user level, as the dataset does not define any specific groups. Our experimental results indicate the following: (1) The local training as opposed to FedAvg (or FRL) does not offer any benefits, motivating all clients to participate in FL for improved accuracy. (2) The q-FFL algorithm [116] provides a more equitable outcome, reducing

Table 6.2. Comparison of utility, equality, and communication cost among different FL algorithms on FEMNIST using 3400 clients. We also show the average accuracies for the worst and best 10% of the clients.

Approach	Metric					
	User-level Fairness (Equality)				Communication Cost	
	Average	Worst (10%)	Best (10%)	Variance	Up (MB)	Down (MB)
Local training	68.74	44.45	87.41	154.50	0	0
FedAvg	85.80	63.51	99.39	104.04	6.23	6.23
FRL	84.20	62.65	98.52	114.49	4.06	4.06
IFCA	87.38	67.31	**100**	88.51	6.23	12.45
IFCA_Avg	84.71	62.73	98.88	108.33	6.23	18.68
q-FFL	84.40	64.09	99.14	100.80	6.23	6.23
Our E2FL (OneShot)	83.52	60.37	98.31	121.44	4.06	4.44
Our E2FL (Rank Clustering)	87.21	66.36	99.87	94.09	4.06	**4.06**
Our E2FL (Lowest Loss)	**87.93**	**68.59**	99.96	**81.88**	**4.06**	4.44

variance by 4% at the cost of a 1.40 reduction in accuracy compared to FedAvg. This reduction in accuracy is due to the emphasis on uniform performance among clients in majority groups, leading to the minority groups receiving less attention. (3) IFCA improved the average and variance of the client's accuracies, but it requires each client to download Q models, each consisting of 32/64 bits weight parameters. (4) Our proposed method, E2FL, demonstrates a clear advantage over the other algorithms. E2FL leverages the benefits of both accuracy and fairness, for instance, the EFFL method with the lowest loss group inference approach improves the average accuracy by 2.13 (3.73) and reduces the variance among clients by 21% (28%) in comparison to FedAvg (FRL). These improvements stem from the training of multiple models, one for each group, and the combination of all group models into a global model that utilizes knowledge from all participating clients by performing a fair majority vote among them. (5) The lowest loss and rank clustering group inference methods perform better than the oneshot inference method, as the latter only requires a single forward and backward pass, while the former require additional operations.

6.4.3 Fair FL when each client has training data of multiple groups

In our experiments, we also evaluate the performance of the E2FL on a real-world dataset, the Adult Census Income Dataset [102]. This dataset contains 48,842 samples collected from the United States Census Database, with the task of classifying individuals' income into two categories: earning less than or equal to 50K per year (mapped to 0) or earning more than 50K per year (mapped to 1) where the output of $\hat{Y} = 1$ is regarded as a positive output (i.e., making more money). In our analysis, gender is considered as the protected attribute, with male samples (represented by $A = 1$) being the privileged group and female samples (represented by $A = 0$) being the unprivileged group. We split the data into train and test, and Table 6.3 shows the bias in this dataset for the difference in their opportunities for making higher income ($Pr[\hat{Y} = 1|A = a]$ where $a \in \{0, 1\}$). The bias towards the male group is evident, with the male group having a 31.4% chance of getting $\hat{Y} = 1$ and the female group having only an 11.3% chance in the training data. During testing, models trained on this dataset further amplify the bias towards the male samples by having a higher chance of making a positive prediction ($\hat{Y} = 1$) for male data inputs.

Table 6.3. Distribution of training and test samples for male and female groups on the Adult dataset.

Protected Attr	Stats	train data	test data
Gender	$\Pr[A = 1]$	67.50%	67.50%
	$\Pr[A = 0]$	32.5%	32.5%
	$\Pr[\hat{Y} = 1\|A = 1]$	**31.4%**	**30.8%**
	$\Pr[\hat{Y} = 1\|A = 0]$	**11.3%**	**11.3%**

In our experiments, we distribute the samples of the Adult Census Income Dataset among five clients, using Dirichlet distribution. The data distribution is done with two settings: (a) Independent and Identically Distributed (IID) with a Dirichlet parameter of $\alpha = 5000$, and (b) Non-Independent and Identically Distributed (Non-IID) with a Dirichlet parameter of $\alpha = 1$.

To evaluate the fairness of the trained global model, we adopt two metrics that have been previously used in related works such as [61, 7, 172]. The first metric is the Equal Opportunity Difference (EOD), i.e., $EOD = Pr(\hat{Y} = 1|A = 0, Y = 1) - Pr(\hat{Y} = 1|A = 1, Y = 1)$, which measures the difference in the True Positive Rate between the privileged and unprivileged groups. The second metric is the Discrimination Index (DI), i.e., $DI = F1(\theta|A = 0) - F1(\theta|A = 1)$, which measures the difference in the F1 score between the two groups.

Table 6.4. Comparison of fairness between E2FL and other baselines on the Adult dataset.

Algorithm	Metric	Heterogeneity Level α	
		5000 (IID)	1 (Non-IID)
FedAvg	Test Accuracy	85.56	85.47
	EOD_{te}	-0.0689	-0.0834
	DI_{te}	-0.0432	-0.0517
FairFed	Test Accuracy	85.1	84.47
	EOD_{te}	-0.0701	-0.069
	DI_{te}	-0.0441	-0.041
E2FL	Test Accuracy	85.22	85.20
	EOD_{te}	-0.0174	-0.0222
	DI_{te}	-0.019	-0.0252

Table 6.4 presents the fairness comparison between FedAvg [103, 126], FairFed [61], and the proposed method, E2FL, for different data distributions on Adult dataset. FairFed is selected as the baseline due to its ability to provide group fairness when different data groups (protected attributes) are present at each client, which is similar to the situation in the Adult Census Income dataset. This algorithm adaptively modifies the aggregation weights at the server in each round. The weights are based on the mismatch between each client's global fairness measure (at the server) and the local fairness measure. This algorithm favors clients whose local measures match the global fairness measure more. The results in the table demonstrate that E2FL reduces the equal opportunity difference (EOD) and discrimination index (DI) between the male and female groups with a minor reduction in the final test accuracy. Specifically,

for non-IID data distribution, E2FL improves the EOD by 73% and the DI by 51% with only a small loss in test accuracy (0.27%) compared to FedAvg. On the other hand, FairFed improves the EOD by 17% and the DI by 20% with a 1% decrease in test accuracy compared to FedAvg. In the case of IID data distribution, FairFed does not provide the same level of improvement as its design is optimized for heterogeneous data distribution. In contrast, E2FL achieves similar improvement in both cases. These results highlight that E2FL enforces a fairer model behavior even when each client has a mixture of training samples belonging to different groups.

6.4.4 Our group inference approaches

In Section 6.3, we proposed four approaches for the estimation of the group to which an FL client belongs based on its data distribution. This section presents a discussion of the performance and utility of these approaches.

Table 6.5. Accuracy of group inference in rank clustering approach on FairMNISTRotate with 1000 clients based on local rankings learned after two local epochs. Results of the accuracy of prediction are presented for rankings of individual layers with varying numbers of parameters.

Layer	No Params	Accuracy of predicting the right group (%)
Conv1	288	91.78
Conv2	18432	94.94
FC1	1605632	93.76
FC2	1280	91.99
ALL	1625632	93.21

Table 6.6. Comparison of utility, equality, and equity of E2FL with rank clustering on FEMNIST using 3400 clients.

Approach	Number of Clusters	Metric							
		Group-level Fairness (Equity)				User-level Fairness (Equality)			
		Average	Worst group	Best group	Variance	Average	Worst (10%)	Best (10%)	Variance
E2FL	2	88.12	85.65	90.60	6.10	87.21	66.36	99.87	94.09
	3	88.03	84.74	91.33	10.82	86.36	64.82	99.66	101.80
	4	88.01	85.01	91.39	5.42	87.12	65.81	100	98.80
	5	87.61	85.31	91.06	3.80	87.40	67.30	100	88.54

Rank clustering approach: Table 6.5 presents the accuracy of the rank clustering approach in E2FL on FairMNISTRotate, in which the local rankings of different network layers are clustered either separately or all together. In this approach, each E2FL client learns local rankings for $E = 2$ local epochs on its local data (each layer has its own ranking) at the beginning of E2FL and sends the local ranking to the server so that the server can assign different group IDs to the clients. From this table, we can see that with larger rankings, we can predict group ID with higher accuracy. Table 6.6 also presents the equity and equality measurements of E2FL on FEMNIST for different numbers of clusters. By increasing the number of clusters, more diverse groups can be covered, leading to fairer results.

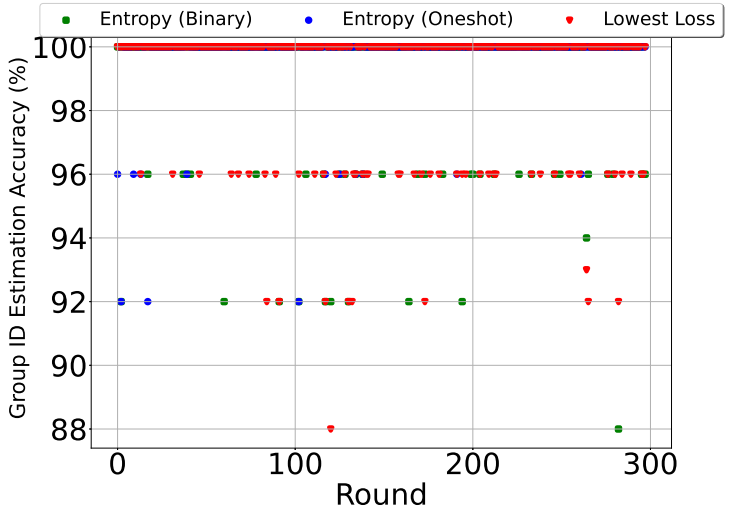


Figure 6.4. Accuracy of client-side group inference approaches in E2FL on FairMNISTRotate during the first 300 global epochs.

Client-based group inference approaches: In this study, we evaluate the performance of three client-side group inference approaches, namely lowest loss, binary search of entropy of outputs, and oneshot method. Figure 6.4 presents the accuracy results of these approaches over the first 300 global epochs of E2FL on FairMNISTRotate. The results indicate that the lowest loss approach outperforms the binary and oneshot methods. Additionally, the binary search approach exhibits better accuracy compared

to the oneshot method as it requires more forward and backward passes to accurately determine the group ID.

6.5 Conclusions

This Chapter addresses the fairness issue in Federated Learning (FL) by introducing two fairness metrics, equality and equity, to assess user-level and group-level fairness, respectively. To achieve both equality and equity in FL, a novel algorithm, named E2FL (Equal and Equitable Federated Learning) was proposed and implemented. Through empirical evaluations on various real-world FL scenarios, the results demonstrated that E2FL outperforms existing methods in terms of efficiency, fairness among different groups, and fairness for individual clients.

CHAPTER 7

CONCLUSION

7.1 Summary

In this dissertation, I focus on addressing the trustworthiness challenges arising in distributed learning systems, particularly in Federated Learning (FL). I carry out extensive measurement studies to shed light on security, privacy, and fairness issues in FL. Moreover, I develop innovative mechanisms to enhance the trustworthiness of existing FL algorithms.

To improve robustness, I introduce a comprehensive framework encompassing various poisoning attacks and defensive aggregation strategies found in FL literature. This framework aims to bridge the gap between existing works and provide a clear understanding of the different types of threats that must be considered when designing FL systems. Additionally, I devise the Federated Rank Learning (FRL) approach, in which clients rank edges within a randomly initialized network. This method was demonstrated to restrict the adversary's options for poisoning the global model.

Regarding access privacy, I create the Heterogeneous Private Information Retrieval (HPIR) mechanism, allowing clients to fetch their specific model parameters from untrusted servers without revealing any information. I also examine the privacy leakage of local rankings in FRL by performing membership inference attacks on them. I demonstrate that FRL offers superior privacy against local and global membership inference attacks compared to standard FL training.

To tackle fairness in FL, I propose learning multiple global models by training on parameter ranks (similar to FRL), enabling each group of clients to benefit from

their personalized model. I establish a fair FL approach based on learning parameter ranks to ensure that the global model performs consistently across various clients and different groups of clients.

In summary, I have contributed to each aspect of trustworthiness in distributed learning systems and provided a clear understanding of the current state of the art in FL security.

7.2 Future work

The research presented in this thesis can be extended in various directions. In this section, I present the potential future works to explore.

7.2.1 FRL with different rank aggregation methods

One potential extension is to investigate the effect of different rank aggregation methods in FRL (Chapter 3). Specifically, the FRL server uses the Borda count rank aggregation method [60], where it assigns a reputation to each edge for each ranking, sums the reputations, and sorts them from least to most to find the global ranking.

The *average Kendall tau distance* of a ranking R_t^g to a ranking dataset $T = \{\theta_t^1, \theta_t^2, ..., \theta_t^n\}$ is defined as $\bar{\mathbf{K}}(R_t^g, T) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{K}(\theta_t^i, R_t^g)$. The Kemeny optimal aggregation [94, 48] is a commonly used criterion for determining the best aggregate ranking, which minimizes the average Kendall tau distance to T. Although computing the Kemeny optimal ranking is NP-hard for rank databases with size $n > 3$, PTIME approximation algorithms are available [58, 79]. Furthermore, numerous heuristic methods have been proposed, including scoring and ranking elements [60], conducting locally directed searches in the rank space [95, 58], and defining a Markov chain from the rank database to rank elements based on their stationary probabilities [58, 108].

By replacing FRL's rank aggregation at the FRL server with alternative methods such as the KwiKSort [10], Footrule and scaled Footrule [55],and Markov chain

methods [58, 108], it is possible to examine the impact on FL accuracy in highly heterogeneous data distributions where all rankings significantly differ from one another. Additionally, the effect of excluding adversarial local updates can be evaluated. The privacy leakage and fairness of different rank aggregation methods at the server can also be assessed, allowing for a more comprehensive understanding of how various aggregation techniques influence the overall trustworthiness of FL systems.

7.2.2 Extending FRL with existing ideas in FL algorithms

Our main argument in Chapter 3 is that FL algorithms based on FedAvg are vulnerable to the same kind of attacks that even a single malicious client can corrupt the model by sending well-crafted updates (e.g., very large updates for FedAvg). This vulnerability comes from sending and receiving weight parameters, as the adversary has more space to find the most damaging updates. Recent studies on FL, such as SCAFFOLD [92], FedProx [115], FedNova [160], and Momentum-based FedAVG [49, 145], have introduced new techniques to improve global model performance in heterogeneous data distributions without malicious clients. These systems' clients also send updates as trained weights (similar to FedAvg), making them vulnerable to the same attacks as FedAvg. Conversely, FRL employs ranking, with free-scale ranks, resulting in greater robustness compared to these systems. However, to enhance performance in the absence of malicious clients, we can incorporate their ideas into FRL as follows:

SCAFFOLD [92] estimates the update direction for the global model and each client's update direction. It then uses the difference as a client drift estimate to correct the local update. This concept can also be applied to FRL for correcting client drifts. In the modified FRL, each client estimates the global reputation of edges (using global ranking) and the local reputation of edges (using local ranking). The client then uses the difference between these reputations to correct the local ranking.

FedProx [115] learns a local model for a chosen client by regularizing the distance between the local and global models. This technique can also be integrated into FRL. In the modified FRL, each client calculates the local rankings of the supernetwork, ensuring that the local ranking (R_t^k) remains close to the global ranking (R_t^g). To measure the difference between the two rankings, we can compute the Kendall tau distance of the two rankings, which measures the number of disagreements between rankings. In this case, we optimize the following objective functions: for FedProx optimization, $w_k^{t+1} = \arg\min_w F_k(w) + \frac{\mu}{2}||w - w^t||$, and for modified FRL optimization, $R_k^{t+1} = \arg\min_R F_k(R) + \frac{\mu}{2} \times K(R, R_t^g)$. Here, $F_k(.)$ is the objective function of the k^{th} client, and μ is a tunable hyperparameter in FedProx.

FedNova [160] averages gradients based on the number of local epochs each client has, rather than using a fixed number of local epochs per client. This idea can also be integrated into FRL. In our FRL experiments, we used 2 local epochs for MNIST and FEMNIST clients and 5 local epochs for CIFAR10 users. However, by using different numbers of local epochs per client, we can employ the same technique by averaging the reputation of edges based on the number of local epochs. In FedAvg, the server announces the global model as $w^{t+1} = \frac{1}{n}\sum_{i=1}^{n} W_i^t$, and in FedNova, the server announces the global model as $w^{t+1} = \frac{1}{n}\sum_{i=1}^{n} \frac{W_i^t}{\tau_i}$, where τ_i is the number of local updates for the i^{th} client. In the modified FRL, the server calculates the global rankings using Algorithm 9, with T being a vector containing the number of local epochs applied by each user.

Algorithm 9 Modified FRL vote to act similar to idea of FedNova [160].

```
function VOTE(R_{u∈U}, T_{u∈U} )
    V ← ARGSORT(R_{u∈U})          ▷ Reputation of each edge in each local ranking
    A ← SUM(V/T)                  ▷ Sum the reputations devices by number of local epochs
    return ARGSORT(A)             ▷ Order of the reputations
end function
```

Momentum-based FLs [49, 145] : Das et al. [49] use a global momentum for updating the global model, and Reddi et al. [145] extend this approach by incorporating

AdaGrad and Adam. All of these methods can be integrated into FRL as well since the server calculates the reputation of each edge and sums them. In momentum-based FRL, the server incorporates previous rankings (reputations) for each edge to update the new rankings as momentum. In this case, Algorithm 10 illustrates the Vote function with momentum, where μ is the momentum, and R_t^g is the previous global ranking.

Algorithm 10 Modified FRL vote to act similar to idea of Momentum-based FLs [49, 145] [160].

```
1: function VOTE(R_{u∈U}, R_t^g )
2:     V_1 ← ARGSORT(R_{u∈U})          ▷ Reputation of each edge in each local ranking
3:     V_2 ← ARGSORT(R_t^g)            ▷ Reputation of each edge in the global ranking
4:     A ← SUM(V_1 + μV_2)
5:     return ARGSORT(A)               ▷ Order of the reputations
6: end function
```

7.2.3 FL personalization with rankings

Personalized FL (PFL) algorithms reduce the variance of clients' performances by personalizing the global model for each client. Incorporating the following ideas in FRL could lead to an increase in performance and further reduction of the variance of the clients' performances:

Personalization through fine-tuning. A natural strategy for personalizing the global model is that each client can fine-tune the global model parameters on its local data [125, 161, 175]. Zhao et al. [175] also show that accuracy of FL will be reduced significantly when the data is non-i.i.d. (up to 55% reduction in test accuracy when each client has a different class of data). For solving this problem, the authors propose to use fine-tuning on a shared global dataset that has all the classes and then send a warm-up model to the clients.

Personalization through mixing global and local models. Hanzely et al. [77] propose a new FL algorithm that jointly learns local representations on each client and a global model across all of them. In the new optimization problem, they are trying

to find local updates for each client to minimize the average of local objective function plus a regularizer consisting of the distance from the global model. They mention that using the new FL algorithm can help in solving performance, communication, and fairness issues of current FL. They also provide convergence guarantees for their algorithm by assuming convex loss functions. Deng et al. [52] propose an adaptive personalized federated learning algorithm where each FL participant will train a local model while contributing to the global model. They aim to balance the tradeoff between getting benefit by using other clients' knowledge and the disadvantage of statistical heterogeneity of data, which may cause divergence on the gradients.

Personalization through meta-data. In meta-learning, the goal is to use data from previous tasks to learn updates or model parameters that can be fine-tuned to perform well on new tasks with a small amount of data. Chen et al. [39] and Jiang et al. [89] use the idea of Model-Agnostic Meta-Learning (MAML) [64] to achieve personalization in FL. The main idea of MAML is to find a meta initialization that performs well on new unseen tasks once it is updated using gradient-based updates of the loss related to the new task. The authors suggest to find an initial model for each client that can be fine-tuned using a few steps of gradient descent. Fallah et al. [62] introduce Per-Fed, a personalized FL using meta-learning that provides the first theoretical guarantees for MAML methods for nonconvex functions.

Personalization through representation learning. Liang et al. [118] propose to mix different network layers for personalization. Collins et al. [47] suggest learning lower layer parameters as a shared global representation, and then each client can learn a customized classifier on top of that. They suggest that each client, after receiving the global representation for base layers, fine-tunes the last layer on its local data, and then performs one round of SGD on the base layer parameters, which are then uploaded as the user's gradients.

BIBLIOGRAPHY

[1] Glove: Pretrained word embeddings by standford nlp group. `http://nlp.stanford.edu/data/glove.840B.300d.zip`.

[2] Sparsemodule: finding score-based subnetworks in any neural network architectures. `https://github.com/dchiji/sparse_module`.

[3] Tiny imagenet challenge [online]. `https://tinyimagenet.herokuapp.com`.

[4] Denoising Diffusion Probabilistic Model, in Tensorflow. `https://github.com/hojonathanho/diffusion`, 2020.

[5] Denoising Diffusion Probabilistic Model, in Pytorch. `https://github.com/lucidrains/denoising-diffusion-pytorch`, 2022.

[6] Abadi, Martín, Chu, Andy, Goodfellow, Ian, McMahan, H Brendan, Mironov, Ilya, Talwar, Kunal, and Zhang, Li. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM.

[7] Abay, Annie, Zhou, Yi, Baracaldo, Nathalie, Rajamoni, Shashank, Chuba, Ebube, and Ludwig, Heiko. Mitigating bias in federated learning. *arXiv preprint arXiv:2012.02447* (2020).

[8] Aguilar-Melchor, Carlos, Barrier, Joris, Fousse, Laurent, and Killijian, Marc-Olivier. Xpir: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies 2016*, 2 (2016), 155–174.

[9] Aguilar-Melchor, Carlos, and Gaborit, Philippe. A lattice-based computationally-efficient private information retrieval protocol. In *Western European Workshop on Research in Cryptology* (2007), Citeseer.

[10] Ailon, Nir, Charikar, Moses, and Newman, Alantha. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM) 55*, 5 (2008), 1–27.

[11] Aji, Alham Fikri, and Heafield, Kenneth. Sparse communication for distributed gradient descent. In *EMNLP* (2017).

[12] Alabi, Daniel, Ghazi, Badih, Kumar, Ravi, and Manurangsi, Pasin. Private rank aggregation in central and local models. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2022), vol. 36, pp. 5984–5991.

[13] Alistarh, Dan, Hoefler, Torsten, Johansson, Mikael, Konstantinov, Nikola, Khirirat, Sarit, and Renggli, Cédric. The convergence of sparsified gradient methods. In *NeurIPS* (2018).

[14] Angel, Sebastian, Chen, Hao, Laine, Kim, and Setty, Srinath. Pir with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 962–979.

[15] Bagdasaryan, Eugene, Veit, Andreas, Hua, Yiqing, Estrin, Deborah, and Shmatikov, Vitaly. How to backdoor federated learning. In *AISTATS* (2020).

[16] Baruch, Moran, Gilad, Baruch, and Goldberg, Yoav. A little is enough: Circumventing defenses for distributed learning. In *NeurIPS* (2019).

[17] Beimel, Amos, and Ishai, Yuval. Information-theoretic private information retrieval: A unified construction. In *International Colloquium on Automata, Languages, and Programming* (2001), Springer, pp. 912–926.

[18] Beimel, Amos, Ishai, Yuval, Kushilevitz, Eyal, and Orlov, Ilan. Share conversion and private information retrieval. In *2012 IEEE 27th Conference on Computational Complexity* (2012), IEEE, pp. 258–268.

[19] Beimel, Amos, Ishai, Yuval, Kushilevitz, Eyal, and Raymond, J-F. Breaking the o (n/sup 1/(2k-1)/) barrier for information-theoretic private information retrieval. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on* (2002), IEEE, pp. 261–270.

[20] Beimel, Amos, and Stahl, Yoav. Robust information-theoretic private information retrieval. In *International Conference on Security in Communication Networks* (2002), Springer, pp. 326–341.

[21] Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[22] Bernstein, Jeremy, Zhao, Jiawei, Azizzadenesheli, Kamyar, and Anandkumar, Anima. signsgd with majority vote is communication efficient and fault tolerant. In *ICLR* (2019).

[23] Bhagoji, Arjun Nitin, Chakraborty, Supriyo, Mittal, Prateek, and Calo, Seraphin. Analyzing federated learning through an adversarial lens. In *ICML* (2019).

[24] Bhat, Radhakrishna, and Sunitha, NR. A novel hybrid private information retrieval with non-trivial communication cost. In *2018 4th International Conference on Recent Advances in Information Technology (RAIT)* (2018), IEEE, pp. 1–7.

[25] Bhowmick, Abhishek, Duchi, John, Freudiger, Julien, Kapoor, Gaurav, and Rogers, Ryan. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984* (2018).

[26] Blakley, George Robert, and Meadows, Catherine. Security of ramp schemes. In *Workshop on the Theory and Application of Cryptographic Techniques* (1984), Springer, pp. 242–268.

[27] Blanchard, Peva, Guerraoui, Rachid, Stainer, Julien, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NeurIPS* (2017), pp. 119–129.

[28] Bonawitz, Keith, Eichner, Hubert, Grieskamp, Wolfgang, Huba, Dzmitry, Ingerman, Alex, Ivanov, Vladimir, Kiddon, Chloé, Konecnỳ, Jakub, Mazzocchi, Stefano, McMahan, H Brendan, et al. Towards federated learning at scale: System design. In *MLSys* (2019).

[29] Borisov, Nikita, Danezis, George, and Goldberg, Ian. Dp5: A private presence service. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 4–24.

[30] Brakerski, Zvika, and Vaikuntanathan, Vinod. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing 43*, 2 (2014), 831–871.

[31] Brendan, McMahan H, Ramage, Daniel, Talwar, Kunal, and Zhang, Li. Learning differentially private recurrent language models. *International Conference on Learning and Representation* (2018).

[32] Cachin, Christian, Micali, Silvio, and Stadler, Markus. Computationally private information retrieval with polylogarithmic communication. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1999), Springer, pp. 402–414.

[33] Caldas, Sebastian, Wu, Peter, Li, Tian, Konečný, Jakub, McMahan, H. Brendan, Smith, Virginia, and Talwalkar, Ameet. LEAF: A benchmark for federated settings. *CoRR abs/1812.01097* (2018).

[34] Cantelli, Francesco Paolo. Sui confini della probabilita. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928* (1929), pp. 47–60.

[35] Cao, Xiaoyu, and Gong, Neil Zhenqiang. Mpaf: Model poisoning attacks to federated learning based on fake clients. *arXiv preprint arXiv:2203.08669* (2022).

[36] Cappos, Justin. Avoiding theoretical optimality to efficiently and privately retrieve security updates. In *International Conference on Financial Cryptography and Data Security* (2013), Springer, pp. 386–394.

[37] Carlini, Nicholas, Chien, Steve, Nasr, Milad, Song, Shuang, Terzis, Andreas, and Tramer, Florian. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)* (2022), IEEE, pp. 1897–1914.

[38] Chang, Hongyan, Shejwalkar, Virat, Shokri, Reza, and Houmansadr, Amir. Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer, 2019.

[39] Chen, Fei, Luo, Mi, Dong, Zhenhua, Li, Zhenguo, and He, Xiuqiang. Federated meta-learning with fast convergence and efficient communication. *arXiv preprint arXiv:1802.07876* (2018).

[40] Chen, Ting, Zhang, Ruixiang, and Hinton, Geoffrey. Analog bits: Generating discrete data using diffusion models with self-conditioning, 2022.

[41] Chien, Hung-Yu, Jan, Jinn-Ke, and Tseng, Yuh-Min. A practical (t, n) multi-secret sharing scheme. *IEICE transactions on fundamentals of electronics, communications and computer sciences 83*, 12 (2000), 2762–2765.

[42] Chijiwa, Daiki, Yamaguchi, Shin'ya, Ida, Yasutoshi, Umakoshi, Kenji, and Inoue, Tomohiro. Pruning randomly initialized neural networks with iterative randomization. *Advances in Neural Information Processing Systems 34* (2021), 4503–4513.

[43] Choi, Jooyoung, Lee, Jungbeom, Shin, Chaehun, Kim, Sungwon, Kim, Hyunwoo J., and Yoon, Sung-Hoon. Perception prioritized training of diffusion models. *ArXiv abs/2204.00227* (2022).

[44] Chor, Benny, and Gilboa, Niv. Computationally private information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (1997), ACM, pp. 304–313.

[45] Chor, Benny, Goldreich, Oded, Kushilevitz, Eyal, and Sudan, Madhu. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on* (1995), IEEE, pp. 41–50.

[46] Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. EMNIST: extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks, IJCNN* (2017).

[47] Collins, Liam, Hassani, Hamed, Mokhtari, Aryan, and Shakkottai, Sanjay. Exploiting shared representations for personalized federated learning. In *International Conference on Machine Learning* (2021).

[48] Conitzer, Vincent, Davenport, Andrew, and Kalagnanam, Jayant. Improved bounds for computing kemeny rankings. In *AAAI* (2006), vol. 6, pp. 620–626.

[49] Das, Rudrajit, Acharya, Anish, Hashemi, Abolfazl, Sanghavi, Sujay, Dhillon, Inderjit S, and Topcu, Ufuk. Faster non-convex federated learning via global and local momentum. In *Uncertainty in Artificial Intelligence* (2022), PMLR, pp. 496–506.

[50] Dauphin, Yann N, and Bengio, Yoshua. Big neural networks waste capacity. *arXiv preprint arXiv:1301.3583* (2013).

[51] Demmler, Daniel, Herzberg, Amir, and Schneider, Thomas. Raid-pir: Practical multi-server pir. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security* (2014), ACM, pp. 45–56.

[52] Deng, Yuyang, Kamani, Mohammad Mahdi, and Mahdavi, Mehrdad. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461* (2020).

[53] Denil, Misha, Shakibi, Babak, Dinh, Laurent, Ranzato, Marc'Aurelio, and de Freitas, Nando. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2* (2013), pp. 2148–2156.

[54] Devet, Casey, Goldberg, Ian, and Heninger, Nadia. Optimally robust private information retrieval. In *USENIX Security Symposium* (2012), pp. 269–283.

[55] Diaconis, Persi, and Graham, Ronald L. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological) 39*, 2 (1977), 262–268.

[56] Dingyi, Pei, Arto, Salomaa, and Cunsheng, Ding. *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.

[57] Dong, Changyu, and Chen, Liqun. A fast single server private information retrieval protocol with low communication cost. In *European Symposium on Research in Computer Security* (2014), Springer, pp. 380–399.

[58] Dwork, Cynthia, Kumar, Ravi, Naor, Moni, and Sivakumar, Dandapani. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web* (2001), pp. 613–622.

[59] Dwork, Cynthia, Roth, Aaron, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* (2014).

[60] Emerson, Peter. The original borda count and partial voting. *Social Choice and Welfare 40*, 2 (2013), 353–358.

[61] Ezzeldin, Yahya H, Yan, Shen, He, Chaoyang, Ferrara, Emilio, and Avestimehr, Salman. Fairfed: Enabling group fairness in federated learning. *arXiv preprint arXiv:2110.00857* (2021).

[62] Fallah, Alireza, Mokhtari, Aryan, and Ozdaglar, Asuman. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems 33* (2020).

[63] Fang, Minghong, Cao, Xiaoyu, Jia, Jinyuan, and Gong, Neil Zhenqiang. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security* (2020).

[64] Finn, Chelsea, Abbeel, Pieter, and Levine, Sergey. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning* (2017), PMLR, pp. 1126–1135.

[65] Frankle, Jonathan, and Carbin, Michael. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR* (2019).

[66] Gao, Yang, Colombo, Nicolo, and Wang, Wei. Adapting by pruning: A case study on bert. *arXiv preprint arXiv:2105.03343* (2021).

[67] Gertner, Yael, Goldwasser, Shafi, and Malkin, Tal. A random server model for private information retrieval (or how to achieve information theoretic pir avoiding data replication). *IACR Cryptology ePrint Archive 1998* (1998), 13.

[68] Geyer, Robin C., Klein, Tassilo, and Nabi, Moin. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[69] Ghinita, Gabriel, Kalnis, Panos, Khoshgozaran, Ali, Shahabi, Cyrus, and Tan, Kian-Lee. Private queries in location based services: anonymizers are not necessary. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), ACM, pp. 121–132.

[70] Ghosh, Avishek, Chung, Jichan, Yin, Dong, and Ramchandran, Kannan. An efficient framework for clustered federated learning. *arXiv preprint arXiv:2006.04088* (2020).

[71] Glorot, Xavier, and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS* (2010).

[72] Goldberg, Ian. Improving the robustness of private information retrieval. In *Security and Privacy, 2007. SP'07. IEEE Symposium on* (2007), IEEE, pp. 131–148.

[73] Goldberg, Ian, Devet, Casey, Lueks, Wouter, Yang, Ann, Hendry, Paul, and Henry, Ryan. Percy++ project on sourceforge, 2014.

[74] Goodfellow, Ian J, Mirza, Mehdi, Xiao, Da, Courville, Aaron, and Bengio, Yoshua. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211* (2013).

[75] Gupta, Trinabh, Crooks, Natacha, Mulhern, Whitney, Setty, Srinath TV, Alvisi, Lorenzo, and Walfish, Michael. Scalable and private media consumption with popcorn. In *NSDI* (2016), pp. 91–107.

[76] Hafiz, Syed Mahbub, and Henry, Ryan. Querying for queries: Indexes of queries for efficient and expressive it-pir. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 1361–1373.

[77] Hanzely, Filip, and Richtárik, Peter. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516* (2020).

[78] Hashimoto, Tatsunori, Srivastava, Megha, Namkoong, Hongseok, and Liang, Percy. Fairness without demographics in repeated loss minimization. In *International Conference on Machine Learning* (2018).

[79] Hay, Michael, Elagina, Liudmila, and Miklau, Gerome. Differentially private rank aggregation. In *Proceedings of the 2017 SIAM International Conference on Data Mining* (2017), SIAM, pp. 669–677.

[80] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1026–1034.

[81] Henry, Ryan. Polynomial batch codes for efficient it-pir. *Proceedings on Privacy Enhancing Technologies 2016*, 4 (2016), 202–218.

[82] Henry, Ryan, Huang, Yizhou, and Goldberg, Ian. One (block) size fits all: Pir and spir with variable-length records via multi-block queries. In *NDSS* (2013).

[83] Henry, Ryan, Olumofin, Femi, and Goldberg, Ian. Practical pir for electronic commerce. In *CCS, 2011. Proceedings* (2011).

[84] Ho, Jonathan. Classifier-free diffusion guidance. *ArXiv abs/2207.12598* (2022).

[85] Ho, Jonathan, Jain, Ajay, and Abbeel, Pieter. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 6840–6851.

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

[87] Ivkin, Nikita, Rothchild, Daniel, Ullah, Enayat, Braverman, Vladimir, Stoica, Ion, and Arora, Raman. Communication-efficient distributed sgd with sketching. In *NeurIPS* (2019).

[88] Jabri, A., Fleet, David J., and Chen, Ting. Scalable adaptive computation for iterative generation.

[89] Jiang, Yihan, Konečnỳ, Jakub, Rush, Keith, and Kannan, Sreeram. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488* (2019).

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

[91] Karimireddy, Sai Praneeth, Jaggi, Martin, Kale, Satyen, Mohri, Mehryar, Reddi, Sashank J, Stich, Sebastian U, and Suresh, Ananda Theertha. Mime: Mimicking centralized stochastic algorithms in federated learning. *arXiv preprint arXiv:2008.03606* (2020).

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

[93] Karras, Tero, Aittala, Miika, Aila, Timo, and Laine, Samuli. Elucidating the design space of diffusion-based generative models. *ArXiv abs/2206.00364* (2022).

[94] Kemeny, John G, and Snell, James Laurie. *Mathematical models in the social sciences*, vol. 9. Blaisdell New York, 1962.

[95] Kenyon-Mathieu, Claire, and Schudy, Warren. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing* (2007), pp. 95–103.

[96] Ker, Justin, Wang, Lipo, Rao, Jai, and Lim, Tchoyoson. Deep learning applications in medical image analysis. *Ieee Access 6* (2017), 9375–9389.

[97] Khodak, Mikhail, Tu, Renbo, Li, Tian, Li, Liam, Balcan, Maria-Florina F, Smith, Virginia, and Talwalkar, Ameet. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. *Advances in Neural Information Processing Systems 34* (2021), 19184–19197.

[98] Kiayias, Aggelos, Leonardos, Nikos, Lipmaa, Helger, Pavlyk, Kateryna, and Tang, Qiang. Optimal rate private information retrieval from homomorphic encryption. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 222–243.

[99] Kingma, Diederik P, Salimans, Tim, Poole, Ben, and Ho, Jonathan. On density estimation with diffusion models. In *Advances in Neural Information Processing Systems* (2021), A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, Eds.

[100] Kirkpatrick, James, Pascanu, Razvan, Rabinowitz, Neil, Veness, Joel, Desjardins, Guillaume, Rusu, Andrei A, Milan, Kieran, Quan, John, Ramalho, Tiago, Grabska-Barwinska, Agnieszka, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences 114*, 13 (2017), 3521–3526.

[101] Knill, Oliver. A multivariable chinese remainder theorem. *arXiv preprint arXiv:1206.5114* (2012).

[102] Kohavi, Ron, et al. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd* (1996), vol. 96, pp. 202–207.

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

[104] Krizhevsky, Alex, and Hinton, Geoffrey. Learning multiple layers of features from tiny images.

[105] Kukkala, Vipin Kumar, Tunnell, Jordan, Pasricha, Sudeep, and Bradley, Thomas. Advanced driver-assistance systems: A path toward autonomous vehicles. *IEEE Consumer Electronics Magazine 7*, 5 (2018), 18–25.

[106] Kurihara, Jun, Kiyomoto, Shinsaku, Fukushima, Kazuhide, and Tanaka, Toshiaki. A fast (k, l, n)-threshold ramp secret sharing scheme. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 92*, 8 (2009), 1808–1821.

[107] Kushilevitz, Eyal, and Ostrovsky, Rafail. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on* (1997), IEEE, pp. 364–373.

[108] Langville, Amy N, and Meyer, Carl D. *Who's# 1?: the science of rating and ranking*. Princeton University Press, 2012.

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

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

[111] Li, Lichun, Militzer, Michael, and Datta, Anwitaman. rpir: ramp secret sharing-based communication-efficient private information retrieval. *International Journal of Information Security 16*, 6 (2017), 603–625.

[112] Li, Tian, Beirami, Ahmad, Sanjabi, Maziar, and Smith, Virginia. Tilted empirical risk minimization. In *International Conference on Learning Representations* (2021).

[113] Li, Tian, Hu, Shengyuan, Beirami, Ahmad, and Smith, Virginia. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning* (2021).

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

[115] Li, Tian, Sahu, Anit Kumar, Zaheer, Manzil, Sanjabi, Maziar, Talwalkar, Ameet, and Smith, Virginia. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[116] Li, Tian, Sanjabi, Maziar, Beirami, Ahmad, and Smith, Virginia. Fair resource allocation in federated learning. In *International Conference on Learning Representations* (2019).

[117] Li, Zhuohang, Zhang, Jiaxin, Liu, Luyang, and Liu, Jian. Auditing privacy defenses in federated learning via generative gradient leakage. *CoRR abs/2203.15696* (2022).

[118] Liang, Paul Pu, Liu, Terrance, Ziyin, Liu, Allen, Nicholas B, Auerbach, Randy P, Brent, David, Salakhutdinov, Ruslan, and Morency, Louis-Philippe. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523* (2020).

[119] Lim, Jia Qi, and Chan, Chee Seng. From gradient leakage to adversarial attacks in federated learning. In *IEEE International Conference on Image Processing (ICIP)* (2021), pp. 3602–3606.

[120] Lipmaa, Helger, and Pavlyk, Kateryna. A simpler rate-optimal cpir protocol. In *International Conference on Financial Cryptography and Data Security* (2017), Springer, pp. 621–638.

[121] Liu, Ruixuan, Cao, Yang, Yoshikawa, Masatoshi, and Chen, Hong. Fedsel: Federated sgd under local differential privacy with top-k dimension selection. In *International Conference on Database Systems for Advanced Applications* (2020), Springer, pp. 485–501.

[122] Lopez-Paz, David, and Ranzato, Marc'Aurelio. Gradient episodic memory for continual learning. *Advances in neural information processing systems 30* (2017), 6467–6476.

[123] Ludwig, Heiko, Baracaldo, Nathalie, Thomas, Gegi, and Zhou, Yi. IBM federated learning: an enterprise framework white paper V0.1. *CoRR abs/2007.10987* (2020).

[124] Maas, Andrew, Daly, Raymond E, Pham, Peter T, Huang, Dan, Ng, Andrew Y, and Potts, Christopher. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (2011), pp. 142–150.

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

[126] McMahan, H Brendan, Moore, Eider, Ramage, Daniel, Hampson, Seth, and Arcas, Blaise Aguera y. Communication-efficient learning of deep networks from decentralized data. *AISTATS* (2017).

[127] Melis, Luca, Song, Congzheng, Cristofaro, Emiliano De, and Shmatikov, Vitaly. Exploiting unintended feature leakage in collaborative learning. *40th IEEE Symposium on Security and Privacy* (2019).

[128] Mhamdi, El Mahdi El, Guerraoui, Rachid, and Rouault, Sébastien. The hidden vulnerability of distributed learning in byzantium. In *ICML* (2018).

[129] Minka, Thomas. Estimating a dirichlet distribution, 2000.

[130] Mittal, Prateek, Olumofin, Femi G, Troncoso, Carmela, Borisov, Nikita, and Goldberg, Ian. Pir-tor: Scalable anonymous communication using private information retrieval. In *USENIX Security Symposium* (2011), p. 31.

[131] Mohri, Mehryar, Sivek, Gary, and Suresh, Ananda Theertha. Agnostic federated learning. In *International Conference on Machine Learning* (2019).

[132] Mozaffari, Hamid, Shejwalkar, Virat, and Houmansadr, Amir. Fsl: Federated supermask learning. *arXiv preprint arXiv:2110.04350* (2021).

[133] Naseri, Mohammad, Hayes, Jamie, and De Cristofaro, Emiliano. Local and central differential privacy for robustness and privacy in federated learning. *arXiv preprint arXiv:2009.03561* (2020).

[134] Nasr, Milad, Shokri, Reza, and Houmansadr, Amir. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *Security and Privacy (SP), 2019 IEEE Symposium on* (2019).

[135] Nichol, Alexander Quinn, and Dhariwal, Prafulla. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning* (18–24 Jul 2021), Marina Meila and Tong Zhang, Eds., vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 8162–8171.

[136] Olumofin, Femi, and Goldberg, Ian. Privacy-preserving queries over relational databases. In *International Symposium on Privacy Enhancing Technologies Symposium* (2010), Springer, pp. 75–92.

[137] Olumofin, Femi, and Goldberg, Ian. Revisiting the computational practicality of private information retrieval. In *International Conference on Financial Cryptography and Data Security* (2011), Springer, pp. 158–172.

[138] Paillier, Pascal. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1999), Springer, pp. 223–238.

[139] Pang, Liao-Jun, and Wang, Yu-Min. A new (t, n) multi-secret sharing scheme based on shamir's secret sharing. *Applied Mathematics and Computation 167*, 2 (2005), 840–848.

[140] Paulik, Matthias, Seigel, Matt, and Mason, Henry. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503* (2021).

[141] Piotrowska, Ania M, Hayes, Jamie, Gelernter, Nethanel, Danezis, George, and Herzberg, Amir. Annotify: A private notification service. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society* (2017), ACM, pp. 5–15.

[142] Qiao, Siyuan, Wang, Huiyu, Liu, Chenxi, Shen, Wei, and Yuille, Alan Loddon. Weight standardization. *ArXiv abs/1903.10520* (2019).

[143] Ramanujan, Vivek, Wortsman, Mitchell, Kembhavi, Aniruddha, Farhadi, Ali, and Rastegari, Mohammad. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020).

[144] Ramanujan, Vivek, Wortsman, Mitchell, Kembhavi, Aniruddha, Farhadi, Ali, and Rastegari, Mohammad. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 11893–11902.

[145] Reddi, Sashank J, Charles, Zachary, Zaheer, Manzil, Garrett, Zachary, Rush, Keith, Konečnỳ, Jakub, Kumar, Sanjiv, and McMahan, Hugh Brendan. Adaptive federated optimization. In *ICLR* (2020).

[146] Salimans, Tim, and Ho, Jonathan. Progressive distillation for fast sampling of diffusion models. *ArXiv abs/2202.00512* (2022).

[147] Shamir, Adi. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[148] Shejwalkar, Virat, and Houmansadr, Amir. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS* (2021).

[149] Shejwalkar, Virat, Houmansadr, Amir, Kairouz, Peter, and Ramage, Daniel. Back to the drawing board: A critical evaluation of poisoning attacks on federated learning. In *Security and Privacy (SP)*. 2021.

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

[151] Shoup, Victor. Number theory library (ntl) for c++. *Available at Shoup's homepage http://shoup. net/ntl* (2010).

[152] Smith, Virginia, Chiang, Chao-Kai, Sanjabi, Maziar, and Talwalkar, Ameet. Federated multi-task learning. In *Advances in Neural Information Processing Systems* (2017).

[153] Smith, Virginia, Chiang, Chao-Kai, Sanjabi, Maziar, and Talwalkar, Ameet. Federated multi-task learning. In *Neural Information Processing Systems (NIPS)* (2017).

[154] Song, Jiaming, Meng, Chenlin, and Ermon, Stefano. Denoising diffusion implicit models. *ArXiv abs/2010.02502* (2021).

[155] Stern, Julien P. A new and efficient all-or-nothing disclosure of secrets protocol. In *International Conference on the Theory and Application of Cryptology and Information Security* (1998), Springer, pp. 357–371.

[156] Stich, Sebastian U, Cordonnier, Jean-Baptiste, and Jaggi, Martin. Sparsified sgd with memory. In *NeurIPS* (2018).

[157] Sun, Ziteng, Kairouz, Peter, Suresh, Ananda Theertha, and McMahan, H Brendan. Can you really backdoor federated learning? In *NeurIPS FL Workshop* (2019).

[158] Sunkara, Raja, and Luo, Tie. No more strided convolutions or pooling: A new cnn building block for low-resolution images and small objects. *ArXiv abs/2208.03641* (2022).

[159] Wang, Hongyi, Sreenivasan, Kartik, Rajput, Shashank, Vishwakarma, Harit, Agarwal, Saurabh, Sohn, Jy-yong, Lee, Kangwook, and Papailiopoulos, Dimitris. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS* (2020).

[160] Wang, Jianyu, Liu, Qinghua, Liang, Hao, Joshi, Gauri, and Poor, H Vincent. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in Neural Information Processing Systems 33* (2020).

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

[162] Wang, Ning, Xiao, Xiaokui, Yang, Yin, Zhao, Jun, Hui, Siu Cheung, Shin, Hyejin, Shin, Junbum, and Yu, Ge. Collecting and analyzing multidimensional data with local differential privacy. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)* (2019), IEEE, pp. 638–649.

[163] Wei, Wenqi, Liu, Ling, Loper, Margaret, Chow, Ka Ho, Gursoy, Mehmet Emre, Truex, Stacey, and Wu, Yanzhao. A framework for evaluating gradient leakage attacks in federated learning. *CoRR abs/2004.10397* (2020).

[164] Wortsman, Mitchell, Ramanujan, Vivek, Liu, Rosanne, Kembhavi, Aniruddha, Rastegari, Mohammad, Yosinski, Jason, and Farhadi, Ali. Supermasks in superposition. In *NeurIPS* (2020).

[165] Xie, Chulin, Huang, Keli, Chen, Pin-Yu, and Li, Bo. Dba: Distributed backdoor attacks against federated learning. In *ICLR* (2019).

[166] Yamamoto, Hirosuke. Secret sharing system using (k, l, n) threshold scheme. *Electronics and Communications in Japan (Part I: Communications) 69*, 9 (1986), 46–54.

[167] Yang, Chou-Chen, Chang, Ting-Yi, and Hwang, Min-Shiang. A (t, n) multi-secret sharing scheme. *Applied Mathematics and Computation 151*, 2 (2004), 483–490.

[168] Yekhanin, Sergey. New locally decodable codes and private information retrieval schemes. In *Electronic Colloquium on Computational Complexity* (2006), vol. 127, p. 2006.

[169] Yin, Dong, Chen, Yudong, Ramchandran, Kannan, and Bartlett, Peter L. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML* (2018).

[170] Yu, Tao, Bagdasaryan, Eugene, and Shmatikov, Vitaly. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758* (2020).

[171] Zari, Oualid, Xu, Chuan, and Neglia, Giovanni. Efficient passive membership inference attack in federated learning. *arXiv preprint arXiv:2111.00430* (2021).

[172] Zhang, Daniel Yue, Kou, Ziyi, and Wang, Dong. Fairfl: A fair federated learning approach to reducing demographic bias in privacy-sensitive classification models. In *2020 IEEE International Conference on Big Data (Big Data)* (2020), IEEE, pp. 1051–1060.

[173] Zhang, Lixia, Afanasyev, Alexander, Burke, Jeffrey, Jacobson, Van, Crowley, Patrick, Papadopoulos, Christos, Wang, Lan, Zhang, Beichuan, et al. Named data networking. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 66–73.

[174] Zhang, Michael, Sapra, Karan, Fidler, Sanja, Yeung, Serena, and Alvarez, Jose M. Personalized federated learning with first order model optimization. In *International Conference on Learning Representations* (2021).

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

[176] Zhou, Hattie, Lan, Janice, Liu, Rosanne, and Yosinski, Jason. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS* (2019).

[177] Zhu, Ligeng, Liu, Zhijian, and Han, Song. Deep leakage from gradients. In *Advances in Neural Information Processing Systems* (2019), pp. 14747–14756.