Securing Multi-Layer Federated Learning: Detecting and Mitigating Adversarial Attacks

Justin Matthew Gouge

A Thesis Submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements
for the Degree of Master of Science

Graduate Program in Electrical Engineering and Computer Science

York University
Toronto, Ontario

February 2025

Abstract

Within the realm of federated learning (FL), adversarial entities can poison models, slowing down or destroying the FL training process. Therefore, attack prevention and mitigation are crucial for FL. Real-world scenarios may necessitate additional separation or abstraction between clients and servers. When considering multi-layer FL systems, which contain edge server layers, the structural differences warrant new strategies to handle adversaries. While existing works primarily address attack prevention and mitigation in conventional two-layer FL systems, research on attack prevention and mitigation in multi-layer federated learning systems remains limited. This thesis aims to address this gap by investigating the defense strategies in a multi-layered FL system. We propose new methods for anomaly detection and removal of attackers/adversarial entities from training in a multi-layer FL system. First, we train a variational autoencoder (VAE) using the model updates collected from the edge servers. This allows the VAE to discern between benign and adversarial model updates. Following that, we deploy the VAE to detect which edge servers at the cohort level contain malicious clients. Subsequently, we devise two malicious client exclusion strategies: the scoring-based method, which applies a score for each client based upon its appearances within cohorts labeled as benign or malicious, and the Bayesian-based method, which uses Bayesian inference to predict if a specific client is malicious based on the statistical performance of the autoencoder. Both approaches are aimed at mitigating potential harm caused by malicious clients during model training. The experimental results demonstrate the superiority of the proposed methods over previous works for traditional FL mitigation under a variety of scenarios.

Acknowledgements

I would like to thank my supervisor, Prof. Ping Wang for her guidance and support throughout my research. I am grateful for her patience and encouragement over the past years. I would like to thank my examiners, Prof. Ruth Urner and Prof. Liang Xue for diligently engaging with my thesis and providing valuable suggestions. I would like to thank the reviewers and team from IEEE ICCC 2024 for providing me with the opportunity to publish conference level research. Finally, thanks to my family for all their emotional and financial support during these hard economic times, and previously during the tail end of the pandemic.

Contents

List of Tables

List of Figures

List of Abbreviations

AE Autoencoder

AI Artificial Intelligence

DP Differential Privacy

FABA Filtering-based Byzantine Aggregation

FedAvg Federated Averaging

FL Federated Learning

HE Homomorphic Encryption

IID Independent & Identically Distributed

MLFL Multi-Layer Federated Learning

non-IID Not Independent & Identically Distributed

SGD Stochastic Gradient Descent

SMPC Secure Multi-Party Computation

VAE Variational Autoencoder

Chapter 1

1 Introduction

This chapter introduces various aspects of federated learning (FL), including traditional FL, multi-layer federated learning (MLFL), and potential vulnerabilities in FL systems. Section 1.1 provides an overview of FL and anomaly detection. Section 1.2 outlines the contributions of this work. Section 1.3 describes the organization of the thesis. Finally, in section 1.4, we reference our publication which is related to the research presented in this thesis.

1.1 Federated Learning and Anomaly Detection

The advancement of FL is driven by the proliferation of edge devices, including mobile phones, Internet of Things devices, and smart watches, etc. These types of devices are equipped with sensors that collect real-time data. With the rapid improvement of wireless networks, we can expect deep learning and other methods to enhance what is possible with a large network of small edge devices. This can include applications such as traffic light timing, disaster monitoring, and self-health alerts [2]. Additionally, these devices usually possess the capability to process data locally. Leveraging training data distributed among such clients allows federated learning to train a highly accurate shared global model [3]. Major companies and telecommunications providers currently employ federated learning, supporting privacy-sensitive applications such as private messaging systems and banking [4].

A classical FL system typically consists of a two-layered architecture. In this setup, participating devices will use their data to train a local model, and the model weights are then shared with a global server. The job of the server is to collect the updates from each device and aggregate a global model. This global model is then shared back with the devices. To enhance privacy preservation, multi-layered federated learning introduces an additional

level: the cohort level. In addition to the client level and server level, this newly-introduced cohort level enables a sufficient separation between client and server. In this case, the model updates are aggregated at two different levels: the cohort and the server [1], [5]. Aramoon *et al.* [6] explore Meta-FL, an attempt to increase privacy and robustness in the multi-layer FL system. This approach also allows for new aggregation strategies to be deployed at both the cohort and server levels. Adding an additional layer to the FL system provides several advantages. For instance, the server can monitor incoming updates without violating the privacy of individual clients or risk vulnerabilities such as gradient leakage. It also makes it harder for adversaries to hinder global model performance, since they would need to have a significant impact on the entire cohort, rather than just being able to upload their attack directly to the server.

FL systems are vulnerable to malicious client attacks [7], which has become a significant issue for their deployment. The server, which performs aggregation, cannot govern or be responsible for the actions taken by clients, nor can it access the data they are using. Consequently, clients can send illegitimate model updates to the server, which enacts adversarial attacks on the global model. Untargeted attacks aim to degrade the global model performance, potentially causing model training failure. In contrast, targeted attacks aim to modify the global model's behaviour in a specific way so that the overall model performance remains unaffected, but certain data instances are handled differently by the model. An example could be miss-recognizing 3 for an 8 in a MNIST data sample [8]. Previous work [8], [9], [10] has shown that both targeted and untargeted attacks can have damaging effects. In order to address this issue, it is necessary to find a method to detect and remove malicious clients to protect the global model's health. Previously, there have been various successful approaches to the detection and removal of malicious clients in a standard 2-layer FL system, as shown in Figure 1. [8], [11], [12], [13].

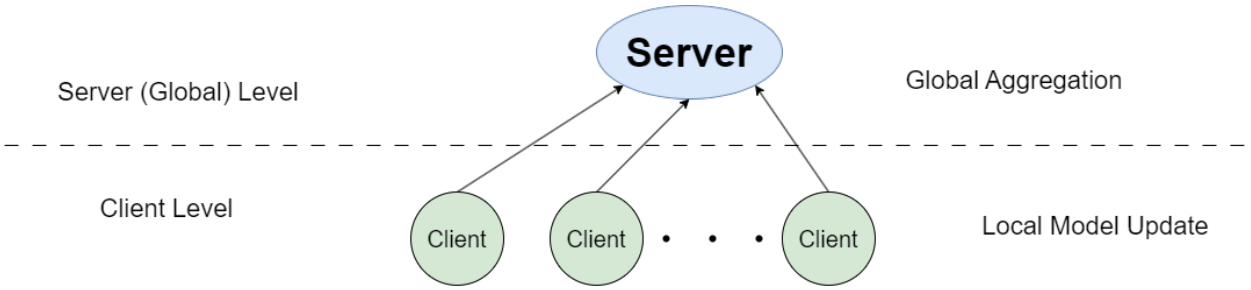

Figure 1: System Model of 2-layer FL

In the context of MLFL systems, as shown in Figure 2, detecting attackers becomes inherently more challenging due to the presence of intermediary layers between the server and clients. Please note this diagram shows the generalized MLFL structure, and is not the one we use in our work. These middle layers shield attackers from direct scrutiny by the server, rendering them more covert. As a result, traditional detection-based approaches employed in 2-layer systems are not directly applicable. This thesis explores how to identify and remove malicious users from participating in multi-layer federated learning systems. Our approach involves several key steps. First, we employ a variational autoencoder (VAE) to detect suspicious model updates aggregated at the cohort level, and then we develop scoring-based and Bayesian-based methods to identify malicious users.

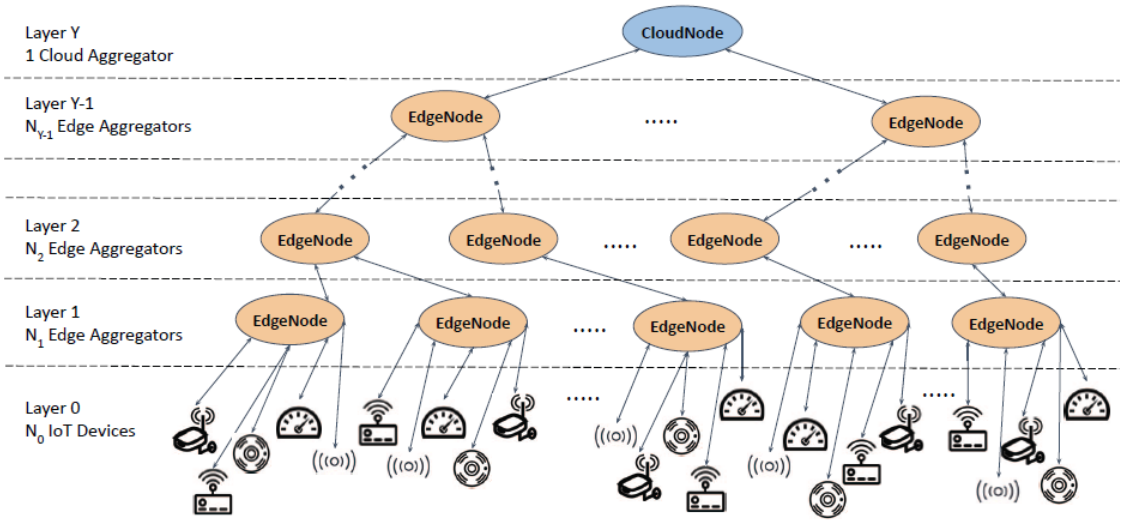

Figure 2: Multi-Layer FL System as displayed in [1]

1.2 Contributions

Our main contributions are as follows:

- To the best of our knowledge, this is the first work that addresses the malicious user detection issue in multi-layer FL systems. In this context, identifying attackers becomes notably intricate due to the absence of a direct communication line between the server and clients. Consequently, traditional methods relying on client weight updates are ineffective for detecting attackers.
- We propose two novel methods for detecting and eliminating attackers from a training network. These methods are designed to adapt to different situations, depending on whether we possess prior knowledge about the attack, such as the proportion of attackers in the system.
- Our proposed methods undergo rigorous evaluation. Experimental results demonstrate that our proposed methods outperform the established benchmarks.

1.3 Organization of Thesis

This thesis is structured as follows: Chapter 2 introduces background information relevant to the work presented in this thesis. Chapter 3 presents a literature review of relevant related works in FL, providing groundwork for the research present in this thesis. Chapter 4 details our methodology, including the use of Variational Autoencoders (VAE), novel approaches for attacker detection and removal, along with the experiment procedures. Finally, Chapter 5 concludes the thesis and discusses future works.

1.4 Related Publication

Gouge, J., and Wang, P. Securing Multi-Layer Federated Learning: Detecting and Mitigating Adversarial Attacks. *IEEE/CIC International Conference on Communications in China* (2024).

Chapter 2

2 Background Knowledge

In this chapter, we provide readers with relevant background information. We also include specific content from related literature that will help support understanding of the thesis topic and new contributions. Specifically, we lay out the basics of federated learning, multi-layer federated learning and attacks that we test against our methods.

2.1 Federated Learning Basics

In traditional machine learning (ML), a central server collects data from user devices to train a global model. This involves uploading data from the user devices to a central server. This server then performs training with the provided data to create the globally trained model which is then shared back to users [14]. Generally, this data comes from mobile devices such as cellular phones. The wealth of data available from personal devices allows for massive leaps in user experience. This can include things such as language models for improvement of speech recognition or more realistic AI voice, in addition to imaging models which may be able to improve or select good photos automatically for your post on social media. In current networks, with many users present, it is quite likely that much of this data is privacy sensitive. Therefore, it is crucial to consider a method of training which does not expose personal data to the central server.

Federated learning is a machine learning method proposed to maintain data privacy on user devices while still achieving a well-trained global model. This is done by having the users perform training locally, then uploading the model/weight updates to the server. The centralized server then performs federated averaging to update the global model [3]. In this approach, the central entity coordinates with every individual client that is participating in

training. It attempts to ensure that training is performed locally on each client and only model updates are shared with the server [15]. The goal is to collaboratively train a machine learning model, with each client contributing valuable updates.

There are several scenarios where FL is preferable to traditional machine learning. Real-world data that we collect from mobile devices may provide an advantage over generated or proxy data. Generally, this data is very large/complex and may also contain private information of the user which the device belongs to [16]. Therefore, it is unprofessional to store this data simply for the purpose of model training. When it comes to supervised FL, it is known that we can infer the labels from user interaction [17]. FL has established privacy advantages when weighed against centralized learning. Even if data is anonymous or the users participating are also anonymous, privacy can be compromised when joined with other data [18]. In the setting of pure FL, only the minimal information required for training/model improvement is shared with the server. This means that transmitted data will not contain any identifying data or personal information [19].

The training process involves computing gradients based on local data, without sharing raw data with the central server. After training, each device sends back only the model updates (usually in the form of gradients) to the central server. The central server aggregates these updates to refine the global model. This aggregation step can be done in various ways, such as averaging the gradients or using more sophisticated techniques like Federated Averaging [3], which adjusts for imbalances in data distribution across devices to ensure fair representation in the global model.

One of the key benefits of federated learning is privacy preservation. Since raw data remains on the devices and only model updates are exchanged, the risk of exposing sensitive information is significantly reduced. This makes federated learning suitable for applications where regulatory compliance or ethical considerations require strict data privacy controls. Another advantage is scalability and efficiency. Federated learning allows for parallel and

distributed model training across numerous devices simultaneously, potentially accelerating the training process compared to centralized approaches. Moreover, it enables inclusion of diverse datasets that may be geographically distributed or subject to local regulatory constraints, thereby improving the robustness and generalization of the global model. However, federated learning also presents challenges. These include managing heterogeneity in data across devices, ensuring consistency and fairness in model updates, handling communication and computational overhead, and addressing potential biases introduced by the distributed nature of data collection [20].

Overall, federated learning is a promising approach that empowers organizations to leverage decentralized data for model training while upholding privacy and security standards. Its adoption continues to grow as industries seek ways to harness the power of machine learning without compromising sensitive user data.

2.1.1 Federated Learning Technical Process

In this subsection, we describe the step-by-step process of FL in detail [3].

1. **Defining the Global Model**

- Initialize a global model θ which represents the parameters (weights) of the neural network.

2. **Client Selection**

- Devices (clients) are selected to participate based on criteria such as connectivity, battery level, or data relevance.

3. **Distribute the Model to Clients**

- The central server distributes the initial global model θ_0 to K clients. Each client k receives θ_0.

4. **Local Training on Clients**

- Each client k has its own local dataset D_k.
- Each client updates the model θ_0 using its local dataset D_k. This involves several iterations of gradient descent to minimize the local loss function $L_k(\theta)$.
- The local training process can be mathematically represented as:

$$\theta_k^{(t+1)} = \theta_k^{(k)} - \eta \nabla L_k(\theta_k^{(t)}) \tag{1}$$

where η is the learning rate, $\theta_k^{(t)}$ are the model parameters at iteration t for client k, and $\nabla L_k(\theta_k^{(t)})$ is the gradient of the loss function L_k with respect to θ_k.

5. Upload Local Models to Server

- After local training, each client k uploads its updated model parameters θ_k to the central server.

6. Aggregate the Local Models

- The central server aggregates the updated model parameters from all clients to form a new global model. A common aggregation method is Federated Averaging (FedAvg), which is mathematically represented as:

$$\theta^{(t+1)} = \sum_{k=1}^{K} \frac{|D_k|}{|D|} \theta_k^{(t+1)} \tag{2}$$

where K is the number of clients, $\theta_k^{(t+1)}$ are the updated model parameters for client k, and D references the dataset.

- The Federated Averaging algorithm combines local models' updates by computing a weighted average of the clients' model parameters. This ensures that the global model reflects the contributions from all participating clients.
- Mathematically, the update rule for FedAvg can be written as shown in Eq. 2, where

$|D_k|$ is the number of data points on client k, and $|D|$ is the total number of data points across all clients, i.e., $|D| = \sum_{k=1}^{K} |D_k|$.

7. Update and Distribute the Global Model

- The central server updates the global model parameters θ to $\theta^{(t+1)}$ and then distributes the updated model to all clients.
- This process is repeated for several rounds until the model converges.

8. Model Evaluation

- Once the model has converged, it can be evaluated on a separate test dataset to determine its performance.

This step-by-step process ensures that the global model is updated in a decentralized manner, leveraging the data from all clients without requiring the data to be centralized. This is particularly useful in scenarios where data privacy and security are critical.

2.2 Multi-Layer Federated Learning

In our work, the considered system model is called multi-layer FL. Multi-layer federated learning (MLFL) is an advanced extension of federated learning (FL). In traditional FL, data remains on local devices (clients), and only model updates are shared with a central server. The server aggregates these updates to improve a global model, ensuring that raw data never leaves the client devices, thus enhancing privacy and security. However, as the number of clients grows, this single-layer aggregation can become inefficient and challenging to manage.

MLFL addresses these scalability and efficiency issues by introducing a hierarchical structure to the aggregation process. The key idea is to add intermediate aggregation points between the clients and the central server. These intermediate servers, which could be cohort or edge servers, aggregate updates from multiple clients within their region before sending a

consolidated update to the central server. This multi-layer approach distributes the computational and communication load more evenly and reduces the direct communication burden on the central server [1], [6], [21].

For example, in a healthcare setting, patient data may be distributed across several hospitals. Each hospital acts as an intermediate server, aggregating updates from individual patient devices within the hospital. These intermediate updates are then sent to a central server that aggregates updates from all participating hospitals to improve the global model. This hierarchical aggregation not only improves scalability by reducing the number of direct communications with the central server but also enhances privacy by minimizing the exposure of individual updates [22].

The benefits of MLFL are widespread. By distributing the aggregation process, MLFL can handle larger numbers of clients and more complex models, making it more scalable. It also reduces communication costs and latency, as updates are partially aggregated before reaching the central server. Furthermore, the additional layer of aggregation adds another level of privacy, as local updates are aggregated and anonymized before being sent to higher levels [5]. Despite these advantages, MLFL also presents several challenges. Implementing and managing a multi-layer federated learning system is more complex than traditional FL. Communication costs, while reduced, can still be significant, especially when intermediate servers are numerous or widely distributed. Ensuring model consistency and convergence across multiple layers of aggregation can be difficult, particularly in non-IID (non-independent and identically distributed) data environments [23].

Security and privacy remain crucial considerations in MLFL. Techniques such as differential privacy, which adds noise to updates to protect individual data points [6], and secure aggregation, which ensures that individual updates cannot be reconstructed from the aggregated updates, are essential to main the level of professionalism and privacy that is desired by users [21]. Additionally, adaptive learning rates and staleness-aware aggregation are op-

timization techniques used to manage the variability and asynchrony in client updates [24]. Applications of MLFL span various domains, including healthcare, where collaborative models can be trained across hospitals while preserving patient data privacy; smart cities, where data from various sensors and devices can be aggregated regionally before central analysis; and autonomous vehicles, where models are shared among vehicles to improve safety and performance [25].

2.2.1 Multi-Layer Federated Learning Technical Process

MLFL extends FL by introducing a hierarchical structure where model updates are aggregated at multiple levels (local, edge/cohort, and global) which improves scalability, efficiency, and privacy [1].

1. Hierarchical Aggregation

- Initialize a global model θ which represents the parameters (weights) of the neural network.

2. Client Selection

- Devices (clients) are selected to participate based on criteria such as connectivity, battery level, or data relevance.

3. Distribute the Model to Clients

- The central server distributes the initial global model θ_0 to K clients by distributing θ_0 through the cohort servers. Each client k receives θ_0.
- Local Updates: Each client k with local data D_k trains a local model and updates the model θ_0. This involves several iterations of gradient descent to minimize the local loss function $L_k(\theta)$.
- Intermediate Aggregation: Local updates from multiple clients are sent to an intermediate/edge server where they are aggregated. The updates are collected from each

client within their respective cohort.

- Global Aggregation: Aggregated updates from intermediate servers are sent to a central server for final aggregation, forming the global model update.

4. Mathematical Formulation

- The global objective can be formulated as:

$$\min_{\theta} L_k(\theta) = \sum_{k=1}^{K} \frac{|D_k|}{|D|} L_k(\theta) \tag{3}$$

where $L_k(\theta)$ is the local loss function for client k and $|D|$ is the total data size across all clients.

- Local training is done to compute local gradients by each client:

$$\theta_k^{(t+1)} = \theta_k^{(k)} - \eta \nabla L_k(\theta_k^{(t)}) \tag{4}$$

- Edge server aggregation is written as:

$$\theta_{edge}^{(t+1)} = \sum_{k \in clients} \frac{|D_k|}{|D_{edge}|} \theta_k^{t+1} \tag{5}$$

where $|D_{edge}|$ is the total data size of clients associated with the edge server, and 'clients' here refers to the clients that send their updates to that edge server.

- When it comes to global aggregation, we consider the following which use the updates from the edge server level:

$$\theta^{(t+1)} = \sum_{edge \in edge\ servers} \frac{|D_{edge}|}{|D|} \theta_{edge}^{(t+1)} \tag{6}$$

5. Security and Privacy

There are various security and privacy in addition to optimization techniques that can be applied.

- Differential Privacy: Adding noise to updates to protect individual data points from being identified.
- Secure Aggregation: Ensuring that individual updates cannot be reconstructed from the aggregated updates.
- Staleness-Aware Aggregation: Managing the staleness of updates due to asynchronous communication.
- Adaptive Learning Rates: Adjusting learning rates based on the variance of updates to improve convergence.

MLFL enhances the traditional federated learning paradigm by introducing a hierarchical structure that improves scalability, efficiency, and privacy. It involves complex communication protocols, security measures, and optimization techniques to ensure effective and efficient training of global models across decentralized data sources. The adoption of MLFL in diverse domains demonstrates its potential and underscores the need for ongoing research to address its challenges and refine its methodologies [25].

2.3 Introduction of Attacks

In our work, we consider 3 different attack scenarios: sign-flipping attack, additive-noise attack, and backdoor attack. These attacks are carried out by malicious clients, and affect the model updates which would be produced from each client. In the multi-layer scenario, it is more challenging to detect and deal with the attackers because of the separation of communications between the server and users. In the following subsections, we explain the details for each type of attack.

2.3.1 Sign-Flipping Attack

In a sign-flipping attack, an adversary intentionally flips the sign of the gradients or updates sent to the central server during the training process [26]. This means that if the true gradient is positive, the adversary sends a negative gradient, and vice versa. Additionally, in federated learning we assume the server which receives the client update (edge or central) has no information about the data, so this may also make it more difficult to detect the attack.

How does the attack work?

1. Gradient Calculation: Each participating client computes the gradient/update based on its own private local data.
2. Gradient Flipping: Each adversarial client flips the sign of its computed gradients.
3. Aggregation: The central or cohort server aggregates the gradients from all nodes, including the malicious ones.
4. Model Update: The aggregated (and now corrupted) gradients are used to update the model parameters.

This attack results in model degradation. The primary effect is a significant degradation in the model's performance. The updates from malicious clients counteract the correct updates from honest clients, leading to slower convergence or even divergence. Additionally, the robustness is also challenged when dealing with the poisoning attack. Sign-flipping attacks exploit the trust that the central server places in the received updates, making the system less robust to adversarial behavior.

Sign-flipping attacks highlight the vulnerabilities in collaborative and distributed learning environments. They emphasize the need for robust aggregation and verification mechanisms to ensure the integrity of the training process and the resulting model. In summary, sign-flipping attacks are a potent adversarial tactic that can significantly impair the training of machine learning models in distributed settings, necessitating the development of sophisti-

cated defense mechanisms to maintain model performance and reliability.

2.3.2 Additive Noise Attack

An additive noise attack attempts to poison model training by adding noise to model updates. The adversary adds noise to the gradients or updates sent to the central server during the training process. This noise can be designed to degrade the model's performance, slow down the convergence, or mislead the training process . Similar to sign-flipping attacks, this attack is relevant in scenarios where multiple clients collaboratively train a model by computing updates locally and then sending these updates to a central server for aggregation [27].

How does the attack work?

1. Gradient Calculation: Each participating client computes the gradient (or update) based on its local data.
2. Noise Addition: An adversarial client adds noise to its computed gradients before sending them to the central or cohort server. This noise can be random or specifically crafted.
3. Aggregation: The central server aggregates the gradients from all updates, including the noisy ones.
4. Model Update: The aggregated (and now corrupted) gradients are used to update the model parameters.

The added noise can lead to incorrect updates to the model parameters, reducing the model's accuracy and overall performance. The presence of noisy gradients can slow down the convergence of the training process, requiring more iterations to reach an acceptable model performance. In extreme cases, the noise can cause the model to diverge, failing to learn anything meaningful [28].

Types of Noise

- Random Noise: Simple, unstructured noise that can still degrade performance but is easier to detect and mitigate.
- Structured Noise: Carefully crafted noise that can be more challenging to identify and counteract, potentially targeting specific weaknesses in the model or training process.

Similarly to sign-flipping attacks, this type of poisoning attack requires robust and secure aggregation mechanisms and the need for continuous monitoring and validation of the training process to maintain the integrity and performance of the model. Overall, additive noise attacks are a significant challenge to the training of machine learning models. Sophisticated defense techniques should be developed and applied to detect and remove attackers from training to prevent model degradation.

2.3.3 Backdoor Attack

Unlike the poisoning attacks described above, a backdoor attack deploys a method to trick a trained model into miss-classifying certain datum. A backdoor attack is a sophisticated type of adversarial attack targeting machine learning models, particularly in distributed or federated learning environments. In a backdoor attack, an adversary injects a hidden trigger into the training process so that the model behaves normally on standard inputs but produces incorrect or malicious outputs when the trigger is present. This type of attack allows the model to be subverted for specific inputs while maintaining overall performance on benign data [29].

How does the attack work?

1. Trigger Design: The adversary designs a trigger pattern (e.g., a specific image pattern, a particular phrase, or any identifiable feature).
2. Gradient Manipulation: During the local training phase, the adversarial client injects the trigger into some training data and assigns the desired malicious labels to these samples.

3. Update Sending: The adversarial client sends the manipulated gradients to the central server along with gradients from normal training data.

4. Aggregation: The central server aggregates the gradients from all clients, including the ones with backdoored data.

5. Model Update: The aggregated gradients are used to update the model, embedding the backdoor.

The effects of backdoor attacks are widespread. Generally, these are covert and well planned so that models are unaware they are infected. The primary power of a backdoor attack is its stealthiness. The model performs well on standard validation and test datasets, making the attack hard to detect. The model behaves maliciously only when the trigger is present, allowing specific inputs to be subverted while maintaining overall performance. This type of attack pose significant security risks, as they can be used to bypass security measures, manipulate outputs, or cause targeted failures.

Types of Triggers

- Visible Triggers: Easily identifiable patterns or features added to the input data, such as a watermark on an image.

- Invisible Triggers: Subtle modifications that are not easily noticeable, such as slight pixel changes in images or minor perturbations in text.

Each of the previously mentioned attack types have various practical implications, underscoring the importance of evaluation and defense techniques. Some important techniques are listed below. A more detailed exploration of these will be provided in Chapter 3.

Defense Mechanisms

- Robust Aggregation: Techniques like median or trimmed mean aggregation reduce the influence of outliers, including flipped gradients.

- Anomaly Detection: Detecting and ignoring or removing suspicious updates.
- Secure Multiparty Computation: Ensuring that updates cannot be tampered with during transmission.
- Redundancy and Cross-Checking: Using redundancy and cross-validation among multiple nodes to identify and mitigate adversarial updates.
- Noise Detection: Implementing anomaly detection methods to identify and disregard suspicious updates.
- Differential Privacy: Adding controlled noise to gradients intentionally in a way that maintains privacy while ensuring the added noise does not degrade the model's performance significantly.
- Robust Training: Using techniques like data augmentation and adversarial training to improve the model's robustness to adversarial manipulation.
- Sanitizing Updates: Examining and sanitizing updates from nodes to remove potential backdoors before aggregation.
- Model Validation: Continuously validating the model against known triggers and potential backdoors to ensure integrity.

Chapter 3

3 Related Works

In this chapter, we will provide an overview of some related works to our research.

3.1 Federated Learning Exploration

FL was introduced to enable training machine learning models across decentralized data sources while preserving data privacy. The foundational work by McMahan et al. [3] proposed the Federated Averaging (FedAvg) algorithm, which aggregates locally computed updates from clients to produce a global model. This approach laid the groundwork for numerous advancements in federated learning. FedAvg operates by iteratively performing local updates on client devices followed by aggregation on a central server. Each client computes updates using its local data, and these updates are then averaged by the server to update the global model. This method significantly reduces communication overhead compared to traditional distributed learning techniques, as clients only need to send model updates rather than raw data.

To address privacy concerns, Differential Privacy (DP) techniques have been integrated into FL frameworks. Differential Privacy ensures that the inclusion or exclusion of a single data point in a dataset does not significantly affect the output of a computation, thereby providing a mathematical guarantee of privacy. In [30] and [31], they emphasize the use of DP in machine learning by adding noise to gradients during training. More recently, the studies in [6] and [32] extend and apply these concepts to federated settings, ensuring that individual client data remains protected from inference attacks. In the federated learning context, differential privacy is typically implemented by adding calibrated noise to the updates sent from the clients to the server. This noise ensures that the updates do not reveal sensitive

information about individual data points. A hybrid method was proposed in [32], combining local and global differential privacy to enhance the privacy guarantees in federated learning.

Secure Multi-Party Computation (SMPC) and Homomorphic Encryption (HE) are other cryptographic approaches that enhance security in federated learning. A practical secure aggregation protocol using SMPC has been developed, which allows the server to aggregate client updates without revealing individual updates [21]. Meanwhile, others have applied HE to encrypt updates, ensuring that the server operates on encrypted data and cannot infer sensitive information [33]. SMPC techniques enable multiple parties to jointly compute a function over their inputs while keeping those inputs private. In FL, SMPC can be used to securely aggregate model updates from different clients. It is a secure aggregation protocol which ensures that individual updates are masked before being sent to the server. The server can then aggregate these masked updates without learning the contributions of individual clients. Homomorphic Encryption allows computations to be performed on encrypted data without decrypting it. Additive homomorphic encryption in FL can encrypt client updates. The server can then aggregate these encrypted updates, and only the aggregated result is decrypted, preserving the privacy of individual updates.

In addition to the aforementioned methods, security can also be applied to prevent gradient leakage which adversaries may use to infer private client data. It has been demonstrated in both FL and traditional ML that adding noise to gradient (updates) makes inference from leakage more difficult [34]. When dealing with a large enough data variance, the accuracy of gradient leakage attacks is significantly lower which causes an inability for adversarial entities to infer the original data. Compressing model gradients may also help with leakage. The idea behind this approach is to sparsify all gradients which affect algorithms whose aim is to infer data from leakage. It has been shown gradients can be compressed to a fraction of their original size, up to hundreds of times, without significantly affecting model accuracy. [35].

FL has been applied in various domains, showcasing its practical benefits. Google's

Gboard [36] uses FL to improve its next-word prediction models by training on user data directly on their devices. The local updates are aggregated to update the global model, enhancing the prediction quality while ensuring user privacy. This approach reduces the need to collect sensitive data on central servers, aligning with privacy regulations. Healthcare is another critical field of application, where FL enables collaborative training of medical models across different institutions [37], facilitating advancements in disease prediction and treatment without exposing sensitive patient data. Sheller et al. [37] demonstrated the use of FL for training models on medical imaging data from multiple hospitals. This approach allows for the development of robust models for disease diagnosis and treatment while preserving patient privacy.

Communication overhead is a significant challenge in FL due to the frequent exchange of model updates between clients and the server. Strategies such as federated dropout [38], more recently ensembling [39], and adaptive federated optimization [23] have been proposed to reduce communication costs by optimizing the frequency and size of updates. Federated dropout [38] reduces communication overhead by allowing clients to send only a subset of model parameters during each update round, thereby decreasing the amount of data transmitted and reducing the communication burden on clients and the server. Building on this concept, it has been shown in [39] that simple ensembling techniques can outperform federated dropout. Additionally, adaptive federated optimization techniques dynamically adjust the learning rate and update frequency based on the convergence behavior of the model. This approach reduces the number of communication rounds needed for model convergence, thereby enhancing the efficiency of the FL process.

Recent research has focused on personalization in FL, aiming to tailor global models to individual client preferences and characteristics. Various frameworks for learning personalized models by have been proposed [40]. Additionally, the challenge of non-iid (independent and identically distributed) data by has been addressed by developing strategies to ensure fair and effective learning across diverse datasets [41]. One approach to personalization is

meta-learning, where the global model is trained to quickly adapt to new tasks based on client-specific data. This approach enhances the relevance and performance of the model for individual users [40]. The challenge of non-iid data in FL has become easier to handle over the past few years by utilizing data-sharing strategies. It has been suggested that sharing a small subset of data among clients can align distributions and improve the convergence of the global model. This approach mitigates the negative impact of data heterogeneity on model performance [41].

With the growing emphasis on privacy regulations, there is an increasing need to explore FL frameworks that comply with stringent privacy constraints. Advances in federated learning with differential privacy and secure computation techniques are expected to continue, along with novel methods for accountability and auditability to ensure compliance with legal and ethical standards. In the following subsection, we will highlight some of the prominent attacks in the federated learning field.

3.2 Attacks in Federated Learning

FL has garnered significant interest due to its potential to train machine learning models across decentralized data sources while preserving data privacy. However, the decentralized and distributed nature of FL introduces new vulnerabilities to various types of attacks. This section reviews the related works on attacks in federated learning, including model poisoning attacks, data poisoning attacks, and backdoor attacks.

In FL, we generally encounter various types of attacks, with our focus primarily on two types known as poisoning (untargeted) attacks and backdoor (targeted) attacks. Poisoning attacks can be further categorized into model poisoning attacks and data poisoning attacks. The primary objective of a poisoning attack is to degrade global model performance by introducing malicious clients that provide corrupted/garbage updates, or damage the data that is used to train the model thus leading to a downturn in integrity which will corrupt the

global model [9]. Data poisoning attacks involve injecting malicious data into the training dataset to manipulate the final model. In the context of FL, this type of attack can be carried out by adversarial clients who contribute poisoned data during local training. The goal is to degrade the overall model performance or introduce specific backdoor behaviors. Adversarial clients can introduce backdoors into federated learning models. These backdoors allow the model to perform well on regular inputs but behave maliciously on inputs containing specific triggers. Studies show that even a small number of malicious clients could significantly impact the global model by carefully crafting their updates [29]. Federated learning has been analyzed through an adversarial lens and it has been highlighted with what ease an attacker can perform model poisoning. By sending malicious updates during the training process, attackers can degrade the model's accuracy or bias it towards specific outcomes [42]. Membership inference attacks aim to determine whether a specific data point was part of the training dataset. In federated learning, adversaries can leverage updates from multiple rounds of communication to infer the presence of particular data points.

Model poisoning attacks corrupt local model updates before they are sent to the server [13], [43]. Adversaries manipulate the model updates directly rather than poisoning the data. This can be achieved by modifying the local model parameters before sending them to the central server. There are various attacks that can be used to poison updates, including the sign-flipping attack and additive noise attack [8]. In the sign-flipping attack, malicious clients flip the sign of the model updates locally. With the additive noise attack, the attacking clients add random Gaussian noise their locally trained model. A data poisoning attack [44] in FL typically involves dirty label attacks instead of the clean label attack. A dirty label attack occurs when an adversary (usually a malicious client) introduces several mislabeled data samples with the intention of having them misclassified [7]. The idea is to have a target label attached to these data samples that does not match the actual classification. For example, assigning the label '2' to the image of '7' (thus flipping all the labels) in the MNIST dataset and forcing the model to be trained with that incorrect label. A clean label

attack occurs when input-label pair cannot be clearly recognized as mislabeled. Clean label attacks simply mean that the attacker has to be very subtle or careful with its changes, as it is assumed that in this case clients cannot change the label of any training data pieces. Backdoor attacks follow a similar concept to model poisoning and data poisoning. However, the goal is not to degrade model performance. In this scenario, attackers aim to achieve a high degree of accuracy for the global model while injecting our backdoor objective into the training [29]. That is, the local models are manipulated in such a way that the global model is compromised. The objective of the malicious entity is to cause the global model to behave in a certain way when a specific label or input is encountered.

It was outlined by Melis et al. [45] that federated learning models are vulnerable to property inference attacks. By analyzing the updates sent by clients, adversaries can infer properties of the local datasets, such as the distribution of certain attributes. This type of attack compromises the privacy guarantees of federated learning and necessitates stronger privacy-preserving techniques. Model inversion attacks in federated learning, where adversaries reconstruct training data by exploiting the shared model parameters demonstrate that even without direct access to the data, attackers could infer sensitive information from the model updates. The authors proposed cryptographic techniques, such as secure multiparty computation (SMPC), to enhance privacy protection [46].

Gradients shared during the federated learning process could leak sensitive information about the training data. By analyzing the gradients, adversaries can reconstruct input data with high accuracy. This highlights the need for secure aggregation methods that protect gradient information while enabling efficient model updates. Adversaries can exploit these shared gradients to recover private data, leading to gradient leakage attacks [47]. Byzantine attacks are a type of poisoning that involve malicious or faulty clients that behave arbitrarily, potentially disrupting the entire learning process. Federated learning systems must be resilient to such attacks to ensure robust model performance. Evasion attacks involve manipulating the input data to cause the model to produce incorrect predictions [48], which

necessitates them to be classified within data poisoning. In the context of federated learning, adversaries can craft adversarial examples that exploit vulnerabilities in the global model. It was demonstrated that federated learning models are vulnerable to adversarial examples. By crafting inputs that are specifically designed to deceive the model, attackers can cause incorrect predictions. The study proposed defense mechanisms, such as adversarial training and robust optimization, to enhance the resilience of federated learning models against evasion attacks [49].

3.3 Defense Techniques in Federated Learning

In FL, robust aggregation strategies are a common method used to defend against adversarial attacks [12], [50]. Existing works primarily focus on robust aggregation where algorithms such as KRUM, TrimmedMean, and GeoMed [12] attempt to find the centre of received model updates. The objective of the KRUM algorithm is to identify the model update that exhibits the least deviation from the other $K - M - 2$ updates, as determined by the squared Euclidean distance. In this context, K represents the total client count, and M is the number of malicious clients. Essentially, it seeks to pinpoint those model update vectors that are most similar to each other. The TrimmedMean method calculates the coordinate-wise trimmed average of the given model updates [12]. The Geometric Median (GeoMed) approach identifies a vector that minimizes the sum of its Euclidean distances to all update vectors in the system. Most previous robust aggregation defense strategies assume Independent and Identically Distributed (IID) data. While some of these strategies mitigate the damage caused by malicious clients, they cannot entirely eliminate them, since they do not differentiate between benign and malicious clients. In FL, the server cannot access the data of the clients. This causes such secure aggregation strategies to be less impactful since a single poisoned client can cause noticeable damage through poisoned updates or data.

In the realm of FL, the majority of existing defensive strategies have primarily focused on targeted attacks [8]. To deal with untargeted attacks such as sign-flipping or additive noise

attacks, prevention-based techniques need to be considered. Defending targeted attacks include estimating the likelihood of a client update being malicious [51], detecting backdoor attacks [52], and assessing the impact of local updates on the global model [53]. Researchers have effectively devised detection-based approaches for handling anomalies in 2-layer FL systems [8]. In [8], they propose a detection-based approach that identifies abnormalities in the model updates based on their low-dimensional embeddings. Reconstruction errors from embedded data can be used to identify differences between benign and malicious model updates. This is often referred to as spectral anomaly detection. A special neural network, known as an autoencoder is employed to create such low-level embeddings of model updates in FL. Then, the reconstruction error is calculated to discern whether an update originates from a malicious or benign client.

Secure aggregation protocols aim to protect the privacy and integrity of the model updates contributed by the clients. They ensure that the server can only access aggregated updates without learning individual client updates. Bonawitz et al. [21] proposed a practical secure aggregation protocol for privacy-preserving machine learning. Their approach uses secure multi-party computation (SMPC) to aggregate client updates securely, ensuring that the server cannot learn individual updates. This protocol significantly enhances privacy without compromising the efficiency of the aggregation process. Byzantine-resilient aggregation methods are designed to mitigate the influence of malicious or faulty clients that send arbitrary or incorrect updates.

Differential Privacy (DP) is a widely adopted technique to protect the privacy of individual data points. In federated learning, DP ensures that the updates from clients do not reveal sensitive information about their local datasets. Differential privacy has been extended to the federated learning context by introducing noise to the updates before sending them to the server. This approach provides strong privacy guarantees while maintaining the utility of the federated learning model. The added noise prevents inference attacks that could otherwise extract sensitive information from the updates [10], [31]. A proposed hybrid approach

combines local and global differential privacy balances the trade-off between privacy and model performance, ensuring robust protection against inference attacks while maintaining the accuracy of the federated learning model [32].

Homomorphic encryption (HE) allows computations to be performed on encrypted data, providing strong privacy guarantees in federated learning. It was previously proposed by Phong et al. [33] to additively apply homomorphic encryption to federated learning, which enables secure computations on encrypted model updates. This approach ensures that the server operates on encrypted data, preserving the confidentiality of client updates while allowing accurate aggregation. Homomorphic encryption protects against inference attacks and unauthorized access to client data.

Adversarial training enhances the robustness of machine learning models against adversarial attacks by incorporating adversarial examples into the training process. It was previously proposed to incorporate adversarial training into federated learning to defend against evasion attacks. This approach involves generating adversarial examples during local training and using them to update the global model. Adversarial training improves the model's resilience to adversarial inputs and enhances overall robustness [49]. Bhagoji et al. [42] analyzed federated learning through an adversarial lens and proposed defense mechanisms such as anomaly detection and robust aggregation. These methods aim to detect and mitigate the impact of poisoned updates, preserving the accuracy and reliability of the global model.

Poisoning attacks involve injecting malicious data or updates to manipulate the global model. Effective defenses are crucial to maintaining the integrity of federated learning. The vulnerability of federated learning to backdoor attacks has been highlighted many times, where adversaries introduce specific triggers to manipulate the model's behavior. It has been proposed to use anomaly detection techniques to identify and exclude malicious updates, ensuring the integrity of the global model [54]. In contrast, Byzantine fault tolerance is another critical mechanism used within federated learning to ensure robustness against mali-

cious or faulty clients that behave arbitrarily. FABA (Filtering-based Byzantine Aggregation or Fast Aggregation against Byzantine Attacks) has been proposed [55], which is an algorithm that uses filtering techniques to exclude potential Byzantine clients. FABA enhances the robustness of federated learning by filtering out clients that deviate significantly from the majority (outlier removal), ensuring that the global model is not adversely affected by malicious updates. Mhamdi et al. [56] introduced Bulyan, a robust aggregation rule that combines multiple robust estimators to improve resilience against Byzantine clients. Bulyan first applies a robust mean estimator to filter out outliers and then uses a second round of aggregation to enhance robustness further. This approach ensures high accuracy and resilience against Byzantine attacks.

As federated learning evolves, there is a growing need for personalized defense mechanisms tailored to the specific requirements and vulnerabilities of individual clients. Future research is expected to focus on developing adaptive and client-specific defenses to enhance the security and privacy of federated learning systems.

Chapter 4

4 Multi-Layer FL Anomaly Detection

In this chapter, we will introduce the model for our research, along with the approaches for anomaly detection, and the algorithms for malicious client removal. Additionally, we will present experimental results and compare those with previous methods. We are able to show that our approach creates a noticeable advantage compared to previous removal strategies that were only directly applicable in 2-layer systems. Section 4.1 highlights the system model for our research. Section 4.2 shows how we utilize a VAE for anomaly detection in the multi-layer FL scenario. Section 4.3 explains the strategy for malicious client removal following 2 distinct algorithms. Finally, Section 4.4 displays our exerpiemental procedure and results.

4.1 System Model

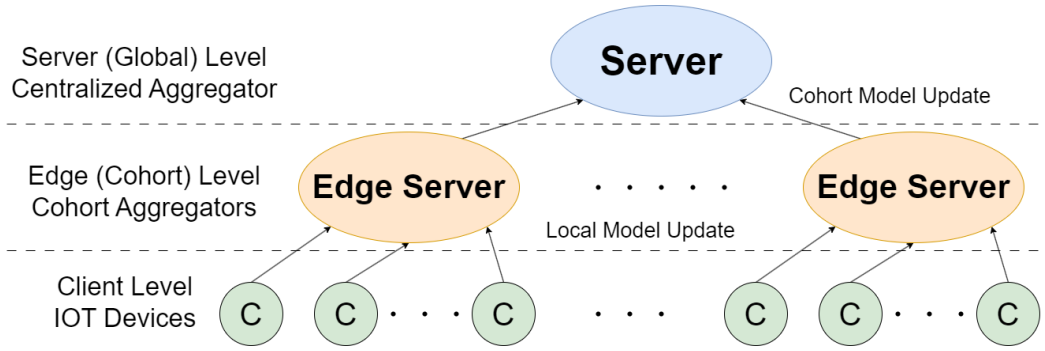

Figure 3: Overview of System Model Architecture

Figure 3 shows the system model used in our research. The clients that wish to participate in training are randomly shuffled among cohorts during each communication round, so that there is a fair chance of a particular client appearing in any of the cohorts. The server then selects a subset of the cohorts for training. This allows easy implementation of different client ratios without granting the server direct access to any of the clients. Before local

training begins, an initial global model is shared with each participating client. This ensures consistency across clients. Each participating client performs local training using stochastic gradient descent (SGD). After local training, the clients share their updated local models with the cohort to which they belong which they belong. Cohort aggregation takes place when the clients belonging to a cohort send their weight updates (i.e., updated local models) to the edge server at the cohort level. The widely adopted FedAvg algorithm [3] is used at the cohort level to aggregate the client updates. The FedAvg algorithm updates the cohort-level model by computing a federated average of each of the client-level updates that are shared with the cohort. Following this, each edge server at the cohort level submits its aggregated model to the server. Therefore, the server receives the updates from the edge servers at the cohort level, not directly from individual clients. Upon receiving all the updates from the edge servers, the server employs the FedAvg algorithm to aggregate these cohort updates. Upon completion, the final aggregated global model is then shared with all clients participating in training through the edge servers.

The multi-layer FL system is vulnerable to attacks. Adversarial clients participating in training may intentionally send poisoned model updates to disrupt global model aggregation. As a result, robust mechanisms must be deployed to detect and mitigate such attacks.

4.2 VAE Based Anomaly Detection

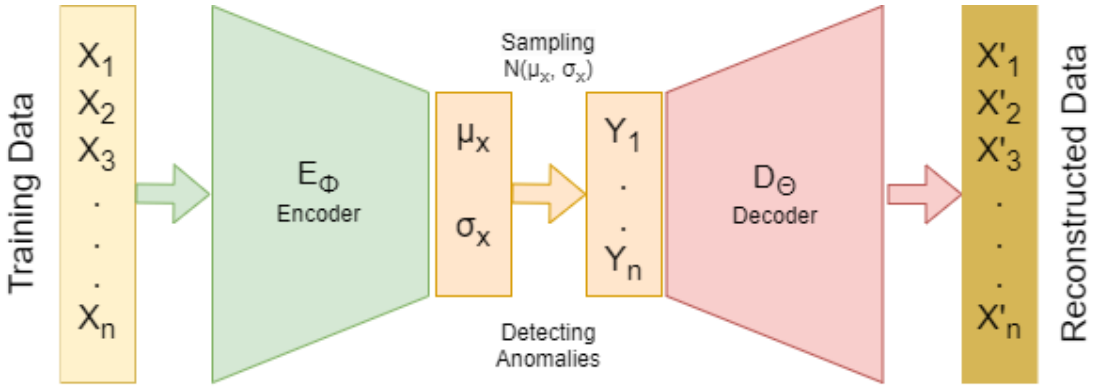

Figure 4: VAE Structure for Anomaly Detection

Before aggregation is performed at the global level, robust defense mechanisms are employed to identify and eliminate malicious clients. This proactive approach minimizes the impact of adversarial interference on the global model. In the following, we introduce our anomaly detection and attack mitigation methods for multi-layer FL systems.

4.2.1 VAE Training

Before global-level aggregation, we deploy a server-side Variational Autoencoder (VAE) to detect malicious clients within cohorts. Upon detection, we employ additional removal methods. For anomaly detection, we train the VAE using a dataset of aggregated model updates from edge servers at the cohort level. Initially, we collect this data by running our multi-layer Federated Learning (FL) system without any malicious clients. The VAE training algorithm, outlined in Algorithm 1, utilizes the collected dataset. Key components include the encoder and decoder, crucial for effective anomaly detection, as shown in Fig. 4.

Algorithm 1 VAE Training Procedure

Input: Dataset of the FL model updates collected at the cohort level

Output: Optimized ϕ, θ for E_ϕ and D_θ

$\phi, \theta \leftarrow$Parameter initialization

repeat

randomly get one sample $\mathbf{X}$ from the input dataset

$(\mu_x, \sigma_x) \leftarrow e_\phi(\mathbf{Y}|\mathbf{X})$

$\mathbf{Y} \sim \mathcal{N}(\mu_x, \sigma_x)$

$\mathbf{X'} \leftarrow d_\theta(\mathbf{X'}|\mathbf{Y})$

$loss = r_e + D_{KL}(\mathcal{N}(\mu_x, \sigma_x)||\mathcal{N}(0, 1))$

$\phi, \theta \leftarrow$update using SGD to minimize the loss

until E_ϕ and D_θ have converged parameters

The purpose of the encoder, denoted as $E_\phi = e_\phi(\mathbf{Y}|\mathbf{X})$ is to transform the input data

$\mathbf{X}$ into a latent vector $\mathbf{Y}$, i.e. the encoder first outputs the mean μ_x and variance σ_x, then the latent vector $\mathbf{Y}$ is generated by sampling from a Gaussian distribution $\mathcal{N}(\mu_x, \sigma_x)$. The decoder, denoted as $D_\theta = d_\theta(\mathbf{X'}|\mathbf{Y})$ will reconstruct the input data $\mathbf{X}$ based on the latent vector $\mathbf{Y}$, denoted as $\mathbf{X'}$. The reconstruction error defined in Eq. (7) refers to the difference between $\mathbf{X}$ and $\mathbf{X'}$. Eq. (8) defines the total loss from the VAE, where r_e represents the reconstruction error, and the D_{KL} represents the Kullback-Leibler divergence between the Gaussian distribution defined by μ_x, σ_x and the standard normal distribution.

$$r_e = MSE(\mathbf{X}, \mathbf{X'}) = ||\mathbf{X} - \mathbf{X'}||^2 \tag{7}$$

$$loss = r_e + D_{KL}(\mathcal{N}(\mu_x, \sigma_x)||\mathcal{N}(0, 1)) \tag{8}$$

The VAE is trained with the goal of minimizing the loss defined in Eq. (8). Algorithm 1 will optimize the parameters of the encoder and decoder subroutines.

4.2.2 Anomaly Detection

In order to detect which cohorts contain malicious clients, we leverage our previously trained VAE. As depicted in Fig. 4, the input vector $\mathbf{X}$, which is collected from the real-scenario edge servers at the cohort level (where anomalies may be present), is fed into the encoder. The encoder will embed or compact the information within $\mathbf{X}$ and output a new vector $\mathbf{Y}$. Then, the decoder takes $\mathbf{Y}$ as input and outputs the reconstruction vector $\mathbf{X'}$. We then use the reconstruction errors of $\mathbf{X'}$ to detect which cohorts contain malicious clients. Algorithm 2 shows this process in detail.

Algorithm 2 Anomaly Detection Algorithm

Input: Test dataset

Output: Detection List *det*

for each sample **X** in the Test dataset **do**

$(\mu_x, \sigma_x) \leftarrow e_\phi(\mathbf{Y}|\mathbf{X})$

$\mathbf{Y} \sim \mathcal{N}(\mu_x, \sigma_x)$

$\mathbf{X'} \leftarrow d_\theta(\mathbf{X'}|\mathbf{Y})$

$loss[\mathbf{X}\_index] = r_e + D_{KL}(\mathcal{N}(\mu_x, \sigma_x)||\mathcal{N}$

▷ $\mathbf{X}\_index$ denotes the index of **X** in the Test dataset.

Append $loss[\mathbf{X}\_index]$ to $loss\_list$

if $\sigma(loss\_list) > \sigma_t$ and $D' >> D$ **then**

for each **X** in the test dataset **do**

if $loss[\mathbf{X}\_index] > \mu_{loss\_list}$ **then**

▷ μ_{loss_list} is the average value in the loss list

Cohort contains a malicious client

Append True to *det*

else

Cohort does not contain a malicious client

Append False to *det*

end if

end for

else

Anomaly detection is skipped

end if

end for

The criterion for detecting if a cohort contains an adversarial client is to check if its loss is greater than the average loss. Additionally, a standard deviation check is applied.

We first compute the standard deviation of the list of loss values returned from the VAE denoted as $\sigma(loss\_list)$. The removal strategies are only activated if this value is greater than a pre-determined value σ_t to prevent unnecessary removal of benign clients. In other words, if the standard deviation is large, presume the presence of malicious users, and if it is small, we assume there are no malicious cohorts. Generally, σ_t is decided upon inspection of the distribution of reconstruction error values, which may change based on system model settings and dataset. This check is effective because the reconstruction errors from anomalous data reconstruction will be different from normal data in the distribution. We compare the spread of the reconstruction errors in the normal (benign) case with the anomalous case. When running the system without malicious clients, we calculate the distance between the mean and the furthest outlier reconstruction error, denoted as D. Then, in the anomaly detection case, we compute D'. If $D' >> D$, we conclude that anomalous data is present, as the reconstruction error distribution significantly deviates from the benign scenario. These additional checks are deemed necessary, otherwise the VAE will still detect attackers even if little or none are present. Based on these detection results from the VAE, we further propose methods for malicious client removal, as explained in the following subsection.

4.3 Malicious Client Removal Strategy

In our work, we have designed two distinct approaches for removal of attacking clients, namely the scoring method and Bayesian method, taking into account varying levels of system transparency. The scoring approach is used when limited information about the system is available. It relies on essential information such as the number of clients per cohort and the total number of clients. The scoring method primarily utilizes the detections from the VAE to employ its removal method, whereas the Bayesian method comes into play when we have more comprehensive information about the system. The additional information includes the attacker ratio and the statistics of the VAE detection performance.

4.3.1 Scoring Method

Algorithm 3 displays how the scoring method removes malicious clients, which is run at each communication round of FL. A score is kept for each client, and the detection decision for each cohort is used to update the client scores. If the VAE detects a cohort containing an adversarial client, we add a score of 1 to each client within that cohort. On the other hand, if the VAE determines that a cohort does not contain an attacker, we subtract a score of 1 from each client within that cohort. The scoring threshold t_s is essentially how many times a client can appear in a malicious cohort in a row before being labelled as an attacker and removed from training. After the threshold t_s is reached for a specific client, we exclude that client from training. For ease of implementation, we simply replace an accused client with a new client. In Algorithm 3, *det* is the list of detection results corresponding to each cohort, which is obtained from Algorithm 2. s_c is the current list of client scores (initially 0), c_c is the current client list, c_u is the updated client list, and s_u is the updated list of client scores.

Algorithm 3 Scoring Method

Input: det, c_c, s_c

Output: c_u, s_u

for each cohort in det **do** ▷ Client score update

  if the detection result is True **then**

    for each client within this cohort **do**

      $s_c[\text{client\_index}] \leftarrow s_c[\text{client\_index}] + 1$

    end for

  else

    for each client within this cohort **do**

      $s_c[\text{client\_index}] \leftarrow s_c[\text{client\_index}] - 1$

    end for

  end if

end for

$s_u \leftarrow s_c$

for each client c in c_c **do** ▷ Malicious client removal

  if $s_u[c] \geq t_s$ **then**

    $c_c \leftarrow c_c \setminus c$

  end if

end for

$c_u \leftarrow c_c$

In the worst-case scenario, the attackers are spread out among the cohorts. That is, no cohort contains more than one malicious client. The probability of an honest client being clumped with an attacker in the same cohort can be expressed as $p(c) = \frac{a_c}{t_c}$, where a_c is the number of cohorts containing an attacker, and t_c is the total number of cohorts.

The threshold t_s determines how many consecutive times a client should appear within a

malicious cohort before we identify that client as an attacker. Striking the right balance is crucial: we want to avoid accidentally removing honest clients while effectively identifying attackers. Therefore, the threshold must be set so that if the score reaches that value, there is a high chance of the client being an attacker. To set the threshold, we first define p_m as the maximum probability we allow for an honest client to be mistakenly labelled as an attacker. For instance, if we set $p_m = 1\%$, our proposed scoring method ensures that the probability of an honest client being misclassified as an attacker does not exceed 1%. The equation $p_m = p(c)^{t_s}$ shows the representation between p_m and t_s. Consequently, the threshold t_s is set as $t_s = \left\lceil \frac{\ln(p_m)}{\ln(p(c))} \right\rceil$, which is formulated. This scoring method, guided by the threshold, aims to strike a balance between efficiently identifying attackers and minimizing the risk of false positives for honest clients. As an example, we would set the threshold to 10 in a the scenario where we have 2 clients per cohort, with 100 total clients and a 30% attacker ratio. If this is the case, there will be less than a 1% chance an honest client has been with an attacker 10 rounds consecutively.

4.3.2 Bayesian Method

To detect individual malicious clients, the Bayesian method utilizes the statistical performance of the VAE, i.e., the precision $P = \frac{TP}{TP+FP}$ and specificity $S = \frac{TN}{TN+FP}$, which are used to gauge how likely a client is malicious or honest, respectively. Here, TN is the number of true negatives (correct honest cohort detection), TP is the number of true positives (correct malicious cohort detection), and FP is the number of incorrect positives (incorrect malicious cohort detection).

After an initial training phase (t_b epochs), we deploy the Bayesian method to start removing malicious clients. Algorithm 4 outlines the elimination process, which is run at each communication round of FL. The primary work of the Bayesian method is to calculate the *ratio* value. This *ratio* compares the probability of a client being an attacker $p(a|h)$ to the probability of a client being benign $p(b|h)$, where h refers to the client's history detection

sequence. In order to formulate the *ratio*, we apply the Bayesian method to find $p(a|h)$ and $p(b|h)$, as expressed in Eqs. (9) and (10). This *ratio* serves as a comparative measure of the likelihood that a client is either malicious or benign Therefore, we apply it as the decision criterion for client removal.

$$p(a|h) = \frac{p(h|a)p(a)}{p(h)} = \frac{p(h|a)r_a}{p(h)} \tag{9}$$

$$p(b|h) = \frac{p(h|b)p(b)}{p(h)} = \frac{p(h|b)(1-r_a)}{p(h)} \tag{10}$$

To derive $p(a|h)$ and $p(b|h)$, we start by finding $p(h|a)$ and $p(h|b)$, as we know $p(a) = r_a$ (attacker ratio) and $p(b) = 1 - r_a$. The probability $p(h)$ is the denominator of $p(a|h)$ and $p(b|h)$, so it is removed from the calculation of *ratio*. To properly derive $p(h|a)$ and $p(h|b)$, we first set the initial value of these two probabilities as 1, then update them at each epoch (i.e., communication round) by multiplying the precision or specificity of the VAE to the current value of $p(h|a)$ or $p(h|b)$ if the detection at the epoch was True or False, respectively. t_r is the threshold, which we compare to the calculated ratio to make a decision to remove a client. It is chosen to balance the maximal amount of malicious clients removed with the minimum amount of honest clients removed.

Algorithm 4 Bayesian Method

Input: det, c_c, $p(h|a)$, $p(h|b)$

Output: c_u

if epoch $> t_b$ **then** ▷ Malicious client removal

for each cohort in det **do**

if the detection result is True **then**

for each client c within this cohort **do**

$p(h|a)$[c_index]$\leftarrow p(h|a)$[c_index]$*P$

$p(h|b)$[c_index]$\leftarrow p(h|b)$[c_index]$*(1-S)$

end for

else

for each client c within this cohort **do**

$p(h|a)$[c_index]$\leftarrow p(h|a)$[c_index]$*(1-P)$

$p(h|b)$[c_index]$\leftarrow p(h|b)$[c_index]$*S$

end for

end if

end for

for each client c in c_c **do**

$ratio[c] = \frac{p(a|h)[c]}{p(b|h)[c]}$

if $ratio[c] > t_r$ **then**

$c_c \leftarrow c_c \setminus c$

end if

end for

$c_u \leftarrow c_c$

end if

4.4 Experimental Results

In our experiments, we consider various attack scenarios and system model settings. We evaluate the effectiveness of a VAE on our detection and removal strategies. We compare our results to benchmarked alternatives from prior studies to exemplify when our approach is well-used. The experiments included in this section are implemented in PyTorch.

4.4.1 Experiment Setup

We consider a 3-layer FL scenario, as shown in Fig. 3. In our system, we have 100 clients available for experiment purposes. The number of clients per cohort can be adjusted. Prior to training, we create cohorts for clients to join. In each round, we randomly assign each client to a cohort. In our experiments, we have opted to consider various cohort sizes: 2, 4, 6, and a variable size, denoted as 'V'. The variable cohort size is randomly chosen between 2 and 6 inclusively. The attacker ratio (i.e., the ratio of the number of attackers to the total number of clients) is set as 30%.

For the dataset used in our experiments, we have chosen MNIST because of its wide use. The data from MNIST is sorted based on the digit labels. The training portion of the dataset is split into 200 shards, each containing 300 data samples. In the experiment, we consider model poisoning attacks where the adversary's objective is to degrade model performance as significantly as possible. We explore two types of poisoning attacks. The first poisoning attack we have considered is the sign-flipping attack. The first one is the sign-flipping attack. In this attack, each adversarial client participating in training deliberately flips the sign of the update they intend to share with the edge server at the cohort level. The second one is the additive noise attack. In this attack, each adversarial client adds a pre-determined amount of Gaussian noise to their update before sharing it with the edge server at the cohort level. For this additive noise attack, we have tested with 30% noise.

For each result graphed, we compare our proposed methods with several benchmark meth-

ods. First is the 'Old Method', which refers to the method used in [8]. Note that this method was originally designed for the 2-layer FL setting, and is not directly applicable to multi-level FL. As our proposed method is the first to address attack mitigation in multi-layer FL systems, this is the most similar work we could find for comparison, which also employs a VAE for anomaly detection. To adapt this method for multi-layer FL, we follow the following process: when a cohort is detected as malicious, the entire cohort will be excluded from the global model aggregation to mitigate the potential impact of the malicious clients within that cohort. However, due to the lack of information on which specific client is malicious within a cohort, there is no mechanism to remove individual clients. Consequently, a malicious client may rejoin the training in the next round. Compared to this previous work, we include the criterion of standard deviation and distance checks for VAE detection to minimize false positives. Second is the 'No Attack' scenario, in which the system has no attacking clients. This serves as an upper-limit benchmark, representing optimal performance. Third is the 'No Defense' scenario. This represents the case where no defense mechanisms are applied, and malicious clients are free to poison the system. This can be considered as the lower limit of system performance.

4.4.2 Experiment Results

Fig. 5 displays the accuracy of the FL model across different experiment settings. In every instance, our methods either match or come remarkably close to the accuracy observed in the no-attack scenario, outperforming the previous approach. Regarding the backdoor attack, one may notice some chaos in the results especially for the no defense scenario. This is expected, as having several clients per cohort affects which cohorts are influencing the backdoor objective.

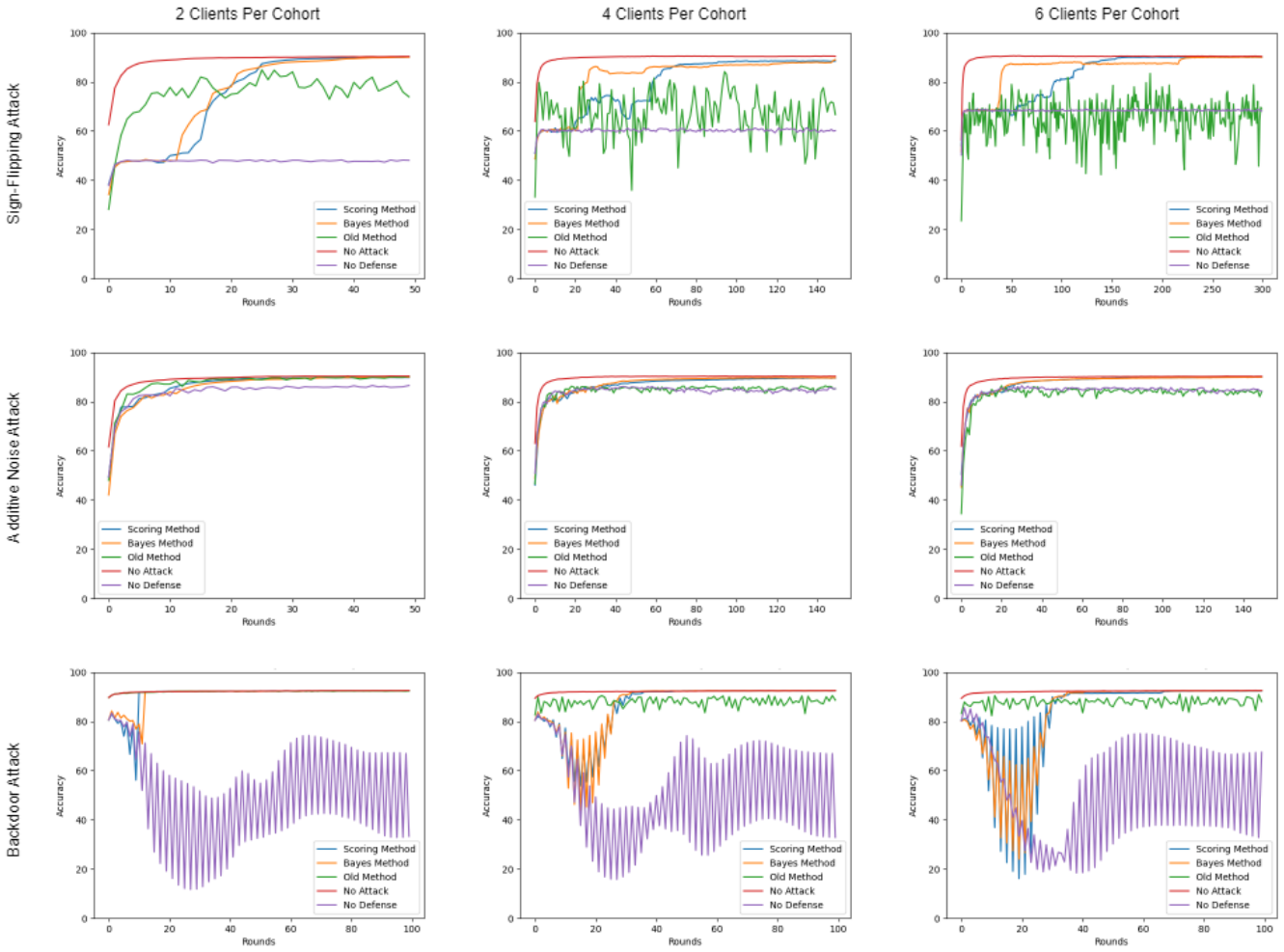

Figure 5: Results under different attacks and cohort sizes. The first and second rows correspond to the sign-flipping attack and the additive noise attack, respectively. The columns refer to the amount of clients in each cohort, as labeled.

In addition to the fixed cohort size, we also studied a variable cohort sized case. Fig. 6 shows the model accuracy for that scenario. In these graphs, we also notice that both of our removal methods are able to meet the no-attack scenario in terms of model accuracy, indicating that our proposed methods can effectively mitigate the impact of adversarial attacks.

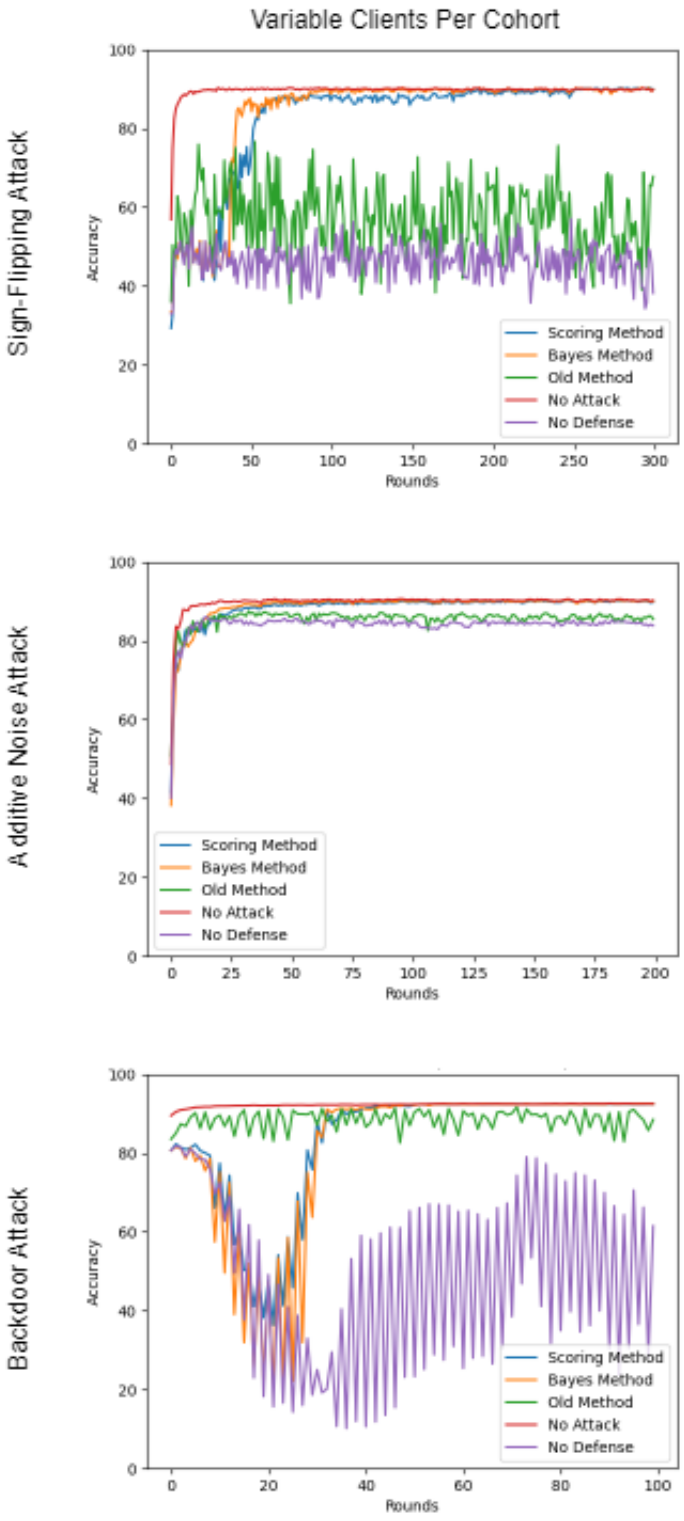

Figure 6: Results with variable size cohorts.

Table 1: Percentage of attackers removed for various scenarios.

Attack Type/Clients within Cohort	2	4	6	V
Sign Attack	97%	93%	90%	90%
	100%	97%	93%	93%
Noise Attack	100%	100%	93%	93%
	100%	100%	93%	93%
Backdoor Attack	100%	97%	87%	93%
	100%	100%	90%	93%

Table I illustrates the effectiveness of our solutions in eliminating attackers, with the tabulated values representing the proportion of attackers successfully eliminated. For each attack, there are two rows, which correspond to the scoring (top) and Bayesian (bottom) methods. It is observed that the Bayesian method outperforms the scoring method in many cases. This superior performance is attributed to its utilization of additional information about the VAE and system variables, enabling it to formulate a more robust and targeted defense.

From the process detailed in this chapter, along with the results that have been displayed, we can see how trans-formative new strategies for anomaly detection in multi-layer systems are required. To have an impact on malicious clients and to stop them from ruining training, full-blown client removal algorithms have been developed, rather than just excluding them from training during each epoch. Looking at our results in detail, we can see a definite improvement when compared to old methods or when no defense is provided. Even when comparing to the scenario where no malicious clients are present in the system, our approach allows us to get to that point after sufficient time.

Chapter 5

5 Conclusion

5.1 Summary of Thesis

In this work, we have proposed novel methods for detecting and removing malicious actors in the MLFL setting. This thesis introduces 2 new methods for detecting and removing malicious clients. Our results demonstrate that adversaries can poison models which disrupts the process of FL in the multi-layer scenario. The choice of method depends on what information is available. Utilizing 2 algorithms with different approaches for detection allows more flexibility in the application of our method. In our research, a VAE is trained on benign data and deployed to detect which cohorts contain malicious clients, and then the proposed malicious client exclusion strategies, i.e., the scoring or Bayesian methods, are used to accurately identify and remove individual malicious clients. Removal of the malicious clients improves the FL model performance. We have thoroughly investigated the necessity of defense strategies that surpass existing approaches. Both of the demonstrated approaches effectively mitigate the harm that would otherwise be caused from open training with malicious users. A variety of experiments have been conducted, and numerical results have been presented. We have analyzed various types of attacks including sign-flipping, additive noise, and backdoor attacks. The results show that our methods are more effective (and in many cases fully successful) at removing malicious users when compared to previous approaches in mitigating the detrimental effects of malicious attacks in MLFL.

5.2 Future Works

Following on from this work, immediate developments should include creating new AEs for other datasets and to evaluate the performance of the proposed scoring and Bayesian methods perform in those scenarios. Some adjustments may be necessary to optimize these methods for various scenarios. Additional work on this topic may include the exploration of systems with more than three layers. As discussed by De Rango et al. [1], the MLFL can include many layers (not just 3). Therefore, adapting the methods proposed here for hierarchical MLFL would be useful as it is unlikely other defense mechanisms can be easily adapted for malicious client removal. With additional experimentation, it might be possible to leverage more recent advancements to expedite the removal of malicious clients. Finally, we observed that in some scenarios our algorithms may take a while to successfully remove most or all malicious clients. Utilizing optimization based techniques could be beneficial in high-stakes settings where swift removal of malicious users is crucial, minimizing the impact on honest users.

Bibliography

[1] F. D. Rango, A. Guerrieri, P. Raimondo, and G. Spezzano, "A novel edge-based multi-layer hierarchical architecture for federated learning," in *Proceedings of IEEE Intl Conf on Dependable, Autonomic and Secure Computing*, 2021.

[2] J. C. Jiang, B. Kantarci, S. Oktug, and T. Soyata, "Federated learning in smart city sensing: Challenges and opportunities," *Sensors*, vol. 20, no. 21, 2020.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2016.

[4] S. Bharati, M. R. H. Mondal, P. Podder, and V. S. Prasath, "Federated learning: Applications, challenges and future scopes," *International Journal of Hybrid Intelligent Systems*, vol. 10, no. 10, pp. 1–17, 2022.

[5] Z. Liu, S. Duan, S. Wang, Y. Liu, and X. Li, "Mflces: Multi-level federated edge learning algorithm based on client and edge server selection," *Electronics*, vol. 12, no. 12, p. 2689, 2023.

[6] O. Aramoon, P.-Y. Chen, G. Qu, and Y. Tian, "Meta federated learning," *Federated Learning: Theory and Practice*, pp. 161–179, 2024.

[7] N. Bouacida and P. Mohapatra, "Vulnerabilities in federated learning," *IEEE Access*, vol. 9, pp. 63229–63249, 2021.

[8] S. Li, Y. Cheng, Y. Liu, W. Wang, and T. Chen, "Abnormal client behavior detection in federated learning," in *Proceedings of FL-NeurIPS'19*, 2019.

[9] L. Muñoz-González, A. D. B. Biggio, A. Paudice, V. Wongrassamee, and F. R. E. C. Lupu, "Towards poisoning of deep learning algorithms with back-gradient op-

timization," in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 2017.

[10] Y. Liang, Y. Li, Y. Zhou, and X. Zheng, "Sybil attacks and defense on differential privacy based federated learning," in *Proceedings of IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2021.

[11] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[12] S. Li, E. C.-H. Ngai, and T. Voigt, "An experimental study of byzantine-robust aggregation schemes in federated learning," *IEEE Transactions on Big Data*, vol. 10, no. 6, pp. 975–988, 2024.

[13] Z. Wang, Q. Kang, X. Zhang, and Q. Hu, "Defense strategies toward model poisoning attacks in federated learning: A survey," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, 2022.

[14] S. Badillo, B. Banfai, F. Birzele, I. Davydov, L. Hutchinson, T. Kam-Thong, J. Siebourg-Polster, B. Steiert, and J. Zhang, "An introduction to machine learning," *Clinical Pharmacology & Therapeutics*, pp. 871–885, 2020.

[15] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, pp. 1–19, 2019.

[16] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 46–51, 2020.

[17] K. C. Yilun Jin, Yang Liu and Q. Yang, "Federated learning without full labels: A survey," *IEEE Data Engineering Bulletin*, vol. 46, no. 1, pp. 27–51, 2023.

[18] L. Sweeney, "Simple Demographics Often Identify People Uniquely," *Health*, vol. 671, 1 2000.

[19] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, p. 84–90, feb 1981.

[20] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, pp. 513–535, 2023.

[21] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," in *Proceedings of the NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[22] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[23] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konecny, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *Proceedings of ICLR*, 2021.

[24] X. Lian, C. Zhang, C.-J. Hsieh, H. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[25] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawit, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar,

M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. Theertha Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, *Advances and Open Problems in Federated Learning*, vol. 14. Now Foundations and Trends, 2021.

[26] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling, "Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19, AAAI Press, 2019.

[27] R. Guerraoui, N. Gupta, and R. Pinot, "Byzantine machine learning: A primer," *ACM Computing Surveys*, vol. 56, apr 2024.

[28] C. Wang, J. Chen, Y. Yang, X. Ma, and J. Liu, "Poisoning attacks and countermeasures in intelligent networks: Status quo and prospects," *Digital Communications and Networks*, vol. 8, no. 2, pp. 225–234, 2022.

[29] T. D. Nguyen, T. Nguyen, P. L. Nguyen, H. H. Pham, K. D. Doan, and K.-S. Wong, "Backdoor attacks and defenses in federated learning: Survey, challenges and future research directions," *Engineering Applications of Artificial Intelligence*, vol. 127, 2024.

[30] M. Abadi, A. Chu, I. Goodfellow, H. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 308–318, 2016.

[31] S. Baraheem and Z. Yao, "A survey on differential privacy with machine learning and future outlook," *arXiv preprint arXiv:2211.10708*, 2022.

[32] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, W. Wei, and L. Wu, "A hybrid approach to privacy-preserving federated learning.," in *Proceedings of the 12th ACM Workshop*

on Artificial Intelligence and Security (AISec), pp. 1–11, 2019.

[33] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[34] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Proceedings of Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[35] L. Cui, X. Su, Y. Zhou, and L. Zhang, "Clustergrad: Adaptive gradient compression by clustering in federated learning," in *Proceedings of IEEE Global Communications Conference*, pp. 1–7, 2020.

[36] Z. Xu, Y. Zhang, G. Andrew, C. Choquette, P. Kairouz, B. McMahan, J. Rosenstock, and Y. Zhang, "Federated learning of gboard language models with differential privacy," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, vol. 5, 2023.

[37] M. Sheller, B. Edwards, G. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. Colen, and S. Bakas, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, 2020.

[38] N. Bouacida, J. Hou, H. Zang, and X. Liu, "Adaptive federated dropout: Improving communication efficiency and generalization for federated learning," in *Proceedings of IEEE Conference on Computer Communications Workshops*, pp. 1–6, 2021.

[39] G. Cheng, Z. Charles, Z. Garrett, and K. Rush, "Does federated dropout actually work?," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3386–3394, 2022.

[40] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 9587–9603,

2023.

[41] Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang, "Federated learning with non-iid data: A survey," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 19188–19209, 2024.

[42] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, pp. 634–643, PMLR, 09–15 Jun 2019.

[43] X. Zhou, M. Xu, Y. Wu, and N. Zheng, "Deep model poisoning attack on federated learning," *Future Internet*, vol. 13, no. 3, 2021.

[44] R. Doku and D. B. Rawat, "Mitigating data poisoning attacks on a federated learning-edge computing network," in *Proceedings of IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–6, 2021.

[45] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," *IEEE Symposium on Security and Privacy (SP)*, pp. 691–706, 2019.

[46] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: Information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, p. 603–618, 2017.

[47] H. Yang, M. Ge, D. Xue, K. Xiang, H. Li, and R. Lu, "Gradient leakage attacks in federated learning: Research frontiers, taxonomy, and future directions," *IEEE Network*, vol. 38, no. 2, pp. 247–254, 2024.

[48] P. Liu, X. Xu, and W. Wang, "Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives," *Cybersecurity*, 2022.

[49] C. Xie, O. Koyejo, and I. Gupta, "Zeno: Byzantine-suspicious stochastic gradient descent," in *Proceedings of the 36th International Conference on Machine Learning*

(ICML), pp. 6893–6901, 2019.

[50] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1142–1154, 2022.

[51] K. Purohit, S. Das, S. Bhattacharya, and S. Rana, "Learndefend: Learning to defend against targeted model-poisoning attacks on federated learning," *arXiv preprint arXiv:2305.02022*, 2023.

[52] S. Shen, S. Tople, and P. Saxena, "Auror: defending against poisoning attacks in collaborative deep learning systems," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016.

[53] F. Fang, X. Y. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020.

[54] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, vol. 108, pp. 2938–2948, 26–28 Aug 2020.

[55] Q. Xia, Z. Tao, Z. Hao, and Q. Li, "Faba: An algorithm for fast aggregation against byzantine attacks in distributed neural networks," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4824–4830, 7 2019.

[56] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The Hidden Vulnerability of Distributed Learning in Byzantium," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 3518–3527, PMLR, 2018.