Distributed Detection and Mitigation of Attacks in Three-tier Federated Learning Systems

***Abstract*—In this paper, we study federated learning in a hierarchical architecture consisting of three tiers, namely, cloud-based central servers at the top tier, edge servers at the middle tier, and end devices at the bottom tier. We propose a distributed framework to efficiently detect/mitigate the poisoning attacks that end devices may launch, through distributed collaborations between the central and edge servers, and to detect attacks that may be launched by compromised middle-tier edge servers, through collaborations between the central server and end devices. Extensive experiments have been conducted, and the results show that our proposed design can effectively detect/mitigate clients' attacks and deliver a level of model training accuracy comparable to or even better than a state-of-the-art centralized scheme, and it incurs moderate overheads for detecting misbehaving edge servers.**

Keywords: Federated Learning, Edge-Cloud Computing, Poisoning Attacks.

I. Introduction

Deep learning has revolutionized the processing of image, audio, video and natural language and many other application domains. Though deep learning models are more convenient to be trained at powerful computation platform with centrally collected massive datasets, in practice data are often generated and distributed at end devices such as sensors and smartphones. Gathering the data to central servers could violate the increasing privacy concerns or even be prohibited by the privacy-protecting laws [1]–[3]. To address the problem, federated learning (FL) [4]–[6] has been proposed as a distributed deep learning paradigm, where end devices download a global model from a central server, and perform local training before upload their updates to the model weights (i.e., gradients) to the server for model aggregation; the process repeats until a global model with a desired accuracy is obtained by the server. The feasibility of FL has been demonstrated in many real-world implementations [7]–[9].

Initial study of FL assumes a two-tier hierarchical architecture, i.e., central servers at the top tier and end devices at the bottom tier. Here, a central server can be either a cloud server residing deeply in the Internet or an edge server located at the edge of the Internet. A cloud server can reach a large base of end devices and thus can train deep learning models based on massive datasets; however, the communication between the server and the end devices can be slow and unpredictable due to the long, dynamic network connections between them. An edge server, on the other hand, can only reach out to the end devices within its proximity to make use of their limited datasets; however, the communication between the server and the end devices can be much more efficient due to the short-range communication. Analyzing the tradeoffs between the cloud server-based and the edge server-based architectures has inspired the proposal of *multi-tier federated learning* architecture, for example, the client-edge-cloud hierarchical federated learning [10]. The system is composed of three tiers: on the bottom of the architecture is a large number of end devices that form a big base of clients, on the top of the architecture is a cloud server that acts as the central parameter server, and a number of edge servers lying at the middle tier that connect the top and bottom tiers.

With the introduction of servers at middle tiers, the large number of model updates generated by the FL clients are only communicated to and aggregated at the middle-tier servers, and the middle-tier servers only need to transfer aggregated model updates to the central parameter server. This way, the incurred communication cost is distributed to multiple tiers of servers and is more scalable as it does not increase at a linear or higher order of the number of clients. However, such advantages do not come without any cost. In particular, the detection/mitigation of poisoning attacks [11]–[14], which have been treated as an important security threat to federated learning, becomes more challenging to address. For two-tier FL systems, most of the proposed methods for detecting/mitigating poisoning attacks of clients [15]–[20] require the access and processing of the clients' model updates; they often assume the central server to be trusted and take the whole detection/mitigation job. For a multi-tier FL system, however, the central server does not have access to the original model updates reported by clients, as the middle-tier servers should aggregate them for communication efficiency and scalability. The middle-tier servers need to participate in processing clients' updates for attack detection/mitigation, while they aggregate the updates. Hence, two problems should be addressed: the mechanisms for detecting/mitigating clients' attacks should be decentralized such that both central server and middle-tier servers can participate in; as the middle-tier servers are not as trusted as the central server, attacks that may be launched by them should be detected and addressed.

Towards addressing the above problems, we propose a distributed FL framework aiming to attain system efficiency and security simultaneously. The framework contains two novel features: First, it can efficiently detect/mitigate the poisoning attacks that end devices may launch, through distributed collaborations between the central and the middle-tier edge servers; second, it can detect the attacks that may be launched by malicious middle-tier edge servers, based on the collaborations between the central server and the end devices. To the best of our knowledge, this is the first effort for distributed detection/mitigation of poisoning attacks and misbehavior of middle-tier servers in a three-tier FL system.

Extensive experiments have been conducted to evaluate the performance of our proposed design. The results show that, our design can effectively detect and mitigate poisoning attacks and attain the level of model training accuracy comparable to or even better than the CONTRA [20], a state-of-the-art centralized detection/mitigation scheme. The results also indicate that, the overheads incurred by our design for detecting misbehaving middle-tier servers are moderate.

In the rest of the paper, we define the problem in Section II, provide an overview of our proposed design in Section III, which is followed by the detailed design for distributed detection/mitigation of clients' attacks in Section IV and for detection of middle-tier edger servers' misbehavior in Section V. Performance evaluation is reported in Section VI. Section VII briefly discusses related works, and finally Section VIII concludes the paper.

II. Problem Definition

A. System Model

We consider a system composed of three tiers of entities: one *FL server*, a moderate set of e *edge servers* denoted as $\mathcal{E}_0, \cdots, \mathcal{E}_{e-1}$, and a large set of c *FL clients* (i.e., end devices). The FL server is connected with all the edge servers. Each edge server is connected with a subset of the FL clients, where for simplicity each client is assumed to be connected with only one edge server. Therefore, we use c_i, where $i = 0, \cdots, e-1$, to denote the number of clients connected with $\mathcal{E}_i$, and let $\mathcal{C}_{i,0}, \cdots, \mathcal{C}_{i,c_i-1}$ denote these clients. Note that, in practice there could be multiple FL processes ongoing concurrently, each edge server could serve multiple FL servers at the same time, and each client might participate in multiple FL processes as well. In this paper, we focus on only one FL process as other concurrent processes can work with the same set of security mechanisms that we propose.

In this three-tier hierarchical system, a *basic FL process* can be modeled generally as follows. Similar to the FL processes commonly modeled in the literature, we also consider a FL process that runs round by round. Each round t is started by the FL server, which disseminates the current model, denoted as $\mathbf{W}^{(t)}$, to all the edge servers. Particularly, for the first round, the model is initialized in a certain way (e.g., randomly); for each of the later rounds, the model should be the result of learning from all the previous rounds. Each edge server $\mathcal{E}_i$ further disseminates the received model to all the clients that it is connected with. Then, each client $\mathcal{C}_{i,j}$ performs local learning based on the received model and its local training data. After the local learning for this round completes, the client obtains the updates to the parameters of the model, which are represented as a vector called *gradient* and denoted as $\mathbf{G}_{i,j}^{(t)}$, and sends the gradient to $\mathcal{E}_i$. For communication and computation efficiency, $\mathcal{E}_i$ aggregates all the gradients that it receives from the connected clients, and reports the aggregated gradient, denoted as $\mathbf{G}_i^{(t)}$, to the FL server. Upon receiving aggregated gradients from all the edge servers, the FL server processes them to obtain a newer model, denoted as $\mathbf{W}^{(t+1)}$, which is to be disseminated in the next round (i.e., round $t+1$).

Our proposed designs are based on the above basic process, but adds security mechanisms to address the security threats as presented in the following.

B. Security Model

We assume the FL server is fully trusted, while the edge servers and FL clients are not trusted.

The FL clients may not honestly conduct local learning or report local updates correctly. The attacks that can be potentially launched by the clients have been studied extensively. Malicious clients could launch simple attacks, for example, label flipping, where they poison their training data by flipping labels. The attacks could also be more sophisticated; for example, the malicious clients may collude and manipulate the global model by taking orchestrated attacks. Lots of research [15]–[20] have been conducted to address such attacks, but they were developed mainly for the two-tier systems where only the central FL server is responsible for attack detection and mitigation. For a three-tier system as studied in this paper, the detection and defense should involve the edge servers in order to leverage them for better scalability in computation and communication. That is, we need more distributed solutions.

In this paper, edge servers are to be employed for efficient gradient aggregation as well as attack detection and mitigation. However, we do not assume the edge servers as trusted. They may not behave honestly in conducting the afore-mentioned aggregation and detection/aggregation tasks. Hence, security mechanisms should also be designed for the FL server and the FL clients to detect and mitigate the misbehavior of the edge servers.

Note that, the edge servers and/or FL clients may also attempt to breach the privacy of other FL clients. Hence, privacy-preservation mechanisms should also be deployed. Due to space limit, we do not study such issue in this paper.

C. Design Goals

Our proposed design aims to attain the goals in both security and performance aspects. In terms of security, we aim to attain the following goals:

- When a non-majority set of FL clients launch data poisoning attacks, the FL and edge servers should be able to mitigate the attacks and continue performing FL process and attaining acceptable levels of accuracy.
- When any edge server does not executes the protocol deployed to them, its misbehavior should be detected by the FL server, which may collaborate with honest FL clients.

In terms of performance, we aim to attain the following goals:

- The communication costs should be as low as possible. The costs are incurred by: the clients reporting their gradients to the edge servers; the edge servers reporting their aggregated gradients to the central server; the demanded communication among the clients and servers for the purpose of attack detection and mitigation. We should aim to lower the sum of such communication costs.

- The computational cost should be as low as possible. The costs for local learning at the clients, and for gradient aggregation and model updating at the server, are inherent. Hence, we mainly target at the extra costs incurred by attack detection and mitigation, and we should aim to make such extra costs as low as possible.
- The system performance should be scalable. The performance of the system can be impacted by the size of the model to be trained, the number of clients and the number of edge servers. The scalability goal of our proposed designs is to make the communication and computational costs not increase at a rate higher than linear to the size of the model or the number of clients and servers.

III. Overview of the Proposed Framework

This section presents the framework of our proposed design. The framework is based on the *basic FL process* presented in Section II-A, but security mechanisms for detecting/mitigating clients' attacks and for detecting edge servers' misbehavior are added. In the framework, a federated learning (FL) process runs round by round, and each round t is started by the central server, which disseminates the current global model $\mathbf{W}^{(t)}$ to all the clients via the edge servers. The rest of the round contain the following components.

A. Local Operations at Client

Upon receiving the current global model $\mathbf{W}^{(t)}$, each client $\mathcal{C}_{i,j}$ works as follows.

1) Local Learning: Client $\mathcal{C}_{i,j}$ conducts local learning based on the received $\mathbf{W}^{(t)}$ and its local dataset, and thus obtains the model update (i.e., gradient) for this round. We denote the gradient as $\mathbf{G}_{i,j}^{(t)}$, which is divided to two parts $\mathbf{G}_{i,j}^{(t,0)}$ and $\mathbf{G}_{i,j}^{(t,1)}$; that is,

$$\mathbf{G}_{i,j}^{(t)} = (\mathbf{G}_{i,j}^{(t,0)},\ \mathbf{G}_{i,j}^{(t,1)}). \tag{1}$$

Here, $\mathbf{G}_{i,j}^{(t,0)}$ is the part of gradient used for computing the similarity between gradients for detection/mitigation of clients' attacks, while $\mathbf{G}_{i,j}^{(t,1)}$ is the rest of the gradient. Also, we use g_0 and g_1 to denote the sizes of (i.e., the number of elements in) $\mathbf{G}_{i,j}^{(t,0)}$ and $\mathbf{G}_{i,j}^{(t,1)}$, respectively.

Note that, the idea of using a part of (instead of the whole) gradient for similarity analysis is consistent with the related works such as CONTRA [20]. Specifically, we only use the parameters in the final output layer, since they map directly to the output probabilities and are relevant to the correctness of the model and the success of a targeted attack [21].

2) Facilitating the Detection of Edge Servers' Misbehavior: In order to facilitate the detection of possible misbehavior by edge server $\mathcal{E}_i$, client $\mathcal{C}_{i,j}$ should compute the *commitments* of its gradients $\mathbf{G}_{i,j}^{(t,0)}$ and $\mathbf{G}_{i,j}^{(t,1)}$, denoted as $com(\mathbf{G}_{i,j}^{(t,0)})$ and $com(\mathbf{G}_{i,j}^{(t,1)})$. Details of the computation are to be elaborated in Section V. Then, the commitments should be securely sent to the central server.

3) Reporting of gradients: After the commitments have been successfully sent to the central server, client $\mathcal{C}_{i,j}$ sends gradients $\mathbf{G}_{i,j}^{(t,0)}$ and $\mathbf{G}_{i,j}^{(t,1)}$ to edge server $\mathcal{E}_i$ for aggregation and clients' attack detection/mitigation.

B. Operations at Edge Server

Each edge server $\mathcal{E}_i$ should conduct two tasks, aggregating gradients reported by its clients and facilitating the detection/mitigation of clients' poisoning attacks.

1) Gradient Aggregation: $\mathcal{E}_i$ aggregates the gradients from its clients. Specifically, the aggregation is implemented as a weighted averaging of all the received local updates. For this purpose, each edge server $\mathcal{E}_i$ maintains a weight $w_{i,j}$ for each client $\mathcal{C}_{i,j}$. We denote the gradients aggregated by $\mathcal{E}_i$ as $\mathbf{G}_i^{(t,0)}$ and $\mathbf{G}_i^{(t,1)}$, where

$$\mathbf{G}_i^{(t,0)} = \sum_{j=0}^{c_i-1} w_{i,j} \cdot \mathbf{G}_{i,j}^{(t,0)} \tag{2}$$

and

$$\mathbf{G}_i^{(t,1)} = \sum_{j=0}^{c_i-1} w_{i,j} \cdot \mathbf{G}_{i,j}^{(t,1)}. \tag{3}$$

Then, the edge server forwards the aggregation results to the FL server.

2) Facilitating Detection/Mitigation of Clients' Poisoning Attacks: We propose a decentralized scheme for detecting/mitigating clients' attacks, as to be elaborated in Section IV. In a nutshell, edge server $\mathcal{E}_i$ should maintain a set of reference vectors. Then, the similarities between the gradients reported by every client $\mathcal{C}_{i,j}$ and every reference vector should be computed, and the similarities should be reported to the central server for further processing.

C. Operations at Central Server

The central server works on updating global model, detecting/mitigating clients' poisoning attacks, and detecting edge servers' misbehavior.

1) Updating Global Model: The central server should aggregate the aggregated gradients from the edge servers, in order to update the global model. Specifically, the central server should maintain the weight $w_{i,j}$ for every client $\mathcal{C}_{i,j}$. Given the aggregated gradients $\mathbf{G}_i^{(t,0)}$ and $\mathbf{G}_i^{(t,1)}$ from every edge server $\mathcal{E}_i$, the central server computes

$$\mathbf{G}^{(t,0)} = \sum_{i=0}^{e-1} \frac{\sum_{j=0}^{c_i} w_{i,j}}{\sum_{i=0}^{e-1}\sum_{j=0}^{c_i-1} w_{i,j}} \cdot \mathbf{G}_i^{(t,0)} \tag{4}$$

and

$$\mathbf{G}^{(t,1)} = \sum_{i=0}^{e-1} \frac{\sum_{j=0}^{c_i} w_{i,j}}{\sum_{i=0}^{e-1}\sum_{j=0}^{c_i-1} w_{i,j}} \cdot \mathbf{G}_i^{(t,1)}. \tag{5}$$

Then, the global model is updated based on the combined gradient $\mathbf{G}^{(t)} = (\mathbf{G}^{(t,0)}, \mathbf{G}^{(t,1)})$.

2) Detecting/mitigating Clients' Poisoning Attacks: As elaborated in Section IV, the central server should gather the similarities reported by all of the edge servers. The similarities for each client $\mathcal{C}_{i,j}$ are treated as a compressed gradient of the client. Then, similarity analysis is conducted over the compressed gradients to detect clients that may have launched poisoning attacks and then update the weights of the clients to mitigate the attacks.

3) Detecting Misbehaving Edge Servers: As elaborated in Section V, the central server should gather the commitments reported by all of the clients. Then, the similarities and the aggregated gradients reported from each edge server $\mathcal{E}_i$ are checked against the commitments for consistency. If inconsistency is identified, the edge server is found misbehaving. Note that, it is also possible for an detected edge server to dispute. In this case, the edge server and its clients can show their evidences to the central server to resolve the dispute; we skip the detail of this due to space limit.

IV. Distributed Detection and Mitigation of Clients' Poisoning Attacks

In this section, we first present the basic ideas of our proposed scheme for distributed detection and mitigation of clients' poisoning attacks, and then describe the details.

A. Basic Ideas

Our proposed scheme is inspired by the CONTRA scheme [20], in which the similarities between the gradients reported by every pair of clients are computed as the *cosine* value of the angle between the compared gradients. Then the cosine values are used to identify suspicious attacking clients and to adjust the weights of the clients' gradients in model updating.

The CONTRA scheme has been shown effective in addressing un-targeted and particularly targeted attacks regardless of whether the clients' training data distribution is I.I.D. or not. However, the design cannot be directly applied to the three-tier FL system. Computing the cosine value for each client pair demands to access all the clients' gradients. This is convenient for the CONTRA scheme that was designed for the two-tier FL system, as all the clients' gradients are directly reported to the central server which performs the computation. In the three-tier FL system, however, each edge server has access to only a subset of the clients' gradients (i.e., the gradients reported by the clients connected to it) and the central server only has access to the aggregated gradients (instead of the original clients' gradients) reported by the edge servers.

To address the above problem, instead of directly computing the pairwise similarities as in the CONTRA scheme, we propose an indirect approach to estimate the similarities. More specifically, the basic ideas are as follows. We first introduce a set of r *reference gradients*, denoted as $\mathbf{R}_0, \cdots, \mathbf{R}_{r-1}$, and have all the edge servers share these gradients. Here, r is a system parameter. At each edge server $\mathcal{E}_i$, the similarity between the gradient $\mathbf{G}_{i,j}^{(t,0)}$ of each client $\mathcal{C}_{i,j}$ at round t and each reference gradient $\mathbf{R}_k$ is computed; this way, a vector of cosine values (denoted as $\mathbf{C}_{i,j}^{(t)}$) is obtained for client $\mathcal{C}_{i,j}$ at edge server $\mathcal{E}_i$. The edge servers report the above vectors to the central server, which then estimates the similarity, denoted as $cos_{i,j;i',j'}^{(t)}$, for each pair of clients $\mathcal{C}_{i,j}$ and $\mathcal{C}_{i',j'}$. The approach for estimation is based on the intuition that, if the reference vectors are properly selected, the similarity between $\mathbf{C}_{i,j}^{(t)}$ and $\mathbf{C}_{i',j'}^{(t)}$ reflects the similarity between $\mathbf{G}_{i,j}^{(t,0)}$ and $\mathbf{G}_{i',j'}^{(t,0)}$. There could be many ways to quantify the similarity between cosine vectors. In this work, we treat each vector of cosine value $\mathbf{C}_{i,j}^{(t)}$ as a compressed version of the original gradient $\mathbf{G}_{i,j}^{(t,0)}$, and hence estimates $cos_{i,j;i',j'}^{(t)}$ for any client pair $\mathcal{C}_{i,j}$ and $\mathcal{C}_{i',j'}$ as the cosine value for the similarity between $\mathbf{C}_{i,j}^{(t)}$ and $\mathbf{C}_{i',j'}^{(t)}$.

The details of our proposed scheme are in as follows.

B. Construction of Reference Gradients

How many reference gradients are needed and how to construct them are two important questions that should be answered in the proposed design. The decision and design should meet two requirements:

- *efficacy* - the constructed reference gradients can be used to estimate the pairwise similarities as accurately as possible;
- *efficiency* - the number of needed reference gradients should be as small as possible in order to incur low communication and computation costs at the edge and central servers.

To meet the requirement for effectiveness while ignoring efficiency, a naive construction could be as following: Supposing each gradient has g_0 elements, we first construct a unit matrix of dimension $g_0 \times g_0$, i.e., a matrix with 1s on the main diagonal while zeros otherwise; then, all the g_0 column vectors would become the reference gradients. Obviously, such a construction is effective; in fact, each $\mathbf{C}_i$ computed based on such reference gradients would be the same as the original gradient $\mathbf{G}_i$ for each $\mathcal{C}_i$. However, this construction is inefficient as all original gradients would be reported to the central server.

To make a proper tradeoff between the two requirements, we propose a heuristic algorithm that randomly combines the g_0 reference gradients constructed as the above into r reference gradients, where system parameter $r < g_0$. More specifically, the construction algorithm is as follows:

1) The g_0 positions of a gradient are randomly divided into r groups (called group $0, \cdots, r-1$ respectively), as evenly as possible.
2) For each group $i \in \{0, \cdots, r-1\}$, a reference vector denoted as $\mathbf{R}_i$ is constructed such that its elements on the positions in group i are set to 1s while the others are set to 0.

As to be shown later, with a properly selected parameter r, the above algorithm can construct effective and efficient reference gradients.

Once the reference gradients have been constructed, they are loaded to every edge server before the FL process starts.

C. Computation of Similarities at Edge Servers

Analyzing a client's behavior can be based solely on the client's one-round gradient or the history of its gradients reported in recent rounds. As the former can be treated as a special case of the latter, here we describe for the more general case, where each client's gradients reported in the recent h rounds should be saved and aggregated for the computation of similarities. Specifically, each edge server $\mathcal{E}_i$ allocates a buffer $B_{i,j}$ to store the most recent h rounds of gradients reported by each client $\mathcal{C}_{i,j}$, and the space is empty initially.

At each round t, each edge server $\mathcal{E}_i$ receives gradient $\mathbf{G}_{i,j}^{(t,0)}$ from each client $\mathcal{C}_{i,j}$ connected to it. The gradient is stored to buffer $B_{i,j}$, and the oldest gradient stored in the buffer is discarded if the buffer has stored more than h gradients. An average of the gradients in $B_{i,j}$ is computed and the average is denoted as $\bar{\mathbf{G}}_{i,j}^{(t,0)}$; that is,

$$\bar{\mathbf{G}}_{i,j}^{(t,0)} = \frac{1}{\min(t+1,h)} \sum_{u=\max(0,t-h+1)}^{t} \mathbf{G}_{i,j}^{(u,0)}. \tag{6}$$

The cosine values of the angles between each $\bar{\mathbf{G}}_{i,j}^{(t,0)}$ and every reference gradients are computed and they are denoted as $cos_{i,j;k}^{(t)}$ for $k = 0, \cdots, r-1$; that is,

$$cos_{i,j;k}^{(t)} = \frac{\langle \bar{\mathbf{G}}_{i,j}^{(t,0)}, \mathbf{R}_k \rangle}{||\mathbf{R}_k||}, \tag{7}$$

where $\langle X, Y \rangle$ denotes the inner product of vectors X and Y, and $||X||$ denotes the L_2-norm of vector X. Then all these computed cosine values, i.e., $\{cos_{i,j;k}^{(t)} \mid k = 0, \cdots, r-1\}$, are reported to the central server.

D. Estimation of Pairwise Similarities at Central Server

After the central server has received the cosine values for round t from all edge servers, it constructs a vector, denoted as $\mathbf{C}_{i,j}^{(t)}$, of the cosine values for each client $\mathcal{C}_{i,j}$. That is,

$$\mathbf{C}_{i,j}^{(t)} = (cos_{i,j;0}^{(t)}, \cdots, cos_{i,j;r-1}^{(t)})^T. \tag{8}$$

Then, it uses the following formula to estimate the cosine value (i.e., similarity) for each pair of clients $\mathcal{C}_{i,j}$ and $\mathcal{C}_{i',j'}$:

$$cos_{i,j;i'.j'}^{(t)} = \frac{\langle \mathbf{C}_{i,j}^{(t)}, \mathbf{C}_{i',j'}^{(t)} \rangle}{||\mathbf{C}_{i,j}^{(t)}|| \cdot ||\mathbf{C}_{i',j'}^{(t)}||}. \tag{9}$$

E. Mitigation of Attacks

Once the pairwise similarities have been computed, the similarities are used to mitigate the attacks. We follow the CONTRA scheme [20] for the mitigation. To be self-contained, we summarize the steps in the following.

- First, alignment level $\tau_{i,j}$ is calculated for each client $\mathcal{C}_{i,j}$ based on the average of its top-k pairwise cosine similarity with all other clients. Parameter k is chosen based on the number of malicious clients. Letting n be the total number of clients, we pick $\frac{n}{5}$ as k in our experiments, as we assume that only a non-majority (i.e., less than 50%) of clients are malicious.
- After getting alignment levels, adaptive learning rates are calculated. Specifically, for client $\mathcal{C}_{i,j}$, its learning rate $lr_{i,j}$ is adjusted to $1 - \tau_{i,j}$ correspondingly.

In a targeted attack, malicious clients tend to align more with each other, and thus have higher alignment levels, which leads to lower adaptive learning rates for them. As a result, the contribution of the malicious clients are downgraded and hence the aggregation result is more accurate. To reduce the chance that honest clients are penalized, all learning rates are normalized and pardon is further used [10]. We notice from our experiments that, both malicious and honest clients have similar learning rates at the early stages of training; but at the later stages, most malicious clients' learning rates are decreased to 0 and honest clients' learning rates are increased instead.

V. Detection of Misbehaving Edge Servers (**This part need revising for the adoption of (leveled) homomorphic encryption system.**)

A. Basic Ideas

Different from related works such as CONTRA [20] where FL is conducted in a two-tier system, we need to place certain mechanisms to detect misbehaving edge servers in the three-tier system. Specifically, we consider the following attacks:

- A misbehaving edge server may fake/modify the gradients reported by its clients.
- A misbehaving edge server may not honestly aggregate the gradients from its clients.
- A misbehaving edge server may not honestly compute the vectors of cosine values for its clients.

To address these attacks, we propose a commitment-verification scheme based on the following ideas:

- For each client, after it has computed its gradient, it should not immediately send the gradient to the edge server it connects with. Instead, it should compute a commitment for the gradient. The commitment is computed by leveraging an homomorphic encryption (HE) algorithm that is parameterized by secrets known only by the FL server; thus, the commitment for a client's gradient cannot be faked by anyone else.
- Each client should submit the commitment of its gradient and make sure the commitment is received correctly by the central server, before the client can submit the gradient to its edge server.
- After receiving the aggregated gradients and the orientation similarities from the edge servers, the central server should verify them based on the commitments that it has received from the clients earlier. This way, the edge server cannot fake/modify its clients' gradients and apply the falsified values in aggregation or computing the orientation similarities, without being detected by the central server.

In the rest of this section, we present the HE primitives and the three key steps, i.e., system initialization, commitment construction/reporting and computation verification, of the proposed scheme.

B. Preliminary: Homomorphic Encryption (HE)

The proposed scheme applies a public-key homomomorphic encryption (HE) algorithm, which has the following primitives:

- $(sk, pk, evk, \mathcal{S}) \leftarrow KeyGen(1^\lambda, \mathcal{L})$: with security parameter λ and the highest level of encryption $\mathcal{L}$ as inputs, the primitive for key generation outputs a secret key sk, a public key pk, an evaluation key evk and the the slot number $\mathcal{S}$ for each ciphertext. Particularly, $\mathcal{S}$ indicates the maximum number of scalar values that can be encoded and encrypted within a single ciphertext.
- $ct \leftarrow Enc_{pk}(\vec{pt})$: provided public key pk and a plaintext vector $\vec{pt} = (pt_0, \cdots, pt_{\mathcal{S}-1})$ of $\mathcal{S}$ elements, the primitive for encryption outputs a ciphertext ct. Note that a newly encrypted ciphertext has the smallest noise or the highest level (i.e., $\mathcal{L}-1$); we denote this as $ct.level = \mathcal{L}-1$.
- $\vec{pt} \leftarrow Dec_{sk}(ct)$: provided secret key sk and a ciphertext ct, the primitive for decryption outputs plaintext vector $\vec{pt}$ s.t. $Enc_{pk}(\vec{pt}) = ct$.
- $ct' \leftarrow ct_1 \oplus ct_2$: provided ciphertexts ct_1 and ct_2, the primitive for addition outputs ciphertext ct' s.t. $Dec_{sk}(ct') = Dec_{sk}(ct_1) + Dec_{sk}(ct_2)$. Here, the $+$ operator stands for element-wise addition between two vectors; that is, if $Dec_{sk}(ct_1) = \vec{pt}_1 = (pt_{1,0}, \cdots, pt_{1,\mathcal{S}-1})$ and $Dec_{sk}(ct_2) = \vec{pt}_2 = (pt_{2,0}, \cdots, pt_{2,\mathcal{S}-1})$, then $Dec_{sk}(ct_1) + Dec_{sk}(ct_2) = (pt_{1,0} + pt_{2,0}, \cdots, pt_{1,\mathcal{S}-1} + pt_{2,\mathcal{S}-1})$. Note that, applying this primitive aggregate the noises in the operands ct_1 and ct_2 to ct', and ct' has the level which is the smaller one between those of ct_1 and ct_2; specifically, $ct'.level = \min\{ct_1.level, ct_2.level\}$.
- $ct' \leftarrow ct_1 \otimes ct_2$: provided ciphertexts ct_1 and ct_2, the primitive for multiplication between ciphertexts outputs ciphertext ct' s.t. $Dec_{sk}(ct') = Dec_{sk}(ct_1) \times Dec_{sk}(ct_2)$. Here, the $\times$ operator stands for element-wise multiplication between two vectors; that is, if $Dec_{sk}(ct_1) = \vec{pt}_1 = (pt_{1,0}, \cdots, pt_{1,\mathcal{S}-1})$ and $Dec_{sk}(ct_2) = \vec{pt}_2 = (pt_{2,0}, \cdots, pt_{2,\mathcal{S}-1})$, then $Dec_{sk}(ct_1) \times Dec_{sk}(ct_2) = (pt_{1,0} \times pt_{2,0}, \cdots, pt_{1,\mathcal{S}-1} \times pt_{2,\mathcal{S}-1})$. Note that, applying this primitive results in larger noise at the resulting ciphertext than that of every operands and hence the level of the result is lower than that of every operands; specifically, $ct'.level = \min\{ct_1.level, ct_2.level\} - 1$.
- $ct' \leftarrow CMult(ct, \vec{pt})$: provided ciphertext ct and plaintext $\vec{pt}$, the primitive for multiplication between plaintext and ciphertext outputs ciphertext ct' s.t. $Dec_{sk}(ct') = \vec{pt} \times Dec_{sk}(ct)$. Note that, applying this primitive also results in larger noise at the resulting ciphertext than that of the ciphertext operand and hence the level of the result is lower than that of the operand; specifically, $ct'.level = ct.level - 1$.
- $ct' \leftarrow Rot(ct, m)$: provided ciphertext ct that encrypts $\vec{pt} = (pt_0, \cdots, pt_{\mathcal{S}-1})$ and integer $m < \mathcal{S}$, the primitive for rotation outputs ct' which is ciphertext for $(pt_m, \cdots, pt_{\mathcal{S}-1}, pt_0, \cdots, pt_{m-1})$. Note that, applying this primitive does not change the level; that is, $ct'.level = ct.level$.

Our design and implementation utilize CKKS [22], which offers all of the aforementioned primitives required by our proposed system.

C. Initialization

To initialize the system for detecting misbehaving edge servers, the central server takes the following steps:

- First, it calls $KeyGen$ with appropriate security parameters to obtain keys pk and sk for AHE.
- Second, it randomly picks r numbers denoted as $a_0, \cdots, a_{r-1}$.
- Third, it derives g numbers denoted as $s_0, \cdots, s_{g-1}$ as follows:
$$s_u = \sum_{v=0}^{r-1} (a_v \cdot R_v[u]), \tag{10}$$
where $u \in \{0, \cdots, g-1\}$ and each $R_v[u]$ denotes the element of R_v at index u.
- Fourth, it encrypts each s_u with the AHE encryption primitive and pk to get ciphertext s'_u, as follows:
$$s'_u = Enc_{pk}(s_u). \tag{11}$$

At the end of the above steps, the central server should keep pk, sk, $S = \{s_u | u = 0, \cdots, g-1\}$ and $S' = \{s'_u | u = 0, \cdots, g-1\}$.

When a new client $\mathcal{C}_{i,j}$ joins the FL process, the central server should provide it with pk and S'. We assume the client and the central server communicates via a secure channel, which can be set up through a protocol such as SSL.

D. Commitment Construction and Sending by Clients

For each client $\mathcal{C}_{i,j}$, after having computed its gradients $\mathbf{G}_{i,j}^{(t,0)}$ and $\mathbf{G}_{i,j}^{(t,1)}$) at round t, it computes commitments for these gradients. Each commitment is a tuple, denoted as $com(\mathbf{G}_{i,j}^{(t,b)}) = (com_0, com_1)$ for $b \in \{0, 1\}$. Here,

$$com_0 = \frac{\sum_{u=0}^{g_b} \mathbf{G}_{i,j}^{(t,b)}[u] \cdot s'_u}{||\mathbf{G}_{i,j}^{(t,b)}||}, \tag{12}$$

where $\mathbf{G}_{i,j}^{(t,b)}[u]$ is the u-the element of $\mathbf{G}_{i,j}^{(t,b)}$; and

$$com_1 = ||\mathbf{G}_{i,j}^{(t,b)}||. \tag{13}$$

Then, the client sends $com(\mathbf{G}_{i,j}^{(t,0)})$ and $com(\mathbf{G}_{i,j}^{(t,1)})$ to the central server via the secure channel between them, which is set up at the time when the client joins the FL process. After the commitments have been successfully received by the central server, the client sends its gradient to the edge server that it is connected with.

Optimization: In practice, the number of parameters in a model for training could be large and so the gradient vector computed by a client can contain a large number of elements. The cost for computing commitments for many elements is high. For computational efficiency, a parameter pruning technique can be applied. Specifically, each client $\mathcal{C}_{i,j}$ may only select $g_1' < g_1$ elements of $\mathbf{G}_{i,j}^{(t,1)}$, where the g_1' elements have larger magnitudes than those that are not selected. This way, the commitment computation only involves $g_0 + g_1' < g$, instead of g, elements, and therefore the computational cost can be reduced. Note that, we assume the client will not prune any parameter from $\mathbf{G}_{i,j}^{(t,0)}$ because this part of gradient is needed for detecting/mitigating clients' poisoning attacks.

E. Verification at Central Server

The central server should check the edge server's behavior on aggregating the clients' gradients and on computing the similarities between the clients' gradients and the reference vectors. We elaborate them as follows.

1) Verifying Edge Server on Gradient Aggregation: Upon receiving an aggregated gradient $\bar{\mathbf{G}}_i^{(t)} = (\bar{\mathbf{G}}_i^{(t,0)}, \bar{\mathbf{G}}_i^{(t,1)})$ from each edge server $\mathcal{E}_i$, the central server first computes $com(\bar{\mathbf{G}}_i^{(t,0)})$ and $com(\bar{\mathbf{G}}_i^{(t,1)})$. Then, it verifies if

$$\begin{aligned} & com(\bar{\mathbf{G}}_i^{(t,0)})_0 \cdot com(\bar{\mathbf{G}}_i^{(t,0)})_1 \\ = & \sum_{j=0}^{c_i-1} com(\mathbf{G}_{i,j}^{(t,0)})_0 \cdot com(\mathbf{G}_{i,j}^{(t,0)})_1, \end{aligned} \tag{14}$$

and

$$\begin{aligned} & com(\bar{\mathbf{G}}_i^{(t,1)})_0 \cdot com(\bar{\mathbf{G}}_i^{(t,1)})_1 \\ = & \sum_{j=0}^{c_i-1} com(\mathbf{G}_{i,j}^{(t,1)})_0 \cdot com(\mathbf{G}_{i,j}^{(t,1)})_1. \end{aligned} \tag{15}$$

Here, for a commitment $com(\mathbf{G})$ for some vector $\mathbf{G}$, $com(\mathbf{G})_0$ and $com(\mathbf{G})_1$ represent the two elements of $com(\mathbf{G})$ respectively. If either of the above equality fails to hold, the edge server is found misbehaving.

2) Verifying Edge Server on Similarity Computation: Upon receiving cosine values $cos_{i,j;k}^{(t)}$ where $k = 0, \cdots, r-1$ for client $\mathcal{C}_{i,j}$, the central server verifies if

$$\begin{aligned} \tfrac{1}{min(t+1,h)} & \sum_{u=max(0,t-h+1)}^{t} com(\mathbf{G}_{i,j}^{(u,0)})_0 \cdot com(\mathbf{G}_{i,j}^{(u,0)})_1 \\ = & \sum_{k=0}^{r-1} (a_k \cdot ||R_k|| \cdot cos_{i,j;k}^{(t)}). \end{aligned} \tag{16}$$

If the quality does not hold, the edge server is found misbehaving.

VI. Performance Evaluations

In this section, we report the settings and results of our experiments for performance evaluation.

A. Experiment Settings

1) Datasets: Our proposed design is evaluated by running experiments with two standard datasets: *MNIST* [23] and *CIFAR-10* [24]. The details of the training data and the validation accuracy when there is no attacks from the clients are shown in Table I. For the *MNIST* dataset, LeNet [25] is used for training. There are 21840 trainable parameters in total. On the client level, the optimizer is chosen to be Stochastic Gradient Descent (SGD) with mini-batch size 20. The initial learning rate is set to 0.01 with an exponential decaying rate of 0.995 on every epoch. For the *CIFAR-10* dataset, a convolution neural network (CNN) with three convolution layers is used for training. The number of trainable parameters is 5852170 in total. On the client level, the optimizer is set to SGD with mini-batch size 20. The initial learning rate is set as 0.1 with an exponential decaying rate of 0.992 on every epoch. Although using SGD with momentum can speed up the training and improve the accuracy, it regulates the gradients' directions, thus leading to difficulty in differentiating honest and malicious clients. Since our design detects malicious clients by comparing the cosine similarity between gradients of clients' updates, momentum is not suitable in our experiments. In order to reach convergence, the numbers of communication rounds are set to 100 and 300 for training *MNIST* and *CIFAR* respectively. We also fix the local update k1 to 60 iterations and edge updates k2 as 1 iteration. That is, each client runs local training $k1 = 60$ iterations and before sending the local updates to its connected edge server; each edge server receives local updates from clients, aggregates them and then sends the aggregated result to central server.

2) Data Distribution: The non-IID data distribution is adopted in our experiments in order to simulate the realistic cases where each client generates data independently and the local data distribution tends to be unbalanced [26]. The non-IID data distribution typically has huge impact on performance, but as we will see our proposed design is robust against data poisoning attacks when data distribution is non-IID. In our experiments, there are two levels of non-IIDness, namely the client level and the edge level [10]. On the client level, we adopt the most commonly used non-IID data partition [27], where each client is assigned with two classes of samples. However, on the edge level, each edge server is assigned with the same number of the clients randomly. This aligns better with realistic cases where clients are distributed randomly. Thus each edge server has both malicious and honest clients at the same time.

3) Simulation of Attacks: In our experiments, we deploy different types of data poisoning attacks in various settings where malicious clients account for varying proportions(m) among all clients. Following the related works such as CONTRA [20], we simulate three types of data poisoning attacks as follows.

- The first type is *label-flipping attacks*, where the adversaries attempt to flip labels of the training samples to a target label while leaving all the training features

Table I: Datasets and performance of unattacked models. Model Accuracy: MA.

	#clients	train/test	feature	model	parameters	K1	K2	MA
MNIST	100	60K/10K	784	LeNet	21840	60	1	92.22%
CIFAR-10	100	50K/10K	1024	CNN	5852170	60	1	71.74%

unchanged.

- The second type is *coordinated attacks*, where the malicious clients share a certain amount of training data and flip the labels of the training samples to a target label. For such attacks, we assume that the malicious clients are able to coordinate and manipulate their data [21]. For each malicious client, the data sharing rate is set as x; that is, $x\%$ of the training samples for each malicious client are uniformly drawn from all classes and the rest (1-x)% training samples are attack samples that belong to this malicious client only [20].
- We also simulate the simple *backdoor attacks*, where the white pixel in the bottom-right corner of a figure is inserted and the corresponding label is flipped to 7 for malicious clients. For the test data, we insert the white pixel backdoor pattern for 20% test data, same as [21].

4) Evaluation Metrics: In evaluating system performance, we use the metrics of model accuracy (MA) and attack success rate (ASR). The attack success rate is defined as the ratio of data points in the test set that are incorrectly classified to 7 (the poisoning target). A successful attack means that final model incorrectly classify the label to the poisoning target label 7. We adopt the related work CONTRA [20] as the baseline in evaluating our design's performance.

We also evaluate the overhead for detecting misbehaving edge servers, in terms of the extra delay for computation incurred at each client and the central server.

B. Evaluation Results

Table II: Performance of our proposed design under various number of references: MA and ASR at model convergence

	MNIST		*CIFAR-10*	
r	MA(%)	ASR(%)	MA(%)	ASR(%)
10	92.03	0.68	63.31	1.31
50	92.30	1.13	68.40	2.37
100	92.41	0.62	67.64	1.34
200	92.40	1.11	68.05	1.74
500	92.33	0.97	67.91	1.67
baseline[1]	92.33	0.97	67.90	2.00

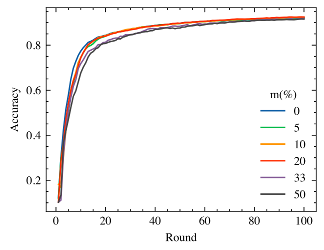
1) Impact of Reference Vector Number: Since our design decentralizes the estimation of cosine similarity values among clients, it sets up a fixed number of reference vectors at the beginning. As we mention earlier, we only use the features in the final output layer to calculate the cosine similarity values between clients' gradient and reference vector. Particularly, for the LeNet and CNN models that we use, there are 500 and 5120 parameters respectively in the final output layer. The number of reference vectors depends on the neural networks' final output layer; more specifically, the number of outputs from the final output layer is the upper-bound of the number of reference vectors. To find out an appropriate number of reference vectors, we run the experiments using *MNIST* and *CIFAR-10* datasets with clients number n = 100 and malicious client ratio m = 40%. The label-flipping attack is deployed with target label 7. When all the reference vectors are stacked in order, we can construct a reference matrix with dimension $r \times 500$ for LeNet and $r \times 5120$ for CNN. If the matrix is an identity matrix, our design is equivalent to the centralized computation of the cosine similarity among the clients in CONTRA; we use such a setting as baseline (in the last row of table II). We show the results in Table II. For *MNIST* dataset, as we can see, when the number of reference vectors is 10, MA is 92.03%. The baseline result in the last row has an MA of 92.30%. When the number of reference vectors is 50, our design achieves an MA of 92.30%, which is the same as CONTRA. Considering that 40% clients are malicious in the experiments, the MAs attained by our design is acceptable. As the number of reference vectors increases, MA is stably around 92%, which indicates that using 50 reference vectors is an appropriate choice for the *MNIST* dataset. For the *CIFAR-10* dataset, when the number of reference vectors is 50, the resulting MA is the best among all settings that we have tested with. Therefore, in the following experiments with n = 100, we set the number of reference vectors r to 50 by default.

[1]$r = 500$ for MNIST or $r = 5120$ for CIFAR-10.

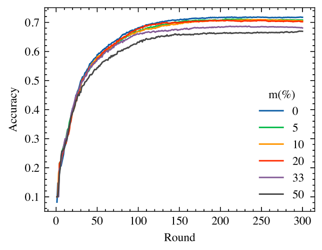
Table III: Performance of our proposed design under label-flipping attacks: MA and ASR at model convergence

	MNIST		*CIFAR-10*	
m(%)	MA(%)	ASR(%)	MA(%)	ASR(%)
5	92.12	0.60	70.81	2.90
10	92.41	0.69	70.47	1.52
20	92.47	0.66	70.22	1.03
33	91.84	0.73	68.07	1.18
50	91.55	0.52	66.93	2.93

2) Impact of Label-flipping Attacks: We evaluate our proposed design under the label-flipping attacks. In the experiments, we use 100 clients for the *MNIST* and the *CIFAR-10* datasets respectively. The ratio of malicious clients m varies from 5% to 50%. The result is shown in Table III. For the *MNIST* dataset, our design outperforms CONTRA [20] in all setting. First, our design achieves 92.12% MA with $m = 5\%$, which is very close to the MA when there is no attack. By contrast, CONTRA only achieved 89.52% MA with $m = 5\%$. Second, the MA attained by our design decreased only by 0.67% when m increases from 5% to 50%. However, CONTRA's MA decreases more than 8% when m increases from 5% to 50%. This indicates that our design is more robust against the increasing ratio of malicious clients. Third, CONTRA's ASR is below 0.01%, which is lower than that of our design. When m increases from 5% to 50%, CONTRA's ASR increases to 1.4%, but our design still has a low ASR of 0.52%. On average, our design reaches an overall MA of 92.07% and an ASR of 0.64%. By contrast, CONTRA has an MA of 80.8% and an ASR of 0.9%. Similar trend can be observed for the *CIFAR-10* dataset, where the MA of our design drops to 66.93% when m increases to 50%. However, the MA is above 70% when m is less than 20%. All these results demonstrate that our design is robust and effective against the label-flipping attacks, and has better performance than CONTRA, a state-of-the-art method.



(a) *MNIST.*



(b) *CIFAR-10.*

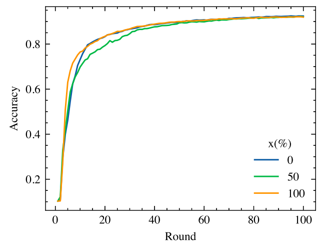
Figure 1: Test accuracy of our proposed design under label-flipping attack w.r.t communication round.

In Figure 1, we show the MA of our design against communication round for both the *MNIST* and the *CIFAR-10* datasets. As we can see, during the early stages, the MA for the no-attack setting (i.e., $m = 0\%$) grows faster than other settings. For example, at communication round 18, the no-attack setting has an MA of 83.60%, but the MA is only 78.86% with $m = 50\%$. At later stages, however, the MA for all settings tend to converge to the same level. For example, at round 70, the MA are 91.03%, 90.06% for settings with $m = 0\%$ and $m = 50\%$ respectively. This indicates that our design needs to accumulate gradients history to calculate similarity between clients and then degrade the learning rates for malicious clients. That is why at later stages after malicious client are detected, our design tends to get updates mostly from honest clients.

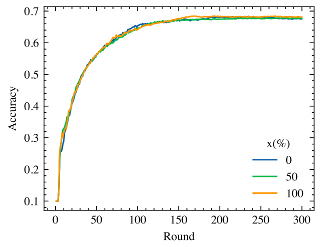
Table IV: Performance comparison of our proposed design under coordinated attacks: MA and ASR at model convergence

	MNIST		*CIFAR-10*	
x(%)	MA(%)	ASR(%)	MA(%)	ASR(%)
0	92.41	0.30	67.58	1.00
50	92.02	1.36	67.73	2.32
100	91.95	0.72	68.09	1.60

3) Impact of Coordinate Attacks: We evaluate our design under the coordinated attacks. Here, the ratio of malicious clients (i.e., m) is fixed at 40%. The experiments are run under three coordinated attacks with sharing rate x set to 0%, 50%, 100% respectively. The results are shown in table IV. As we can see, for the *MNIST* data set, the MA decreases as coordinate rate increases. However, the MA is still above 91.95% even if the coordinate rate increased to 100%. In comparison, the best MA of CONTRA [20] is only 87.04% in all setting under coordinated attacks. For the *CIFAR-10* dataset, the trends are similar. But CONTRA's ASR is lower than ours. Overall, our model outperforms CONTRA in term of MA while CONTRA has lower ASR.



(a) *MNIST.*



(b) *CIFAR-10.*

Figure 2: Test accuracy of our design under coordinated attacks w.r.t communication round.

Figure 2 shows how the MA of our design changes along with the communication round. As we can see, with different settings of parameter x, the MA of our design converge to a high value at a similar pace.

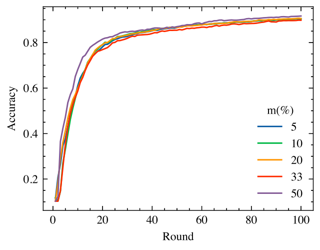
Table V: Performance of our proposed design under backdoor attacks: MA and ASR at model convergence

	MNIST		*CIFAR-10*	
m(%)	MA(%)	ASR(%)	MA(%)	ASR(%)
5	90.36	1.34	69.48	2.45
10	90.42	1.28	70.40	1.92
20	90.55	1.01	70.09	1.66
33	89.75	1.54	69.10	1.77
50	91.56	0.06	66.26	2.17

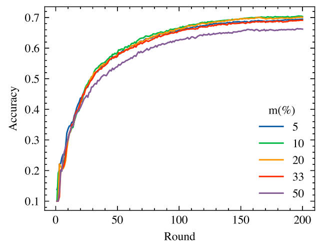
4) Impact of Backdoor Attacks: We also evaluate our design under the backdoor attacks. For backdoor attack, we change a single white pixel in the bottom-right corner of the figure [28] and the corresponding label is flipped to 7. When a figure with backdoor pattern inserted is incorrectly classified as 7, then the attack is considered as a success. For the test data, we inserts backdoor pattern to 20% of the pictures.

The evaluation result is reported in Table V. As we can see, when the ratio of malicious clinets, i.e., m, increases from 5% to 20%, the MA of our design is still above 90%. When m increases to 33%, the MA drops to 89.75%. However, when m increases to 50%, the MA increases to 91.56% and the ASR decreases to below 0.1%, which is lowest among all the settings. This can be explained as follows. Our design tends to be strict in detecting malicious clients; that is, it sometimes incorrectly penalized honest clients (which are detected as malicious). When m is low, this does impact the MA more compared to the situations where m is high. Our design outperforms CONTRA [20] both in MA and ASR with the *MNIST* dataset. With the *CIFAR-10* dataset, our design also attains high MA and low ASR.

Figure 3 shows the MA of our design against communication round. As shown in the figure, at the early stage,



(a) *MNIST.*



(b) *CIFAR-10.*

Figure 3: Test accuracy of our proposed design under backdoor attack w.r.t communication round.

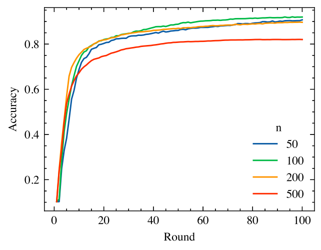


Figure 4: Test accuracy of our proposed design under various numbers of clients w.r.t communication round.

the MA attained by our design varies with various ratio of malicious clients. However, as time passes by, the settings converge to similar values of the MA. This indicates that our design detect/contain the malicious clients successfully in these settings. Thus it is able to defend against backdoor attack.

Table VI: Performance of our proposed design under label-flipping attack with various number of clients and fixed percent of malicious clients(33%)

n	*MNIST* MA(%)	ASR(%)
50	90.72	0.88
100	91.97	1.04
200	89.64	0.52
500	82.02	0.07

5) Impact of Client Number: To find out how the number of clients, i.e., n, influences the performance our our proposed design, we simulate multiple settings where the number of clients n are set to 50, 100, 200, and 500 and the ratio of malicious clients, i.e., m, is fixed as 33%. Experiments are run using the *MNIST* dataset and the results are shown in table VI. As shown in the table, the MA decreases as n increases, but the ASR is still below 2%. We traced the changes of learning rates during the training process and found that most of the malicious clients can be detected, but our design also incorrectly penalized a part of honest clients.

Figure 4 shows the MA of our design with various number of clients against communication round for the *MNIST* dataset. As we can see, at the early stage, the MA with $n = 500$ is higher compared to the MA with $n = 500$. But after 10 rounds, the trend is reversed. This is mainly because the accuracy for estimating cosine similarity in our design is not accurate and depends on the number of reference vectors. In these experiments, we use 50 vectors which was tuned for $n = 100$. When there are more clients joining the training, more honest clients are incorrectly penalized. As a result, the MA with $n = 500$ is the lowest among these settings. Hence, we suggest that the number of reference vectors should be increased as the number of client increases.

6) Computation Overhead for Detecting Misbehaving Edge Servers: We run experiments on a machine with a 8 core processor and 32 GB RAM, and evaluate the time for a client to conduct local training and computing commitments, and for the central server to verify if the edge servers behave normally, at each round. Here, there are 100 clients and five edge servers. Two datasets are used. For the MNIST dataset, LeNet is used for training, and the model to be trained has 21840 trainable parameters. Among them, 500 parameters for the final layer are used for similarity analysis (i.e., $\mathbf{G}_{i,j}^{(t,0)}$ in the scheme description) and so the updates to all of them are reported; for the rest parameters, each client picks only 500 the most significant parameter updates to report. For the CIFAR-10 dataset, a CNN model is used with 5852170 parameters. Among them, 5120 parameters for the final layer are used for similarity analysis; for the rest, each client picks 5120 the most significant parameter updates to report. The results are reported in Table VII.

Table VII: The latency (in second) for the training, commitment generation and verification.

Dataset	T^{comit}	T^{verify}	T^{T}
MNIST	8.20	35.70	0.56
CIFAR-10	80.36	268.06	3.33

For the MNIST dataset, by average each client spends 0.56 seconds for training (denoted as T^T in the table) and 8.20 seconds for computing commitments, while the central server spends 35.60 seconds for verifying the behavior of edge servers. For the CIFAR-10 dataset, where the model to be trained is much larger, each spends 3.33 seconds for training, 80.36 seconds for verifying the behavior of edge servers. As we can see, the incurred overhead is not high, especially when the model side is not too large.

VII. Related Works

Many detection and defense approaches were developed to address data poisoning attacks in FL. Byzantine robust aggregation is one of them. Compared with traditional aggregation schemes, this approach uses different rules to aggregate the model updates reported by clients by estimating a robust mean from local updates. To some extent, this approach decreases the impact of malicious clients on aggregated central model. There are a few methods proposed based on the Byzantine robust aggregation approach, such as median aggregation [15], trimmed mean aggregation [16], and Krum aggregation [17]. Clustering-based detection is another approach to address data poisoning attacks in federated learning. This approach first checks the model updates from the clients and then use clustering to distinguish malicious from honest clients, and aggregate local updates according to the clustering results. There are a few common clustering methods used in such approach, for example, principal component analysis [29] and k-means [30]. What's more, behavior-based defensing scheme is also a key approach for defense. This approach detects malicious clients by measuring the behavioral difference between malicious and honest clients using the metrics such as Euclidean distance or cosine similarity. For example, the adaptive federated averaging scheme [18] implements this approach. The scheme detects malicious clients by measuring the cosine similarity between the gradients of every clients' local updates and the average of the gradients of all the clients' local updates. FLTrust [19] is another scheme falling in the same category. FLTrust detects malicious clients by measuring the cosine similarity between a client's gradient and the server's gradient where the server is trained with a small clean root dataset. Recently, Foolsgold [21] is developed to also detect malicious clients by measuring the cosine similarities between a client's gradient history and the server's gradient. After measuring the cosine similarities, Foolsgold downgrades the impact of potentially malicious clients on updating the global model by decreasing their learning rates accordingly. Foolsgold performs well when the training dataset is i.i.d. But Foolsgold does not perform well and non-malicious client could be incorrectly penalized when the training data is non-i.i.d [20]. As a solution to this issue, CONTRA [20] is proposed, which detects malicious clients by measuring cosine similarity between the clients and then dynamically promotes or penalizes each individual client based on its reputation score. The central server calculates cosine similarity between pairs of clients, and assigns each client corresponding reputation score and learning rates based on reputation scores. For target attack, malicious clients tend to have the same global goal; thus CONTRA is able to detect them and then dynamically change their learning rates. As a result, the attained model accuracy and attack success rate are both promising.

However, all of the above approaches were proposed for the two-tier architecture with a cloud-based server as FL central server, where the cloud server can collect and process gradients from all the clients, and in the meantime it has excessive communication overhead. The Client-Edge-Cloud FL [10], also known as HierFL, is a different type of system developed to come over the long latency in the above cloud-based FL systems. In HierFL, the cloud server collects and processes the gradients aggregated at edge servers, and each edge server collects and aggregates model updates from the clients connected to it. As a result, HierFL is able to get better model accuracy with less communication overhead. However, there is no any security measures in HierFL to defend against data poisoning attacks. Thus, we propose in this paper a new scheme to take advantage of the potential performance benefits of Client-Edge-Cloud FL and in the meantime provide the robustness against data poisoning attacks. Results from extensive experiments on standard machine learning datasets indicate that our design is able to defend against various data poisoning attacks, and achieve high MA and low ASR in the same time. As a key component of our proposed scheme, the decentralized mechanism for estimating cosine similarity values among clients has also been proved to be effective under different experimental settings, compared with the state-of-the-art defense mechanisms.

VIII. Conclusions And Future Works

In this paper, We proposed a distributed framework for a three-tier federated learning system to efficiently detect/mitigate the clients' poisoning attacks, through distributed collaboration between the central and the middle-tier edge servers, and to detect attacks that may be launched by compromised middle-tier edge servers, through collaboration between the central server and the clients. Extensive experiments have been conducted, and the results show that our proposed design can effectively detect/mitigate clients' attacks and deliver a level of model training accuracy comparable to or even better than the state-of-the-art centralized scheme, and it incurs moderate overheads for detecting misbehaving edge servers.

In the future, we will enhance and extend the proposed design to also protect the privacy of clients' gradients from the edge servers, while still providing the security protection against malicious clients and edge servers.

IX. Privacy-preserving Detection and Mitigation of Poisoning Attacks

We consider a federated learning system with three tiers, federated learning (FL) servers, edge servers and clients.

A. Preparation

Let c be the total number of clients connected to an edge server, and n be the size (i.e., the number of elements) of a gradient vector reported by each client.

To facilitate noise-based additive homomorphism, all arithmetic operations are modular and over integers. Let every element in a gradient be in the range of $\{0, \cdots, g-1\}$, and the weight (used in weighted averaging of the clients' gradients) assigned to each client be in the range of $\{0, \cdots, w-1\}$.

Note that, in a practical system, the gradient elements could be fractions when they are used in learning; to be compatible with our design, each client needs to cast the gradient elements to the afore-specified range properly. Similarly for the weights, the FL server should cast the real weights to the specified range for the clients and edge servers to conduct weighed averaging, and the results from the averaging are then converted back; for example, if the real weights are $\{0.1, 0.2, \cdots, 1.0\}$, they can be converted to $\{1, \cdots, 10\}$ and later the averaging results should be divided by 10 to get the real average values.

The modulus number used in the arithmetic modular operations is picked as a prime number denoted as p, which should be larger than $g \cdot w \cdot c$; in other the size of p is at least $log(g \cdot w \cdot c)$ bits.

For each round t of client-server FL interaction, the FL server randomly picks two secret numbers:

- $c_{0,t}$ and $c_{1,t}$,

and also secretly shares with each client $\mathcal{C}_i$:

- $a_{i,t}$;
- $b_{i,t}$;
- noises $\gamma_{i,t,j}$ and $\xi_{i,t,j}$ for $j = 0, \cdots, n-1$ where n is the size (i.e., the number of elements) of each gradient vector reported by each client.

Here, all the above numbers should be in the range of $\{0, p-1\}$.

Also, letting $w_{i,t}$ be the weight that the FL server assigns to client $\mathcal{C}_i$ for weighed aggregation (averaging) of gradients, the following equality should be met:

$$a_{i,t} \cdot c_{0,t} + b_{i,t} \cdot c_{1,t} = w_{i,t} \ (mod\ p). \tag{17}$$

In practice, $w_{i,t}$ is determined based on the attack mitigation mechanisms (e.g., as in CONTRA), $c_{0,t}$ and $c_{1,t}$ are integers randomly picked from $\{0, \cdots, p-1\}$ for all clients, $a_{i,t}$ is picked for client $\mathcal{C}_i$ from $\{1, \cdots, p-1\}$ randomly; then, $b_{i,t}$ can be derived as

$$(w_{i,t} - a_{i,t} \cdot c_{0,t}) \cdot c_{1,t}^{-1} \ (mod\ p). \tag{18}$$

B. Reporting from Each Client

Denote the gradient vector of client $\mathcal{C}_i$ as $\mathbf{G}_{i,t} = \{g_{i,t,j}\}$ for $j = 0, \cdots, n-1$.

For each $g_{i,t,j}$, the client perturbs it into the following numbers:

$$\hat{g}_{i,t,j} = a_{i,t} \cdot g_{i,t,j} + \gamma_{i,t,j} \ (mod\ p), \tag{19}$$

and

$$\hat{g'}_{i,t,j} = b_{i,t} \cdot g_{i,t,j} + \xi_{i,t,j} \ (mod\ p). \tag{20}$$

Then the two numbers are reported to the edge server that the client connects with.

C. Aggregation at Edge Server and FL Server

The edge server can sum up the received vectors for aggregation:

$$\hat{g}_{t,j} = \sum_{i=0}^{c-1} \hat{g}_{i,t,j} \ (mod\ p), \tag{21}$$

and

$$\hat{g'}_{t,j} = \sum_{i=0}^{c-1} \hat{g'}_{i,t,j} \ (mod\ p). \tag{22}$$

The FL server can derive the sum of all the clients' noises and remove it from the aggregated reports to get the overall model gradient vector:

$$(\hat{g}_{t,j} - \sum_{i=0}^{c-1} \gamma_{i,t,j}) \cdot c_{0,t} + (\hat{g'}_{t,j} - \sum_{i=0}^{c-1} \xi_{i,t,j}) \cdot c_{1,t} (mod\ p). \tag{23}$$

According to equation (17), the above computation should result in the weighed sum of the clients' gradient element at index j.

D. Attack Detection Based on Reference Vectors

The FL server and the edge server share m reference vectors, denoted as $\hat{r}_k$ for $k = 0, \cdots, m-1$. The vectors should be diverse.

The set of the these vectors can be denoted as a matrix R with dimension $n \times m$, i.e., m of n-element vectors.

The edge server receives two vectors from each client $\mathcal{C}_i$:

$$\hat{g}_{i,t} = (\hat{g}_{i,t,0} \cdots \hat{g}_{i,t,n-1}), \tag{24}$$

and

$$\hat{g}'_{i,t} = (\hat{g}'_{i,t,0} \cdots \hat{g}'_{i,t,n-1}). \tag{25}$$

Such vectors from all the clients form two matrices:

$$G_t = (\hat{g}_{0,t} \cdots \hat{g}_{c-1,t})^T, \tag{26}$$

and

$$G'_t = (\hat{g}'_{0,t} \cdots \hat{g}'_{c-1,t})^T. \tag{27}$$

Note that, each of these matrices has the dimension of $c \times n$, where c is the number of clients and n is the number of elements at each gradient.

The edge server multiplies each received report by the reference vectors and report the results to the FL server. Note that, as the reports from clients are perturbed with random noises, their gradients are protected. The FL server then removes the noises from the vectors received from the edge server.

Specifically, the edge server computes

$$G_t \times R \tag{28}$$

and sends it to the FL server.

The FL server computes

$$\Gamma_t \times R, \tag{29}$$

where Γ_t is the $c \times n$ matrix of all the clients' noises $\gamma_{i,t,j}$ for $i = 0, \cdots, c-1$ and $j = 0, \cdots, n-1$. Then, it computes matrix

$$M_t = (G_t \times R - \Gamma_t \times R). \tag{30}$$

For each row of the matrix, denoted as $M_{t,i}$, which corresponds to client $\mathcal{C}_i$, it computes

$$\frac{1}{a_{i,t}} \cdot M_{t,i} \tag{31}$$

to get a compressed version of gradient reported from client $\mathcal{C}_i$. Such gradients can then be used for follow-up processing based on schemes such as CONTRA.

E. Discussion

The above design works when the FL server and the edge server do not collude, and each party is semi-honest.

To make each party semi-honest, TEE can be used for protecting integrity and endorsing its computation result. The TEE only needs to assure the digital signature scheme to be secure against side-channel attacks.

References

[1] A. Act, "Health insurance portability and accountability act of 1996," *Public Law 104, 191*, 1996.

[2] G. Regulation, "egulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46," *Official Journal of the European Union (OJ) 59(1-88), 294*, 2016.

[3] K. Mathews and C. Bowman, "The california consumer privacy act of 2018," 2018.

[4] J. Konecny, H. McMahan, F. Yu, P. Richtarik, A. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv:1610.05492*, 2016.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Arcas, "Communication-efficient learning of deep networks from decentralized data," *Artificial Intelligence and Statistics*, 2017.

[6] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv:1811.04017*, 2018.

[7] G. Kaissis, M. Makowski, D. Ruckert, and R. Braren, "Secure, privacy-preserving and federated machine learning in medical imaging," *Nature Machine Intelligence*, 2020.

[8] M. Sheller and et al., "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Sci. Rep.*, vol. 10, no. 1, 2020.

[9] D. Ye, R. Yu, M. Pan, and Z. Han, "Federated learning in vehicular edge computing: a selective model aggregation approach," *IEEE Access*, 2020.

[10] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. IEEE, Jun. 2020. [Online]. Available: https://doi.org/10.1109/icc40277.2020.9148862

[11] V. Tolpegin, S. Truex, M. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," *ESORICS*, 2020.

[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *Proceedings of Machine Learning Research (PMLR)*, 2020.

[13] A.N., S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," *International Conference on Machine Learning (PMLR)*, 2019.

[14] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," *USENIX Security Symposium*, 2020.

[15] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," in *PERV*, 2017.

[16] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," *ArXiv*, vol. abs/1803.01498, 2018.

[17] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NIPS*, 2017.

[18] L. Muñoz-González, K. T. Co, and E. C. Lupu, "Byzantine-robust federated machine learning through adaptive model averaging," *ArXiv*, vol. abs/1909.05125, 2019.

[19] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping." in *Network and Distributed System Security Symposium*, 2021.

[20] S. M. Awan, B. Luo, and F. Li, "Contra: Defending against poisoning attacks in federated learning," in *European Symposium on Research in Computer Security*, 2021.

[21] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "The Limitations of Federated Learning in Sybil Settings," in *Symposium on Research in Attacks, Intrusion, and Defenses*, ser. RAID, 2020.

[22] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[23] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[24] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, 1998.

[26] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[27] H. B. McMahan, E. Moore, D. Ramage, and S. Hampson, "Communication-efficient learning of deep networks from decentralized data," *Artificial Intelligence and Statistics*, pp. 1273–1282, April. 2017.

[28] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *ArXiv*, vol. abs/1708.06733, 2017.

[29] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*, 2020.

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