FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping

Xiaoyu Cao*[1], Minghong Fang*[2], Jia Liu[2], Neil Zhenqiang Gong[1]
[1] Duke University, {xiaoyu.cao, neil.gong}@duke.edu
[2] The Ohio State University, {fang.841, liu.1736}@osu.edu

***Abstract*—Byzantine-robust federated learning aims to enable a service provider to learn an accurate global model when a bounded number of clients are malicious. The key idea of existing Byzantine-robust federated learning methods is that the service provider performs statistical analysis among the clients' local model updates and removes suspicious ones, before aggregating them to update the global model. However, malicious clients can still corrupt the global models in these methods via sending carefully crafted local model updates to the service provider. The fundamental reason is that there is no root of trust in existing federated learning methods, i.e., from the service provider's perspective, every client could be malicious.**

In this work, we bridge the gap via proposing *FLTrust*, a new federated learning method in which the service provider itself bootstraps trust. In particular, the service provider itself collects a clean small training dataset (called *root dataset*) for the learning task and the service provider maintains a model (called *server model*) based on it to bootstrap trust. In each iteration, the service provider first assigns a trust score to each local model update from the clients, where a local model update has a lower trust score if its direction deviates more from the direction of the server model update. Then, the service provider normalizes the magnitudes of the local model updates such that they lie in the same hyper-sphere as the server model update in the vector space. Our normalization limits the impact of malicious local model updates with large magnitudes. Finally, the service provider computes the average of the normalized local model updates weighted by their trust scores as a global model update, which is used to update the global model. Our extensive evaluations on six datasets from different domains show that our FLTrust is secure against both existing attacks and strong adaptive attacks. For instance, using a root dataset with less than 100 examples, FLTrust under adaptive attacks with 40%-60% of malicious clients can still train global models that are as accurate as the global models trained by FedAvg under no attacks, where FedAvg is a popular method in non-adversarial settings.

I. Introduction

Federated learning (FL) [22], [28] is an emerging distributed learning paradigm on decentralized data. In FL, there are multiple clients (e.g., smartphones, IoT devices, and edge devices) and a service provider (e.g., Google, Apple, and IBM). Each client holds a local training dataset; and the service provider enables the clients to jointly learn a model (called *global model*) without sharing their raw local training data with the service provider. Due to its potential promise of protecting private/proprietary client data, particularly in the age of emerging privacy regulations such as General Data Protection Regulation (GDPR), FL has been deployed by high-profile companies. For instance, Google has deployed FL for next-word prediction on Android Gboard [1]; WeBank uses FL for credit risk prediction [3]; and more than 10 leading pharmaceutical companies leverage FL for drug discovery in the project MELLODDY [2]. Roughly speaking, FL iteratively performs the following three steps: the server provided by the service provider sends the current global model to the clients or a selected subset of them; each selected client trains a model (called *local model*) via fine-tuning the global model using its own local training data and sends the local model updates back to the server[1]; and the server aggregates the local model updates to be a *global model update* according to an *aggregation rule* and uses it to update the global model. For instance, FedAvg [28], a popular FL method in non-adversarial settings developed by Google, computes the average of the local model updates weighted by the sizes of local training datasets as the global model update.

However, due to its distributed nature, FL is vulnerable to adversarial manipulations on malicious clients, which could be fake clients injected by an attacker or genuine clients compromised by an attacker. For instance, malicious clients can corrupt the global model via poisoning their local training data (known as *data poisoning attacks* [8], [32]) or their local model updates sent to the server (called *local model poisoning attacks* [15], [7], [5], [43]). The corrupted global model makes incorrect predictions for a large number of testing examples indiscriminately (called *untargeted attacks*) [15], or it predicts attacker-chosen target labels for attacker-chosen target testing examples while the predicted labels for other non-target testing examples are unaffected (called *targeted attacks*) [5], [7], [43]. For instance, the global model in FedAvg can be arbitrarily manipulated by a single malicious client [9], [48].

Byzantine-robust FL methods [9], [12], [29], [46], [48] aim to address malicious clients. The goal therein is to learn an accurate global model when a bounded number of clients are malicious. Their key idea is to leverage *Byzantine-robust aggregation rules*, which essentially compare the clients' local model updates and remove statistical outliers before using them to update the global model. For instance, Median [48] computes the coordinate-wise median of the clients' local

*Equal contribution.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24434
www.ndss-symposium.org

[1] It is algorithmically equivalent to send local models instead of their updates to the server.

model updates as the global model update. However, recent studies [7], [15] showed that existing Byzantine-robust FL methods are still vulnerable to local model poisoning attacks on malicious clients. The fundamental reason is that they have no root of trust. Specifically, from the server's perspective, every client could be malicious, providing no root of trust for the server to decide which local model updates are suspicious.

Our work: In this work, we propose a new Byzantine-robust FL method called *FLTrust*. Instead of completely relying on the local model updates from clients, the server itself bootstraps trust in FLTrust. Specifically, the service provider manually collects a small clean training dataset (called *root dataset*) for the learning task. The server maintains a model (called *server model*) for the root dataset just like how a client maintains a local model. In each iteration, the server updates the global model by considering both its server model update and the clients' local model updates.

Our new Byzantine-robust aggregation rule: Specifically, we design a new Byzantine-robust aggregation rule in FLTrust to incorporate the root of trust. A model update can be viewed as a vector, which is characterized by its *direction* and *magnitude*. An attacker can manipulate both the directions and magnitudes of the local model updates on the malicious clients. Therefore, our aggregation rule takes both the directions and magnitudes into considerations when computing the global model update. Specifically, the server first assigns a trust score (TS) to a local model update, where the trust score is larger if the direction of the local model update is more similar to that of the server model update. Formally, we use the *cosine similarity* between a local model update and the server model update to measure the similarity of their directions. However, the cosine similarity alone is insufficient because a local model update, whose cosine similarity score is negative, can still have a negative impact on the aggregated global model update. Therefore, we further clip the cosine similarity score using the popular ReLU operation. The ReLU-clipped cosine similarity is our trust score. Then, FLTrust normalizes each local model update by scaling it to have the same magnitude as the server model update. Such normalization essentially projects each local model update to the same hyper-sphere where the server model update lies in the vector space, which limits the impact of the poisoned local model updates with large magnitudes. Finally, FLTrust computes the average of the normalized local model updates weighted by their trust scores as the global model update, which is used to update the global model.

FLTrust can defend against existing attacks: We perform extensive empirical evaluation on six datasets from different domains, including five image classification datasets (MNIST-0.1, MNIST-0.5, Fashion-MNIST, CIFAR-10, and CH-MNIST) and a smartphone-based human activity recognition dataset (Human Activity Recognition). We compare FLTrust with multiple existing Byzantine-robust FL methods including Krum [9], Trimmed mean [48], and Median [48]. Moreover, we evaluate multiple poisoning attacks including label flipping attack (a data poisoning attack), Krum attack and Trim attack (untargeted local model poisoning attacks) [15], as well as Scaling attack[2] (targeted local model poisoning attack) [5]. Our results show that FLTrust is secure against these attacks even if the root dataset has less than 100 training examples, while existing Byzantine-robust FL methods are vulnerable to them or a subset of them. For instance, a CNN global model learnt using FLTrust has a testing error rate of 0.04 under all the evaluated attacks on MNIST-0.1. However, the Krum attack can increase the testing error rate of the CNN global model learnt by Krum from 0.10 to 0.90. Moreover, we treat FedAvg under no attacks as a baseline and compare our FLTrust under attacks with it. Our results show that FLTrust under attacks achieves similar testing error rates to FedAvg under no attacks. We also study different variants of FLTrust and the impact of different system parameters on FLTrust. For instance, our results show that FLTrust works well once the root dataset distribution does not diverge too much from the overall training data distribution of the learning task.

FLTrust can defend against adaptive attacks: An attacker can adapt its attack to FLTrust. Therefore, we also evaluate FLTrust against adaptive attacks. Specifically, Fang et al. [15] proposed a general framework of local model poisoning attacks, which can be applied to optimize the attacks for any given aggregation rule. An attacker can substitute the aggregation rule of FLTrust into the framework and obtain an adaptive attack that is particularly optimized against FLTrust. Our empirical results show that FLTrust is still robust against such adaptive attacks. For instance, even when 60% of the clients are malicious and collude with each other, FLTrust can still learn a CNN global model with testing error rate 0.04 for MNIST-0.1. This testing error rate is the same as that of the CNN global model learnt by FedAvg under no attacks.

Our contributions can be summarized as follows:

- We propose the first federated learning method FLTrust that bootstraps trust to achieve Byzantine robustness against malicious clients.
- We empirically evaluate FLTrust against existing attacks. Our results show that FLTrust can defend against them.
- We design adaptive attacks against FLTrust and evaluate their performance. Our results show that FLTrust is also robust against the adaptive attacks.

II. Background and Related Work

A. Background on Federated Learning (FL)

Suppose we have n clients and each client has a local training dataset $D_i, i = 1, 2, \cdots, n$. We use $D = \bigcup_{i=1}^{n} D_i$ to denote the joint training data. Each training example in D is drawn from an unknown distribution $\mathcal{X}$. The clients aim to collaboratively learn a shared global model with the help of a service provider. The optimal global model $\boldsymbol{w}^*$ is a solution to the following optimization problem: $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} F(\boldsymbol{w})$, where $F(\boldsymbol{w}) = \mathbb{E}_{D\sim\mathcal{X}}[f(D, \boldsymbol{w})]$ is the expectation of the empirical loss $f(D, \boldsymbol{w})$ on the joint training dataset D. Since the expectation is hard to evaluate, the global model is often learnt via minimizing the empirical loss in practice, i.e., $\arg\min_{\boldsymbol{w}} f(D, \boldsymbol{w})$ is the learnt global model. Specifically, each client maintains a local model for its local training dataset. Moreover, a service provider's server maintains the global model via aggregating the local model updates from

[2]The Scaling attack is also known as a backdoor attack.

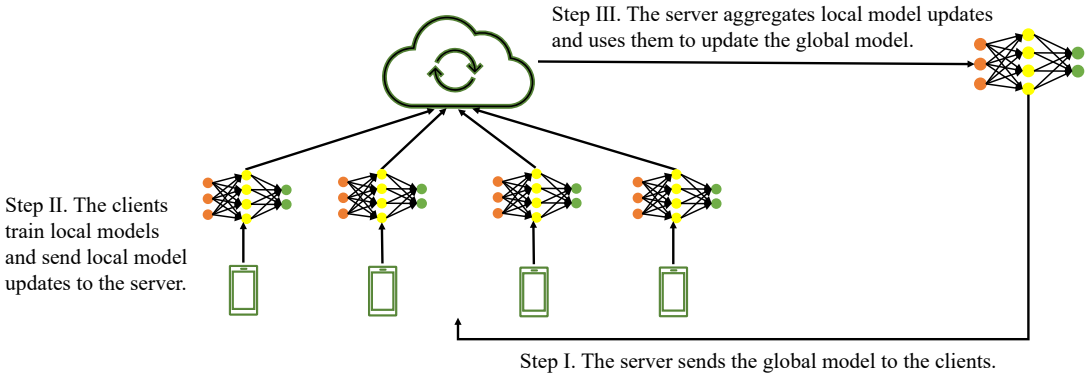


Fig. 1: Illustration of the three steps in FL.

the clients. Specifically, FL iteratively performs the following three steps (illustrated in Figure 1):

- **Step I: Synchronizing the global model with clients.** The server sends the current global model $\boldsymbol{w}$ to the clients or a subset of them.
- **Step II: Training local models.** Each client trains a local model via fine-tuning the global model using its local training dataset. Formally, the ith client solves the optimization problem $\min_{\boldsymbol{w}_i} f(D_i, \boldsymbol{w}_i)$, where $\boldsymbol{w}_i$ is the client's local model. In particular, the client initializes its local model as the global model and uses stochastic gradient descent to update the local model for one or more iterations. Then, each client sends its local model update $\boldsymbol{g}_i = \boldsymbol{w}_i - \boldsymbol{w}$ (i.e., the difference between its local model and the current global model) to the server.
- **Step III: Updating the global model via aggregating the local model updates.** The server computes a *global model update* $\boldsymbol{g}$ via aggregating the local model updates according to some *aggregation rule*. Then, the server updates the global model using the global model update, i.e., $\boldsymbol{w} = \boldsymbol{w} - \alpha \cdot \boldsymbol{g}$, where α is the global learning rate.

The aggregation rule plays a key role in FL. Different FL methods essentially use different aggregation rules. Next, we discuss popular aggregation rules.

1) FedAvg: FedAvg [28] was proposed by Google. FedAvg computes the average of the clients' local model updates as the global model update, where each client is weighted by its number of training examples. Formally, $\boldsymbol{g} = \sum_{i=1}^{n} \frac{|D_i|}{N} \boldsymbol{g}_i$, where $|D_i|$ is the local training dataset size on the ith client and N is the total number of training examples. FedAvg is the state-of-the-art FL method in non-adversarial settings. However, the global model in FedAvg can be arbitrarily manipulated by a single malicious client [9], [48].

2) Byzantine-robust Aggregation Rules: Most Byzantine-robust FL methods use Byzantine-robust aggregation rules (see, e.g., [9], [12], [29], [34], [44], [47], [48], [31]) that aim to tolerate Byzantine client failures. One exception is that Li et al. [26] introduced a norm regularization term into the loss function. Examples of Byzantine-robust aggregation rules include Krum [9], Trimmed mean [48], and Median [48], which we discuss next.

Krum [9]: Krum selects one of the n local model updates in each iteration as the global model update based on a square-distance score. Suppose at most f clients are malicious. The score for the ith client is computed as follows:

$$s_i = \sum_{\boldsymbol{g}_j \in \Gamma_{i,n-f-2}} \|\boldsymbol{g}_j - \boldsymbol{g}_i\|_2^2, \tag{1}$$

where $\Gamma_{i,n-f-2}$ is the set of $n-f-2$ local model updates that have the smallest Euclidean distance to $\boldsymbol{g}_i$. The local model update of the client with the minimal score will be chosen as the global model update to update the global model.

Trimmed Mean (Trim-mean) [48]: Trimmed mean is a coordinate-wise aggregation rule that considers each model parameter individually. For each model parameter, the server collects its values in all local model updates and sorts them. Given a trim parameter $k < \frac{n}{2}$, the server removes the largest k and the smallest k values, and then computes the mean of the remaining $n-2k$ values as the value of the corresponding parameter in the global model update. The trim parameter k should be at least the number of malicious clients to make Trim-mean robust. In other words, Trim-mean can tolerate less than 50% of malicious clients.

Median [48]: Median is another coordinate-wise aggregation rule. Like Trim-mean, in Median, the server also sorts the values of each individual parameter in all local model updates. Instead of using the mean value after trim, Median considers the median value of each parameter as the corresponding parameter value in the global model update.

Existing FL methods suffer from a key limitation: they are vulnerable to sophisticated local model poisoning attacks on malicious clients, which we discuss in the next section.

B. Poisoning Attacks to Federated Learning

Poisoning attacks generally refer to attacking the training phase of machine learning. One category of poisoning attacks called *data poisoning attacks* aim to pollute the training data to corrupt the learnt model. Data poisoning attacks have been demonstrated to many machine learning systems such as spam detection [32], [35], SVM [8], recommender systems [16],

[17], [25], [45], neural networks [11], [18], [27], [30], [36], [37], and graph-based methods [20], [41], [49], as well as distributed privacy-preserving data analytics [10], [14]. FL is also vulnerable to data poisoning attacks [38], i.e., malicious clients can corrupt the global model via modifying, adding, and/or deleting examples in their local training datasets. For instance, a data poisoning attack known as *label flipping attack* changes the labels of the training examples on malicious clients while keeping their features unchanged.

Moreover, unlike centralized learning, FL is further vulnerable to *local model poisoning attacks* [5], [7], [15], [43], in which the malicious clients poison the local models or their updates sent to the server. Depending on the attacker's goal, local model poisoning attacks can be categorized into *untargeted attacks* [15] and *targeted attacks* [5], [7], [43]. Untargeted attacks aim to corrupt the global model such that it makes incorrect predictions for a large number of testing examples indiscriminately, i.e., the testing error rate is high. Targeted attacks aim to corrupt the global model such that it predicts attacker-chosen target labels for attacker-chosen target testing examples while the predicted labels for other non-target testing examples are unaffected.

Note that any data poisoning attack can be transformed to a local model poisoning attack, i.e., we can compute the local model update on a malicious client's poisoned local training dataset and treat it as the poisoned local model update. Moreover, recent studies [7], [15] showed that local model poisoning attacks are more effective than data poisoning attacks against FL. Therefore, we focus on local model poisoning attacks in this work. Next, we discuss two state-of-the-art untargeted attacks (i.e., Krum attack and Trim attack) [15] and one targeted attack (i.e., Scaling attack) [5].

Krum attack and Trim attack [15]: Fang et al. [15] proposed a general framework for local model poisoning attacks, which can be applied to optimize the attacks for any given aggregation rule. Assuming the global model update without attack is $\boldsymbol{g}$, Fang et al. [15] formulate the attack as an optimization problem that aims to change the global model update the most along the opposite direction of $\boldsymbol{g}$, by optimizing the poisoned local model updates sent from the malicious clients to the server. Different aggregation rules lead to different instantiations of the optimization problem. Fang et al. applied the framework to optimize local model poisoning attacks for Krum (called Krum attack) as well as Trim-mean and Median (called Trim attack).

Scaling attack [5]: This attack aims to corrupt the global model to predict attacker-chosen target labels for attacker-chosen target testing examples, while the predicted labels for other testing examples are unaffected (i.e., the normal testing error rate remains the same). For instance, the attacker-chosen target testing examples can be normal testing examples embedded with a predefined backdoor trigger (e.g., a logo, a specific feature pattern). To achieve the goal, the Scaling attack adds trigger-embedded training examples with the attacker-chosen target label to the local training data of malicious clients. The local model updates on malicious clients are then computed based on the local training datasets augmented with the trigger-embedded examples. However, the poisoned local model updates may have limited impact on the global model update because it is aggregated over all clients' local model updates. For instance, in FedAvg, the effect of the attack will be diluted after the averaging [5]. Therefore, the attack further scales the poisoned local model updates on malicious clients by a factor that is much larger than 1. The scaled poisoned local model updates are then sent to the server.

III. Problem Setup

Attack model: We follow the attack model in previous works [5], [7], [15]. Specifically, an attacker controls some malicious clients, which can be fake clients injected by the attacker or genuine ones compromised by the attacker. However, the attacker does not compromise the server. The malicious clients can send arbitrary local model updates to the server in each iteration of the FL training process. Typically, an attacker has the following partial knowledge about an FL system: local training data and local model updates on the malicious clients, loss function, and learning rate. We notice that the Scaling attack [5] only requires such partial knowledge. The Krum and Trim attacks [15] are also applicable in this partial-knowledge setting. However, they are stronger in the full-knowledge setting [15], where the attacker knows everything about the FL training process, including the local training data and local model updates on all clients in each iteration, as well as the FL's aggregation rule. Therefore, we consider such full-knowledge setting to show that our method can defend against strong attacks. Moreover, the attacker can perform adaptive attacks to FLTrust, which we discuss in Section V.

Defense goals: We aim to design an FL method that achieves Byzantine robustness against malicious clients without sacrificing the fidelity and efficiency. In particular, we treat FedAvg under no attacks as a baseline to discuss fidelity and efficiency, i.e., our method should be robust against malicious clients while being as accurate and efficient as FedAvg under no attacks. Specifically, we aim to design a Byzantine-robust FL method that achieves the following defense goals:

- **Fidelity.** The method should not sacrifice the classification accuracy of the global model when there is no attack. In particular, under no attacks, the method should be able to learn a global model that is as accurate as the global model learnt by FedAvg, a popular FL method in non-adversarial settings.
- **Robustness.** The method should preserve the classification accuracy of the global model in the presence of malicious clients performing strong poisoning attacks. In particular, we aim to design a method that can learn a global model under attacks that is as accurate as the global model learnt by FedAvg under no attacks. Moreover, for targeted attacks, our goal further includes that the global model is unlikely to predict the attacker-chosen target labels for the attacker-chosen target testing examples.
- **Efficiency.** The method should not incur extra computation and communications overhead, especially to the clients. Clients in FL are often resource-constrained devices. Therefore, we aim to design a method that does not increase the workload of the clients, compared to FedAvg under no attacks.

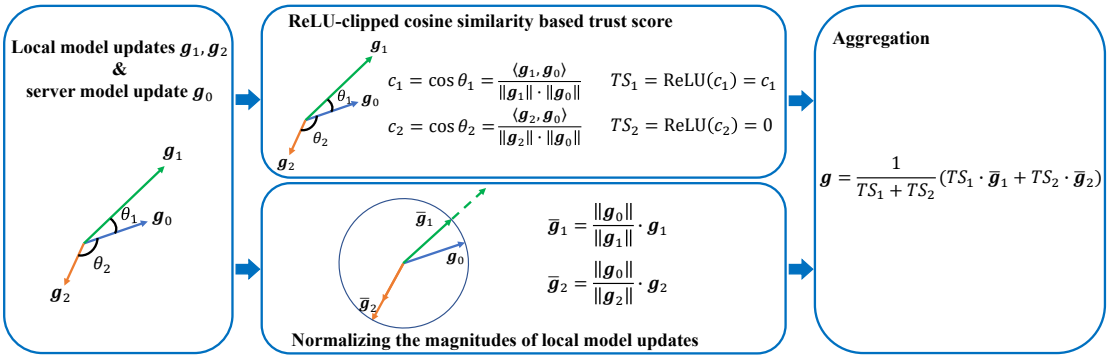


Fig. 2: Illustration of our aggregation rule, which is applied in each iteration of FLTrust.

Existing Byzantine-robust FL methods such as Krum, Trim-mean, and Median do not satisfy the fidelity and robustness goals. Moreover, Krum does not satisfy the efficiency goal because it requires the server to compute pairwise distances of the clients' local model updates, which is computationally expensive when the number of clients is large.

Defender's knowledge and capability: We consider the defense is performed on the server side. The server does not have access to the raw local training data on the clients, and the server does not know the number of malicious clients. However, the server has full access to the global model as well as the local model updates from all clients in each iteration. Moreover, the server itself can collect a clean small training dataset (we call it root dataset) for the learning task. We require the root dataset to be clean from poisoning. The server can collect a clean root dataset by manual labeling. For instance, Google enlists its employees to type with Gboard to create the root dataset for its federated next-word prediction [1]; when the learning task is digit recognition, the service provider can hire human workers to label some digits. Since we only require a small root dataset, e.g., 100 training examples, it is often affordable for the server to perform manual collection and labeling. The root dataset may or may not follow the same distribution as the overall training data distribution of the learning task. Our experimental results show that our method is effective once the root dataset distribution does not deviate too much from the overall training data distribution.

IV. Our FLTrust

A. Overview of FLTrust

In our FLTrust, the server itself collects a small clean training dataset (called *root dataset*) and maintains a model (called *server model*) for it just like how a client maintains a local model. In each iteration, our FLTrust follows the general three steps of FL discussed in Section II-A. However, our FLTrust is different from existing FL methods in Step II and Step III. Specifically, in Step II, each client trains its local model in existing FL methods, while the server also trains its server model via fine-tuning the current global model using the root dataset in FLTrust. In Step III, existing FL methods only consider the clients' local model updates to update the global model, which provides no root of trust. On the contrary, FLTrust considers both the server model update and the clients' local model updates to update the global model.

Specifically, an attacker can manipulate the directions of the local model updates on the malicious clients such that the global model is updated towards the opposite of the direction along which it should be updated; or the attacker can scale up the magnitudes of the local model updates to dominate the aggregated global model update. Therefore, we take both the directions and the magnitudes of the model updates into consideration. In particular, FLTrust first assigns a trust score (TS) to a local model update based on its direction similarity with the server model update. Formally, our trust score of a local model update is its ReLU-clipped cosine similarity with the server model update. Then, FLTrust normalizes each local model update by scaling it to have the same magnitude as the server model update. Such normalization essentially projects each local model update to the same hyper-sphere where the server model update lies in the vector space, which limits the impact of the poisoned local model updates with large magnitudes. Finally, FLTrust computes the average of the normalized local model updates weighted by their trust scores as the global model update, which is used to update the global model.

B. Our New Aggregation Rule

Our new aggregation rule considers both the directions and magnitudes of the local model updates and the server model update to compute the global model update. Figure 2 illustrates our aggregation rule.

ReLU-clipped cosine similarity based trust score: An attacker can manipulate the directions of the local model updates on the malicious clients such that the global model update is driven to an arbitrary direction that the attacker desires. Without root of trust, it is challenging for the server to decide which direction is more "promising" to update the global model. In our FLTrust, the root trust origins from the direction of the server model update. In particular, if the direction of a local model update is more similar to that of the server model update, then the direction of the local model update may be more "promising". Formally, we use the *cosine similarity*, a popular metric to measure the angle between two vectors, to measure the direction similarity between a local model update and the server model update.

Algorithm 1 ModelUpdate($\boldsymbol{w}$, D, b, β, R)

Output: Model update.

$\boldsymbol{w}^0 \leftarrow \boldsymbol{w}$.
for $r = 1, 2, \cdots, R$ **do**
  Randomly sample a batch D_b from D.
  $\boldsymbol{w}^r \leftarrow \boldsymbol{w}^{r-1} - \beta \nabla Loss(D_b; \boldsymbol{w})$.
end for
return $\boldsymbol{w}^R - \boldsymbol{w}$.

However, the cosine similarity alone faces a challenge. Specifically, if a local model update and the server model update are in opposite directions, their cosine similarity is negative, which still has negative impact on the aggregated global model update (see our experimental results in Section VI-B). Therefore, we exclude such local model updates from the aggregation by clipping the cosine similarity. In particular, we use the popular ReLU operation for clipping. Formally, our trust score is defined as follows:

$$TS_i = ReLU(c_i), \tag{2}$$

where TS_i is the trust score for the ith local model update $\boldsymbol{g}_i$, and c_i is the cosine similarity between $\boldsymbol{g}_i$ and the server model update $\boldsymbol{g}_0$, i.e., $c_i = \frac{\langle \boldsymbol{g}_i, \boldsymbol{g}_0 \rangle}{||\boldsymbol{g}_i|| \cdot ||\boldsymbol{g}_0||}$. ReLU is defined as follows: $ReLU(x) = x$ if $x > 0$ and $ReLU(x) = 0$ otherwise.

Normalizing the magnitudes of local model updates: An attacker can also scale the magnitudes of the local model updates on the malicious clients by a large factor such that they dominate the global model update. Therefore, we normalize the magnitude of each local model update. Without root of trust, it is challenging to decide what quantity we should normalize to. However, the server has the root dataset to bootstrap trust in FLTrust. Therefore, we normalize each local model update such that it has the same magnitude as the server model update. Such normalization means that we rescale local model updates to be the same hyper-sphere where the server model update lies in the vector space. Formally, we have the following:

$$\bar{\boldsymbol{g}}_i = \frac{||\boldsymbol{g}_0||}{||\boldsymbol{g}_i||} \cdot \boldsymbol{g}_i, \tag{3}$$

where $\boldsymbol{g}_i$ is the local model update of the ith client in the current iteration, $\bar{\boldsymbol{g}}_i$ is the *normalized local model update* of the ith client, $\boldsymbol{g}_0$ is the server model update, and $||\cdot||$ means ℓ_2 norm of a vector. Our normalization ensures that no single local model update has too much impact on the aggregated global model update. Note that our normalization also enlarges a local model update with a small magnitude to have the same magnitude as the server model update. This is based on the intuition that local model updates with small magnitudes are more likely from benign clients, and thus enlarging their magnitudes helps reduce the impact of the poisoned local model updates from the malicious clients, leading to a better global model (see our experimental results in Section VI-B).

Aggregating the local model updates: We compute the average of the normalized local model updates weighted by their trust scores as the global model update:

$$\boldsymbol{g} = \frac{1}{\sum_{j=1}^{n} TS_j} \sum_{i=1}^{n} TS_i \cdot \bar{\boldsymbol{g}}_i$$

Algorithm 2 FLTrust

Input: n clients with local training datasets $D_i, i = 1, 2, \cdots, n$; a server with root dataset D_0; global learning rate α; number of global iterations R_g; number of clients τ sampled in each iteration; local learning rate β; number of local iterations R_l; and batch size b.

Output: Global model $\boldsymbol{w}$.

$\boldsymbol{w} \leftarrow$ random initialization.
for $r = 1, 2, \cdots, R_g$ **do**
  // Step I: The server sends the global model to clients.
  The server randomly samples τ clients $C_1, C_2, \cdots, C_\tau$ from $\{1, 2, \cdots, n\}$ and sends $\boldsymbol{w}$ to them.
  // Step II: Training local models and server model.
  // Client side.
  for $i = C_1, C_2, \cdots, C_\tau$ **do in parallel**
    $\boldsymbol{g}_i = ModelUpdate(\boldsymbol{w}, D_i, b, \beta, R_l)$.
    Send $\boldsymbol{g}_i$ to the server.
  end for
  // Server side.
  $\boldsymbol{g}_0 = ModelUpdate(\boldsymbol{w}, D_0, b, \beta, R_l)$.
  // Step III: Updating the global model via aggregating the local model updates.
  for $i = C_1, C_2, \cdots, C_\tau$ **do**
    $TS_i = ReLU\left(\frac{\langle \boldsymbol{g}_i, \boldsymbol{g}_0 \rangle}{\|\boldsymbol{g}_i\| \|\boldsymbol{g}_0\|}\right)$.
    $\bar{\boldsymbol{g}}_i = \frac{||\boldsymbol{g}_0||}{||\boldsymbol{g}_i||} \cdot \boldsymbol{g}_i$.
  end for
  $\boldsymbol{g} = \frac{1}{\sum_{j=1}^{\tau} TS_{C_j}} \sum_{i=1}^{\tau} TS_{C_i} \cdot \bar{\boldsymbol{g}}_{C_i}$.
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \boldsymbol{g}$.
end for
return $\boldsymbol{w}$.

$$= \frac{1}{\sum_{j=1}^{n} ReLU(c_j)} \sum_{i=1}^{n} ReLU(c_i) \cdot \frac{||\boldsymbol{g}_0||}{||\boldsymbol{g}_i||} \cdot \boldsymbol{g}_i, \tag{4}$$

where $\boldsymbol{g}$ is the global model update. Note that if the server selects a subset of clients in an iteration, the global model update is aggregated from the local model updates of the selected clients. In principle, the server model update can be treated as a local model update with a trust score of 1 and the global model update can be weighted average of the clients' local model updates together with the server model update. However, such variant may negatively impact the global model because the root dataset is small and may not have the same distribution as the training data, but the server model update derived from it has a trust score of 1, reducing the contributions of the benign clients' local model updates (see our experimental results in Section VI-B). Finally, we update the global model as follows:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \boldsymbol{g}, \tag{5}$$

where α is the global learning rate.

C. Complete FLTrust Algorithm

Algorithm 2 shows our complete FLTrust method. FLTrust performs R_g iterations and has three steps in each iteration.

In Step I, the server sends the current global model to the clients or a subset of them. In Step II, the clients compute the local model updates based on the global model and their local training data, which are then sent to the server. Meanwhile, the server itself computes the server model update based on the global model and the root dataset. The local model updates and the server model update are computed by the function ModelUpdate in Algorithm 1 via performing stochastic gradient descent for R_l iterations with a local learning rate β. In Step III, the server computes the global model update by aggregating the local model updates and uses it to update the global model with a global learning rate α.

D. Formal Security Analysis

As we discussed in Section II-A, the optimal global model $\boldsymbol{w}^*$ is a solution to the following optimization problem: $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\Theta} F(\boldsymbol{w}) \triangleq \mathbb{E}_{D\sim\mathcal{X}}\left[f(D,\boldsymbol{w})\right]$, where Θ is the parameter space of the global model, $D = \bigcup_{i=1}^{n} D_i$ is the joint training dataset of the n clients, $\mathcal{X}$ is the training data distribution, $f(D,\boldsymbol{w})$ is the empirical loss function on the training data D, and $F(\boldsymbol{w})$ is the expected loss function. Our FLTrust is an iterative algorithm to find a global model to minimize the empirical loss function $f(D,\boldsymbol{w})$. We show that, under some assumptions, the difference between the global model learnt by FLTrust under attacks and the optimal global model $\boldsymbol{w}^*$ is bounded. Next, we first describe our assumptions and then describe our theoretical results.

Assumption 1. *The expected loss function $F(\boldsymbol{w})$ is μ-strongly convex and differentiable over the space Θ with L-Lipschitz continuous gradient. Formally, we have the following for any $\boldsymbol{w}, \widehat{\boldsymbol{w}} \in \Theta$:*

$$F(\widehat{\boldsymbol{w}}) \geq F(\boldsymbol{w}) + \langle \nabla F(\boldsymbol{w}), \widehat{\boldsymbol{w}} - \boldsymbol{w}\rangle + \frac{\mu}{2}\|\widehat{\boldsymbol{w}} - \boldsymbol{w}\|^2,$$
$$\|\nabla F(\boldsymbol{w}) - \nabla F(\widehat{\boldsymbol{w}})\| \leq L\|\boldsymbol{w} - \widehat{\boldsymbol{w}}\|,$$

where ∇ represents gradient, $\|\cdot\|$ represents ℓ_2 norm, and $\langle\cdot,\cdot\rangle$ represents inner product of two vectors. Moreover, the empirical loss function $f(D,\boldsymbol{w})$ is L_1-Lipschitz probabilistically. Formally, for any $\delta \in (0,1)$, there exists an L_1 such that:

$$Pr\left\{\sup_{\boldsymbol{w},\widehat{\boldsymbol{w}}\in\Theta:\boldsymbol{w}\neq\widehat{\boldsymbol{w}}} \frac{\|\nabla f(D,\boldsymbol{w}) - \nabla f(D,\widehat{\boldsymbol{w}})\|}{\|\boldsymbol{w} - \widehat{\boldsymbol{w}}\|} \leq L_1\right\} \geq 1 - \frac{\delta}{3}.$$

Assumption 2. *The gradient of the empirical loss function $\nabla f(D,\boldsymbol{w}^*)$ at the optimal global model $\boldsymbol{w}^*$ is bounded. Moreover, the gradient difference $h(D,\boldsymbol{w}) = \nabla f(D,\boldsymbol{w}) - \nabla f(D,\boldsymbol{w}^*)$ for any $\boldsymbol{w} \in \Theta$ is bounded. Specifically, there exist positive constants σ_1 and γ_1 such that for any unit vector $\boldsymbol{v}$, $\langle \nabla f(D,\boldsymbol{w}^*), \boldsymbol{v}\rangle$ is sub-exponential with σ_1 and γ_1; and there exist positive constants σ_2 and γ_2 such that for any $\boldsymbol{w} \in \Theta$ with $\boldsymbol{w} \neq \boldsymbol{w}^*$ and any unit vector $\boldsymbol{v}$, $\langle h(D,\boldsymbol{w}) - \mathbb{E}\left[h(D,\boldsymbol{w})\right], \boldsymbol{v}\rangle / \|\boldsymbol{w} - \boldsymbol{w}^*\|$ is sub-exponential with σ_2 and γ_2. Formally, for $\forall\,|\xi| \leq 1/\gamma_1$, $\forall\,|\xi| \leq 1/\gamma_2$, we have:*

$$\sup_{\boldsymbol{v}\in\boldsymbol{B}} \mathbb{E}\left[\exp(\xi\,\langle \nabla f(D,\boldsymbol{w}^*), \boldsymbol{v}\rangle)\right] \leq e^{\sigma_1^2\xi^2/2},$$

$$\sup_{\boldsymbol{w}\in\Theta,\boldsymbol{v}\in\boldsymbol{B}} \mathbb{E}\left[\exp\left(\frac{\xi\,\langle h(D,\boldsymbol{w}) - \mathbb{E}\left[h(D,\boldsymbol{w})\right], \boldsymbol{v}\rangle}{\|\boldsymbol{w} - \boldsymbol{w}^*\|}\right)\right] \leq e^{\sigma_2^2\xi^2/2},$$

where $\boldsymbol{B}$ is the unit sphere $\boldsymbol{B} = \{\boldsymbol{v} : \|\boldsymbol{v}\| = 1\}$.

Assumption 3. *Each client's local training dataset D_i ($i = 1, 2, \cdots, n$) and the root dataset D_0 are sampled independently from the distribution $\mathcal{X}$.*

Theorem 1. *Suppose Assumption 1-3 hold and FLTrust uses $R_l = 1$ and $\beta = 1$. For an arbitrary number of malicious clients, the difference between the global model learnt by FLTrust and the optimal global model $\boldsymbol{w}^*$ under no attacks is bounded. Formally, we have the following with probability at least $1 - \delta$:*

$$\left\|\boldsymbol{w}^t - \boldsymbol{w}^*\right\| \leq (1-\rho)^t \left\|\boldsymbol{w}^0 - \boldsymbol{w}^*\right\| + 12\alpha\Delta_1/\rho,$$

where $\boldsymbol{w}^t$ is the global model in the tth iteration, $\rho = 1 - \left(\sqrt{1 - \mu^2/(4L^2)} + 24\alpha\Delta_2 + 2\alpha L\right)$, α is the learning rate, $\Delta_1 = \sigma_1\sqrt{\frac{2}{|D_0|}}\sqrt{d\log 6 + \log(3/\delta)}$, $\Delta_2 = \sigma_2\sqrt{\frac{2}{|D_0|}}\sqrt{d\log\frac{18L_2}{\sigma_2} + \frac{1}{2}d\log\frac{|D_0|}{d} + \log\left(\frac{6\sigma_2^2 r\sqrt{|D_0|}}{\gamma_2\sigma_1\delta}\right)}$, $|D_0|$ is the size of the root dataset, d is the dimension of $\boldsymbol{w}$, $L_2 = \max\{L, L_1\}$, and r is some positive number such that $\|\boldsymbol{w} - \boldsymbol{w}^*\| \leq r\sqrt{d}$ for any $\boldsymbol{w} \in \Theta$ (i.e., the parameter space Θ is constrained). When $|1-\rho| < 1$, we have $\lim_{t\to\infty}\|\boldsymbol{w}^t - \boldsymbol{w}^*\| \leq 12\alpha\Delta_1/\rho$.

Proof: See Appendix A. ■

V. Adaptive Attacks

When an attacker knows our FLTrust is used to learn the global model, the attacker can adapt its attacks to FLTrust. Therefore, in this section, we design strong adaptive attacks to FLTrust. In particular, Fang et al. [15] proposed the state-of-the-art framework that can optimize local model poisoning attacks for any given aggregation rule. We generate adaptive attacks to FLTrust via instantiating this framework with our aggregation rule. Next, we first describe the general attack framework in [15], then we discuss how to design adaptive attacks to FLTrust based on the framework.

A. Local Model Poisoning Attack Framework

The framework of local model poisoning attacks introduced in [15] is general to all aggregation rules. Specifically, in each iteration of FL, the attacker aims to change the global model update the most along the opposite direction of the global model update under no attacks, by carefully crafting the local model updates on the malicious clients. Assuming the first m clients are malicious. The local model poisoning attack is formulated as the following optimization problem[3]:

$$\begin{aligned}\max_{\boldsymbol{g}_1',\boldsymbol{g}_2',\cdots,\boldsymbol{g}_m'} \; & \boldsymbol{s}^T(\boldsymbol{g} - \boldsymbol{g}'),\\ \text{subject to } & \boldsymbol{g} = \mathcal{A}(\boldsymbol{g}_1,\cdots,\boldsymbol{g}_m,\boldsymbol{g}_{m+1},\boldsymbol{g}_n),\\ & \boldsymbol{g}' = \mathcal{A}(\boldsymbol{g}_1',\cdots,\boldsymbol{g}_m',\boldsymbol{g}_{m+1},\boldsymbol{g}_n),\end{aligned} \tag{6}$$

where $\mathcal{A}$ is the aggregation rule of the FL method, $\boldsymbol{g}_i'$ is the poisoned local model update on the ith malicious client for $i = 1, 2, \cdots, m$, $\boldsymbol{g}$ is the global model update before attack, $\boldsymbol{g}'$ is the global model update after attack, and $\boldsymbol{s}$ is a column vector of the sign of the global model update before attack.

[3]Fang et al. formulates the framework based on local models, which is equivalent to formulating the framework based on local model updates.

Algorithm 3 Our Adaptive Attack to FLTrust.

Input: $\boldsymbol{g}_0; \boldsymbol{g}_i$ for $i = 1, 2, \cdots, n; m; \sigma; \eta; \gamma; Q; V$.
Output: $\boldsymbol{e}'_i$ for $i = 1, 2, \cdots, m$.

Compute $\boldsymbol{e}_0, \boldsymbol{e}_i, c_i$ for $i = 1, 2, \cdots, n$.
Initialize $\boldsymbol{e}'_i$ using Trim attack for $i = 1, 2, \cdots, m$.
for $v = 1, 2, \cdots, V$ **do**
  for $i = 1, 2, \cdots, m$ **do**
    for $t = 1, 2, \cdots, Q$ **do**
      Randomly sample $\boldsymbol{u} \sim N(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$.
      Compute $\nabla_{\boldsymbol{e}'_i} h$ according to (11).
      Update $\boldsymbol{e}'_i = \boldsymbol{e}'_i + \eta \nabla_{\boldsymbol{e}'_i} h$.
      Normalize $\boldsymbol{e}'_i$ such that $\|\boldsymbol{e}'_i\| = 1$.
    end for
  end for
end for
return $\boldsymbol{e}'_i$ for $i = 1, 2, \cdots, m$.

B. Adaptive Attack to Our FLTrust

We leverage the state-of-the-art framework to design adaptive attacks to our FLTrust. The idea is to instantiate the aggregation rule $\mathcal{A}$ with our aggregation rule in FLTrust in the framework. We denote by $\boldsymbol{e}_i = \frac{\boldsymbol{g}_i}{||\boldsymbol{g}_i||}$ the unit vector whose direction is the same as $\boldsymbol{g}_i$. Then, our aggregation rule in Equation (4) can be rewritten as follows:

$$\boldsymbol{g} = ||\boldsymbol{g}_0|| \sum_{i=1}^{n} \frac{ReLU(c_i)}{\sum_{j=1}^{n} ReLU(c_j)} \boldsymbol{e}_i. \tag{7}$$

Suppose there are m malicious clients, and without loss of generality, we assume the first m clients are malicious. These malicious clients send poisoned local model updates $\boldsymbol{g}'_i, i = 1, 2, \cdots, m$ to the server. Let $\boldsymbol{e}'_i$ $(i = 1, 2, \cdots, m)$ be the corresponding unit vectors. We note that the cosine similarity c'_i between a poisoned local model update $\boldsymbol{g}'_i$ and the server model update $\boldsymbol{g}_0$ is the same as the cosine similarity between the corresponding unit vectors, i.e., $c'_i = \langle \boldsymbol{e}'_i, \boldsymbol{e}_0 \rangle$, where $\langle \cdot, \cdot \rangle$ means the inner product of two vectors. Therefore, we have the poisoned global model update $\boldsymbol{g}'$ under attacks as follows:

$$\boldsymbol{g}' = ||\boldsymbol{g}_0|| \left[\sum_{i=1}^{m} \frac{ReLU(\langle \boldsymbol{e}'_i, \boldsymbol{e}_0 \rangle)}{\sum_{j=1}^{m} ReLU(\langle \boldsymbol{e}'_j, \boldsymbol{e}_0 \rangle) + \sum_{j=m+1}^{n} ReLU(c_j)} \boldsymbol{e}'_i \right.$$
$$\left. + \sum_{i=m+1}^{n} \frac{ReLU(c_i)}{\sum_{j=1}^{m} ReLU(\langle \boldsymbol{e}'_j, \boldsymbol{e}_0 \rangle) + \sum_{j=m+1}^{n} ReLU(c_j)} \boldsymbol{e}_i \right]. \tag{8}$$

Substituting Equations (7) and (8) into (6), and noticing that optimizing $\boldsymbol{g}'_i$ is equivalent to optimizing $\boldsymbol{e}'_i$ for $i = 1, 2, \cdots, m$, we can instantiate the attack framework in Equation (6) as the following optimization problem:

$$\max_{\boldsymbol{e}'_1, \boldsymbol{e}'_2, \cdots, \boldsymbol{e}'_m} \quad h(\boldsymbol{e}'_1, \boldsymbol{e}'_2, \cdots, \boldsymbol{e}'_m), \tag{9}$$

where $h(\boldsymbol{e}'_1, \boldsymbol{e}'_2, \cdots, \boldsymbol{e}'_m)$ is defined as follows:

$$h(\boldsymbol{e}'_1, \boldsymbol{e}'_2, \cdots, \boldsymbol{e}'_m) = ||\boldsymbol{g}_0|| \boldsymbol{s}^T \left[\sum_{i=1}^{n} \frac{ReLU(c_i)}{\sum_{j=1}^{n} ReLU(c_j)} \boldsymbol{e}_i \right.$$
$$- \sum_{i=1}^{b} \frac{ReLU(\langle \boldsymbol{e}'_i, \boldsymbol{e}_0 \rangle)}{\sum_{j=1}^{m} ReLU(\langle \boldsymbol{e}'_j, \boldsymbol{e}_0 \rangle) + \sum_{j=m+1}^{n} ReLU(c_j)} \boldsymbol{e}'_i$$
$$\left. - \sum_{i=m+1}^{n} \frac{ReLU(c_i)}{\sum_{j=1}^{m} ReLU(\langle \boldsymbol{e}'_j, \boldsymbol{e}_0 \rangle) + \sum_{j=m+1}^{n} ReLU(c_j)} \boldsymbol{e}_i \right], \tag{10}$$

where $\boldsymbol{s}^T = \text{sgn}(\boldsymbol{g})^T$ is the sign of the global model update without attacks. Solving the optimization problem generates an adaptive attack to FLTrust. We consider a strong adaptive attacker who has full knowledge about the FL system when solving the optimization problem. In particular, $||\boldsymbol{g}_0||, \boldsymbol{s}, c_i$ $(i = 1, 2, \cdots, n), \boldsymbol{e}_0$, and $\boldsymbol{e}_i$ $(i = 1, 2, \cdots, n)$ are all available to the attacker.

Solving the optimization problem: We use a standard gradient ascent approach to solve the optimization problem. Specifically, we can compute the gradient $\nabla_{\boldsymbol{e}'_i} h$ of the objective function h with respect to each $\boldsymbol{e}'_i$ and move $\boldsymbol{e}'_i$ a small step along the gradient. Since the gradient $\nabla_{\boldsymbol{e}'_i} h$ involves a Jacobian matrix of $\boldsymbol{e}'_i$, it is not practical to directly compute the gradient. Therefore, we leverage a zeroth-order method [13], [33] to compute the gradient, which is a standard method to solve such optimization problems with computationally intractable objective functions. Specifically, we compute the gradient $\nabla_{\boldsymbol{e}'_i} h$ as follows:

$$\nabla_{\boldsymbol{e}'_i} h \approx \frac{h(\boldsymbol{e}'_i + \gamma \boldsymbol{u}) - h(\boldsymbol{e}'_i)}{\gamma} \cdot \boldsymbol{u}, \tag{11}$$

where $\boldsymbol{u}$ is a random vector sampled from the multivariate Gaussian distribution $N(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ with zero mean and diagonal covariance matrix, and $\gamma > 0$ is a smoothing parameter.

We optimize $\boldsymbol{e}'_i$ one by one following the standard coordinate ascent approach, i.e., when optimizing $\boldsymbol{e}'_i$, all other $\boldsymbol{e}'_j, j \neq i$ are fixed. Specifically, we use projected gradient ascent to iteratively optimize $\boldsymbol{e}'_i$. In the beginning, we initialize $\boldsymbol{e}'_i$ using the Trim attack, i.e., we use the Trim attack to compute the poisoned local model updates and initialize $\boldsymbol{e}'_i$ as the corresponding unit vector. Then, in each iteration, we sample a random vector $\boldsymbol{u}$ from $N(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ and compute the gradient $\nabla_{\boldsymbol{e}'_i} h$ following Equation (11). We multiply the gradient by a step size η and add it to $\boldsymbol{e}'_i$ to get the new $\boldsymbol{e}'_i$. Finally, we project $\boldsymbol{e}'_i$ to the unit sphere to ensure that $\boldsymbol{e}'_i$ is a valid unit vector. We repeat the gradient ascent process for Q iterations. Moreover, we repeat the iterations over the unit vectors for V iterations. Algorithm 3 shows our adaptive attack. We let $\boldsymbol{g}'_i = \|\boldsymbol{g}_0\| \cdot \boldsymbol{e}'_i$ after $\boldsymbol{e}'_i$ is solved for $i = 1, 2, \cdots, m$.

VI. Evaluation

We evaluate our FLTrust against both existing poisoning attacks to FL and adaptive attacks in this section.

A. Experimental Setup

1) Datasets: We use multiple datasets from different domains in our evaluation, including five image classification datasets and a human activity recognition dataset. We follow previous work [15] to distribute the training examples in a dataset among clients. Assuming there are M classes in a dataset. We randomly split the clients into M groups. A training example with label l is assigned to group l with probability $q > 0$ and to any other group with probability $\frac{1-q}{M-1}$. Within the same group, data are uniformly distributed to each client. q controls the distribution difference of the clients' local training data. If $q = 1/M$, then the clients' local training data are independent and identically distributed (IID), otherwise the clients' local training data are non-IID. Moreover, a larger q indicates a higher degree of non-IID among the clients' local training data. One characteristic of FL is that clients often have non-IID local training data [22], [28]. Therefore, we will set $q > 1/M$ by default to simulate the non-IID settings.

Next, we use the MNIST dataset as an example to show the distribution process. Assume we have 100 clients in total and set $q = 0.5$. $M = 10$ for the MNIST dataset. We first randomly split the clients into 10 groups, each containing 10 clients. For a training image of digit l (e.g., $l = 5$), we first assign it to group 5 with probability 0.5, and to any other group with probability $\frac{1-0.5}{10-1} \approx 0.056$. Once the group is determined, e.g., group 5 is chosen, we will select a client from group 5 uniformly at random and assign this training image to the selected client.

MNIST-0.1: MNIST [24] is a 10-class digit image classification dataset, which consists of 60,000 training examples and 10,000 testing examples. We set $q = 0.1$ in MNIST-0.1, which indicates local training data are IID among clients. We use MNIST-0.1 to show that FLTrust is also effective in the IID setting.

MNIST-0.5: In MNIST-0.5, we simulate non-IID local training data among the clients via setting $q = 0.5$.

Fashion-MNIST: Fashion-MNIST [42] is a 10-class fashion image classification task, which has a predefined training set of 60,000 fashion images and a testing set of 10,000 fashion images. Like the MNIST-0.5 dataset, we distribute the training examples to the clients with $q = 0.5$ to simulate non-IID local training data.

CIFAR-10: CIFAR-10 [23] is a color image classification dataset consisting of predefined 50,000 training examples and 10,000 testing examples. Each example belongs to one of the 10 classes. To simulate non-IID local training data, we distribute the training examples to clients with $q = 0.5$.

Human activity recognition (HAR): The HAR dataset [4] consists of human activity data collected from the smartphones of 30 real-world users. The data are signals from multiple sensors on a user's smartphone, and the task is to predict the user's activity among 6 possible activities, i.e., WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, and LAYING. Each example includes 561 features and there are 10,299 examples in total. Unlike the previous datasets, we don't need to distribute the data to clients in this dataset, as each user is naturally considered as a client. HAR represents a real-world FL scenario, where each user is considered as a client. We use 75% of each client's data as training examples and the rest 25% as testing examples. We note that HAR has unbalanced local training data on clients: the maximum number of training examples on a client is 409, the minimum number is 281, and the mean is 343.

CH-MNIST: CH-MNIST [21] is a medical image classification dataset consisting of 5,000 images of histology tiles collected from colorectal cancer patients. Each example has 64×64 gray-scale pixels and belongs to one of the 8 classes. We use 4,000 images selected randomly as the training examples and use the other 1,000 images as the testing examples. To simulate non-IID local training data, we distribute the training examples to clients with $q = 0.5$.

2) Evaluated Poisoning Attacks: We consider both data poisoning attacks and local model poisoning attacks. For data poisoning attack, we consider the popular label flipping attack. For local model poisoning attacks, we evaluate Krum attack, Trim attack, and our adaptive attack (untargeted attacks) [15], as well as Scaling attack (targeted attack) [5].

Label flipping (LF) attack: We use the same label flipping attack setting as [15]. In particular, for each training example on the malicious clients, we flip its label l to $M-l-1$, where M is the total number of labels and $l \in \{0, 1, \cdots, M-1\}$.

Krum attack: Krum attack is an untargeted local model poisoning attack optimized for the Krum aggregation rule. We use the default parameter settings in [15] for the Krum attack.

Trim attack: Trim attack is an untargeted local model poisoning attack optimized for the Trim-mean and Median aggregation rules. We use the default parameter settings in [15] for the Trim attack.

Scaling attack: Scaling attack is a targeted local model poisoning attack. Specifically, we consider the attacker-chosen target testing examples are normal testing examples with a predefined feature-pattern trigger embedded. Following [5], we use the data augmentation scheme in [18] to implement the Scaling attack. Specifically, each malicious client copies p fraction of its local training examples, embeds the trigger to them, changes their labels to the attacker-chosen target label, and uses them to augment its local training data. Then, in each iteration of FL, each malicious client computes its local model update based on the augmented local training data and scales it by a factor $\lambda \gg 1$ before sending it to the server.

Specifically, we use the same pattern trigger in [18] as our trigger for MNIST-0.1, MNIST-0.5, Fashion-MNIST, and CH-MNIST, and we set the attacker-chosen target label as 0; for CIFAR-10, we consider the same pattern trigger and target label (i.e., "bird") in [5]; and for HAR, we create a feature-pattern trigger by setting every 20th feature to 0 and we set the target label as "WALKING_UPSTAIRS". Following previous work [5], we set the scaling factor $\lambda = n$, where n is the number of clients. In each dataset, the attacker-chosen target testing examples consist of the trigger-embedded normal testing examples whose true labels are not the target label.

Adaptive attack: We evaluate the adaptive attack proposed in Section V. Our adaptive attack leverages an zeroth-order

TABLE I: The default FL system parameter settings.

	Explanation	MNIST-0.1	MNIST-0.5	Fashion-MNIST	CIFAR-10	HAR	CH-MNIST
n	# clients	100				30	40
τ	# clients selected in each iteration	n					
R_l	# local iterations	1					
R_g	# global iterations	2,000		2,500	1,500	1,000	2,000
b	batch size	32			64	32	
$\alpha \cdot \beta$	combined learning rate	3×10^{-4}		6×10^{-3}	2×10^{-4}	3×10^{-3}	3×10^{-4} (decay at the 1500th and 1750th iterations with factor 0.9)
m/n	fraction of malicious clients (%)	20					
m	# malicious clients	20				6	8
f	Krum parameter	m					
k	Trim-mean parameter	m					
$\|D_0\|$	size of the root dataset	100					

TABLE II: The CNN architecture of the global model used for MNIST-0.1, MNIST-0.5, and Fashion-MNIST.

Layer	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$3 \times 3 \times 30$
Max Pooling	2×2
Convolution + ReLU	$3 \times 3 \times 50$
Max Pooling	2×2
Fully Connected + ReLU	100
Softmax	10

optimization method. Following the suggestions by previous work [13], [33], we set $\sigma^2 = 0.5$ and $\gamma = 0.005$ in the zeroth-order method. Moreover, we set $\eta = 0.01$ and $V = Q = 10$ so that the adaptive attack converges.

3) Evaluation Metrics: For the LF attack, Krum attack, Trim attack, and adaptive attack, we use the standard *testing error rate* of the global model to evaluate an FL method since these attacks aim to increase the testing error rate. Specifically, the testing error rate of a global model is the fraction of testing examples whose labels are incorrectly predicted by the global model. An FL method is more robust against these attacks if its global models achieve lower testing error rates under these attacks. The Scaling attack is a targeted attack, which aims to preserve the testing error rate of normal testing examples while making the global model predict the attacker-chosen target label for the attacker-chosen target testing examples. Therefore, other than the testing error rate, we further use *attack success rate* to measure the Scaling attack. Specifically, the attack success rate is the fraction of the attacker-chosen target testing examples whose labels are predicted as the attacker-chosen target label by the global model. An FL method is more robust against the Scaling attack if its global model achieves a lower attack success rate.

4) FL System Settings: By default, we assume there are $n = 100$ clients in total for each dataset except HAR and CH-MNIST. For HAR, the data are collected from 30 users, each of which is treated as a client. Therefore, HAR has 30 clients in total. For CH-MNIST, there are only 4,000 training examples in total and thus we assume 40 clients such that each client has 100 training examples on average. Unless otherwise mentioned, we assume 20% of the clients are malicious for each dataset. However, we will also explore the impact of the fraction of malicious clients. Table I shows the default FL system settings that we will use unless otherwise mentioned.

Global models: We train different types of global models on different datasets to show the generality of our method. Specifically, for MNIST-0.1, MNIST-0.5, and Fashion-MNIST, we train a convolutional neural network (CNN) as the global model. Table II shows the architecture of the CNN. And we train a logistic regression (LR) classifier as the global model for HAR. For CIFAR-10 and CH-MNIST, we consider the widely used ResNet20 architecture [19] as the global model.

Parameter settings of the FL methods: We compare FLTrust with FedAvg [22], [28], Krum [9], Trim-mean [48], and Median [48]. Details of these FL methods can be found in Section II-A. FedAvg is a popular FL method in non-adversarial settings, while Krum, Trim-mean, and Median are Byzantine-robust FL methods. These methods all follow the three-step framework described in Algorithm 2, though they use different aggregation rules. Therefore, they all use the parameters τ, R_l, R_g, α, β, and b. Following previous work [15], we set $\tau = n$, i.e., all clients are selected in each iteration; and we set $R_l = 1$, in which we can treat the product of the global learning rate α and the local learning rate β as a single learning rate. We set this combined learning rate on each dataset to achieve small training error rates and fast convergence. We set the batch size $b = 32$ for all datasets except CIFAR-10, where we set $b = 64$. We set the number of global iterations R_g such that the FL methods converge. Specifically, $R_g = 2,000$ for MNIST-0.1, MNIST-0.5, and CH-MNIST; $R_g = 2,500$ for Fashion-MNIST; $R_g = 1,500$ for CIFAR-10; and $R_g = 1,000$ for HAR.

Krum further has the parameter f and Trim-mean further has the trim parameter k, both of which are an upper bound of the number of malicious clients. We set $f = k = m$, which assumes that the server knows the exact number of malicious clients and gives advantages to Krum and Trim-mean.

Root dataset: Our FLTrust requires a small root dataset. By default, we assume the root dataset has only 100 training examples. Moreover, we consider the following two cases depending on how the root dataset is created.

- **Case I.** We assume the service provider can collect a representative root dataset for the learning task, i.e., the root dataset has the same distribution as the overall training data distribution of the learning task. In particular, we sample the root dataset from the union of the clients' clean local training data uniformly at random. For instance, for MNIST-0.5, we sample the root dataset from its 60,000 training examples uniformly at random.
- **Case II.** We assume the root dataset has a distribution different from the overall training data distribution of

the learning task. In particular, we assume the root dataset is biased towards a certain class. Specifically, we sample a fraction of the examples in the root dataset from a particular class (class 1 in our experiments) in the union of the clients' clean local training data and the remaining examples are sampled from the remaining classes uniformly at random, where we call the fraction *bias probability*. Note that, for all the datasets except HAR and CH-MNIST, the root dataset has the same distribution as the overall training data, i.e., Case II reduces to Case I, when the bias probability is 0.1 because they have 10 classes; for HAR and CH-MNIST, Case II reduces to Case I when the bias probability is 0.17 and 0.125 because they have 6 and 8 classes, respectively. The root data distribution deviates more from the overall training data distribution when the bias probability is larger.

In both cases, we exclude the sampled root dataset from the clients' local training data, indicating that the root dataset is collected independently by the service provider. Unless otherwise mentioned, we consider Case I.

B. Experimental Results

Our FLTrust achieves the three defense goals: Recall that we have three defense goals (discussed in Section III): fidelity, robustness, and efficiency. Table III shows the testing error rates of different FL methods under different attacks including our adaptive attack, as well as the attack success rate of the Scaling attack on the six datasets. Our results show that FLTrust achieves the three goals.

First, when there is no attack, our FLTrust has testing error rates similar to FedAvg, achieving the fidelity goal. However, existing Byzantine-robust FL methods may have higher or much higher testing error rates under no attacks. For instance, on MNIST-0.1, the testing error rates for FedAvg and FLTrust are both 0.04, while they are 0.10, 0.06, and 0.06 for Krum, Trim-mean, and Median, respectively; on CH-MNIST, FedAvg, Trim-mean, and FLTrust achieve testing error rates 0.10, while Krum and Median achieve testing error rates 0.24 and 0.11, respectively. Our results indicate that FLTrust is more accurate than existing Byzantine-robust FL methods in non-adversarial settings. This is because existing Byzantine-robust FL methods exclude some local model updates when aggregating them as the global model update, while FLTrust considers all of them with the help of the root dataset.

Second, our FLTrust achieves the robustness goal, while existing FL methods do not. Specifically, the testing error rates of FLTrust under the untargeted attacks including our adaptive attack are at most 0.04 higher than those of FedAvg under no attacks on the six datasets. On the contrary, every existing Byzantine-robust FL method has much higher testing error rates, especially under the untargeted attack that is optimized for the method. For instance, on MNIST-0.5, Krum attack increases the testing error rate of Krum from 0.10 to 0.91, while Trim attack increases the testing error rates of Trim-mean and Median from 0.06 to 0.23 and 0.43, respectively. FedAvg may have lower testing error rates than existing Byzantine-robust FL methods under the evaluated untargeted attacks. This is because these untargeted attacks are not optimized for FedAvg. Previous work [9] showed that FedAvg can be arbitrarily manipulated by a single malicious client.

TABLE III: The testing error rates of different FL methods under different attacks and the attack success rates of the Scaling attacks. The results for the Scaling attacks are in the form of "testing error rate / attack success rate".

(a) CNN global model, MNIST-0.1

	FedAvg	Krum	Trim-mean	Median	FLTrust
No attack	0.04	0.10	0.06	0.06	0.04
LF attack	0.06	0.10	0.05	0.05	0.04
Krum attack	0.10	0.90	0.07	0.07	0.04
Trim attack	0.16	0.10	0.13	0.13	0.04
Scaling attack	0.02 / 1.00	0.10 / 0.00	0.05 / 0.01	0.05 / 0.01	0.03 / 0.00
Adaptive attack	0.08	0.10	0.11	0.13	0.04

(b) CNN global model, MNIST-0.5

	FedAvg	Krum	Trim-mean	Median	FLTrust
No attack	0.04	0.10	0.06	0.06	0.05
LF attack	0.06	0.10	0.06	0.06	0.05
Krum attack	0.10	0.91	0.14	0.15	0.05
Trim attack	0.28	0.10	0.23	0.43	0.06
Scaling attack	0.02 / 1.00	0.09 / 0.01	0.06 / 0.02	0.06 / 0.01	0.05 / 0.00
Adaptive attack	0.13	0.10	0.22	0.90	0.06

(c) CNN global model, Fashion-MNIST

	FedAvg	Krum	Trim-mean	Median	FLTrust
No attack	0.10	0.16	0.14	0.14	0.11
LF attack	0.14	0.15	0.26	0.21	0.11
Krum attack	0.13	0.90	0.18	0.23	0.12
Trim attack	0.90	0.16	0.24	0.27	0.14
Scaling attack	0.90 / 1.00	0.16 / 0.03	0.17 / 0.85	0.16 / 0.05	0.11 / 0.02
Adaptive attack	0.90	0.18	0.34	0.24	0.14

(d) ResNet20 global model, CIFAR-10

	FedAvg	Krum	Trim-mean	Median	FLTrust
No attack	0.16	0.54	0.24	0.25	0.18
LF attack	0.21	0.56	0.27	0.45	0.18
Krum attack	0.24	0.90	0.52	0.64	0.18
Trim attack	0.81	0.51	0.72	0.75	0.20
Scaling attack	0.90 / 1.00	0.44 / 0.07	0.22 / 0.96	0.25 / 0.96	0.18 / 0.02
Adaptive attack	0.90	0.58	0.69	0.82	0.20

(e) LR global model, HAR

	FedAvg	Krum	Trim-mean	Median	FLTrust
No attack	0.03	0.12	0.04	0.05	0.04
LF attack	0.17	0.10	0.05	0.05	0.04
Krum attack	0.03	0.22	0.05	0.05	0.04
Trim attack	0.32	0.10	0.36	0.13	0.05
Scaling attack	0.04 / 0.81	0.10 / 0.03	0.04 / 0.36	0.05 / 0.13	0.05 / 0.01
Adaptive attack	0.04	0.19	0.05	0.06	0.05

(f) ResNet20 global model, CH-MNIST

	FedAvg	Krum	Trim-mean	Median	FLTrust
No attack	0.10	0.24	0.10	0.11	0.10
LF attack	0.12	0.39	0.15	0.13	0.12
Krum attack	0.11	0.95	0.13	0.13	0.12
Trim attack	0.64	0.21	0.55	0.44	0.13
Scaling attack	0.26 / 0.20	0.34 / 0.03	0.14 / 0.02	0.11 / 0.01	0.14 / 0.03
Adaptive attack	0.14	0.29	0.50	0.47	0.13

Moreover, for the Scaling attack, FLTrust substantially reduces its attack success rates. Specifically, the attack success rates for FLTrust are at most 0.03. On the contrary, the attack success rates for FedAvg are always high on the six datasets, and they are also high for the existing Byzantine-robust FL methods on multiple datasets, indicating that existing FL methods are not robust against the Scaling attack. One interesting observation is that the Scaling attack may decrease the testing error rates in some cases. We suspect the reason may be that the data augmentation in the Scaling attack positively impacts the aggregation of the local model updates. Specifically, the

TABLE IV: The testing error rates of different variants of FLTrust under different attacks and the attack success rates of the Scaling attacks on MNIST-0.5. The results for the Scaling attacks are in the form of "testing error rate / attack success rate". "–" means that the attacks are not applicable.

	No attack	LF attack	Krum attack	Trim attack	Scaling attack	Adaptive attack
FLTrust-Server	0.21	–	–	–	–	–
FLTrust-withServer	0.07	0.08	0.09	0.10	0.08 / 0.01	0.94
FLTrust-NoReLU	0.28	0.90	0.90	0.90	0.94 / 0.08	0.90
FLTrust-NoNorm	0.05	0.06	0.06	0.08	0.94 / 0.08	0.06
FLTrust-ParNorm	0.06	0.06	0.06	0.06	0.06 / 0.01	0.06
FLTrust	0.05	0.05	0.05	0.06	0.05 / 0.00	0.06

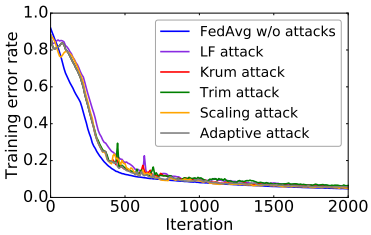
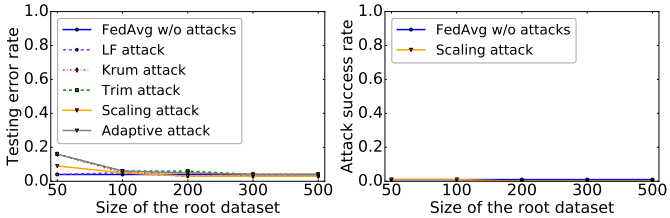


Fig. 3: The training error rates vs. the number of iterations for FLTrust under different attacks and FedAvg without attacks on MNIST-0.5.



(a) Testing error rate (b) Attack success rate

Fig. 4: Impact of the root dataset size on FLTrust under different attacks for MNIST-0.5.

data augmentation in the Scaling attack improves the diversity of the training data, and thus helps the learned global model better generalize to the testing dataset.

Third, FLTrust achieves the efficiency goal. Specifically, in each iteration, FLTrust does not incur extra overhead to the clients; and compared to FedAvg, the extra computation incurred to the server by FLTrust includes computing a server model update, computing the trust scores, and normalizing the local model updates, which are negligible for the powerful server. Moreover, Figure 3 shows the training error rates versus the global iteration number for FLTrust under different attacks and FedAvg under no attack on MNIST-0.5. Our results show that FLTrust converges as fast as FedAvg, which means that FLTrust also does not incur extra communications cost for the clients (each iteration of FL requires communications between clients and server), compared to FedAvg under no attacks. We note that Krum, Trim-mean, and Median do not incur extra overhead to the clients. However, Krum incurs significant computational overhead to the server when there are a large number of clients. This is because Krum requires calculating pairwise distance between local model updates in each iteration.

Comparing different variants of FLTrust: FLTrust has three key features: a root dataset, using ReLU to clip the cosine similarity scores, and normalizing each local model update. Depending on how each feature is used, we consider the following five variants of FLTrust:

- **FLTrust-Server.** In this variant, the server only uses the root dataset to train the global model. Therefore, there is no communications between the clients and the server during the training process. We use this variant to show that the server cannot obtain a good model using its root dataset alone. In other words, even if some clients are malicious, communicating with clients still improves the global model.
- **FLTrust-withServer.** In this variant, the server computes the weighted average of the clients' local model updates together with the server model update whose trust score is 1.
- **FLTrust-NoReLU.** In this variant, the server does not use ReLU to clip the cosine similarity scores of the local model updates when computing their trust scores.
- **FLTrust-NoNorm.** In this variant, the server does not normalize the local model updates to have the same magnitude as the server model update.
- **FLTrust-ParNorm.** In this variant, the server applies partial normalization, i.e., only normalizes the local model updates whose magnitudes are larger than that of the server model update to have the same magnitude as the server model update.

Table IV compares the variants with respect to their testing error rates under different attacks and the attack success rates of the Scaling attacks on MNIST-0.5. The attacks are not applicable to FLTrust-Server as it does not require communications from the clients. Our results show that FLTrust outperforms the five variants. FLTrust outperforms FLTrust-Server and FLTrust-withServer because the root dataset is small. The fact that FLTrust outperforms FLTrust-NoReLU, FLTrust-NoNorm, and FLTrust-ParNorm indicates the necessity of our ReLU operation and normalization.

Impact of the root dataset: Our root dataset can be characterized by its size and how it is sampled (i.e., Case I vs. Case II). Therefore, we study the impact of the root dataset on FLTrust with respect to its size and how it is sampled. Figure 4 shows the testing error rates of FLTrust under different attacks and the attack success rates under the Scaling attack on MNIST-0.5 when the size of the root dataset increases from 50 to 500, where the root dataset is sampled uniformly in Case I. We observe that a root dataset with only 100 training examples is sufficient for FLTrust to defend against the attacks.

TABLE V: The testing error rates of FLTrust under different attacks and the attack success rates of the Scaling attacks when the root dataset is sampled with different bias probabilities in Case II.

(a) MNIST-0.1

Bias probability	0.1	0.2	0.4	0.6	0.8	1.0
No attack	0.04	0.04	0.04	0.05	0.05	0.34
LF attack	0.04	0.04	0.04	0.05	0.78	0.84
Krum attack	0.04	0.04	0.07	0.89	0.89	0.89
Trim attack	0.04	0.05	0.08	0.12	0.46	0.89
Scaling attack	0.03 / 0.00	0.03 / 0.01	0.04 / 0.00	0.04 / 0.00	0.06 / 0.01	0.42 / 0.01
Adaptive attack	0.04	0.05	0.08	0.12	0.90	0.90

(b) MNIST-0.5

Bias probability	0.1	0.2	0.4	0.6	0.8	1.0
No attack	0.05	0.05	0.06	0.08	0.11	0.80
LF attack	0.05	0.05	0.08	0.10	0.25	0.89
Krum attack	0.05	0.05	0.08	0.12	0.86	0.89
Trim attack	0.06	0.06	0.08	0.12	0.16	0.89
Scaling attack	0.05 / 0.00	0.05 / 0.01	0.06 / 0.00	0.07 / 0.01	0.12 / 0.00	0.86 / 0.01
Adaptive attack	0.06	0.07	0.08	0.13	0.90	0.90

(c) Fashion-MNIST

Bias probability	0.1	0.2	0.4	0.6	0.8	1.0
No attack	0.11	0.11	0.12	0.15	0.16	0.90
LF attack	0.11	0.11	0.12	0.12	0.14	0.90
Krum attack	0.12	0.12	0.16	0.90	0.90	0.90
Trim attack	0.14	0.14	0.15	0.21	0.90	0.90
Scaling attack	0.11 / 0.02	0.12 / 0.04	0.12 / 0.04	0.13 / 0.02	0.15 / 0.03	0.90 / 0.00
Adaptive attack	0.14	0.14	0.16	0.90	0.90	0.90

(d) CIFAR-10

Bias probability	0.1	0.2	0.4	0.6	0.8	1.0
No attack	0.18	0.18	0.18	0.21	0.90	0.90
LF attack	0.18	0.19	0.20	0.24	0.90	0.90
Krum attack	0.18	0.18	0.19	0.33	0.90	0.90
Trim attack	0.20	0.20	0.24	0.63	0.90	0.90
Scaling attack	0.18 / 0.02	0.18 / 0.00	0.18 / 0.03	0.22 / 0.04	0.90 / 0.00	0.90 / 0.00
Adaptive attack	0.20	0.20	0.27	0.68	0.90	0.90

(e) HAR

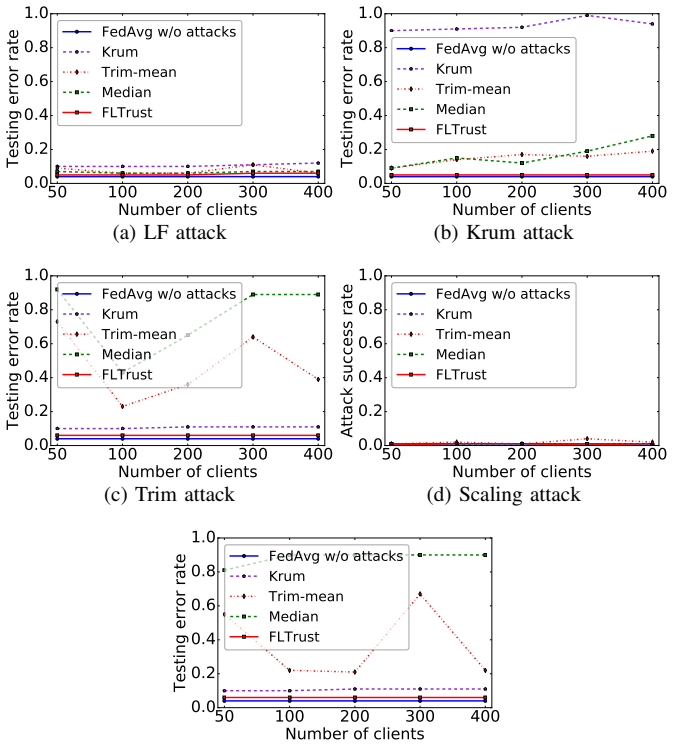
Bias probability	0.17	0.2	0.4	0.6	0.8	1.0
No attack	0.04	0.04	0.06	0.06	0.07	0.48
LF attack	0.04	0.05	0.06	0.05	0.07	0.48
Krum attack	0.04	0.05	0.05	0.05	0.09	0.48
Trim attack	0.05	0.05	0.06	0.09	0.14	0.48
Scaling attack	0.05 / 0.01	0.05 / 0.01	0.06 / 0.02	0.06 / 0.03	0.07 / 0.05	0.48 / 0.34
Adaptive attack	0.05	0.05	0.06	0.09	0.48	0.48

(f) CH-MNIST

Bias probability	0.125	0.2	0.4	0.6	0.8	1.0
No attack	0.10	0.10	0.11	0.13	0.13	0.89
LF attack	0.12	0.12	0.12	0.17	0.21	0.89
Krum attack	0.12	0.12	0.14	0.17	0.19	0.89
Trim attack	0.13	0.13	0.14	0.20	0.20	0.89
Scaling attack	0.14 / 0.03	0.14 / 0.02	0.15 / 0.02	0.16 / 0.06	0.14 / 0.01	0.89 / 0.01
Adaptive attack	0.13	0.14	0.14	0.89	0.89	0.89

Specifically, when the root dataset has 100 training examples, the testing error rates of FLTrust under attacks are similar to that of FedAvg without attacks, and the attack success rate of the Scaling attack is close to 0. When the size of the root dataset increases beyond 100, the testing error rates and attack success rates of FLTrust further decrease slightly.

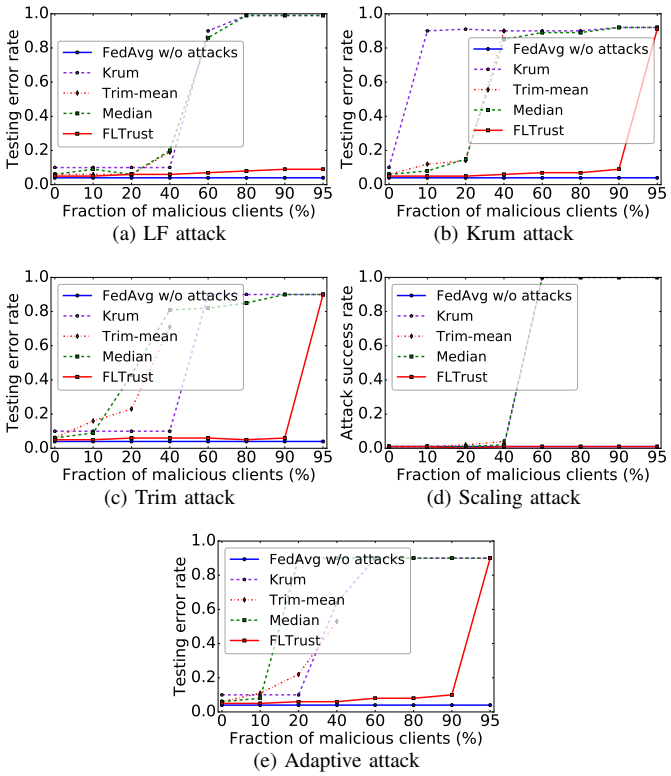
We also evaluate the impact of the bias probability in Case II. Table V shows the testing error rates of FLTrust under different attacks and the attack success rates of the Scaling attacks when the bias probability varies. The second column in each table corresponds to the bias probability with which Case II reduces to Case I. We increase the bias probability up to 1.0 to simulate larger difference between the root data distribution and the overall training data distribution. We observe that FLTrust is accurate and robust when the bias probability is not too large. For instance, when the bias probability is no more than 0.4 for MNIST-0.5, the testing error rates of FLTrust under attacks are at most 0.08, compared to 0.05 when the bias probability is 0.1. Our results show that FLTrust works well when the root data distribution does not diverge too much from the overall training data distribution.



(a) LF attack (b) Krum attack (c) Trim attack (d) Scaling attack (e) Adaptive attack

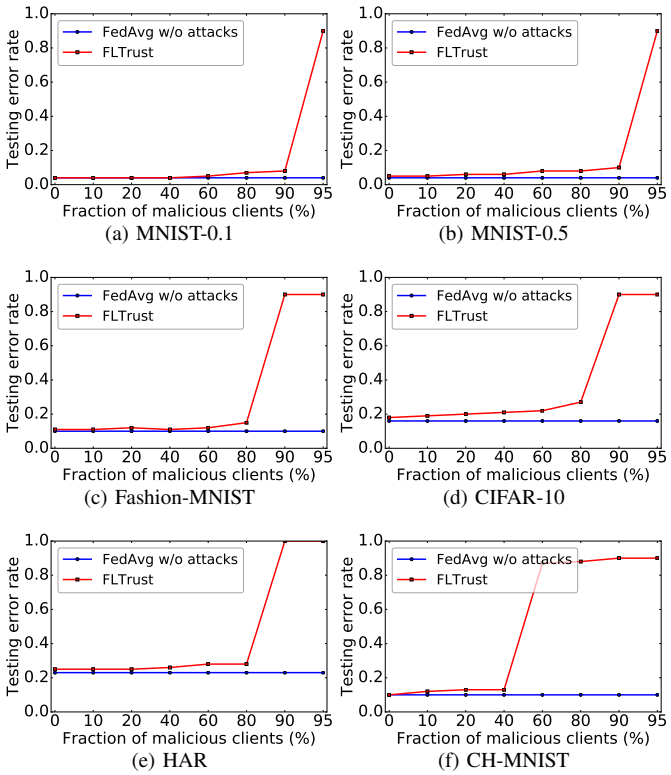
Fig. 5: Impact of the total number of clients on the testing error rates of different FL methods under different attacks ((a)-(c)) and the attack success rates of the Scaling attacks, where MNIST-0.5 is used. The testing error rates of all the compared FL methods are similar and small under the Scaling attacks, which we omit for simplicity.

Impact of the total number of clients: Figure 5 shows the testing error rates of different FL methods under different attacks, as well as the attack success rates of the Scaling attacks, when the total number of clients n increases from 50 to 400. We set the fraction of malicious clients to be $\frac{m}{n} = 20\%$. We observe that FLTrust can defend against the attacks for all considered total number of clients. Specifically, FLTrust under attacks achieves testing error rates similar to FedAvg under no attacks, while the attack success rates of the Scaling attacks are close to 0 for FLTrust. Existing methods can defend against the Scaling attacks on MNIST-0.5, i.e., the attack success rates are close to 0. However, they cannot defend against the Krum attack, Trim attack, and/or our adaptive attack, i.e., their corresponding testing error rates are large.



(a) LF attack (b) Krum attack (c) Trim attack (d) Scaling attack (e) Adaptive attack

Fig. 6: Impact of the fraction of malicious clients on the testing error rates of different FL methods under different attacks ((a)-(c)) and the attack success rates of the Scaling attacks, where MNIST-0.5 is used. The testing error rates of all the compared FL methods are similar and small under the Scaling attacks, which we omit for simplicity.



(a) MNIST-0.1 (b) MNIST-0.5 (c) Fashion-MNIST (d) CIFAR-10 (e) HAR (f) CH-MNIST

Fig. 7: Impact of the fraction of malicious clients on the testing error rates of FLTrust under the adaptive attacks.

Impact of the number of malicious clients: Figure 6 shows the testing error rates of different FL methods under different attacks and the attack success rates of the Scaling attacks on MNIST-0.5, when the fraction of malicious clients increases from 0 to 95%. Trim-mean cannot be applied when the fraction of malicious clients exceeds 50% because the number of local model updates removed by Trim-mean is twice of the number of malicious clients. Therefore, for Trim-mean, we only show the results when the malicious clients are less than 50%.

We observe that, under existing attacks and our adaptive attacks, FLTrust can tolerate up to 90% of malicious clients. Specifically, FLTrust under these attacks still achieves testing error rates similar to FedAvg without attacks when up to 90% of the clients are malicious, while the attack success rates of the Scaling attacks for FLTrust are still close to 0 when up to 95% of the clients are malicious. However, existing Byzantine-robust FL methods can tolerate much less malicious clients. For instance, under Krum attack, the testing error rate of the global model learnt by Krum increases to 0.90 when only 10% of the clients are malicious, while the testing error rates of the global models learnt by Trim-mean and Median become larger than 0.85 when the fraction of malicious clients reaches 40%.

Figure 7 further shows the testing error rates of the global models learnt by FLTrust as a function of the fraction of malicious clients under the adaptive attacks on all datasets. Our results show that FLTrust is robust against adaptive attacks even if a large fraction of clients are malicious on all datasets. Specifically, for MNIST-0.1 (MNIST-0.5, Fashion-MNIST, CIFAR-10, HAR, or CH-MNIST), FLTrust under adaptive attacks with over 60% (over 40%, up to 60%, up to 60%, up to 40%, or over 40%) of malicious clients can still achieve testing error rates similar to FedAvg under no attack.

VII. Discussion and Limitations

FLTrust vs. fault-tolerant computing: Fault-tolerant computing [6] aims to remain functional when there are malicious clients. However, conventional fault-tolerant computing and federated learning have the following key difference: the clients communicate with each other to compute results in fault-tolerant computing [6], while clients only communicate with a cloud server in federated learning. Our FLTrust leverages such unique characteristics of federated learning to bootstrap trust, i.e., the server collects a root dataset, and uses it to guide the aggregation of the local model updates.

Different ways of using the root dataset: Fang et al. [15] also proposed to use a root dataset (they called it validation dataset). However, we use the root dataset in a way that is different from theirs. In particular, they use the root dataset to remove potentially malicious local model updates in each iteration, while we use it to assign trust scores to clients and normalize local model updates. As shown by Fang et al. [15], their way of using the root dataset is not effective in many cases.

Poisoned root dataset: Our FLTrust requires a clean root dataset. We acknowledge that FLTrust may not be robust against poisoned root dataset. The root dataset may be poisoned when it is collected from the Internet or by an insider attacker. However, since FLTrust only requires a small root dataset, a service provider can collect a clean one by itself with a small cost, e.g., asking its employees to generate and manually label a clean root dataset.

Adaptive attacks and hierarchical root of trust: We considered an adaptive attack via extending the state-of-the-art framework of local model poisoning attacks to our FLTrust. We acknowledge that there may exist stronger local model poisoning attacks to FLTrust, which is an interesting future work to explore. Moreover, it is an interesting future work to consider a hierarchical root of trust. For instance, the root dataset may contain multiple subsets with different levels of trust. The subsets with higher trust may have a larger impact on the aggregation.

VIII. Conclusion and Future Work

We proposed and evaluated a new federated learning method called FLTrust to achieve Byzantine robustness against malicious clients. The key difference between our FLTrust and existing federated learning methods is that the server itself collects a clean small training dataset (i.e., root dataset) to bootstrap trust in FLTrust. Our extensive evaluations on six datasets show that FLTrust with a small root dataset can achieve Byzantine robustness against a large fraction of malicious clients. In particular, FLTrust under adaptive attacks with a large fraction of malicious clients can still train global models that are as good as the global models learnt by FedAvg under no attacks. Interesting future work includes 1) designing stronger local model poisoning attacks to FLTrust and 2) considering a hierarchical root of trust.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. This work was supported in part by NSF grants No. 1937786, 1943226, and 2110252, an IBM Faculty Award, and a Google Faculty Research Award.

References

[1] *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[2] *Machine Learning Ledger Orchestration For Drug Discovery (MELLODDY)*. [Online]. Available: https://www.melloddy.eu/

[3] *Utilization of FATE in Risk Management of Credit in Small and Micro Enterprises*. [Online]. Available: https://www.fedai.org/cases/utilization-of-fate-in-risk-management-of-credit-in-small-and-micro-enterprises/

[4] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *ESANN*, 2013.

[5] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *AISTATS*, 2020, pp. 2938–2948.

[6] M. Barborak, A. Dahbura, and M. Malek, "The consensus problem in fault-tolerant computing," *ACM Computing Surveys (CSur)*, vol. 25, no. 2, pp. 171–220, 1993.

[7] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *ICML*, 2019, pp. 634–643.

[8] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *ICML*, 2012.

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

[10] X. Cao, J. Jia, and N. Z. Gong, "Data poisoning attacks to local differential privacy protocols," *arXiv preprint arXiv:1911.02046*, 2019.

[11] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," in *arxiv*, 2017.

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

[13] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-efficient hard-label black-box attack: An optimization-based approach," in *ICLR*, 2019.

[14] A. Cheu, A. Smith, and J. Ullman, "Manipulation attacks in local differential privacy," *arXiv preprint arXiv:1909.09630*, 2019.

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

[16] M. Fang, N. Z. Gong, and J. Liu, "Influence function based data poisoning attacks to top-n recommender systems," in *Proceedings of The Web Conference 2020*, 2020, pp. 3019–3025.

[17] M. Fang, G. Yang, N. Z. Gong, and J. Liu, "Poisoning attacks to graph-based recommender systems," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 381–392.

[18] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," in *Machine Learning and Computer Security Workshop*, 2017.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[20] J. Jia, B. Wang, X. Cao, and N. Z. Gong, "Certified robustness of community detection against adversarial structural perturbation via randomized smoothing," in *Proceedings of The Web Conference 2020*, 2020, pp. 2718–2724.

[21] J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Zöllner, "Multi-class texture analysis in colorectal cancer histology," *Scientific reports*, vol. 6, p. 27988, 2016.

[22] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[23] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *Available: http://yann. lecun. com/exdb/mnist*, 1998.

[25] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *NIPS*, 2016.

[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 *AAAI*, vol. 33, 2019, pp. 1544–1551.

[27] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *NDSS*, 2018.

[28] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[29] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *ICML*, 2018.

[30] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *AISec*, 2017.

[31] L. Muñoz-González, K. T. Co, and E. C. Lupu, "Byzantine-robust federated machine learning through adaptive model averaging," *arXiv preprint arXiv:1909.05125*, 2019.

[32] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia., "Exploiting machine learning to subvert your spam filter," in *LEET*, 2008.

[33] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," vol. 17, no. 2. Springer, 2017, pp. 527–566.

[34] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, "Detox: A redundancy-based framework for faster and more robust gradient aggregation," in *NIPS*, 2019, pp. 10 320–10 330.

[35] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. Tygar, "Antidote: understanding and defending against poisoning of anomaly detectors," in *ACM IMC*, 2009.

[36] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *NIPS*, 2018.

[37] O. Suciu, R. Marginean, Y. Kaya, H. D. III, and T. Dumitras, "When does machine learning fail? generalized transferability for evasion and poisoning attacks," in *USENIX Security Symposium*, 2018.

[38] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," *arXiv preprint arXiv:2007.08432*, 2020.

[39] R. Vershynin, "Introduction to the non-asymptotic analysis of random matrices," *arXiv preprint arXiv:1011.3027*, 2010.

[40] M. J. Wainwright, "High-dimensional statistics: A non-asymptotic viewpoint," vol. 48. Cambridge University Press, 2019.

[41] B. Wang and N. Z. Gong, "Attacking graph-based classification via manipulating the graph structure," in *CCS*, 2019.

[42] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[43] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *ICLR*, 2020.

[44] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *ICML*, 2019, pp. 6893–6901.

[45] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems," in *NDSS*, 2017.

[46] H. Yang, X. Zhang, M. Fang, and J. Liu, "Byzantine-resilient stochastic gradient descent for distributed learning: A lipschitz-inspired coordinate-wise median approach," in *CDC*. IEEE, 2019, pp. 5832–5837.

[47] Z. Yang and W. U. Bajwa, "Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 4, pp. 611–627, 2019.

[48] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *ICML*, 2018.

[49] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong, "Backdoor attacks to graph neural networks," *arXiv preprint arXiv:2006.11165*, 2020.

APPENDIX

A. Proof of Theorem 1

Before proving Theorem 1, we first restate our FLTrust algorithm and prove some lemmas. We note that in our setting where $R_l = 1$, only the combined learning rate $\alpha \cdot \beta$ influences FLTrust. Therefore, given a combined learning rate, we can always set $\beta = 1$ and let α be the combined learning rate. In this case, the local model update $\boldsymbol{g}_i$ and server update $\boldsymbol{g}_0$ are equivalent to the gradients of the ith client and the server, respectively. We denote by $\mathcal{S}$ the set of clients whose cosine similarity c_i is positive in the tth global iteration. Let $\bar{\boldsymbol{g}}_i = \frac{\|\boldsymbol{g}_0\|}{\|\boldsymbol{g}_i\|} \cdot \boldsymbol{g}_i$ and $\varphi_i = \frac{ReLU(c_i)}{\sum_{j \in \mathcal{S}} ReLU(c_j)} = \frac{c_i}{\sum_{j \in \mathcal{S}} c_j}$, where $i \in \mathcal{S}$. Then, we can rewrite Equation (4) as:

$$\boldsymbol{g} = \sum_{i \in \mathcal{S}} \varphi_i \bar{\boldsymbol{g}}_i, \quad \text{s.t.} \sum_{i \in \mathcal{S}} \varphi_i = 1, 0 < \varphi_i < 1. \tag{12}$$

Lemma 1. *For an arbitrary number of malicious clients, the distance between $\boldsymbol{g}$ and $\nabla F(\boldsymbol{w})$ is bounded as follows in each iteration:*

$$\|\boldsymbol{g} - \nabla F(\boldsymbol{w})\| \leq 3 \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| + 2 \|\nabla F(\boldsymbol{w})\|.$$

Proof: We have the following equations:

$$\begin{aligned}
&\|\boldsymbol{g} - \nabla F(\boldsymbol{w})\| && (13)\\
&= \left\| \sum_{i \in \mathcal{S}} \varphi_i \bar{\boldsymbol{g}}_i - \nabla F(\boldsymbol{w}) \right\| && (14)\\
&= \left\| \sum_{i \in \mathcal{S}} \varphi_i \bar{\boldsymbol{g}}_i - \boldsymbol{g}_0 + \boldsymbol{g}_0 - \nabla F(\boldsymbol{w}) \right\| && (15)\\
&\leq \left\| \sum_{i \in \mathcal{S}} \varphi_i \bar{\boldsymbol{g}}_i - \boldsymbol{g}_0 \right\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (16)\\
&\overset{(a)}{\leq} \left\| \sum_{i \in \mathcal{S}} \varphi_i \bar{\boldsymbol{g}}_i + \boldsymbol{g}_0 \right\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (17)\\
&\leq \left\| \sum_{i \in \mathcal{S}} \varphi_i \bar{\boldsymbol{g}}_i \right\| + \|\boldsymbol{g}_0\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (18)\\
&\leq \sum_{i \in \mathcal{S}} \varphi_i \|\bar{\boldsymbol{g}}_i\| + \|\boldsymbol{g}_0\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (19)\\
&\overset{(b)}{=} \sum_{i \in \mathcal{S}} \varphi_i \|\boldsymbol{g}_0\| + \|\boldsymbol{g}_0\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (20)\\
&\overset{(c)}{=} 2 \|\boldsymbol{g}_0\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (21)\\
&= 2 \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w}) + \nabla F(\boldsymbol{w})\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (22)\\
&\leq 2 \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| + 2 \|\nabla F(\boldsymbol{w})\| + \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| && (23)\\
&= 3 \|\boldsymbol{g}_0 - \nabla F(\boldsymbol{w})\| + 2 \|\nabla F(\boldsymbol{w})\|, && (24)
\end{aligned}$$

where (a) is because $\langle \bar{\boldsymbol{g}}_i, \boldsymbol{g}_0 \rangle > 0$ for $i \in \mathcal{S}$; (b) is because FLTrust normalizes the local model updates to have the same magnitude as the server model update, i.e., $\|\bar{\boldsymbol{g}}_i\| = \|\boldsymbol{g}_0\|$; and (c) is because $\sum_{i \in \mathcal{S}} \varphi_i = 1$. ■

Lemma 2. *Assume Assumption 1 holds. If we set the learning rate as $\alpha = \mu/(2L^2)$, then we have the following in any global iteration $t \geq 1$:*

$$\begin{aligned}
&\left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* - \alpha \nabla F(\boldsymbol{w}^{t-1}) \right\| \\
&\leq \sqrt{1 - \mu^2/(4L^2)} \left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* \right\|.
\end{aligned}$$

Proof: Since $\nabla F(\boldsymbol{w}^*) = 0$, we have the following:

$$\begin{aligned}
&\left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* - \alpha \nabla F(\boldsymbol{w}^{t-1}) \right\|^2 && (25)\\
&= \left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* - \alpha \left(\nabla F(\boldsymbol{w}^{t-1}) - \nabla F(\boldsymbol{w}^*) \right) \right\|^2 && (26)\\
&= \left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* \right\|^2 + \alpha^2 \left\| \nabla F(\boldsymbol{w}^{t-1}) - \nabla F(\boldsymbol{w}^*) \right\|^2 \\
&\quad - 2\alpha \left\langle \boldsymbol{w}^{t-1} - \boldsymbol{w}^*, \nabla F(\boldsymbol{w}^{t-1}) - \nabla F(\boldsymbol{w}^*) \right\rangle. && (27)
\end{aligned}$$

By Assumption 1, we have:

$$\begin{aligned}
&\left\| \nabla F(\boldsymbol{w}^{t-1}) - \nabla F(\boldsymbol{w}^*) \right\| \leq L \left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* \right\|, && (28)\\
&F(\boldsymbol{w}^*) + \left\langle \nabla F(\boldsymbol{w}^*), \boldsymbol{w}^{t-1} - \boldsymbol{w}^* \right\rangle \leq F(\boldsymbol{w}^{t-1}) \\
&\quad - \frac{\mu}{2} \left\| \boldsymbol{w}^{t-1} - \boldsymbol{w}^* \right\|^2, && (29)\\
&F(\boldsymbol{w}^{t-1}) + \left\langle \nabla F(\boldsymbol{w}^{t-1}), \boldsymbol{w}^* - \boldsymbol{w}^{t-1} \right\rangle \leq F(\boldsymbol{w}^*). && (30)
\end{aligned}$$

Summing up inequalities (29) and (30), we have:

$$\left\langle \boldsymbol{w}^* - \boldsymbol{w}^{t-1}, \nabla F(\boldsymbol{w}^{t-1}) - \nabla F(\boldsymbol{w}^*) \right\rangle$$

$$\leq -\frac{\mu}{2}\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\|^2. \tag{31}$$

Substituting inequalities (28) and (31) into (27), we have:

$$\begin{aligned}&\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*-\alpha\nabla F(\boldsymbol{w}^{t-1})\right\|^2\\&\leq\left(1+\alpha^2L^2-\alpha\mu\right)\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\|^2.\end{aligned} \tag{32}$$

By choosing $\alpha=\mu/(2L^2)$, we have:

$$\begin{aligned}&\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*-\alpha\nabla F(\boldsymbol{w}^{t-1})\right\|^2\\&\leq\left(1-\mu^2/(4L^2)\right)\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\|^2,\end{aligned} \tag{33}$$

which concludes the proof. ■

Lemma 3. *Suppose Assumption 2 holds. For any $\delta\in(0,1)$ and any $\boldsymbol{w}\in\Theta$, we let $\Delta_1=\sqrt{2}\sigma_1\sqrt{(d\log 6+\log(3/\delta))/|D_0|}$ and $\Delta_3=\sqrt{2}\sigma_2\sqrt{(d\log 6+\log(3/\delta))/|D_0|}$. If $\Delta_1\leq\sigma_1^2/\gamma_1$ and $\Delta_3\leq\sigma_2^2/\gamma_2$, then we have:*

$$Pr\left\{\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*)\right\|\geq 2\Delta_1\right\}\leq\frac{\delta}{3},$$

$$\begin{aligned}Pr\Bigg\{&\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla h(X_i,\boldsymbol{w})-\mathbb{E}\left[h(X,\boldsymbol{w})\right]\right\|\\&\geq 2\Delta_3\|\boldsymbol{w}-\boldsymbol{w}^*\|\Bigg\}\leq\frac{\delta}{3}.\end{aligned}$$

Proof: We prove the first inequality of Lemma 3. The proof of the second inequality is similar, and we omit it for brevity. Let $\boldsymbol{V}=\{\boldsymbol{v}_1,,\cdots,\boldsymbol{v}_{N_{\frac{1}{2}}}\}$ be an $\frac{1}{2}$-cover of the unit sphere $\boldsymbol{B}$. It is shown in [12], [39] that we have $\log N_{\frac{1}{2}}\leq d\log 6$ and the following:

$$\begin{aligned}&\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*)\right\|\leq\\&2\sup_{\boldsymbol{v}\in\boldsymbol{V}}\left\{\left\langle\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*),\boldsymbol{v}\right\rangle\right\}.\end{aligned} \tag{34}$$

According to the concentration inequalities for sub-exponential random variables [40], when Assumption 2 and condition $\Delta_1\leq\sigma_1^2/\gamma_1$ hold, we have:

$$\begin{aligned}&Pr\left\{\left\langle\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*),\boldsymbol{v}\right\rangle\geq\Delta_1\right\}\\&\leq\exp\left(-\left|D_0\right|\Delta_1^2/(2\sigma_1^2)\right).\end{aligned} \tag{35}$$

Taking the union bound over all vectors in $\boldsymbol{V}$ and combining it with inequality (34), we have:

$$\begin{aligned}&Pr\left\{\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*)\right\|\geq 2\Delta_1\right\}\\&\leq\exp\left(-\left|D_0\right|\Delta_1^2/(2\sigma_1^2)+d\log 6\right).\end{aligned} \tag{36}$$

We conclude the proof by letting $\Delta_1=\sqrt{2}\sigma_1\sqrt{(d\log 6+\log(3/\delta))/|D_0|}$ in (36). ■

Lemma 4. *Suppose Assumptions 1-3 hold and $\Theta\subset\left\{\boldsymbol{w}:\|\boldsymbol{w}-\boldsymbol{w}^*\|\leq r\sqrt{d}\right\}$ holds for some positive parameter r. Then, for any $\delta\in(0,1)$, if $\Delta_1\leq\sigma_1^2/\gamma_1$ and $\Delta_2\leq\sigma_2^2/\gamma_2$, we have the following for any $\boldsymbol{w}\in\Theta$:*

$$Pr\left\{\|\boldsymbol{g}_0-\nabla F(\boldsymbol{w})\|\leq 8\Delta_2\|\boldsymbol{w}-\boldsymbol{w}^*\|+4\Delta_1\right\}\geq 1-\delta,$$

where $\Delta_2=\sigma_2\sqrt{\frac{2}{|D_0|}}\sqrt{K_1+K_2}$, $K_1=d\log\frac{18L_2}{\sigma_2}$, $K_2=\frac{1}{2}d\log\frac{|D_0|}{d}+\log\left(\frac{6\sigma_2^2r\sqrt{|D_0|}}{\gamma_2\sigma_1\delta}\right)$, $L_2=\max\{L,L_1\}$, and $|D_0|$ is the size of the root dataset.

Proof: Our proof is mainly based on the ε-net argument [39] and [12]. We let $\tau=\frac{\gamma_2\sigma_1}{2\sigma_2^2}\sqrt{\frac{d}{|D_0|}}$ and ℓ^* be an integer that satisfies $\ell^*=\left\lceil r\sqrt{d}/\tau\right\rceil$. For any integer $1\leq\ell\leq\ell^*$, we define $\Theta_\ell=\{\boldsymbol{w}:\|\boldsymbol{w}-\boldsymbol{w}^*\|\leq\tau\ell\}$. Given an integer ℓ, we let $\boldsymbol{w}_1,\cdots,\boldsymbol{w}_{N_{\varepsilon_\ell}}$ be an ε_ℓ-cover of Θ_ℓ, where $\varepsilon_\ell=\frac{\sigma_2\tau\ell}{L_2}\sqrt{\frac{d}{|D_0|}}$ and $L_2=\max\{L,L_1\}$. From [39], we know that $\log N_{\varepsilon_\ell}\leq d\log\left(\frac{3\tau\ell}{\varepsilon_\ell}\right)$. For any $\boldsymbol{w}\in\Theta_\ell$, there exists a j_ℓ ($1\leq j_\ell\leq N_{\varepsilon_\ell}$) such that:

$$\|\boldsymbol{w}-\boldsymbol{w}_{j_\ell}\|\leq\varepsilon_\ell. \tag{37}$$

According to the triangle inequality, we have:

$$\begin{aligned}&\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w})-\nabla F(\boldsymbol{w})\right\|\leq\|\nabla F(\boldsymbol{w})-\nabla F(\boldsymbol{w}_{j_\ell})\|\\&+\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\left(\nabla f(X_i,\boldsymbol{w})-\nabla f(X_i,\boldsymbol{w}_{j_\ell})\right)\right\|\\&+\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}_{j_\ell})-\nabla F(\boldsymbol{w}_{j_\ell})\right\|.\end{aligned} \tag{38}$$

According to Assumption 1 and inequality (37), we have:

$$\|\nabla F(\boldsymbol{w})-\nabla F(\boldsymbol{w}_{j_\ell})\|\leq L\|\boldsymbol{w}-\boldsymbol{w}_{j_\ell}\|\leq L\varepsilon_\ell \tag{39}$$

Next, we define an event $\mathcal{E}_1$ as follows:

$$\mathcal{E}_1=\left\{\sup_{\boldsymbol{w},\widehat{\boldsymbol{w}}\in\Theta:\boldsymbol{w}\neq\widehat{\boldsymbol{w}}}\frac{\|\nabla f(X,\boldsymbol{w})-\nabla f(X,\widehat{\boldsymbol{w}})\|}{\|\boldsymbol{w}-\widehat{\boldsymbol{w}}\|}\leq L_1\right\}.$$

According to Assumption 2, we have $Pr\{\mathcal{E}_1\}\geq 1-\frac{\delta}{3}$. Moreover, we have the following:

$$\begin{aligned}&\sup_{\boldsymbol{w}\in\Theta}\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\left(\nabla f(X_i,\boldsymbol{w})-\nabla f(X_i,\boldsymbol{w}_{j_\ell})\right)\right\|\\&\leq L_1\|\boldsymbol{w}-\boldsymbol{w}_{j_\ell}\|\leq L_1\varepsilon_\ell.\end{aligned} \tag{40}$$

According to the triangle inequality, we have:

$$\begin{aligned}
&\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}_{j_\ell})-\nabla F(\boldsymbol{w}_{j_\ell})\right\| \\
&\le \left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*)\right\| \\
&\quad+\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\left(\nabla f(X_i,\boldsymbol{w}_{j_\ell})-\nabla f(X_i,\boldsymbol{w}^*)\right)\right. \\
&\qquad\left.-\left(\nabla F(\boldsymbol{w}_{j_\ell})-\nabla F(\boldsymbol{w}^*)\right)\right\| \\
&\overset{(a)}{\le} \left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*)\right\| \\
&\quad+\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}h(X_i,\boldsymbol{w}_{j_\ell})-\mathbb{E}\left[h\left(X,\boldsymbol{w}_{j_\ell}\right)\right]\right\|, \qquad (41)
\end{aligned}$$

where (a) is due to $\mathbb{E}\left[h\left(X,\boldsymbol{w}\right)\right]=\nabla F(\boldsymbol{w})-\nabla F(\boldsymbol{w}^*)$.

We also define events $\mathcal{E}_2$ and $\mathcal{E}_3(\ell)$ as:

$$\begin{aligned}
\mathcal{E}_2&=\left\{\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w}^*)-\nabla F(\boldsymbol{w}^*)\right\|\le 2\Delta_1\right\}, \\
\mathcal{E}_3(\ell)&=\left\{\sup_{1\le j\le N_\varepsilon}\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}h(X_i,\boldsymbol{w}_j)-\mathbb{E}\left[h\left(X,\boldsymbol{w}_j\right)\right]\right\|\right. \\
&\qquad\left.\le 2\Delta_2\tau\ell\right\}.
\end{aligned}$$

According to Lemma 3 and [12], $\Delta_1\le\sigma_1^2/\gamma_1$, and $\Delta_2\le\sigma_2^2/\gamma_2$, we have $Pr\{\mathcal{E}_2\}\ge 1-\frac{\delta}{3}$ and $Pr\{\mathcal{E}_3(\ell)\}\ge 1-\frac{\delta}{3\ell^*}$. Therefore, on event $\mathcal{E}_1\cap\mathcal{E}_2\cap\mathcal{E}_3(\ell)$, we have:

$$\begin{aligned}
&\sup_{\boldsymbol{w}\in\Theta_\ell}\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w})-\nabla F(\boldsymbol{w})\right\| \\
&\le L\varepsilon_\ell+L_1\varepsilon_\ell+2\Delta_1+2\Delta_2\tau\ell, \qquad (42)\\
&\overset{(a)}{\le} 2L_2\varepsilon_\ell+2\Delta_1+2\Delta_2\tau\ell\overset{(b)}{\le}4\Delta_2\tau\ell+2\Delta_1, \qquad (43)
\end{aligned}$$

where (a) holds because $(L+L_1)\le 2L_2$ and (b) is due to $\Delta_2\ge\sigma_2\sqrt{d/|D_0|}$.

Thus, according to the union bound, we have probability at least $1-\delta$ that event $\mathcal{E}_1\cap\mathcal{E}_2\cap(\cap_{\ell=1}^{\ell^*}\mathcal{E}_3(\ell))$ holds. On event $\mathcal{E}_1\cap\mathcal{E}_2\cap(\cap_{\ell=1}^{\ell^*}\mathcal{E}_3(\ell))$, for any $\boldsymbol{w}\in\Theta_{\ell^*}$, there exists an $1\le\ell\le\ell^*$ such that $(\ell-1)\tau<\|\boldsymbol{w}-\boldsymbol{w}^*\|\le\ell\tau$ holds. If $\ell=1$, then we have:

$$\begin{aligned}
&\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w})-\nabla F(\boldsymbol{w})\right\| \\
&\le 4\Delta_2\tau+2\Delta_1\overset{(a)}{\le}4\Delta_1, \qquad (44)
\end{aligned}$$

where (a) holds because $\Delta_2\le\sigma_2^2/\gamma_2$ and $\Delta_1\ge\sigma_1\sqrt{d/|D_0|}$. If $\ell\ge 2$, then we have $2(\ell-1)\ge\ell$ and the following:

$$\begin{aligned}
&\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w})-\nabla F(\boldsymbol{w})\right\| \\
&\le 8\Delta_2\left\|\boldsymbol{w}-\boldsymbol{w}^*\right\|+2\Delta_1. \qquad (45)
\end{aligned}$$

Combining inequalities (44) and (45), we have:

$$\begin{aligned}
&\sup_{\boldsymbol{w}\in\Theta_{\ell^*}}\left\|\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w})-\nabla F(\boldsymbol{w})\right\| \\
&\le 8\Delta_2\left\|\boldsymbol{w}-\boldsymbol{w}^*\right\|+4\Delta_1. \qquad (46)
\end{aligned}$$

We conclude the proof since $\Theta\subset\Theta_{\ell^*}$ and $\boldsymbol{g}_0=\frac{1}{|D_0|}\sum_{X_i\in D_0}\nabla f(X_i,\boldsymbol{w})$. ■

Proof of Theorem 1: With the lemmas above, we can prove Theorem 1 next. We have the following equations for the tth global iteration:

$$\begin{aligned}
&\left\|\boldsymbol{w}^t-\boldsymbol{w}^*\right\| \\
&=\left\|\boldsymbol{w}^{t-1}-\alpha\boldsymbol{g}^{t-1}-\boldsymbol{w}^*\right\| \\
&=\left\|\boldsymbol{w}^{t-1}-\alpha\nabla F(\boldsymbol{w}^{t-1})-\boldsymbol{w}^*+\alpha\nabla F(\boldsymbol{w}^{t-1})-\alpha\boldsymbol{g}^{t-1}\right\| \\
&\le\left\|\boldsymbol{w}^{t-1}-\alpha\nabla F(\boldsymbol{w}^{t-1})-\boldsymbol{w}^*\right\|+\alpha\left\|\boldsymbol{g}^{t-1}-\nabla F(\boldsymbol{w}^{t-1})\right\| \\
&\overset{(a)}{\le}\left\|\boldsymbol{w}^{t-1}-\alpha\nabla F(\boldsymbol{w}^{t-1})-\boldsymbol{w}^*\right\|+3\alpha\left\|\boldsymbol{g}_0^{t-1}-\nabla F(\boldsymbol{w}^{t-1})\right\| \\
&\quad+2\alpha\left\|\nabla F(\boldsymbol{w}^{t-1})\right\| \\
&\overset{(b)}{=}\underbrace{\left\|\boldsymbol{w}^{t-1}-\alpha\nabla F(\boldsymbol{w}^{t-1})-\boldsymbol{w}^*\right\|}_{A_1}+3\alpha\underbrace{\left\|\boldsymbol{g}_0^{t-1}-\nabla F(\boldsymbol{w}^{t-1})\right\|}_{A_2} \\
&\quad+2\alpha\underbrace{\left\|\nabla F(\boldsymbol{w}^{t-1})-\nabla F(\boldsymbol{w}^*)\right\|}_{A_3} \\
&\overset{(c)}{\le}\sqrt{1-\mu^2/(4L^2)}\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\|+2\alpha L\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\| \\
&\quad+3\alpha\left(8\Delta_2\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\|+4\Delta_1\right) \\
&=\left(\sqrt{1-\mu^2/(4L^2)}+24\alpha\Delta_2+2\alpha L\right)\left\|\boldsymbol{w}^{t-1}-\boldsymbol{w}^*\right\| \\
&\quad+12\alpha\Delta_1, \qquad (47)
\end{aligned}$$

where (a) is obtained based on Lemma 1; (b) is due to $\nabla F(\boldsymbol{w}^*)=0$; and (c) is obtained by plugging Lemma 2, Lemma 4, and Assumption 1 into A_1, A_2, and A_3, respectively. By recursively applying the inequality for each global iteration, we have:

$$\left\|\boldsymbol{w}^t-\boldsymbol{w}^*\right\|\le(1-\rho)^t\left\|\boldsymbol{w}^0-\boldsymbol{w}^*\right\|+12\alpha\Delta_1/\rho, \qquad (48)$$

where $\rho=1-\left(\sqrt{1-\mu^2/(4L^2)}+24\alpha\Delta_2+2\alpha L\right)$. Thus, we conclude the proof.