electronics

MDPI

Article

Better Safe than Sorry: Constructing Byzantine-Robust Federated Learning with Synthesized Trust

Gangchao Geng, Tianyang Cai and Zheng Yang *

College of Computer and Information Science College of Software, Southwest University, Chongqing 400010, China; ggczmd@email.swu.edu.cn (G.G.); cai12345678@email.swu.edu.cn (T.C.)
* Correspondence: youngzheng@swu.edu.cn

Abstract: Byzantine-robust federated learning empowers the central server to acquire high-end global models amidst a restrictive set of malicious clients. The general idea of existing learning methods requires the central server to statistically analyze all local parameter (gradient or weight) updates, and to delete suspicious ones. The drawback of these approaches is that they lack a root of trust that would allow us to identify which local parameter updates are suspicious, which means that malicious clients can still disrupt the global model. The machine learning community has recently proposed a new method, FLTrust (NDSS'2021), where the server achieves robust aggregation by using a tiny, uncontaminated dataset (denoted as the *root dataset*) to generate the root of trust; however, the global model's accuracy will significantly decline if the root dataset greatly deviates from the client's dataset. To address the above problems, we propose FLEST: a Federated LEarning with Synthesized Trust method. Our method considers that trust and anomaly detection methods can complementarily solve their respective problems; therefore, we designed a new robust aggregation rule with synthesized trust scores (STS). Specifically, we propose the *trust synthesizing mechanism*, which can aggregate trust scores (TS) and confidence scores (CS) into STS through a dynamic trust ratio γ, and we use STS as the weight for aggregating the local parameter updates. Our experimental results demonstrated that FLEST is capable of resisting existing attacks, even when the root dataset distribution significantly differs from the total dataset distribution: for example, the global model trained by FLEST is 41% more accurate than FLTrust for adaptive attacks using the mnist-0.5 dataset with the bias probability set to 0.8.

Keywords: federated learning; Byzantine robust; synthesized trust

Citation: Geng, G.; Cai, T.; Yang, Z. Better Safe than Sorry: Constructing Byzantine-Robust Federated Learning with Synthesized Trust. *Electronics* **2023**, *12*, 2926. https://doi.org/10.3390/electronics12132926

Academic Editors: Charalabos Skianis, Philippe Krief, Enric Pages Montanera and John Soldatos

Received: 6 June 2023
Revised: 29 June 2023
Accepted: 30 June 2023
Published: 3 July 2023

1. Introduction

Federated learning (FL) [1–6] is a cooperative machine learning technique that makes use of several clients (users) and a centralized server. Each client keeps track of its own private dataset, and only transfers its local parameter updates (gradient or weight) during iterative processes. The central server gathers the client-provided local parameter updates, to produce the global parameter update. FL frequently repeats the following actions. Firstly, the central server transmits the global model and global parameter to a subset of clients. Secondly, the clients that receive the global model use local datasets to train the model, and transmit local parameter updates to the central server. Finally, the central server aggregates the local parameter updates according to aggregation rules, which are used to derive the new global parameter: for instance, the traditional FL method FedAvg [2] aggregates clients' local parameter updates by weighted averaging based on the size of the clients' local dataset as the weight.

As FL is distributed, it is impossible to totally rule out the prospect of malicious clients that not only disrespect the training requirements but also desire to alter the global model: for example, malicious clients can use data poisoning attacks [7–11] to corrupt the local dataset, or parameter poisoning attacks [12–16] to directly corrupt local parameters, and

both types of attacks can lead to impaired performance of the global model. If the FL aggregation rule cannot fight against these poisoning attacks, then the attackers will control the global model's performance: for example, FedAvg's global model is easily vulnerable to tampering by a small number of malicious clients.

The primary objective of Byzantine-robust FL [17–21] is to facilitate the central server's acquisition of a superior global model, despite the onslaught of vicious attacks perpetrated by malevolent clients. In essence, "Byzantine-robust" connotes the unyielding fortitude and resistance exhibited in the face of Byzantine attackers that seek to undermine the integrity of the learning process. The general idea of existing state-of-the-art defense methods is that the central server removes suspicious local parameter updates before aggregation by mathematical statistics among the clients' local parameter updates: for example, Krum [21] exploited *L2-Norm* to filter out malicious parameter updates. Unfortunately, recent studies [12] have demonstrated that these robust FL defense mechanisms can be destroyed by poisoning attacks, especially in the dishonest majority setting, due to the lack of a root of trust in the currently prevalent Byzantine-robust learning methodologies. In particular, the central server is unable to identify which local parameter updates are corrupted.

The machine learning community has recently proposed a new federated learning method named FLTrust [22]. According to this method, the central server needs to collect a root dataset, and the model parameter update trained from this dataset is regarded as the root of trust; however, because the central server cannot access the client's datasets, it is unaware of their dispersion. In addition, in practical learning activities, client datasets are typically not independent and identically distributed (non-IID); therefore, it is likely that the root dataset varies from the client datasets, resulting in the severe accuracy reduction of the global model.

Our work. To address the above problems, we propose a Byzantine-robust FL method called FLEST. In FLEST, the server itself bootstraps both trust and anomaly detection. From the above description, it is apparent that introducing anomaly detection methods, or only relying on the trust bootstraps of the server itself, will create its own challenges: we believe that these two methods can solve their problems in a complementary way; therefore, we devised a complementary process named the trust synthesizing mechanism. The purpose of this mechanism is to integrate the trust score (*TS*) and the confidence score (*CS*) of local parameter updating in various proportions. The central server assigns a *TS* and a *CS* to each local parameter update, based on the root of trust and anomaly detection, respectively. In order to realize this mechanism, we designed a synthesized trust score (*STS*), which is composed of a *TS*, a *CS*, and γ. The parameter γ controls the influence of bias in the root of trust, which is called the *dynamic trust ratio*. Our proposed dynamic trust ratio γ can be computed adaptively, based on the results of each round of *TS*s and *CS*s, thus aggregating them into the *STS* according to the appropriate ratio. Therefore, during the federation learning training process, the central server, after receiving the local parameter updates from clients, first obtains the *TS* for each local parameter update based on the root of trust, and then, using the anomaly detection method, obtains their *CS*. Next, the central server uses the *TS* and the *CS* to dynamically compute the *STS*, by synthesizing the trust mechanism. Finally, the server aggregates each local parameter update, using the *STS* as the weight. In our experiments, we used four datasets and three different attacks to verify the effectiveness of FLEST. And we compared FLEST to the current state-of-the-art methods. Moreover, we analyzed the effectiveness of the dynamic trust ratio γ. Our main contributions can be summarized as follows:

- *New federated learning method FLEST*. The proposed FLEST bootstraps both trust and anomaly detection, in order to be resilient against Byzantine attackers;
- *Initial method to tolerate biases at the root of trust*. Experiments show that FLEST works well, even when the root dataset acquired by the central server is distributed differently from the distribution of clients' local datasets: for example, when the bias probability of mnist-0.5 was 0.8, the testing accuracy of FLEST under Trim attacks was 0.94, which was only 0.01 lower than that when the bias probability was 0.1;

- *FLEST can effectively defend against existing attacks*. We evaluated FLEST using existing attacks. Our experimental results showed that FLEST can effectively defend against existing attacks while training a high-performance global model: for example, the testing accuracy of FLEST under LF attack was the same as that of FedAvg under no attacks on mnist-0.1, both of which were 0.97; on fashion-mnist, the testing accuracy of FLEST under Trim attack was best; on cifar-10, the testing accuracy of FLEST under adaptive attack was best.

2. Background and Related Work

The FL system is highly vulnerable to poisoning attacks from malicious clients, which can take various forms, including Label Flipping attack and Trim attack [12,23]. In order to address this critical issue, numerous studies [24–27] have proposed diverse FL defense methods that aim to prevent these types of attacks. It is worth noting that different FL methods essentially utilize different aggregation rules; therefore, we explored the popular aggregation rules and the poisoning attacks in detail, with the aim of developing more efficient and effective defense mechanisms to bolster the security of the FL system.

2.1. Aggregation Rules

In the following, we present and discuss some of the currently popular aggregation rules.

FedAvg. FedAvg [2] adopts FedAverage as an aggregation rule. FedAverage is a weighted average of local parameter (gradient or weight) updates, based on the size of the clients' local datasets. Suppose an FL has n clients with local datasets $\{d_k\}_{k\in[n]}$, where $[n] = \{1, \ldots, n\}$. We use d to denote the overall local dataset. Formally, the global parameter update $\Delta\boldsymbol{\theta} = \sum_{k=1}^{n} \frac{|d_k|}{\sum_{j=1}^{n}|d_j|}\Delta\boldsymbol{\theta}_k$, where $|d_k|$ is the size of the k-th client's local dataset. FedAverage is normally used as the state-of-the-art baseline solution, due to its high performance without attackers; however, FedAverage is vulnerable to poisoning attacks: specifically, it is not Byzantine-robust.

Byzantine-robust aggregation rules. Distance statistical aggregation (DSA) [18,21,24,25,28] and contribution statistical aggregation (CSA) [26,27] are used in the majority of Byzantine-robust aggregation methods. DSA-based schemes discard statistical outliers, by analyzing the distribution of poisoning and benign local parameter updates, before aggregating them into a global parameter update. In CSA-enabled schemes, local parameter updates of clients are labeled with priorities, in terms of their contributions to the performance of the global model. Essentially, these schemes evaluate local parameter updates and delete statistical outliers before aggregating a global parameter update: for example, the Krum method proposed by Blanchard et al. [21] can resist up to 33% of attackers' poisoning attacks, utilizing Euclidean distance to decide which local parameter updates should be dropped. As Krum removes outliers by analyzing the Euclidean distance, it does not remove the parameter updates of the coordinate direction anomaly. To improve Krum, Yin et al. [18] proposed two new FL schemes called Trim-Mean and Median, which are based on the trimmed mean of coordinate directions and the median of coordinate directions, respectively. Each of the techniques mentioned above suffers a common drawback: the absence of a trusted root for the central server to authenticate which local parameter updates from clients are potentially dubious.

Recent studies [12,22] have demonstrated that the above robust FL defense rules can be subverted by poisoning attacks. Cao et al. [22] improved previous FL approaches, by proposing FLTrust, which mainly bootstraps a root of trust at the central server, so that the server can compare the local parameter updates to the root of trust, to filter abnormal local parameter updates; however, the performance of the global model will decline dramatically if the root of trust deviates. The major difference between FLTrust and FLEST is that FLEST does not rely only on the trust mechanism, but dynamically combines the anomaly detection and trust mechanisms through the proposed trust synthesizing mechanism. By

this approach, FLEST avoids the impact on the global model due to the deviation of the root of trust. We have summarized the existing work in Table 1 below.

Table 1. Summary of previous work on federated learning.

Approach	Robustness	Root of Trust
FedAvg	no	no
Krum	yes	no
Trim-Mean	yes	no
Median	yes	no
FLTrust	yes	yes

2.2. Poisoning Attacks

Attackers can perform targeted and untargeted poisoning attacks in the process of training the model. In targeted poisoning attacks [29], attackers want the trained global model to output their specified target classification. In untargeted poisoning attacks [12,30,31], attackers want to disrupt the global model, so as to make it indiscriminately produce incorrect predictions. Furthermore, poisoning attacks can be classified into local data poisoning attacks [29,32–40] (e.g., the Label Flipping attack [23]) and local parameter poisoning attacks [12–16]. If an FL system has tens of thousands of users, the attackers will not be present in the training iterations very often, so that the effectiveness of data poisoning attacks will be weakened [24]. Local data poisoning attacks are equivalent to indirectly manipulating local parameters to corrupt the global parameter; therefore, FL is more vulnerable to local parameter poisoning attacks compared to local data poisoning attacks. Bhagoji et al. [14] demonstrated that a single attacker [18,21] in FL can take control of the global model by directly transmitting carefully designed local parameters. Essentially, both kinds of poisoning attacks corrupt the local parameters, resulting in the impaired performance of the global model.

Recent works (e.g., [12,14,22]) have shown that Byzantine-robust federated learning schemes can also suffer poisoning attacks. Bhagoji et al. [14] showed that the Krum and Median schemes cannot defend against local parameter poisoning attacks. Fang et al. [12] proposed untargeted local parameter poisoning attacks—the Krum attack and the Trim attack. Before sending the local parameter updates to the server, the attackers can modify the local parameter updates in the opposite direction, along the global parameter update $\Delta\boldsymbol{\theta}$, thereby reducing the performance of the global model. Most recently, Cao et al. [22] proposed an adaptive attack that can adaptively adjust the attack process, in terms of FLTrust aggregation rules.

3. Threat Model and Security Goals

In this section, we present a thorough and detailed overview of the threat model we employed, as well as the security goals that we strove to realize.

3.1. Threat Model

Attack model. We considered the following threats and assumptions, as in many existing works (e.g., [12,15,22]):

- The attackers can create and control many clients in the FL system: such clients are therefore malicious, and can train local models using poisoned local datasets, or directly upload poisoned local parameter (gradient or weight) updates. The proportion of malicious clients, however, is less than half of the total;
- The attackers can collude with each other (e.g., by exchanging poisoned local datasets or local parameter updates);
- The attackers have full knowledge of the target FL system, including the aggregation rule of the FL system, the learning rates, and all clients' local datasets and local parameter updates during the whole learning process;

- The attackers can launch various FL attacks (such as the Label Flipping attack [23] and the Krum attack [12]).

Server's knowledge and capability. As in [22], we assumed that the central server had no access to the local datasets of the clients, and did not know the exact number of malicious clients; however, it was fully aware of the global parameters and local parameter updates for each iteration for all clients. Additionally, a trustworthy, small, and clean dataset for the root of trust had to be acquired and tagged by the central server. Due to the small number of data samples in the root dataset, its construction process would not bring too much time overhead to the server; however, without any prior knowledge of the local datasets of the clients, the distribution of the root dataset might deviate from that of clients' datasets with a non-small bias.

3.2. Security Goals

Our goal was to develop a federated learning approach that satisfied the three security goals listed below, while being Byzantine-robust:

- Fidelity. The target method ought not to lose the testing accuracy of the global model in a benign setting without attackers: that is, fidelity required that the target FL method would have no accuracy loss compared to the baseline method FedAvg without attacks;
- Robustness. The FL method could generate a global model providing classification accuracy (in comparison with FedAvg in the no-attack scenario) in the Byzantine client setting, as mentioned in the attack model (i.e., <50% malicious clients). To achieve robustness, we did not assume that the central server could establish, in advance, a training dataset that was unbiased towards all classes of clients' local data;
- Efficiency. Compared to the FedAvg without attacks, the designed method should not introduce many additional computation and communication overheads, especially for clients who might be resource constrained.

4. FLEST: A Federated Learning with Synthesized Trust Method

In this section, we provide a comprehensive overview of FLEST, followed by a detailed presentation of its design and algorithm. Our FLEST considers not only the direction between local parameter updates but also the direction between the root of trust and each local parameter update. The workflow of FLEST is shown in Figure 1 below:

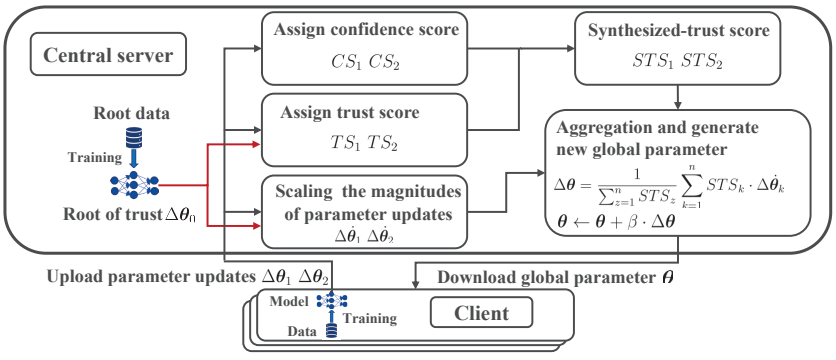


Figure 1. The overall process of FLEST.

4.1. Overview of FLEST

The key idea of FLEST is to combine the trust mechanism with the anomaly detection mechanism, to achieve robust aggregation rules. Regarding the trust mechanism, the central server uses a root of trust trained on an uncontaminated root dataset, to assign a trust score (*TS*) to each client's local parameter updates. As for the anomaly detection mechanism, the server uses a clustering-based anomaly detection method, to assign a confidence score (*CS*) to each local parameter update. Combining these two mechanisms in an effective and

adaptive way is important but challenging because, in practice, the server cannot know whether the root dataset is biased or not, due to the lack of access to the client datasets, and the weights of the *TS* and the *CS* should also be changed dynamically, according to each round of local parameter updates. If the *TS* and the *CS* are combined in an inappropriate way, it is difficult to avoid the global model accuracy degradation caused by the root dataset bias. To address this problem, we propose a trust synthesizing mechanism with dynamic trust ratio γ, to aggregate the *TS* and the *CS* into a synthesized trust score (*STS*) for each local parameter update: in this way, FLEST reduces the performance degradation of the global model due to the bias of the root dataset.

4.2. Detailed Design of FLEST

In this section, we provide the detailed constructions of each part of FLEST, including *TS*, *CS*, and the trust synthesizing mechanism.

Cosine. Cosine similarity is very useful in the Byzantine-robust rule. Given two vectors, $\Delta\boldsymbol{\theta}_a$ and $\Delta\boldsymbol{\theta}_b$, their cosine similarity is $Cos(\Delta\boldsymbol{\theta}_a, \Delta\boldsymbol{\theta}_b) = \frac{\langle \Delta\boldsymbol{\theta}_a, \Delta\boldsymbol{\theta}_b \rangle}{\|\Delta\boldsymbol{\theta}_a\| \cdot \|\Delta\boldsymbol{\theta}_b\|}$.

Trust score. In FLEST, the central server uses the model parameter update, trained from an uncontaminated root dataset, as the root of trust: we considered it to be more trustworthy if the local parameter update had a similar direction to the root of trust, and we computed the directional similarity between them, using the cosine function.

If the local parameter updates deviate too far from the direction of the root of trust, they have a negative cosine similarity, which can lead to a decrease in global model performance: to avoid this effect, we set the negative cosine similarity to zero. Formally, we defined the trust score TS_k of the kth local parameter update $\Delta\boldsymbol{\theta}_k$ as

$$TS_k = ReLU(Cos(\Delta\boldsymbol{\theta}_k, \Delta\boldsymbol{\theta}_0)), \tag{1}$$

where $\Delta\boldsymbol{\theta}_0$ was the root of trust. We utilized the ReLU function, to set the negative directional similarity score to zero.

Confidence score. As the central server does not have access to clients' local datasets, there will be a deviation in the distribution between the root dataset and local datasets, and such a deviation can degrade the accuracy of the final global model. To reduce the side effect of a biased root dataset, we introduced anomaly detection techniques. Specifically, the central server used the K-means clustering algorithm to cluster the local parameter updates into several clusters. We could also have used other clustering algorithms to aggregate the local parameter updates, and we will consider more advanced clustering algorithms in future work. We considered the largest cluster as the cluster of benign local parameter updates. The central server measured the similarity between each local parameter update and the mean of the benign cluster, and assigned a *CS* to each local parameter update. A local parameter update would receive a higher *CS* if it was closer to the mean of the benign cluster. Specifically, we computed the directional similarity between each local parameter update and the mean of the benign local parameter updates, using the cosine function.

For the *TS*, we needed to set the negative directional similarity to zero. Formally, the confidence score of the kth local parameter update $\Delta\boldsymbol{\theta}_k$ was defined as follows:

$$CS_k = ReLU(Cos(\Delta\boldsymbol{\theta}_k, \Delta\boldsymbol{\theta}_L)), \tag{2}$$

where $\Delta\boldsymbol{\theta}_L$ was the mean of the benign local parameter updates.

Trust synthesizing mechanism. Although using either the *TS* or the *CS* has deficiencies in FL, we noted that these two kinds of scores could be complementary to each other in the training procedure; therefore, we designed a complementary mechanism called the *trust synthesizing mechanism*. The idea of this mechanism was to proportionally combine the *TS* and the *CS* of each local parameter update, without completely relying on either of them: when there was a large bias in the root of trust, the trust synthesizing mechanism would

automatically reduce the weight of the *TS*, thus reducing its negative impact. In order to realize this mechanism, we designed a synthesized trust score (*STS*), which was composed of the *TS* and the *CS*, using a dynamic trust ratio $\gamma \in (0,1)$. Formally, we defined the synthesized trust score STS_k of the kth local parameter update $\Delta\boldsymbol{\theta}_k$:

$$STS_k = \gamma \cdot TS_k + (1-\gamma) \cdot CS_k, \tag{3}$$

where γ was used to determine the proportion of the *TS* and the *CS* in the *STS*.

This led to the question: what is the appropriate value of γ? Consequently, we now discuss the strategy of taking γ in different cases, and we illustrate the adaptive calculation method of γ.

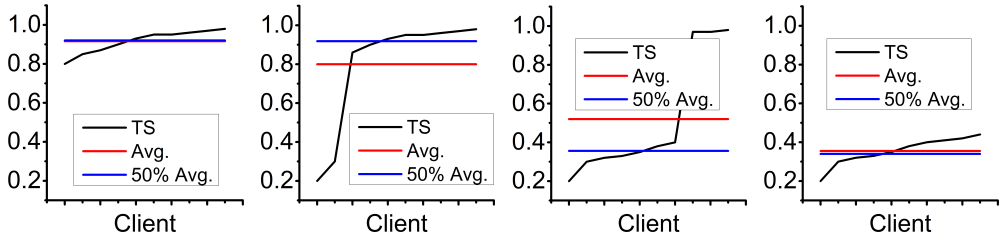
After a round of training, each client's local parameter update was assigned a *TS* and a *CS*. Taking the *TS* as an example, the trust scores of these clients could be sorted, to form a sequence $[TS_1, TS_2, \ldots, TS_n]$. Assuming that less than 50% of the clients were malicious, we considered the following four cases of the distribution of this sequence:

- Case 1: All *TS*s were high;
- Case 2: Most of the *TS*s were high, and the rest were low;
- Case 3: Most of the *TS*s were low, and the rest were high;
- Case 4: All the *TS*s were low.

For ease of understanding, Figure 2 illustrates these four cases. We divided the four cases into two groups, for analysis, according to their characteristics.

- **Analysis of Case 1 and Case 2:** As the more trusted local parameter updates were assigned higher *TS* according to the rules of the *TS* calculation, we deduced that Case 1 occurred when there was no bias in the root of trust, and when there were no malicious clients. By analogy, Case 2 occurred when the trust root bias was small, or when there was a small number of malicious clients. These two cases indicate that the *TS* is not invalid, and thus, should account for a larger proportion of the *STS*, which means that γ should be higher.
- **Analysis of Case 3 and Case 4:** As it was assumed that there were fewer than 50% of malicious clients, Case 3 occurred when there was a large deviation in the root of trust, in which case the TS was close to failure. Case 4, on the other hand, showed that the TS had failed completely. Therefore, in these two cases, the *TS* should have accounted for a smaller proportion of the *STS*, and γ should have been lower.

We first considered using the average of all the *TS*s, to determine their proportion in the *STS*; however, it can be seen from Figure 2 that although the *TS* did not fail in Case 2, it reduced the average value, because of the small number of low *TS*s. Similarly, in Case 3, the *TS* failed, but a small number of local parameters updated by a high *TS* increased the average *TS*. This led to similar average *TS*s for Case 2 and Case 3 in some scenarios, which were difficult to distinguish.



(**a**) Case 1 (**b**) Case 2 (**c**) Case 3 (**d**) Case 4

Figure 2. Various distributions of the *TS*.

To avoid these cases, we used a lightweight calculation method inspired by box plots [41]. Specifically, we used the average of the interquartile range of the *TS* (the middle 50% of the sorted *TS*), noted as the $\overline{TS}$, to determine the proportion of the *TS* in the *STS*.

As shown in Figure 2, the $\overline{TS}$ of Case 2 was close to Case 1, and the $\overline{TS}$ of Case 3 was close to Case 4. In order to consider both the *TS* and the *CS* in the calculation of γ, we performed the same calculation for the *CS*, to obtain the $\overline{CS}$, after which, the dynamic trust ratio γ was defined as

$$\gamma = \overline{TS}/(\overline{TS} + \overline{CS}). \tag{4}$$

From Equation (4), we can see that during each training round, γ could be dynamically adjusted to a proper value, according to the results of the *TS* and the *CS*, thus adaptively adjusting the *STS*. We also verified the effectiveness of the dynamic γ by experiment.

Scaling the magnitudes of local parameter updates. The attackers can modify the magnitudes of local parameter updates, and transmit them to the central server, so that they can manipulate the global parameter update. To weaken the contribution of poisoned local parameter updates, we needed to uniformly scale the magnitudes of the local parameter updates. Without a root of trust, it was difficult to determine how much we should scale, and the mean of the benign local parameter updates was not trusted for the server. As the central server had introduced a root of trust through the root dataset, we used the root of trust to determine how much scaling was needed. As a result, each local parameter update was normalized, so that it had the same magnitude as the root of trust. Formally, the scaling of the magnitudes of the local parameter updates $\Delta\dot{\boldsymbol{\theta}}_k$ of the kth client was defined as follows:

$$\Delta\dot{\boldsymbol{\theta}}_k = \frac{\|\Delta\boldsymbol{\theta}_0\|}{\|\Delta\boldsymbol{\theta}_k\|} \cdot \Delta\boldsymbol{\theta}_k, \tag{5}$$

where $\| * \|$ denoted the modulo operation of the vector.

Aggregation of local parameter updates. We then calculated global parameter update $\Delta\boldsymbol{\theta}$ as the mean of the scaled local parameter updates with their weight *STS*s as follows:

$$\begin{aligned} \Delta\boldsymbol{\theta} &= \sum_{k=1}^{n} \frac{STS_k}{\sum_{z=1}^{n} STS_z} \Delta\dot{\boldsymbol{\theta}}_k \\ &= \sum_{k=1}^{n} \frac{(\gamma \cdot TS_k + (1-\gamma) \cdot CS_k)}{\sum_{z=1}^{n} (\gamma \cdot TS_z + (1-\gamma) \cdot CS_z)} \frac{\|\Delta\boldsymbol{\theta}_0\|}{\|\Delta\boldsymbol{\theta}_k\|} \cdot \Delta\boldsymbol{\theta}_k. \end{aligned} \tag{6}$$

Finally, we changed the global parameter $\boldsymbol{\theta}$ with the rate of global learning β as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \cdot \Delta\boldsymbol{\theta}. \tag{7}$$

4.3. Algorithm of FLEST

The detailed implementation of FLEST is shown in Algorithm 1. This implementation can be divided into three steps, which, during the training process of the global model, assuming I_g iterations, will be performed in each iteration. The three steps are described as follows:

- In Step I, the central server transmits the current parameter $\boldsymbol{\theta}$ to selected partial clients;
- In Step II, based on the global parameter and their local datasets, the clients compute local parameter updates, which are subsequently transmitted to the central server. Using the root dataset and the global parameter, the central server determines the root of trust. After receiving the local parameter updates, the server uses K-means clustering to cluster them, in order to obtain the largest cluster (the benign local parameter updates cluster). The server computes this benign cluster's average value. The function ParameterUpdate in Algorithm 2 uses gradient descent via I_l iterations to determine the local parameter updates and the root of trust;
- In Step III, in order to obtain the global parameter update, and to use it to change the global parameter $\boldsymbol{\theta}$ with β, the central server of FLEST does a weighted mean of the received parameter updates of clients with *STS*s.

Algorithm 1 FLEST

Input: n training datasets of clients $\{d_k\}_{1\leq k\leq n}$; dataset of server d_0; learning rate of global β; iterations of global I_g; amount of clients σ chosen in a single cycle; learning rate of clients ρ; iterations clients I_l; batch size x; proportion of trust score γ; all local parameter updates of the clients ω.

Output: Global parameter $\boldsymbol{\theta}$.

$\boldsymbol{\theta} \leftarrow$ random initialization.
for $i = 1, 2, \ldots I_g$ **do**
  // Step I: Client selection.
  The central server randomly selects σ clients, and sends $\boldsymbol{\theta}$ to them.
  // Step II: Training at the central server and clients.
  // Client.
  for $k = c_1, c_2, \ldots, c_\sigma$ **do**
    $\Delta\boldsymbol{\theta}_k = ParameterUpdate(\boldsymbol{\theta}, d_k, x, \rho, I_l)$.
    Transmit $\Delta\boldsymbol{\theta}_k$ to the central server.
  end for
  // Central server.
  $\Delta\boldsymbol{\theta}_0 = ParameterUpdate(\boldsymbol{\theta}, d_0, x, \rho, I_l)$.
  $\Delta\boldsymbol{\theta}_L \xleftarrow{average}$ K-means(ω). // Average of benign local parameter updates computation.
  // Step III: Robust aggregation and update global parameter.
  for $k = c_1, c_2, \ldots, c_\sigma$ **do**
    $TS_k = ReLU(Cos(\Delta\boldsymbol{\theta}_k, \Delta\boldsymbol{\theta}_0))$.
    $CS_k = ReLU(Cos(\Delta\boldsymbol{\theta}_k, \Delta\boldsymbol{\theta}_L))$.
    $\gamma = \overline{TS}/(\overline{TS} + \overline{CS})$.
    $STS_k = \gamma \cdot TS_k + (1-\gamma) \cdot CS_k$.
    $\Delta\dot{\boldsymbol{\theta}}_k = (\|\Delta\boldsymbol{\theta}_0\|/\|\Delta\boldsymbol{\theta}_k\|) \cdot \Delta\boldsymbol{\theta}_k$.
  end for
  $\Delta\boldsymbol{\theta} = \frac{1}{\sum_{z=1}^{\sigma} STS_{c_z}} \sum_{k=1}^{\sigma} STS_{c_k} \cdot \Delta\dot{\boldsymbol{\theta}}_{c_k}$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \cdot \Delta\boldsymbol{\theta}$.
end for
return $\boldsymbol{\theta}$.

Algorithm 2 ParameterUpdate($\boldsymbol{\theta}, d, x, \rho, I$)

Output: Parameter update.

$\boldsymbol{\theta}^0 \leftarrow \boldsymbol{\theta}$.
for $i = 1, 2, \ldots, I$ **do**
  Select a batch of d_x at random from d.
  $\boldsymbol{\theta}^i \leftarrow \boldsymbol{\theta}^{i-1} - \rho \nabla Loss(\boldsymbol{\theta}; d_x)$.
end for
return $\boldsymbol{\theta}^I - \boldsymbol{\theta}$.

5. Security Analysis

In this section, we formally present our security analysis of FLEST. Our goal was to prove that the global model of FL can still converge with the introduction of FLEST.

Assume that samples from distribution $\mathcal{D}$ are used to create training datasets. The goal of FLEST is to learn an optimal global model $\boldsymbol{\theta}^*$, which can be regarded as an optimization problem: $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} F(\boldsymbol{\theta})$, where Θ is the parameter space, $d = \bigcup_{k=1}^{n} d_k$ is the overall local dataset, $f(\boldsymbol{\theta}; d)$ is the loss function of a parameter vector $\boldsymbol{\theta} \in \Theta$ associated with the training data d, and $F(\boldsymbol{\theta}) \triangleq \mathbb{E}_{d\sim\mathcal{D}}[f(\boldsymbol{\theta}; d)]$ is the population loss function. We followed the three assumptions of [22] for analysis. Our assumptions and analysis results are as follows:

Assumption 1. *$F(\boldsymbol{\theta})$ are μ_F-strongly convex and L-Lipschitz if $F(\boldsymbol{\theta}') \geq F(\boldsymbol{\theta}) + (\boldsymbol{\theta}' - \boldsymbol{\theta})^T \nabla F(\boldsymbol{\theta}) + \frac{\mu_F}{2}\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|^2$ and $\|\nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}')\| \leq L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$, $\forall \boldsymbol{\theta}, \boldsymbol{\theta}'$. Moreover, $f(\boldsymbol{\theta}; d)$ is L_1-Lipschitz with a probability of at least $1 - \frac{\delta}{3}$ for any $\delta \in (0, 1)$. The symbol ∇ represents gradient.*

Assumption 2. *The gradient of $f(\boldsymbol{\theta}^*;d)$ at $\boldsymbol{\theta}^*$ is constrained. The $h(\boldsymbol{\theta};d) = \nabla f(\boldsymbol{\theta};d) - \nabla f(\boldsymbol{\theta}^*;d)$ is also bounded. In detail, $\langle \nabla f(\boldsymbol{\theta}^*;d), v\rangle$ is sub-exponential, with positive constants λ_1 and λ_2 for any unit vector, and $\langle h(\boldsymbol{\theta};d) - \mathbb{E}[h(\boldsymbol{\theta};d)], v\rangle / \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|$ is also sub-exponential, with positive constants ϱ_1 and ϱ_2 for any $\boldsymbol{\theta} \in \Theta$ with $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$. The symbol $\langle \cdot, \cdot \rangle$ represents inner product between vectors.*

Assumption 3. *Let local data d_k and root data d_0 be IID.*

Theorem 1. *Suppose the three Assumptions are true, and FLEST uses $I_l = 1$ and $\rho = 1$. For the percentage of clients who are malicious, less than half of the total clients, $\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|$, is bounded. Formally, for any $\delta \in (0,1)$ with a probability of at least $1-\delta$, after t parallel iterations, we obtain*

$$\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\| \leq \varpi + \varsigma^t\left(\left\|\boldsymbol{\theta}^0 - \boldsymbol{\theta}^*\right\| - \varpi\right),$$

where $\varpi = 12\beta\Delta_1/(1-\varsigma)$, β is the learning rate, $\varsigma = \left(24\beta\Delta_2 + \sqrt{1-\mu_F^2/(2L)^2} + 2\beta L\right)$, $\Delta_1 = \lambda_1\sqrt{\frac{2}{|d_0|}}\sqrt{\log(3/\delta) + D\log 6}$, and $\Delta_2 = \sqrt{D\left(\frac{1}{2}\log\frac{|d_0|}{D} + \log\frac{18L_2}{\varrho_1}\right) + \log\left(\frac{6o\varrho_1^2\sqrt{|d_0|}}{\lambda_1\delta\varrho_2}\right)}$. Furthermore, $\varrho_1\sqrt{\frac{2}{|d_0|}}$, $L_2 = \max\{L, L_1\}$, $|d_0|$ is the number of the root data, D is the dimension of $\boldsymbol{\theta}$, and $\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \leq o\sqrt{D}$ with positive constants o.

Proof. We mainly prove Theorem 1 by following the proof framework of [22]: that is, we show that the difference between $\boldsymbol{\theta}^t$ learned by FLEST under attacks and $\boldsymbol{\theta}^*$ is bounded; in the sequel, we only present the key idea of our proof, to avoid repetition.

Firstly, we first prove that the distance between the global parameter update $\Delta\boldsymbol{\theta}$ and the gradient of population loss $\nabla F(\boldsymbol{\theta})$ is bounded. As our $\sum_{k\in n}\varphi_k = 1$ and $\left\|\Delta\dot{\boldsymbol{\theta}}_k\right\| = \|\Delta\boldsymbol{\theta}_0\|$, and because these conditions are consistent with the counterparts in [22] [Lemma 1], we can apply reduction steps similar to those of [22] [Lemma 1] to obtain the bound $\|\Delta\boldsymbol{\theta} - \nabla F(\boldsymbol{\theta})\| \leq 3\|\Delta\boldsymbol{\theta}_0 - \nabla F(\boldsymbol{\theta})\| + 2\|\nabla F(\boldsymbol{\theta})\|$.

Secondly, as we have $\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} - \beta\Delta\boldsymbol{\theta}^{t-1}$ in any global iteration $t \geq 1$, the meaning of this formula is that the server uses the global parameter update of round $t-1$ to update the global parameter of round $t-1$ with learning rate β, thereby generating the global parameter of round t. We add and subtract $\beta\nabla F\left(\boldsymbol{\theta}^{t-1}\right)$ in $\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|$ at the same time; therefore, we can write $\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|$ as

$$\|\boldsymbol{\theta}^{t-1} - \beta\nabla F\left(\boldsymbol{\theta}^{t-1}\right) - \boldsymbol{\theta}^* + \beta\nabla F(\boldsymbol{\theta}^{t-1}) - \beta\Delta\boldsymbol{\theta}^{t-1}\|. \tag{8}$$

Thirdly, as $\nabla F(\boldsymbol{\theta}^*) = 0$, Equation (8) is bounded by

$$\begin{aligned}&\left\|\boldsymbol{\theta}^{t-1} - \beta\nabla F\left(\boldsymbol{\theta}^{t-1}\right) - \boldsymbol{\theta}^*\right\| + 3\beta\left\|\Delta\boldsymbol{\theta}_0^{t-1} - \nabla F\left(\boldsymbol{\theta}^{t-1}\right)\right\| \\ &+ 2\beta\left\|\nabla F\left(\boldsymbol{\theta}^{t-1}\right) - \nabla F(\boldsymbol{\theta}^*)\right\|,\end{aligned} \tag{9}$$

Equation (9) is a simple split of Equation (8), based on $\nabla F(\boldsymbol{\theta}^*) = 0$ for further proof.

Then, by applying [22, Lemma 2, Lemma 4], we can prove that the distance between $\boldsymbol{\theta}^{t-1} - \beta\nabla F\left(\boldsymbol{\theta}^{t-1}\right)$ and $\boldsymbol{\theta}^*$ and the distance between $\Delta\boldsymbol{\theta}_0$ and $\nabla F(\boldsymbol{\theta})$ are bounded, so that we obtain $\left\|\boldsymbol{\theta}^{t-1} - \boldsymbol{\theta}^* - \beta\nabla F\left(\boldsymbol{\theta}^{t-1}\right)\right\| \leq \sqrt{1-\mu_F^2/(4L^2)}\left\|\boldsymbol{\theta}^{t-1} - \boldsymbol{\theta}^*\right\|$ and $1-\delta \leq \Pr\{\|\Delta\boldsymbol{\theta}_0 - \nabla F(\boldsymbol{\theta})\| \leq 8\Delta_2\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| + 4\Delta_1\}$ for the learning rate $\beta = \mu_F/\left(2L^2\right)$; therefore, Equation (9) has the following upper bound $\left(\sqrt{1-\mu_F^2/(4L^2)} + 24\beta\Delta_2 + 2\beta L\right)\left\|\boldsymbol{\theta}^{t-1} - \boldsymbol{\theta}^*\right\|$ $+12\beta\Delta_1$, which replaces the three sub formulas of Equation (9) based on the results of two lemmas [22][Lemma 2, Lemma 4] and the first Assumption.

By combining Equation (8), Equation (9), and the upper bound of Equation (9), we obtain Equation (10):

$$\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\| \leq \left(\sqrt{1 - \mu_F^2/(4L^2)} + 24\beta\Delta_2 + 2\beta L\right)\left\|\boldsymbol{\theta}^{t-1} - \boldsymbol{\theta}^*\right\| + 12\beta\Delta_1. \tag{10}$$

Finally, by recursively applying Equation (10) to all global iterations (i.e., with concrete iteration index t' from $t' = 0$ to $t' = t - 1$), we can obtain all such bounds for those iterations. We conclude that

$$\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\| \leq 12\beta\Delta_1/(1-\varsigma) + \varsigma^t\left(\left\|\boldsymbol{\theta}^0 - \boldsymbol{\theta}^*\right\| - 12\beta\Delta_1/(1-\varsigma)\right), \tag{11}$$

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

6. Performance Evaluation

We utilized the MXNet platform to implement FLEST. We conducted all the trials on a server with an Intel Core i7-8700K CPU running at 3.70 GHz × 12, a GeForce GTX 1080 Ti GPU, and Ubuntu 18.04 LTS. The three assaults that we used were as follows: the LF attack [12,22]; the Trim attack [12]; and the Adaptive attack [22]. We also implemented aggregation rules, including FedAvg [1,2], Trim-Mean [18], and FLTrust [22], as the comparison points' baselines.

6.1. Experimental Setting

This section details the experimentation performed to validate our proposed FLEST method. To verify the fidelity, robustness, and efficiency of FLEST, we executed experiments using the Label Flipping attack, the Trim attack, and the Adaptive attack on FLEST and on the current top three FL algorithms (FedAvg [1,2], Trim-Mean [18], and FLTrust [22]), to evaluate their defense effectiveness, by comparing the global model prediction accuracies: this section will showcase the experimental environments, settings, and results of all these experiments.

6.1.1. Datasets

We evaluated FLEST on the mnist-0.1 [42], mnist-0.5, cifar-10 [43], and fashion-mnist [44] datasets, mnist being 10-class handwritten digits image datasets, cifar-10 being a 10-class color image classification dataset, and fashion-mnist being a 10-class collection of images of clothing. We generated the clients' local training dataset, following previous work [22]: specifically, we divided the clients randomly into 10 groups, as there were 10 classes in the above datasets. With probability m and with probability $\frac{1-m}{9}$, we allocated training data with data label l to the lth group, and to any other group, respectively. Data from different clients enrolled in the same group were distributed evenly. The parameter m determined the variation in the clients' local datasets distribution. When $m = 0.1$, it meant that the local datasets were independent and identically distributed (IID). When $m > 0.1$, it meant that the local datasets were non-IID [12,22], and a larger m indicated a greater level of non-IID. In the real-world learning tasks of FL, the client's local datasets are usually non-IID; therefore, we set $m > 0.1$ by default.

Dataset mnist. There are 10,000 testing data and 60,000 training data in the mnist [42]. We set $m = 0.1$ in the mnist-0.1 dataset, and set $m = 0.5$ in the mnist-0.5 dataset.

Dataset fashion-mnist. There are 10,000 testing data and 60,000 training data in the 10-class fashion picture dataset known as fashion-mnist [44]. We set $m = 0.5$ in the fashion-mnist dataset.

Dataset cifar-10. 10,000 testing data and 50,000 training data make up the 10-class color image classification dataset known as cifar-10 [43]. We set $m = 0.5$ in the cifar-10 dataset.

6.1.2. Evaluated Attacks

Our attacks included local parameter poisoning and data poisoning, to evaluate FLEST. We employed a representative label flipping attack (LF) in the data poisoning attacks. As in the prior studies [12,22], we employed the identical LF attack configuration. Specifically, we flipped label l of the malicious clients' training data to $9 - l$, where $l \in \{0, 1, \cdots, 9\}$. For the parameter poisoning attacks, we considered the Trim attack [12] and the Adaptive attack [22], for which we used the parameter values from [22].

6.1.3. FL System Settings

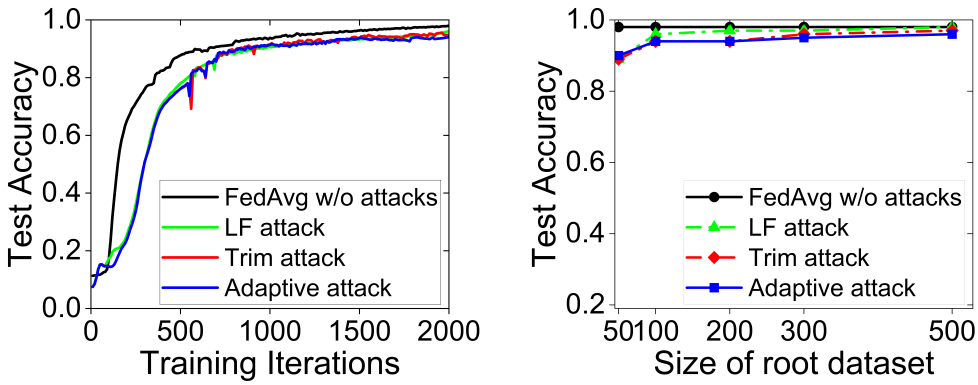
The number of clients was set at $n = 100$ by default, and we turned 20% of the clients into malicious clients by default. The number of K-means clusters was fixed at 10, and the number of iterations to 300.

Selection of global model. For the selection of the global model, we employed a convolutional neural network (CNN) for mnist and fashion-mnist. For the CNN, we used a convolutional layer with $3 \times 3 \times 30$, a convolutional layer with $3 \times 3 \times 50$, a fully connected layer with 100, two max-pooling layers with 2×2, and an output layer with a softmax function. The widely utilized ResNet20 [45] architecture was taken into account as the global model for cifar-10. Our experiments focused on the performance of other aggregation rules and the FLEST in the face of attacks, rather than on the best global model; therefore, we did not need to use an overly complex neural network model.
Parameter settings. We compared FLEST to the works FedAvg [1,2], Trim-Mean [18], and FLTrust [22]. As they obeyed a workflow similar to Algorithm 1, they also used the parameters σ, I_g, I_l, β, ρ, and x. For comparison, we made use of the similar FL settings in [22]. Specifically, we set all clients to be selected in each iteration, i.e., $\sigma = n$, and we set $I_l = 1$. We considered the product $\beta \cdot \rho$ of the local learning rate ρ and the global learning rate β as a combined learning rate. Specifically, $\beta \cdot \rho = 0.0003$ for mnist-0.1 and mnist-0.5, $\beta \cdot \rho = 0.004$ for fashion-mnist, and $\beta \cdot \rho = 0.0005$ for cifar-10. With the exception of cifar-10, we set the batch size to $x = 128$ instead of $x = 64$ for the other three datasets. We determined the total iterations $I_g = 1200$ for mnist-0.1, $I_g = 2000$ for mnist-0.5, $I_g = 2500$ for fashion-mnist, and $I_g = 1600$ for cifar-10.
Generation of root dataset. We built a root dataset with only 100 data, following [22]; however, the root dataset in our experiments was biased towards one class of data. Specifically, we drew a portion of data in the root dataset from a specific kind of local dataset, and sampled the remaining portion evenly and randomly from the other classes: this is known as the bias probability (BP). When each class of the root dataset had the same number of samples, the BP was 0.1, which meant that the distribution of the root dataset was identical to the distribution of the local datasets. Whenever the amount of data in the root dataset that was extracted from one class of local datasets was more than that of other classes, the BP would be larger. Unless otherwise mentioned, we set the BP to 0.1 by default.

6.2. Experimental Results

Evaluation Metric. As this work focused on defending against untargeted attacks, which were designed to reduce the global classification accuracy of the model, we used the accuracy of the model as an evaluation metric: the higher the accuracy of the global model, the more effective the corresponding defense method.

Table 2 displays the testing accuracy of every FL technique when subjected to current assaults, while Table 3 displays the performance of FLTrust and FLEST when subjected to various attacks and various BPs. Figure 3a shows the number of global iterations versus testing accuracies for FedAvg under no attack, and for FLEST under attacks on mnist-0.5. The outcomes demonstrate that FLEST succeeded in terms of the three objectives listed below.



(**a**) mnist-0.5: Iteration (**b**) mnist-0.5: root dataset

Figure 3. The testing accuracy during FLEST's training process under attacks and FedAvg with no attack (**a**) and the impact of the root dataset (**b**).

Goal 1: Fidelity. As shown in Table 2, when there was no attack, the testing accuracies of FLEST were similar to that of FedAvg, which is the baseline FL method. Unfortunately, the testing accuracies of current FL approaches can be lower, because, while aggregating local parameter updates, the current FL approach discards part of them, whereas FLEST considers all local parameter updates: for example, in Table 2, under the fashion-mnist, the testing accuracies of FedAvg, FLTrust, and FLEST are 0.90, 0.89, and 0.90, respectively, while the testing accuracy of Trim-mean is 0.86.

Goal 2: Robustness. The testing accuracies of FLEST under three different attacks were, at most, 0.04 lower than those of FedAvg under no attacks on the mnist and fashion-mnist datasets. In particular, as shown in Table 2, the testing accuracy of FLEST under LF attack was the same as that of FedAvg under no attacks on mnist-0.1, both of which were 0.97. The testing accuracies of the presently used FL techniques under three different attacks on the four datasets were much lower: for example, the Trim attack reduced the testing accuracy of Trim-mean by 0.11 on mnist-0.5, and the Adaptive attack reduced the testing accuracy of Trim-mean by 0.35 on fashion-mnist.

Moreover, Table 3 shows that FLEST and FLTrust were both robust and accurate when the BP was small; however, when the BP was large, FLEST maintained high testing accuracies, while the testing accuracies of FLTrust decreased substantially. In particular, under no attacks, regardless of the BP, the testing accuracies of FLEST were similar to those of FedAvg on the four datasets, which means that, compared to FLTrust, FLEST could still protect the global model effectively when the BP was large: for example, when the BP of mnist-0.1 exceeded 0.4, the testing accuracies of FLTrust under different attacks started to decrease significantly, while FLEST was not significantly lower until the BP was greater than 0.6; when the BP of fashion-mnist was 0.8, the FLTrust method basically failed, but the testing accuracies of FLEST under different attacks were still much higher. Significantly, even when the BP was 1 on all four datasets, FLEST still achieved the same testing accuracies as FedAvg under no attacks, while FLTrust not only could not defend against attacks but also affected the global model without attacks: this was because when the BP was large, the *TS* of each local parameter was almost zero, but the *CS* of each local parameter was not affected. According to the experimental findings, FLEST continues to function well even when the root dataset differs significantly from the local datasets.

Goal 3: Efficiency. Like FedAvg, FLEST does not cause additional computation to the clients in each iteration. In addition, Figure 3a shows the testing accuracy of FLEST, during the training process, under different attacks, and of the baseline (the testing accuracy of FedAvg under no attack). The experimental results show that FLEST converges at a similar rate as FedAvg, which indicates that FLEST does not introduce much communication overhead between the central server and the clients. Compared to FedAvg, FLEST intro-

duces some computation overhead to the central server, including computing the root of trust, computing *STS*, and scaling all local parameter updates; however, a powerful server requires only a negligible time overhead.

Evaluation of dynamic trust ratio γ. To verify the validity of the dynamic trust ratio γ, we evaluated the performance of the global model, using different fixed gamma, thus verifying whether the dynamic γ adaptively adjusted to the appropriate range. Table 4 records the accuracy of the model after fixing different values of γ for different deviation probabilities and different attacks, as well as the average value of the dynamic γ. As shown in Table 4a, when the deviation probability was small, the average value of gamma was around 60–80%; however, when the deviation probability was very high, as shown in Table 4c, the average value of γ dropped to about 20–50%, and the accuracy of FLEST, by applying the dynamic γ, was also at a high level. Thus, these results demonstrate that the trust synthesizing mechanism can mitigate the degradation of model accuracy due to root dataset bias.

Impact of the root dataset. The effect of the root dataset sizes on FLEST, under various assaults, is depicted in Figure 3b. We note that FLEST could fight against various assaults with a root dataset of only 100 training data. In particular, the testing accuracy of FLEST under assaults was comparable to that of FedAvg under no attacks when the root dataset contained 100 training data. When the number of training data in the root dataset exceeded 100, FLEST's testing accuracy increased slightly.

Impact of the number of clients. The effect of the total clients on each FL method under attacks is shown in Figure 4a–c. The findings of the trial indicate that the impact of the total clients on FLTrust and FLEST was small, while Trim-mean was greatly affected: for example, when there were 300 clients, the global model taught by Trim had a testing accuracy of 0.46 under the Trim attack, whereas FLTrust and FLEST trained global models to have a testing accuracy of 0.95.

Impact of the percentage of malicious clients. Figure 4d–f show the effect of the percentage of malicious clients on each FL method under attacks. We observed that the effect of the percentage of malicious clients on FLTrust and FLEST was small, while Trim-mean was greatly affected: for example, when there was 20% malicious clients, the global model learned by Trim had a testing accuracy of 0.89 under the Trim attack, whereas the testing accuracies of the global model learned by both FLTrust and FLEST were 0.95 and 0.94, respectively.

Table 2. Accuracy comparison of FL methods under different attacks.

Attack	Mnist-0.1				Mnist-0.5				Fashion-Mnist				Cifar-10			
	FedAvg	Trim-Mean	FLTrust	FLEST	FedAvg	Trim-Mean	FLTrust	FLEST	FedAvg	Trim-Mean	FLTrust	FLEST	FedAvg	Trim-Mean	FLTrust	FLEST
No	0.97	0.98	0.97	0.98	0.98	0.98	0.96	0.97	0.89	0.86	0.89	0.90	0.64	0.54	0.62	0.63
LF	0.96	0.97	0.97	0.97	0.97	0.97	0.95	0.96	0.84	0.80	0.89	0.90	0.54	0.46	0.57	0.57
Trim	0.91	0.93	0.95	0.95	0.91	0.89	0.95	0.94	0.66	0.44	0.88	0.88	0.19	0.28	0.48	0.48
Adaptive	0.94	0.95	0.95	0.95	0.94	0.92	0.93	0.94	0.78	0.65	0.88	0.88	0.10	0.31	0.49	0.49

Table 3. Accuracy Comparison between FLTrust and FLEST under different attacks, and given root dataset with various bias probabilities.

(a) Mnist-0.1 and Mnist-0.5

Dataset	Mnist-0.1												Mnist-0.5											
BP	0.1		0.2		0.4		0.6		0.8		1		0.1		0.2		0.4		0.6		0.8		1	
Method	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours
No	0.97	0.97	0.97	0.97	0.97	0.97	0.96	**0.97**	0.96	**0.97**	0.66	**0.98**	0.96	0.97	0.96	**0.97**	0.95	**0.96**	0.93	**0.97**	0.93	**0.97**	0.20	**0.97**
LF	0.97	0.97	0.97	0.97	0.96	0.96	0.95	**0.96**	0.22	**0.71**	0.16	**0.64**	0.95	0.96	0.95	0.95	0.92	**0.93**	0.90	**0.92**	**0.78**	0.72	0.11	**0.56**
Trim	0.95	0.95	0.95	0.95	0.92	**0.94**	0.90	**0.93**	0.54	**0.93**	0.11	**0.93**	**0.95**	0.94	0.94	0.94	0.92	**0.93**	0.91	**0.93**	0.86	**0.94**	0.11	**0.94**
Adaptive	0.95	0.95	0.94	**0.95**	0.93	0.93	0.87	**0.93**	0.10	**0.93**	0.10	**0.92**	0.93	**0.94**	0.93	**0.94**	0.92	0.92	0.87	**0.92**	0.50	**0.91**	0.10	**0.86**

(b) Fashion-Mnist and Cifar-10

Dataset	Fashion-Mnist												Cifar-10											
BP	0.1		0.2		0.4		0.6		0.8		1		0.1		0.2		0.4		0.6		0.8		1	
Method	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours	FLTrust	Ours
No	0.89	**0.90**	0.89	0.89	0.88	**0.89**	0.86	**0.88**	0.85	**0.89**	0.10	**0.89**	0.62	0.62	0.62	**0.63**	0.61	0.61	0.59	**0.62**	0.10	**0.62**	0.10	**0.62**
LF	0.89	**0.90**	0.89	0.89	0.88	**0.89**	0.85	**0.88**	0.86	**0.87**	0.10	**0.69**	0.57	0.57	0.56	0.56	0.45	**0.47**	0.41	**0.47**	0.10	**0.44**	0.10	**0.40**
Trim	0.88	0.88	0.87	0.87	0.87	0.87	0.84	0.84	0.10	**0.79**	0.10	**0.78**	0.48	0.48	0.47	**0.48**	0.40	**0.43**	0.33	**0.35**	0.10	**0.32**	0.10	**0.30**
Adaptive	0.88	0.88	0.86	0.86	0.84	**0.86**	**0.82**	0.81	0.10	**0.79**	0.10	**0.69**	0.49	0.49	0.45	**0.46**	0.42	**0.44**	0.32	**0.34**	0.10	**0.33**	0.10	**0.29**

Table 4. Impact of the trust ratio γ on mnist-0.5.

(a) BP = 0.1 and 0.2

Fixed γ	BP = 0.1								BP = 0.2							
	0	10%	20%	40%	60%	80%	100%	γ	0	10%	20%	40%	60%	80%	100%	γ
No	0.97	0.97	0.97	0.97	0.97	**0.97**	0.96	85.4%	0.97	0.97	0.97	0.97	0.97	**0.97**	0.96	83.7%
LF	0.30	0.70	0.94	0.95	0.95	**0.95**	0.95	78.6%	0.30	0.70	0.94	0.95	0.95	**0.95**	0.95	73.2%
Trim	0.94	0.94	0.95	0.94	0.94	**0.95**	0.95	76.3%	0.95	0.94	0.95	0.94	**0.94**	0.94	0.94	64.9%
Adaptive	0.93	0.94	0.94	0.94	**0.94**	0.94	0.93	66.8%	0.94	0.94	0.94	0.94	**0.94**	0.93	0.93	63.4%

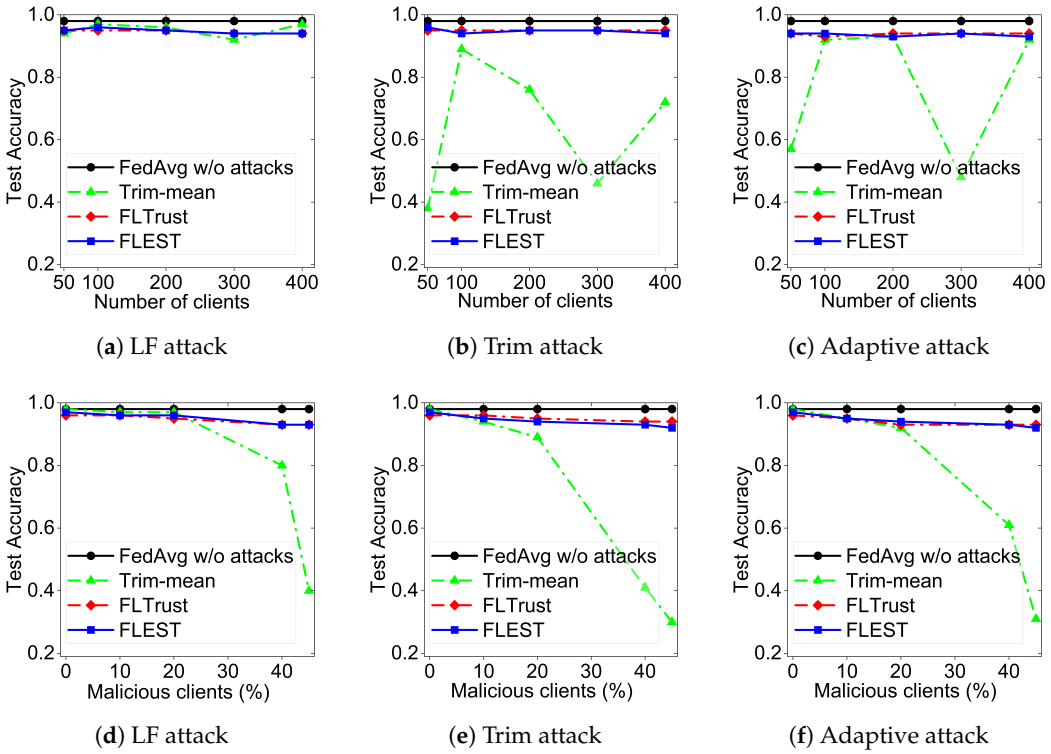
Table 4. *Cont.*

(b) BP = 0.4 and 0.6

	BP = 0.4								BP = 0.6							
Fixed γ	**0**	**10%**	**20%**	**40%**	**60%**	**80%**	**100%**	γ	**0**	**10%**	**20%**	**40%**	**60%**	**80%**	**100%**	γ
No	0.97	0.97	0.97	0.96	0.96	**0.97**	0.95	71.6%	0.97	0.97	0.97	**0.97**	0.96	0.96	0.93	43.3%
LF	0.30	0.58	0.84	0.92	**0.92**	0.92	0.92	65.6%	0.20	0.36	0.75	0.85	0.91	**0.91**	0.90	74.7%
Trim	0.94	0.95	0.94	0.94	**0.93**	0.92	0.92	66.2%	0.94	0.94	0.94	0.94	**0.93**	0.91	0.91	61.8%
Adaptive	0.93	0.94	0.94	0.93	**0.93**	0.92	0.92	72.5%	0.94	0.94	0.93	0.93	**0.92**	0.89	0.87	65.4%

(c) BP = 0.8 and 1

	BP = 0.8								BP = 1							
Fixed γ	**0**	**10%**	**20%**	**40%**	**60%**	**80%**	**100%**	γ	**0**	**10%**	**20%**	**40%**	**60%**	**80%**	**100%**	γ
No	0.98	0.97	**0.97**	0.97	0.97	0.96	0.93	34.9%	0.98	0.98	**0.97**	0.97	0.97	0.97	0.20	26.3%
LF	0.16	0.37	0.48	0.53	0.68	**0.80**	0.78	74.6%	0.26	0.31	0.38	**0.39**	0.39	0.40	0.11	53.8%
Trim	0.95	0.95	0.94	**0.94**	0.93	0.92	0.86	42.8%	0.94	0.94	0.94	**0.94**	0.91	0.40	0.11	47.1%
Adaptive	0.94	0.94	**0.93**	0.90	0.83	0.87	0.50	31.3%	0.95	0.92	**0.89**	0.56	0.23	0.11	0.10	32.4%



(**a**) LF attack (**b**) Trim attack (**c**) Adaptive attack

(**d**) LF attack (**e**) Trim attack (**f**) Adaptive attack

Figure 4. On mnist-0.5, impact of the number of clients (**a**–**c**), and fraction of malicious clients (**d**–**f**).

7. Discussion and Limitations

FLEST vs. existing works. Prior research has exclusively focused on examining the direction between either local parameter updates or the root of trust and each local parameter update. By contrast, our novel FLEST extends this approach, by taking into account not just the direction between the local parameter updates and the root of trust, but also the direction between each individual local parameter update.

Poisoned root dataset. Our FLEST does not require a root dataset with all data samples pollution-free, as FLTrust does; however, we acknowledge that if FLEST uses a poisoned root dataset, it may not be able to defend against existing attacks. In order to avoid the possibility of root dataset pollution, service providers may opt to have their personnel manually curate the root dataset.

Targeted poisoning attacks and root dataset with low BP. We acknowledge that the various poisoning attacks used in the experimental evaluation did not specifically target our FLEST approach. We recognize that stronger poisoning attacks may exist against FLEST, which would be an interesting avenue for future exploration. Additionally, considering the collection of root dataset with low BP is a promising direction for future work: for instance, if the dynamic trust ratio γ of an FL system using FLEST falls below a certain threshold (e.g., $\gamma < 0.4$) during an iteration, it indicates a significant distributional bias between the root dataset collected by the server and the overall local training data distribution on the clients; in such cases, the server can re-collect the root dataset before the next iteration, thus avoiding the use of a biased root dataset from the initial iteration onward.

8. Conclusions and Future Work

We have shown a new robust FL approach, FLEST, which can achieve Byzantine robustness with a biased root dataset. In FLEST, we introduce a trust synthesizing mechanism that can yield synthesized trust scores for weighting the local parameter updates of clients. Meanwhile, the synthesizing procedure combines the trust score and the confidence score (calculated based on an anomaly detection scheme) of each local parameter update with a trust ratio (for scaling both kinds of scores). Consequently, FLEST, with our trust synthesiz-

ing mechanism, can reduce the negative impact of the biased root dataset compared to the previous scheme, FLTrust, using the root dataset only. We have shown the effectiveness of FLEST via both theoretical and experimental analysis: that is, FLEST can provide much better testing accuracy compared to FLTrust with a highly biased root dataset. In this work, we have mainly shown that our new trust synthesizing mechanism is able to improve the robustness of FL significantly; there may be many feasible ways to improve it in future work. We will consider developing an optimal approach (possibly with other anomaly detection methods) to generating more accurate confidence scores. We will also consider designing a local parameter poisoning attack against FLEST, and optimizing FLEST, based on it. Furthermore, we will consider re-collecting low-biased root datasets based on the dynamic trust ratio γ. Specifically, if the dynamic trust ratio of an FL system using FLEST falls below a certain threshold (e.g., $\gamma < 0.4$) during a single iteration, this indicates a significant distributional bias between the root dataset collected by the server and the overall local training data distribution on the clients: in such cases, the server can choose to re-collect the root dataset before the next iteration, thereby avoiding the continuous use of biased root datasets from the initial iteration throughout the training process.

Author Contributions: Conceptualization, G.G. and T.C.; methodology, G.G.; software, G.G.; investigation, G.G.; writing—original draft preparation, G.G. and T.C.; writing—review and editing, G.G., T.C. and Z.Y.; supervision, Z.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Natural Science Foundation of Chongqing under Grant No. CSTB2022NSCQ-MSX0437, and the Fundamental Research Funds for the Central Universities under Grant No. SWU-KR22003.

Data Availability Statement: The mnist dataset can be downloaded at "http://yann.lecun.com/exdb/mnist/" (accessed on 2 June 2023), the fashion-mnist dataset can be downloaded at "https://github.com/zalandoresearch/fashion-mnist" (accessed on 2 June 2023), and the cifar-10 dataset can be downloaded at "http://www.cs.toronto.edu/~kriz/cifar.html" (accessed on 2 June 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Konečný, J.; McMahan, H.B.; Yu, F.X.; Richtarik, P.; Suresh, A.T.; Bacon, D. Federated Learning: Strategies for Improving Communication Efficiency. In Proceedings of the NIPS Workshop on Private Multi-Party Machine Learning, Barcelona, Spain, 5–10 December 2016.
2. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. *Proc. Artif. Intell. Stat. PMLR* **2017**, *54*, 1273–1282.
3. Zhang, Y.; Bai, G.; Li, X.; Nepal, S.; Grobler, M.; Chen, C.; Ko, R.K. Preserving Privacy for Distributed Genome-Wide Analysis Against Identity Tracing Attacks. *IEEE Trans. Dependable Secur. Comput.* **2022**, 1–17. [CrossRef]
4. Fang, M.; Liu, J.; Gong, N.Z.; Bentley, E.S. AFLGuard: Byzantine-robust Asynchronous Federated Learning. In Proceedings of the 38th Annual Computer Security Applications Conference, Austin, TX, USA, 5–9 December 2022; pp. 632–646.
5. Nguyen, J.; Malik, K.; Zhan, H.; Yousefpour, A.; Rabbat, M.; Malek, M.; Huba, D. Federated learning with buffered asynchronous aggregation. In Proceedings of the International Conference on Artificial Intelligence and Statistics, Valencia, Spain, 28–30 March 2016; pp. 3581–3607.
6. Huba, D.; Nguyen, J.; Malik, K.; Zhu, R.; Rabbat, M.; Yousefpour, A.; Wu, C.J.; Zhan, H.; Ustinov, P.; Srinivas, H.; et al. Papaya: Practical, private, and scalable federated learning. *Proc. Mach. Learn. Syst.* **2022**, *4*, 814–832.
7. Biggio, B.; Nelson, B.; Laskov, P. Poisoning Attacks against Support Vector Machines. In Proceedings of the 29th International Coference on International Conference on Machine Learning, Edinburgh, Scotland, 26 June–1 July 2012.
8. Jagielski, M.; Oprea, A.; Biggio, B.; Liu, C.; Nita-Rotaru, C.; Li, B. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–24 May 2018; pp. 19–35.
9. Li, B.; Wang, Y.; Singh, A.; Vorobeychik, Y. Data poisoning attacks on factorization-based collaborative filtering. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
10. Xiao, H.; Biggio, B.; Brown, G.; Fumera, G.; Eckert, C.; Roli, F. Is feature selection secure against training data poisoning? In Proceedings of the International Conference on Machine Learning. PMLR, Lille, France, 7–9 July 2015; pp. 1689–1698.
11. Mei, S.; Zhu, X. Using machine teaching to identify optimal training-set attacks on machine learners. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015.

12. Fang, M.; Cao, X.; Jia, J.; Gong, N. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Boston, MA, USA, 12–14 August 2020; pp. 1605–1622.
13. Xie, C.; Huang, K.; Chen, P.Y.; Li, B. Dba: Distributed backdoor attacks against federated learning. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
14. Bhagoji, A.N.; Chakraborty, S.; Mittal, P.; Calo, S. Analyzing federated learning through an adversarial lens. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 634–643.
15. Bagdasaryan, E.; Veit, A.; Hua, Y.; Estrin, D.; Shmatikov, V. How to backdoor federated learning. In Proceedings of the International Conference on Artificial Intelligence and Statistics. PMLR, Online, 26–28 August 2020; pp. 2938–2948.
16. Lyu, L.; Yu, H.; Ma, X.; Sun, L.; Zhao, J.; Yang, Q.; Yu, P.S. Privacy and robustness in federated learning: Attacks and defenses. *arXiv* **2020**, arXiv:2012.06337.
17. Chen, Y.; Su, L.; Xu, J. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proc. ACM Meas. Anal. Comput. Syst.* **2017**, *1*, 1–25. [CrossRef]
18. Yin, D.; Chen, Y.; Kannan, R.; Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 5650–5659.
19. Sohn, J.y.; Han, D.J.; Choi, B.; Moon, J. Election coding for distributed learning: Protecting signsgd against byzantine attacks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 14615–14625.
20. Yu, L.; Wu, L. Towards byzantine-resilient federated learning via group-wise robust aggregation. In *Federated Learning*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 81–92.
21. Blanchard, P.; El Mhamdi, E.M.; Guerraoui, R.; Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. In Proceedings of the Advances in Neural Information Processing Systems , Long Beach, CA, USA, 4–9 December 2017.
22. Cao, X.; Fang, M.; Liu, J.; Gong, N.Z. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS), Virtually, 21–25 February 2021.
23. Tolpegin, V.; Truex, S.; Gursoy, M.E.; Liu, L. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 480–501.
24. Li, L.; Xu, W.; Chen, T.; Giannakis, G.B.; Ling, Q. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. *Proc. AAAI Conf. Artif. Intell.* **2019**, *33*, 1544–1551. [CrossRef]
25. Zhao, B.; Sun, P.; Wang, T.; Jiang, K. FedInv: Byzantine-robust Federated Learning by Inversing Local Model Updates. *Proc. Aaai Conf. Artif. Intell.* **2022**, *36*, 9171–9179. [CrossRef]
26. Xie, C.; Koyejo, S.; Gupta, I. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6893–6901.
27. Xie, C.; Koyejo, S.; Gupta, I. Zeno++: Robust fully asynchronous sgd. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 10495–10503.
28. Guerraoui, R.; Rouault, S. The hidden vulnerability of distributed learning in byzantium. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 3521–3530.
29. Chen, X.; Liu, C.; Li, B.; Lu, K.; Song, D. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv* **2017**, arXiv:1712.05526.
30. Wang, N.; Xiao, Y.; Chen, Y.; Hu, Y.; Lou, W.; Hou, Y.T. FLARE: Defending Federated Learning against Model Poisoning Attacks via Latent Space Representations. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May–3 June 2022; pp. 946–958.
31. Shen, L.; Zhang, Y.; Wang, J.; Bai, G. Better Together: Attaining the Triad of Byzantine-robust Federated Learning via Local Update Amplification. In Proceedings of the 38th Annual Computer Security Applications Conference, Austin, TX, USA, 5–9 December 2022; pp. 201–213.
32. Fang, M.; Yang, G.; Gong, N.Z.; Liu, J. Poisoning attacks to graph-based recommender systems. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 381–392.
33. Gu, T.; Dolan-Gavitt, B.; Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv* **2017**, arXiv:1708.06733.
34. Muñoz-González, L.; Biggio, B.; Demontis, A.; Paudice, A.; Wongrassamee, V.; Lupu, E.C.; Roli, F. Towards poisoning of deep learning algorithms with back-gradient optimization. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Dallas, TX, USA, 3 November 2017; pp. 27–38.
35. Nelson, B.; Barreno, M.; Chi, F.J.; Joseph, A.D.; Rubinstein, B.I.; Saini, U.; Sutton, C.; Tygar, J.D.; Xia, K. Exploiting machine learning to subvert your spam filter. *LEET* **2008**, *8*, 16–17.
36. Rubinstein, B.I.; Nelson, B.; Huang, L.; Joseph, A.D.; Lau, S.H.; Rao, S.; Taft, N.; Tygar, J.D. Antidote: Understanding and defending against poisoning of anomaly detectors. In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, Chicago, IL, USA, 4–6 November 2009; pp. 1–14.
37. Shafahi, A.; Huang, W.R.; Najibi, M.; Suciu, O.; Studer, C.; Dumitras, T.; Goldstein, T. Poison frogs! targeted clean-label poisoning attacks on neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; Volume 31.

38. Suciu, O.; Marginean, R.; Kaya, Y.; Daume III, H.; Dumitras, T. When does machine learning fail? generalized transferability for evasion and poisoning attacks. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1299–1316.
39. Wang, B.; Gong, N.Z. Attacking graph-based classification via manipulating the graph structure. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, Baltimore, MD, USA, 15–17 August 2018; pp. 2023–2040.
40. Yang, G.; Gong, N.Z.; Cai, Y. Fake Co-visitation Injection Attacks to Recommender Systems. In Proceedings of the NDSS, San Diego, CA, USA, 26 February–1 March 2017.
41. Box Plot—Wikipedia. Available online: https://en.wikipedia.org/wiki/Box_plot (accessed on 4 October 2022).
42. Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
43. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. 2009. Available online: https://www.semanticscholar.org/paper/Learning-Multiple-Layers-of-Features-from-Tiny-Krizhevsky/5d90f06bb70a0a3dced62413346235c02b1aa086 (accessed on 2 July 2023).
44. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
45. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.