Model Poisoning Attacks to Federated Learning via Multi-Round Consistency

Supplementary Material

A. Motivation

We first introduce our key observation about why existing model poisoning attacks achieve suboptimal attack effectiveness, which motivates the design of our PoisonedFL in the next section. We show experimental results on the MNIST dataset with the default parameter settings described in Section 5.1. We assume the server uses Trimmed Mean [39] as the aggregation rule. Moreover, we assume the ratio of fake clients to genuine clients is 20%. We consider MPAF [13], an attack not requiring genuine clients' information, and Fang [17], a representative attack that requires genuine clients' information. To give advantages to Fang, we assume the attacker knows the local models of *all* participating genuine clients in each training round when crafting malicious model updates on fake clients.

In each training round, these attacks craft a malicious model update for each fake client selected by the server to participate in training. A malicious model update $\boldsymbol{g}_i^t$ of fake client i in training round t can be decomposed into a dimension-wise product of a *sign vector* $\boldsymbol{s}_i^t$ and a *magnitude vector* $|\boldsymbol{g}_i^t|$, where $|\cdot|$ means dimension-wise absolute value of a vector. Each dimension of $\boldsymbol{s}_i^t$ is either +1 or -1, while all dimensions of $|\boldsymbol{g}_i^t|$ are non-negative. A dimension with a +1 (or -1) sign aims to increase (or decrease) the corresponding dimension/parameter of the global model. For a fake client, we say a dimension is flipped in training round t if its sign in the malicious model update is flipped, compared to that in the previous training round. A flipped dimension in a training round t may cancel the attack effect of the previous training round $t-1$. This is because the malicious model updates aim to increase the dimension of the global model in one training round but decrease it in the other. For each training round t, we define *flipping rate* as the fraction of flipped dimensions of a malicious model update averaged over the fake clients.

Fig. 1a shows the testing error rate of the global model as a function of training round when no attack, MPAF, or Fang is used; while Fig. 1b shows the flipping rate as a function of training round when MPAF or Fang is used to craft malicious model updates on the fake clients. We observe that the flipping rate of MPAF is always larger than 35% and that of Fang is larger than 40% in the last around 1,500 training rounds, leading to self-cancellation of attack effect in many training rounds. For instance, in Fang attack, the flipping rate is around 10% in nearly the first 1,000 training rounds, i.e., the malicious model updates maintain some degree of "consistency". As a result, the global model under Fang attack has a large testing error rate in those training rounds as shown in Fig. 1a. However, the flipping rate rapidly increases as the training proceeds, eventually leading to self-cancellation of attack effect and an accurate global model.

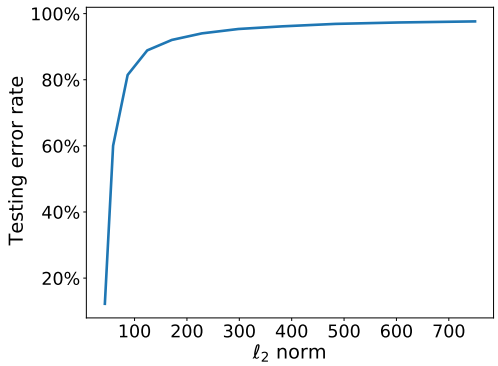


Figure 6. The testing error rate of a global model as a function of the magnitude (measured by ℓ_2 norm) of the noise added to it along a random update direction. We assume the initial global model is accurate and trained under no attack. The dataset is Purchase and FL defense is Median.

Table 2. Model architecture for MNIST and FashionMNIST.

Layer Type	Size
Convolution + ReLU	3x3x30
Max Pooling	2x2
Convolution + ReLU	3x3x50
Max Pooling	2x2
Fully Connected + ReLU	100
Softmax	10

Table 3. Model architecture for CIFAR-10.

Layer Type	Size
Convolution + ReLU	3x3x32
Max Pooling	2x2
Convolution + ReLU	3x3x64
Max Pooling	2x2
Fully Connected + ReLU	512
Softmax	10

B. Additional Details of Experimental Setup

B.1. Datasets

We use the following five datasets from different domains.

MNIST [24]: MNIST is a 10-class handwritten digits clas-

Table 4. Model architecture for FEMNIST.

Layer Type	Size
Convolution + ReLU	3x3x30
Max Pooling	2x2
Convolution + ReLU	3x3x50
Max Pooling	2x2
Fully Connected + ReLU	200
Softmax	62

sification dataset. It comprises 60,000 training examples and 10,000 testing examples. In FL, the training data is typically not independently and identically distributed (non-IID) among clients. Following [13, 17], we distribute the training examples among clients based on a non-IID degree of $q = 0.5$ by default. We train a convolutional neural network (CNN) for MNIST dataset. The architecture of CNN is shown in Table 2 in Appendix.

FashionMNIST [35]: FashionMNIST is a benchmark dataset of Zalando's article images, which contains 60,000 training examples and 10,000 testing examples. For FashionMNIST, the degree of non-IID is also set to $q = 0.5$ by default. We use the same CNN architecture for FashionMNIST dataset as employed in MNIST dataset.

Purchase [7]: Purchase is a 100-class customer purchase style prediction dataset. Each input consists of 600 binary features. Following [13], we split the total 197,324 examples into 180,000 training examples and 17,324 testing examples. Following [13], we evenly distribute the training examples to clients and train a fully connected neural network.

CIFAR-10 [23]: CIFAR-10 is 10-class color image classification dataset. It comprises 50,000 training examples and 10,000 testing examples. To consider a different degree of non-IID, we set $q = 0.2$ for CIFAR-10 dataset. We train a CNN for CIFAR-10, whose architecture is shown in Table 3.

FEMNIST [12]: FEMNIST is a 62-class classification dataset. The dataset is already distributed among 3,550 clients with a total of 805,263 examples. We randomly sample 1,200 clients. We train a CNN for FEMNIST, whose architecture is shown in Table 4.

B.2. Defenses

We evaluate eight state-of-the-art defenses, including six Byzantine-robust aggregation rules (Multi-Krum [11], Median [39], Trimmed Mean [39], Norm Bound [34], FLTrust [14], and FLAME [28]), one provably robust defense (FLCert [16]) and one malicious clients detection method (FLDetector [40]). We also consider the non-robust FedAvg [26] as a baseline.

Multi-Krum [11]: Multi-Krum uses an iterative method to select a subset of clients' model updates. In each step, it selects the model update that has the smallest sum of Euclidean distance to its $n - 2$ neighbors, where n is the number of genuine clients. This selection process continues until n clients are chosen. Finally, the server computes the average of the n selected model updates as the aggregated model update.

Median [39]: In Median aggregation rule, the server computes the coordinate-wise median of the clients' model updates as the aggregated model update.

Trimmed Mean (TrMean) [39]: For each dimension, the server first removes the largest m values and smallest m values, and then computes the average of the remaining $n - m$ values, where m is the number of fake clients.

Norm Bound [34]: In Norm Bound aggregation rule, the server first clips each client's model update to have a predetermined norm and then computes the average of the clipped model updates as the aggregated model update. We follow prior work [33] to set the predetermined norm as the average norm of the genuine model updates in each training round.

FLTrust [14]: FLTrust assumes that the server has a small, clean dataset to bootstrap trust. We assume the server's dataset includes training examples selected uniformly at random. The size of the server's dataset is 100 for MNIST, FashionMNIST, and CIFAR-10, and 200 for Purchase and FEMNIST.

FLAME [28]: In each training round, FLAME considers the cosine similarity between clients' local models (i.e., a client's model update + current global model) to divide clients into clusters and filter out the clusters containing potentially malicious model updates.

FLCert [16]: FLCert divides clients into multiple groups and trains a global model for each group using any existing FL algorithm (we use Median [39]). Given a testing input, FLCert predicts its label based on the majority vote among these global models. We divide the clients into 10 disjoint groups uniformly at random in our experiments.

FLDetector [40]: FLDetector aims to detect malicious clients during training. We first apply FLDetector with the full participation of clients for detection for 300 communication rounds. Following [40], if some clients are detected as malicious, we remove them from the system. Then we re-train a global model from scratch based on the remaining clients using Median [39] aggregation rule with the default setting.

B.3. Compared Attacks

We compare our PoisonedFL with seven attacks, including five attacks (Fang [17], Opt. Fang [32], LIE [10], Min-Max [32], and Min-Sum [32]) that require genuine clients'

information and two attacks (Random [13] and MPAF [13]) that do not require such information. Note that when applying the first category of attacks to craft the malicious model updates on fake clients, we assume that the attacker has access to the model updates of *all* genuine clients and the aggregation rule used by the server. In Section D, we also study the scenarios where an attacker uses the global models to reconstruct synthetic data on the fake clients and uses the local models trained on them to perform attacks. We do not make these assumptions for Random, MPAF, and our PoisonedFL. Therefore, we give advantages to the first category of attacks.

Fang [17]: In this attack, the attacker crafts malicious model updates for fake clients such that the aggregated model update after attack deviates substantially from the before-attack one. Fang has different versions for different aggregation rules. We apply their Krum attack for Multi-Krum defense and TrMean attack for others.

Opt. Fang [32]: Following Fang, this attack also aims to maximize the deviation between the after-attack aggregated model update and before-attack one, but it uses a different way to solve the malicious model update. Specifically, a malicious model update is a variant of the average of all genuine clients' model updates.

A little is enough (LIE) [10]: In LIE attack, fake clients craft their malicious model updates by adding a small amount of noise to the average of genuine model updates. Specifically, for each dimension j, LIE calculates the mean μ_j and standard deviation σ_j among the genuine model updates, and the dimension j of the malicious model update is set as $\mu_j + 0.74 \cdot \sigma_j$.

Min-Max [32]: In this attack, a fake client crafts its malicious model update such that its distance to any genuine model update is no larger than the maximum distance between any two genuine model updates.

Min-Sum [32]: Each fake client crafts its malicious model update such that the sum of the distances between the malicious model update and all genuine model updates is not larger than the sum of distances between any genuine model update and other genuine model updates.

Random [13]: In this attack, each fake client $i \in [n+1, n+m]$ sends a scaled random vector $\boldsymbol{g}_i^t = \lambda \cdot \boldsymbol{\epsilon}$ to the server, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and λ is a scaling factor. Following prior work [13], we set $\lambda = 1e6$ in our experiments.

MPAF [13]: In MPAF attack, during training round t, fake client $i \in [n+1, n+m]$ crafts its malicious model update as $\boldsymbol{g}_i^t = \lambda \cdot (\boldsymbol{w}' - \boldsymbol{w}^{t-1})$, where $\boldsymbol{w}'$ is an attacker-chosen random model, $\boldsymbol{w}^{t-1}$ is the global model in the previous training round $t-1$, and λ is a scaling factor. Following prior work [13], we set $\lambda = 1e6$ in our experiments.

B.4. FL Settings

We primarily establish the FL setting in a cross-device scenario, which is more feasible for model poisoning attacks, as discussed in Section D. Therefore, we assume $1,200$ genuine clients and 20% fake clients. The learning rates for the five datasets are selected within the range of 0.01 to 0.1 for optimal training effectiveness. Specifically, the learning rates for MNIST, FashionMNIST, and FEMNIST are set to 0.01, while CIFAR-10 is set to 0.03, and Purchase is set to 0.1. The number of training rounds is configured to ensure full convergence under various scenarios: $6,000$ for MNIST, FashionMNIST, and FEMNIST, and $10,000$ for CIFAR-10 and FEMNIST. The batch size is set to 32 for MNIST, FashionMNIST, CIFAR-10, and FEMNIST, and to 128 for Purchase, based on the number of local data samples.

C. Additional Experimental Results

C.1. Analyzing the Effectiveness of PoisonedFL

Why PoisonedFL breaks FL defenses: State-of-the-art defenses leverage filtering and clipping to reduce the impact of malicious model updates. They can weaken the attack effect in individual training rounds. However, since PoisonedFL crafts malicious model updates that are consistent across training rounds and avoid being entirely filtered out by dynamically adjusting attack magnitudes, its cumulative attack effect still substantially moves the global model towards the designated random update direction $\boldsymbol{s}$, leading to a large testing error rate. For instance, Fig. 7 in Appendix shows that a large fraction of the dimensions of the total aggregated model updates have signs matching with $\boldsymbol{s}$ and Fig. 8 in Appendix shows that the magnitude of the total aggregated model updates becomes very large, after our attack effect accumulates over multiple training rounds, for all defenses.

Specifically, for Median and TrMean, even though the malicious model updates are not necessarily selected, their consistent nature over multiple training rounds biases the aggregation of each dimension towards the designated random update direction. For Norm Bound, even if our attack effect is weakened in each training round, its cumulative effect over multiple training rounds remains pronounced with a significant magnitude for the total aggregated model update. For filtering-involved defenses based on Euclidean distance or cosine similarity, such as Multi-Krum, FLTrust, and FLAME, PoisonedFL maintains effectiveness because the malicious model updates are not entirely filtered out in many training rounds due to dynamic magnitude adjustment. For instance, Table 5 in Appendix shows that a large fraction of or all malicious model updates are not filtered out by FLAME and Multi-Krum, while Table 6 shows similar results for FLTrust. FLCert is vulnerable to PoisonedFL

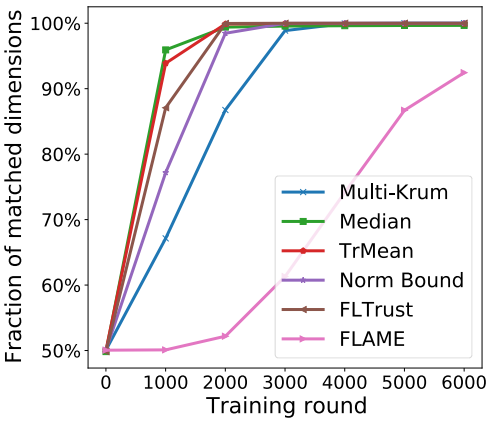


Figure 7. Fraction of dimensions of the aggregated model update whose signs match with the sign vector $\boldsymbol{s}$ as a function of training round under PoisonedFL. The dataset is Purchase.

Table 5. The fraction (mean $\pm$ standard deviation, %) of malicious model updates in PoisonedFL that are not filtered out by FLAME and Multi-Krum (i.e., false negative rate) across rounds. The dataset is Purchase.

	Multi-Krum	FLAME
False Negative Rate	32.38±2.04	44.25±8.18

since the global models for ensembling are learnt by existing aggregation rules, which are vulnerable to PoisonedFL. The detection mechanism FLDetector relies on inconsistent model updates of malicious clients. Since the malicious model updates are consistent in PoisonedFL, FLDetector falsely classifies all malicious clients as benign across all datasets.

Why PoisonedFL outperforms existing attacks: PoisonedFL outperforms existing attacks because their crafted malicious model updates are inconsistent across training rounds, leading to self-cancellation of attack effect. In contrast, PoisonedFL enforces the malicious model updates to be consistent (i.e., have the same sign vector) across training rounds, making the aggregated model updates add up in the global model across training rounds. As a result, the final learnt global model has a very large magnitude, leading to a large testing error rate. For instance, Fig. 8 in Appendix shows the magnitude (measured by ℓ_2 norm) of the total aggregated model updates as a function of training round for different attacks on the Purchase dataset. Our results show that the total aggregated model updates under existing attacks have small magnitudes while those under our PoisonedFL attack have much larger magnitudes, which is caused by the inconsistent malicious model updates in existing attacks vs. consistent malicious model updates in our attack.

Table 6. The normalized trust score (mean $\pm$ standard deviation) of the malicious/benign model updates in FLTrust against PoisonedFL across rounds. The positive normalized trust scores indicate that the malicious model updates are not entirely filtered out. The dataset is Purchase.

	Malicious	Benign
Normalized Trust Score	0.008±0.010	0.007±0.002

C.2. Impact of the Participation Rate

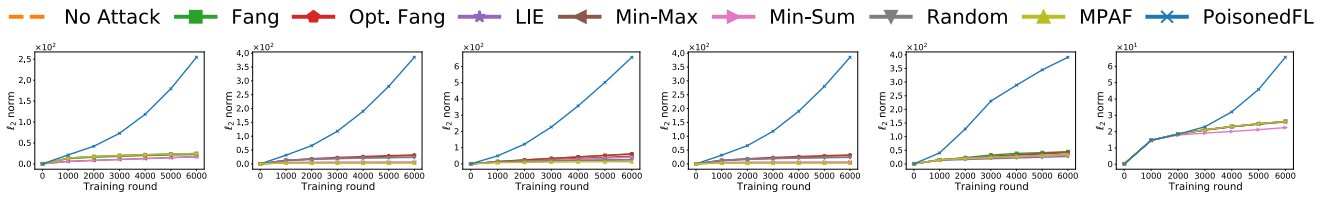
In each training round, the server typically selects a fraction (called *participation rate*) of the clients to participate in training. We use participation rate of 0.1 by default in our experiments. Fig. 9 shows the testing error rates under different defenses and attacks when the participation rate ranges from 0.01 to 1, where FashionMNIST dataset is used. Note that for these experiments, we use FashionMNIST instead of CIFAR-10 dataset, due to limited memory of our GPU server. Moreover, we do not have results for Multi-Krum in Fig. 9 when the participation rate is 1 due to the same reason on insufficient memory. For FLCert, the smallest participation rate is set to 0.03 instead of 0.01, because FLCert divides the clients into 10 groups, and if a participation rate of 0.01 is used, only one client is selected to participate in each training round. Consequently, attacks requiring genuine local models are not applicable. We observe that participation rate almost has no impact on the effectiveness of both existing attacks and our attack. Moreover, our attack consistently outperforms existing attacks across all considered participation rates.

C.3. Impact of Different PoisonedFL Variants

PoisonedFL crafts a malicious model update in each training round as a product of a given sign vector and a dynamically set magnitude vector, which is further decomposed into a product of an unit magnitude vector and a scaling factor. The unit magnitude vector and scaling factor play important roles in a malicious model update. Thus, we study different variants for designing them. We show results on CIFAR-10 dataset for simplicity.

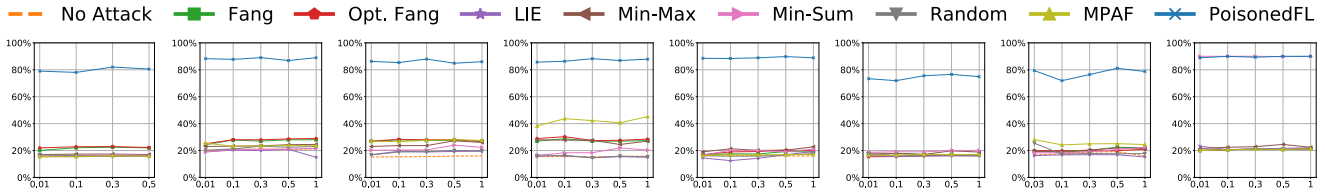
Variants of the unit magnitude vector: For the unit magnitude vector, we consider the following two variants.

- **Same magnitude**: In this variant, the attacker sets each dimension of the unit magnitude vector $\boldsymbol{v}^t$ to the same value. In other words, we set each dimension to be $\frac{1}{\sqrt{d}}$, where d is the number of dimensions/parameters.
- **Adaptive magnitude**: In this variant, the dimensions of the unit magnitude vector have different magnitudes and are dynamically set in each training round based on how the global model changes and the malicious model updates in the previous training round. Equation 6 shows our adaptive unit magnitude vector adopted by Poi-



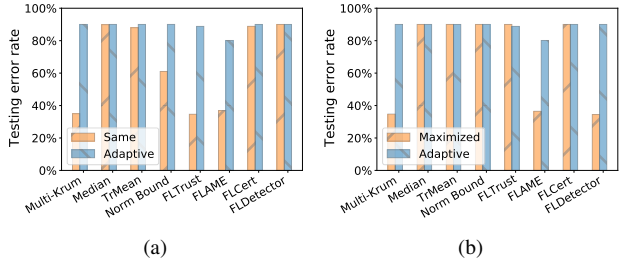
(a) Multi-Krum (b) Median (c) TrMean (d) Norm Bound (e) FLTrust (f) FLAME

Figure 8. ℓ_2 norm of the total aggregated model updates $\left\|\sum_{t'=1}^{t} \boldsymbol{g}^{t'}\right\|$ (i.e., $\left\|\boldsymbol{w}^t - \boldsymbol{w}^0\right\|$) as a function of training round t for different attacks on the Purchase dataset.



(a) Multi-Krum (b) Median (c) TrMean (d) Norm Bound (e) FLTrust (f) FLAME (g) FLCert (h) FLDetector

Figure 9. Testing error rate of the global model as a function of the participation rate under different defenses and attacks.



(a) (b)

Figure 10. (a) Comparing the two variants of setting the unit magnitude vector in PoisonedFL. (b) Comparing the two variants of setting the scaling factor in PoisonedFL.

sonedFL.

Fig. 10a shows the testing error rates under different defenses when PoisonedFL uses the two variants to set the unit magnitude vector, where the scaling factor is set using our adaptive method in Equation 7. Our results show that adaptive magnitude substantially outperforms same magnitude for some defenses (e.g., Multi-Krum and FLTrust) while the two variants achieve comparable effectiveness for other defenses. This is because adaptive magnitude can target specific dimensions with larger effectiveness.

Variants of the scaling factor: We also consider two variants of setting the scaling factor as follows:

- **Maximized scaling factor**: This variant sets a very large scaling factor with a goal to maximize the magnitude of the aggregated model update. We use $100,000$ in our experiments.
- **Adaptive scaling factor**: This variant, adopted by PoisonedFL, leverages how the global model changes in the previous training round to set the scaling factor (i.e., Equation 7) to avoid that the malicious model updates are filtered out by the server.

Fig. 10b illustrates the testing error rates under different defenses when PoisonedFL uses the two variants of setting the scaling factor, where adaptive magnitude is used to set the unit magnitude vector. We observe that, under some defenses (e.g., Median, TrMean, and Norm Bound), the variant of maximized scaling factor achieves comparable testing error rates with adaptive scaling factor. This is because, for dimension-wise aggregation rules like Median and TrMean, although the malicious model updates with very large magnitudes are filtered out, they still deviate the aggregated model update; and although Norm Bound normalizes the norms of malicious model updates, the normalized malicious model updates still maintain consistency across training rounds. However, for other defenses like Multi-Krum and FLDetector that filter out entire model updates, adaptive scaling factor substantially outperforms maximized scaling factor. This is because these defenses filter out the entire malicious model updates with very large magnitudes. In contrast, adaptive scaling factor leverages how the global model changes in the past to achieve a balance between attack effectiveness and undetectability. Since PoisonedFL aims to be defense-agnostic, adaptive scaling factor is preferred.

C.4. Impact of the Sign Vector

We further analyze the impact of the random sign vectors. We experiment with PoisonedFL on CIFAR-10 with 6 random seeds (1-6) for various random sign vectors and de-

Table 7. Testing error rate of the global model under PoisonedFL with various random sign vectors obtained from different random seeds on CIFAR-10.

	seed=1	seed=2	seed=3	seed=4	seed=5	seed=6
Norm Bound	90.00	90.00	89.95	90.00	88.94	90.00
FLTrust	90.00	88.15	89.09	90.00	90.00	90.00
FLAME	80.79	86.59	84.43	81.39	80.68	82.37

fenses including TrMean, FLTrust, and FLAME, as shown in Table 7. The result aligns with our intuition that these random sign vectors do not affect the attack performance and PoisonedFL can successfully break the defenses with these 6 different random sign vectors. The reason is that in a continuous attack with the same random direction, the model inevitably updates substantially in a random direction in an uncontrolled way, and therefore leads to significantly degraded accuracy.

D. Discussion and Limitations

D.1. Countermeasures and Adaptive Attacks

We explore new defenses tailored to PoisonedFL. Specifically, we study a new defense that normalizes the total aggregated model update since PoisonedFL aims to increase its magnitude. We also explore new tailored defenses to detect fake clients based on how PoisonedFL crafts the malicious model updates. We show that the normalization-based defense has limited effectiveness at mitigating PoisonedFL and we can still adapt PoisonedFL to break the detection based defenses.

In our tailored detection based defenses, the server extracts a feature for each client in each training round and then divides the clients into two clusters via fitting a two-component Gaussian Mixture Model (GMM) [29] for the features. We detect a particular cluster (e.g., with the smaller mean value) as fake clients. After detecting fake clients, the server removes them and re-trains a global model using the remaining clients. Note that if the distance between the mean values of the two clusters is smaller than the intra-cluster standard deviations of both clusters, no clients are detected as fake and the training continues since the difference between the two clusters may simply be caused by randomness.

We extract features for clients based on how PoisonedFL crafts malicious model updates. Depending on which information of the malicious model updates is used to extract features, we consider two variants of GMM, i.e., *GMM-Magnitude* and *GMM-Sign*, which extract features based on the magnitudes and signs of the model updates, respectively. Due to limited space, we discuss GMM-Magnitude in Appendix D.1.2.

D.1.1. GMM-Sign

Feature: GMM-Sign exploits the consistency of the malicious model updates to detect fake clients. Specifically, for a client i in training round t, we define the feature x_i^t as the number of flipped dimensions of its model updates in two consecutive training rounds averaged in the recent N training rounds. Formally, we have:

$$x_i^t = \sum_{t'=t-N+1}^{t} \sum_{j=1}^{d} \mathbb{I}(\text{sign}(\boldsymbol{g}_i^{t'}[j]) \neq \text{sign}(\boldsymbol{g}_i^{t'-1}[j])), \quad (8)$$

where d is the number of dimensions/parameters of a model update; $\mathbb{I}$ is the indicator function that has a value of 1 if $\text{sign}(\boldsymbol{g}_i^{t'}[j]) \neq \text{sign}(\boldsymbol{g}_i^{t'-1}[j])$ and 0 otherwise; and $\boldsymbol{g}_i^{t'}[j]$ is the j-th dimension of $\boldsymbol{g}_i^{t'}$. For PoisonedFL, we have $x_i^t = 0$ for all fake clients. After dividing the clients into two clusters, the cluster with a smaller mean value is detected as fake clients.

Adaptive attack: In response to the defense, we adapt PoisonedFL. Specifically, we randomly flip α (called *flipping rate*) fraction of the signs in each fake client's malicious model update and adjust the magnitudes of the corresponding dimensions to a small random value. Formally, we set the d-th dimension of the sign vector $\boldsymbol{s}_i^t$ and magnitude vector $\boldsymbol{k}_i^t$ for fake client i in training round t as follows:

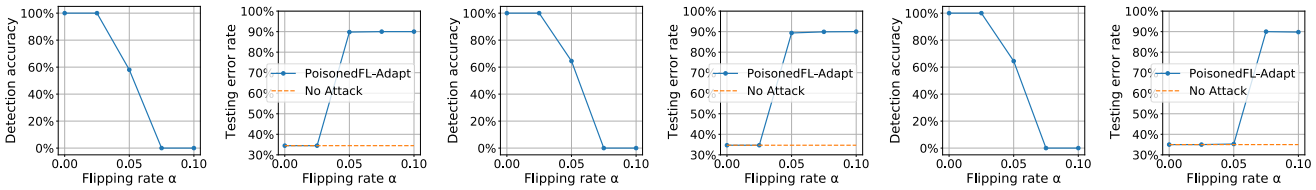
$$\begin{aligned} \boldsymbol{s}_i^t[j] &= \begin{cases} -\boldsymbol{s}[j] & \text{with probability } \alpha \\ \boldsymbol{s}[j] & \text{with probability } 1-\alpha \end{cases}, \\ \boldsymbol{k}_i^t[j] &= \begin{cases} \epsilon & \text{if } \boldsymbol{s}_i^t[j] = -\boldsymbol{s}[j] \\ \boldsymbol{k}^t[j] & \text{otherwise} \end{cases}, \end{aligned} \quad (9)$$

where $j \in [1, d]$, $\boldsymbol{s}$ is the sign vector picked randomly at the beginning of an attack, $\boldsymbol{k}^t$ is the magnitude vector for training round t crafted by PoisonedFL, and ϵ is a small value.

Experimental results: Fig. 11 shows the detection accuracy of GMM-Sign and the testing error rates as the flipping rate α varies when the server uses different aggregation rules, where the dataset is CIFAR-10 and $\epsilon = 1e{-}6$. We observe that GMM-Sign can accurately detect the fake clients when α is small. However, the detection accuracy reduces to 0 and PoisonedFL-Adapt still makes the learnt global models random guessing when α is larger than a threshold, e.g., 0.08.

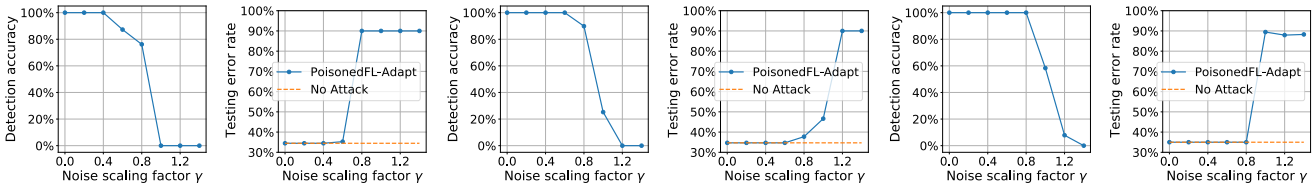
D.1.2. GMM-Magnitude

Feature: PoisonedFL uses the same malicious model update (i.e., $\boldsymbol{g}_i^t = \boldsymbol{k}^t \odot \boldsymbol{s}$) on all fake clients in training round t. Therefore, the server can extract features for the clients based on the similarity between their model updates. Specifically, for each client i in training round t, we propose



(a) Median (b) TrMean (c) Norm Bound

Figure 11. Detection accuracy of GMM-Sign and testing error rate of the global model under PoisonedFL-Adapt attack as the flipping rate α varies when the server uses different aggregation rules.



(a) Median (b) TrMean (c) Norm Bound

Figure 12. Detection accuracy of GMM-Magnitude and testing error rate of the global model under PoisonedFL-Adapt attack as the noise scaling factor γ varies when the server uses different aggregation rules.

to calculate a feature x_i^t as the sum of the average square distance between the client's model update and its closest $m-1$ model updates in the recent N training rounds. Formally, we have:

$$x_i^t = \sum_{t'=t-N+1}^{t} \sum_{j \in \mathbb{N}_i^{t'}} \frac{\left\| \boldsymbol{g}_i^{t'} - \boldsymbol{g}_j^{t'} \right\|^2}{m-1}, \tag{10}$$

where $\mathbb{N}_i^{t'}$ is the set of $m-1$ clients whose model updates are the closest to that of client i in training round t' and m is the number of fake clients. Note that we give advantages to the defense by assuming that the number of fake clients is known to the server. In a training round, the server divides the clients into two clusters via GMM based on the features. The cluster with a smaller mean value is detected as fake clients. The mean value is exactly zero for PoisonedFL since the malicious model updates are the same for the fake clients in each training round.

Adaptive attack: In response to such a defense, we can further adapt our PoisonedFL by introducing random noise to the magnitude vector $\boldsymbol{k}^t$. Specifically, we can add a normalized noise to the magnitude vector for each fake client i as follows:

$$\boldsymbol{k}_i^t = \boldsymbol{k}^t + \gamma \cdot \frac{||\boldsymbol{k}^t||}{||\boldsymbol{\epsilon}_i||} \cdot \boldsymbol{\epsilon}_i, \tag{11}$$

where $\boldsymbol{k}^t$ is the magnitude vector crafted by PoisonedFL, $\boldsymbol{k}_i^t$ is the magnitude vector for fake client i crafted by PoisonedFL-Adapt, $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the random noise sampled from the standard Gaussian distribution, $\frac{||\boldsymbol{k}^t||}{||\boldsymbol{\epsilon}_i||}$ normalizes the random noise to have the same magnitude as $\boldsymbol{k}^t$, and γ is a scaling factor for the random noise. The noise makes the malicious model updates of the fake clients less similar to each other and thus evade detection.

Experimental results: Fig. 12 shows the detection accuracy of GMM-Magnitude and the testing error rates of the global models as the noise scaling factor γ varies when the server uses different aggregation rules, where the dataset is CIFAR-10 and $N = 20$. We observe that when γ is small, i.e., a small amount of noise is added to each malicious model update, GMM-Magnitude can accurately detect the fake clients and thus our PoisonedFL-Adapt is not effective. However, when γ is larger than a threshold (e.g., 1.2), GMM-Magnitude fails to detect the fake clients and PoisonedFL-Adapt still makes the learnt global models nearly random guessing (i.e., testing error rates are around 90%). This is because fake clients and genuine clients have distinguishable features when γ is small while their features are indistinguishable when γ is large. PoisonedFL-Adapt still substantially increases the testing error rates when γ is large because the malicious model updates still have consistent sign vectors.

D.1.3. Normalization-based Defense

PoisonedFL aims to increase the magnitude of the total aggregated model update, i.e., $\left\|\boldsymbol{w}^T - \boldsymbol{w}^0\right\|$. Therefore, one tailored defense is to normalize the total aggregated model update. Specifically, the server can normalize $\boldsymbol{w}^T - \boldsymbol{w}^0$ to have a predefined ℓ_2 norm b and add the normalized total aggregated model update to the initial global model $\boldsymbol{w}^0$ to get a new global model, i.e., $\boldsymbol{w}^T \leftarrow \boldsymbol{w}^0 + b\frac{\boldsymbol{w}^T - \boldsymbol{w}^0}{\|\boldsymbol{w}^T - \boldsymbol{w}^0\|}$. Fig. 13 shows the testing error rate of the final global model under PoisonedFL as the predefined norm b varies. Our results show that the testing error rate can be decreased by this

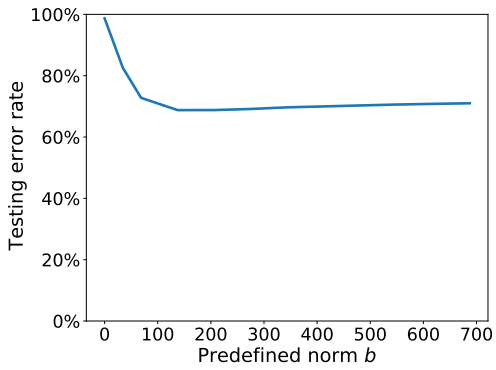


Figure 13. The testing error rate of a global model as a function of the predefined norm b used to rescale the total aggregated model update. The dataset is Purchase and FL defense is Median.

defense but is still high. This is because the update direction of the total aggregated model update is randomly picked by the attacker, which normalization cannot change.

D.2. Using Synthetic Data

In our previous experiments, when applying attacks that require genuine clients' local models to craft malicious model updates on fake clients, we assume the attacker has access to the local models on all genuine clients. This assumption is strong and not realistic. Therefore, we also explore that an attacker uses the global models to reconstruct synthetic data, train local models using the synthetic data, and then craft malicious model updates based on them.

Reconstructing training data from a model is known as *model inversion* [20]. Previous work [30] shows that the global-model trajectory in multiple training rounds (especially early ones) of FL can be used to reconstruct synthetic data that is similar to the clients' genuine local training data. We follow the setting and generation strategy in [30], which is the state-of-the-art reconstruction method in FL, to generate synthetic data for MNIST and FashionMNIST. We use MNIST and FashionMNIST because they are simpler datasets and easier to reconstruct, giving advantages to these attacks. Specifically, we assume no attacks in the first 40 training rounds and use the corresponding clean global models to reconstruct synthetic data. Table 8 shows the testing error rates of the classifiers trained on the synthetic data and evaluated on the genuine testing datasets of MNIST and FashionMNIST. The results show that the classifiers have decent testing error rates and thus the synthetic data mimic the genuine data well. These results are also consistent with prior work [30].

We then use the synthetic data on the fake clients to perform model poisoning attacks in the remaining training rounds. Since each genuine client has 50 local training examples, we sample 50 synthetic training examples for each fake client. In each training round, the fake clients selected to participate in training train local models using their synthetic local training examples; and then we craft malicious model updates based on the local models for existing attacks. Table 9 shows the testing error rates of the final global models under different attacks and defenses. We observe that an existing attack achieves higher testing error rates when having access to local models on the genuine clients in most cases, compared to using local models trained on synthetic data. PoisonedFL does not require genuine local models nor synthetic data but substantially outperforms these attacks.

Table 8. Testing error rate of a classifier trained on synthetic data and evaluated on genuine testing data.

	MNIST	FashionMNIST
Testing error rate	9.94	24.74

Table 9. Testing error rates of attacks using synthetic data on fake clients (i.e., "Syn. Data + ") and local models of all genuine clients (i.e., "Genuine + "), as well as our PoisonedFL that does not require synthetic data nor genuine local models.

		MNIST	FashionMNIST
Median	Syn. Data + Fang	4.71	23.30
	Genuine + Fang	8.43	27.82
	Syn. Data + Min-Max	4.79	22.09
	Genuine + Min-Max	8.52	23.30
	PoisonedFL	**86.49**	**87.75**
TrMean	Syn. Data + Fang	5.46	25.48
	Genuine + Fang	5.66	27.96
	Syn. Data + Min-Max	5.43	19.77
	Genuine + Min-Max	5.66	23.68
	PoisonedFL	**88.73**	**85.36**
FLTrust	Syn. Data + Fang	3.72	19.50
	Genuine + Fang	4.61	18.78
	Syn. Data + Min-Max	3.68	16.53
	Genuine + Min-Max	12.56	21.32
	PoisonedFL	**88.65**	**88.41**
FLCert	Syn. Data + Fang	3.73	20.80
	Genuine + Fang	4.61	18.78
	Syn. Data + Min-Max	3.06	16.27
	Genuine + Min-Max	4.57	19.28
	PoisonedFL	**88.06**	**71.92**

D.3. Cross-silo FL

PoisonedFL primarily targets *cross-device FL* [21], in which the clients are end-user devices such as smartphones. An attacker can inject fake clients into a cross-device FL system due to the open nature of the system. In particular, any client including fake ones can participate in the FL system. We acknowledge that it is harder to apply both PoisonedFL and existing attacks to *cross-silo FL*. Specifically, the clients are often a small number of verified institutions such as banks and hospitals in cross-silo FL. These institu-

tions often go through certain verification processes before jointly training a model using FL. As a result, it is harder for an attacker to compromise genuine clients/institutions or inject fake clients/institutions in such a cross-silo FL system, making full-client control attacks like PoisonedFL—and all model poisoning attacks—difficult to execute. We believe it is an interesting future work to identify a realistic threat model for cross-silo FL that takes into account the unique challenges and constraints.