SIREN: Byzantine-robust Federated Learning via Proactive Alarming

Hanxi Guo
Shanghai Jiao Tong University
Shanghai
hanxiguo@sjtu.edu.cn

Hao Wang
Louisiana State University
Baton Rouge, Louisiana
haowang@lsu.edu

Tao Song
Shanghai Jiao Tong University
Shanghai
songt333@sjtu.edu.cn

Yang Hua
Queen's University Belfast
Belfast, Northern Ireland
Y.Hua@qub.ac.uk

Zhangcheng Lv
Huawei Technologies Co., Ltd.
Hangzhou, Zhejiang
lvzhangcheng@huawei.com

Xiulang Jin
Huawei Technologies Co., Ltd.
Hangzhou, Zhejiang
jinxiulang@huawei.com

Zhengui Xue
Shanghai Jiao Tong University
Shanghai
zhenguixue@sjtu.edu.cn

Ruhui Ma
Shanghai Jiao Tong University
Shanghai
ruhuima@sjtu.edu.cn

Haibing Guan
Shanghai Jiao Tong University
Shanghai
hbguan@sjtu.edu.cn

ABSTRACT

With the popularity of machine learning on many applications, data privacy has become a severe issue when machine learning is applied in the real world. Federated learning (FL), an emerging paradigm in machine learning, aims to train a centralized model while distributing training data among a large number of clients in order to avoid data privacy leaking, which has attracted great attention recently. However, the distributed training scheme in FL is susceptible to different kinds of attacks. Existing defense systems mainly utilize model weight analysis to identify malicious clients with many limitations. For example, some defense systems must know the exact number of malicious clients beforehand, which can be easily bypassed by well-designed attack methods and become impractical for real-world scenarios.

This paper presents SIREN, a Byzantine-robust federated learning system via a proactive alarming mechanism. Compared with current Byzantine-robust aggregation rules, SIREN can defend against attacks from a higher proportion of malicious clients in the system while keeping the global model performing normally. Extensive experiments against different attack methods are conducted under diverse settings on both independent and identically distributed (IID) and non-IID data. The experimental results illustrate the effectiveness of SIREN comparing with several state-of-the-art defense methods.

KEYWORDS

Federated Learning, Byzantine-robust, Attack-agnostic Defense System

ACM Reference Format:
Hanxi Guo, Hao Wang, Tao Song, Yang Hua, Zhangcheng Lv, Xiulang Jin, Zhengui Xue, Ruhui Ma, and Haibing Guan. 2021. SIREN: Byzantine-robust Federated Learning via Proactive Alarming. In *ACM Symposium on Cloud Computing (SoCC '21), November 1–4, 2021, Seattle, WA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3472883.3486990

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SoCC '21, November 1–4, 2021, Seattle, WA, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8638-8/21/11…$15.00
https://doi.org/10.1145/3472883.3486990

1 INTRODUCTION

Federated learning (FL) has become a new machine learning paradigm that enables distributed training with heterogeneous devices such as edge servers, mobile phones, and Internet-of-Things devices without violating data privacy [13, 22]. Unlike traditional machine learning that centralizes data to a cluster of servers for training [16, 18, 30], FL provides a strong privacy guarantee by sending training tasks to a swarm of loosely connected devices and exchanging model weights trained on devices to update a global model iteratively without leaking any sensitive raw data. FL has an extensive range of applications [20, 31], including natural language models, health tracking, and fintech with

privacy-sensitive data such as user keystrokes, photos, and geo-locations.

However, the distributed and iterative nature of FL has introduced new vulnerabilities in defending various attacks from malicious clients. FL performs distributed machine learning with a network of loosely connected devices that volunteer to participate in training, making it hardly possible to determine a concrete number of malicious clients. Besides, the local data on client devices are typically non-independent and identically distributed (non-IID). Such data skews across participating devices aggravate the divergence between local models, further obfuscating the boundary between malicious clients and benign ones.

As a result, federated learning is vulnerable to Byzantine attacks [15]—an attacker corrupts the federated model by smuggling malicious model updates through compromised and fake clients lurking among federated learning clients. Based on attackers' objectives, existing attacks can be classified into two types of attacks. *Untargeted attacks* aim to degrade the global model quality or slow down its convergence with carefully crafted model updates and data, such as sign-flipping and label-flipping [12, 17, 19, 28]. *Targeted attacks* manipulate the global model's accuracy over specific data categories [3, 5, 26].

To defend against these attacks, researchers have developed various Byzantine-robust FL frameworks [2, 6, 8–11, 33]. The main idea of these studies is to detect malicious clients by analyzing model gradients and measuring the difference between clients' model updates. However, these methods leave the following critical concerns unaddressed: *First*, attackers can still sabotage the global model and work around the detection of outlier model weights by adaptively fiddling their model updates. *Second*, non-IID data amplifies the divergence in model weights trained different clients [34], triggering false alarms that drive existing gradient-based methods to drop model updates from benign clients. *Finally*, existing methods only rely on the detection mechanism on the FL server to detect attacks that benign clients have to passively accept the corrupted global model without any resistance once the global model has been successfully affected by malicious clients.

This paper presents Siren, a Byzantine-robust federated learning system that orchestrates clients with the FL server to defend a wide spectrum of attacking methods on both IID and non-IID data by carefully analyzing both model accuracies and gradients. We design a proactive and distributed alarming system that enables clients to collaborate with the FL server on attack detection. Siren clients reserve a small partition of the local dataset to test the global model accuracy and trigger alarms, and the FL server jointly analyzes clients' alarms, model weight updates, and accuracies to detect attacks. Based on this distributed alarming system, we carefully craft a decision process that detects the intentions of malicious clients and sanitizes the model aggregation effectively.

Extensive experimental results of training a CNN model with the Fashion-MNIST dataset [25] and the CIFAR-10 dataset [14] under a system of up to 200 clients show that Siren can achieve the best performance in all scenarios. Compared with SOTA methods, Siren can defend more malicious clients while providing near-baseline performance, *i.e.*, 90.94% (Siren), 89.47% (Krum), and 91.46% (Baseline) under attacks using 40% malicious clients. Besides, Siren can restore the FL system even though the server has already been attacked successfully.

2 BACKGROUND

2.1 Federated Learning

Federated learning (FL) iteratively aggregates model updates from multiple client devices to train a shared global model without violating clients' data privacy. As shown in Fig. 1, in a communication round t, a remote FL server first pushes a global model $\boldsymbol{g}_t$ to client devices and collects model updates $\{\Delta \boldsymbol{g}_t^{(i)} | i \in K\}$ from a set of $|K|$ randomly selected client devices to update the global model from $\boldsymbol{g}_t$ to $\boldsymbol{g}_{t+1}$. The updated global model $\boldsymbol{g}_{t+1}$ is then pushed to client devices for the next round [13, 22]. Clients' local datasets may follow different distributions and are inaccessible for the FL server or other clients.

Specifically, we select $|K|$ client devices to participate in federated learning every round, and each client device i has a local dataset $D^{(i)}$, $i \in K$. Each participating client trains a local model $\boldsymbol{g}_t^{(i)}$ using its own data with an objective to jointly solve the following optimization problem—minimizing the expected empirical loss $F(\boldsymbol{g})$ on the training data across client devices:

$$\min_{\boldsymbol{g}} F(\boldsymbol{g}) := \min_{\boldsymbol{g}} \sum_{i=1}^{|K|} \mathbb{E}_{D^{(i)} \sim \mathcal{X}^{(i)}} [f(D^{(i)}, \boldsymbol{g})],$$

where $\boldsymbol{g}$ is the global model, $D^{(i)}$ is a set of local training data samples following an unknown distribution $\mathcal{X}^{(i)}$ on client i, and f denotes the local loss function.

The FL server orchestrates participating client devices to jointly solve this optimization problem. In the t-th communication round, each client i initializes its local model with the global model $\boldsymbol{g}_t$ and trains the model locally with gradient descent algorithms: $\boldsymbol{g}_{t+1}^{(i)} := \arg\min_{\boldsymbol{g}} \mathbb{E}_{D^{(i)} \sim \mathcal{X}^{(i)}} [f(D^{(i)}, \boldsymbol{g})]$. When the local training completes, the client i calculates and pushes the local model update $\Delta \boldsymbol{g}_{t+1}^{(i)} := \boldsymbol{g}_{t+1}^{(i)} - \boldsymbol{g}_t$ to the FL server. Typically, the FL server updates the global model

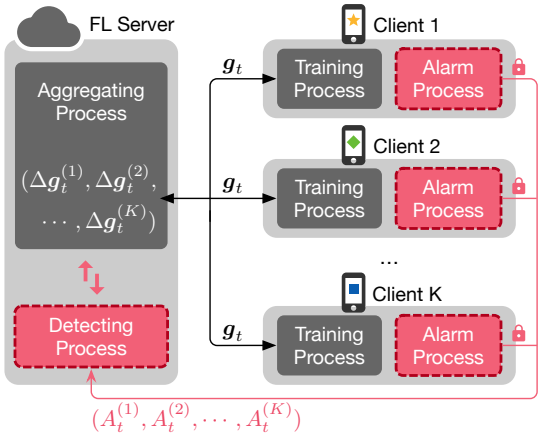


Figure 1: The architecture of FL. The gray blocks belong to the default FL paradigm, and the red blocks with dotted borders are SIREN's components.

using the federated averaging (FEDAVG) algorithm [22]:

$$g_{t+1} \leftarrow g_t + \eta \sum_{i=1}^{|K|} \frac{|D^{(i)}|}{|D|} \Delta g_{t+1}^{(i)},$$

where η is the learning rate, and $D := \sum_{i=1}^{|K|} D^{(i)}$ denotes the total data samples of the $|K|$ client devices. Then, the FL server sends the new global model g_{t+1} to client devices and starts the next communication round.

2.2 Prevailing Attacks to FL

Due to its distributed nature, existing federated learning systems are vulnerable to various types of attacks [21]. These attacks can be divided into three categories: manipulating gradients, poisoning local data, and the compound of the previous two categories of attacks.

Sign-flipping Attack [17], where malicious clients train the model as what benign clients do and obtain normal gradients[1] but multiplying the normal gradients by a negative constant when uploading.

Label-flipping Attack [12] is a data poisoning attack that lets the malicious client train with normal images but with flipped labels ($class_i$ to total class number $-$ $class_i - 1$).

Targeted Model Poisoning [5]'s objective is not to converge the global model to a sub-optimal point or diverge, unlike the two former types of attack. Instead, it manipulates the global model to perform as well as a normal model but degrading the performance when the global model faces one or several specific classes of images. To achieve this, the targeted model poisoning attack only changes one class's or several specific classes' labels and train the model. It also uses explicit boosting to increase the impacts of malicious updates.

[1]In this paper, gradients and weight updates are used interchangeably.

2.3 Byzantine-robust Aggregation Rules

This section introduces three prevailing types of defense methods used as the baseline defense schemes in the following experiments.

Krum [6]. In Krum, the FL server computes the score $s_t^{(i)}$ of each weight update in each communication round, where $s_t^{(i)} = \sum_{i \to j} ||\Delta g_t^{(i)} - \Delta g_t^{(j)}||^2$. Krum uses $i \to j (i \neq j)$ to select the indexes j of the $K-f-2$ nearest neighbors of $\Delta g_t^{(i)}$, measured by Euclidean distances, where f is the number of malicious clients selected for the aggregation in the system. After computing all the scores of all the weight updates, the FL server uses the weight update with the smallest score to do the aggregation. Meanwhile, other weight updates are dropped.

Coordinate-wise Median [33]. In coordinate-wise median, the FL server picks the medians of each coordinate from all the weight updates to build the global weights. Given a set of weight updates $\{\Delta g_t^{(i)}\}_{i=1}^K$ at a communication round t, the FL server uses the coordinate-wise median to do the aggregation, which is $\Delta g_t^{coomed} = coomed\{\{\Delta g_t^{(i)}\}_{i=1}^K\}$ with Δg_t^{coomed} being a vector with its j^{th} coordinate $\Delta g_t^{coomed}(j) = med\{\{\Delta g_t^{(i)}(j)\}_{i=1}^K\}$.

FLTrust [7]. In FLTrust, the server collects a root dataset and uses this root dataset to train an auxiliary server model in each communication round. Then, when receiving weight update $g_t^{(i)}$ from a client i, the server calculates a trust score TS_i of the client i, where $TS_i = ReLU(c_i)$ and c_i is the cosine similarity between $g_t^{(i)}$ and the update of the server model. After calculating all the trust scores, the server updates the global model $g_t = \frac{1}{TS_1 + \cdots + TS_K}(TS_1 \cdot \overline{g}_t^{(1)} + \cdots + TS_K \cdot \overline{g}_t^{(K)})$, where $\overline{g}_t^{(i)}$ is the normalized weight updates of the client i.

3 PROBLEM SETUP

Attack model: We use a similar attack model applied in previous works [5, 7, 12]. The attacker compromises several clients, has complete control over these clients and has full access to the local data. While the server is assumed not to be attacked by the attacker and there at least exists one benign client in the system since if all the clients are compromised, no defense methods could save the system. The attacker can only influence the server indirectly through uploading malicious weight updates using compromised clients. It does not know the data on other benign clients or the aggregation rules on the server. However, the network between the FL server and clients might be compromised, thus exposing data transmitted in the network to attackers. Besides, a malicious client may not attack the model in every communication

Table 1: Notations used in the paper.

Symbol	Meaning
t	the FL communication round index
K	the set of participating clients
S_a	the set of clients with activated alarms
S_s	the set of clients with no alarms, $S_a \cup S_s = K$
S_b	the set of benign clients, $S_b \subseteq K$
$\boldsymbol{g}_t$	the global model weight at round t
$\boldsymbol{g}_t^{(i)}$	Client i's local model weight at round t
$\Delta\boldsymbol{g}_t^{(i)}$	the local model update of client i
ω_t	the global model's testing accuracy at round t
$\omega_t^{(i)}$	the local model's testing accuracy of client i
$A_t^{(i)}$	the alarm on client i at round t
$D^{(i)}$	the local training dataset of client i
D_0	the root test dataset of the FL server
$D_0^{(i)}$	the local test dataset of client i
C_c	a user-defined threshold for clients
C_s	a user-defined threshold for the FL server

round and can upload correct model updates to camouflage itself from time to time.

Defender's setting: In Krum and Coordinate-wise Median, the defense is considered to be performed only on the server, while in Siren, the defense is performed on both server and client sides. However, no matter the types of defense methods, the server cannot access the local data of clients. It knows neither the raw local data nor the data distribution of clients. The server only knows the global model and the weight updates that the clients upload. Besides, in Siren, the server also has the capability to collect and keep a small root test dataset that cannot be poisoned by the attacker. The root test dataset on the server can be collected following different distributions, while we recommend that the distribution of this root test dataset should be similar to the overall data distribution across all the clients. On the client-side, compared with other defense mechanisms, Siren sets up a detecting process on the client to help the client check the integrity of the global model. This detecting process runs locally on each client, guaranteeing that no knowledge of clients will be leaked to the server through this process. This detecting process can only obtain the local test data, the global models distributed by the server, and the local models derived by the client in each communication round.

4 THE DESIGN OF SIREN

4.1 Overview

Since the only intermediate parameter between clients and FL server is the weight updates, malicious clients can only poison the global model through modifying their weight updates, no matter data poisoning [24] or model poisoning [4]. In this case, most current Byzantine-robust aggregation rules [10, 27, 32] only do the weight analysis and only deploy defense on the server, which makes the system vulnerable.

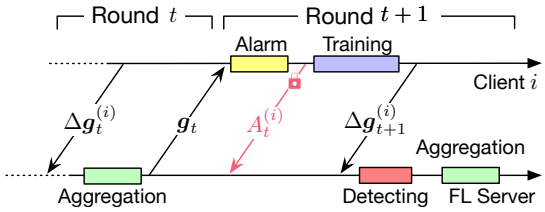


Figure 2: Illustration of interactions between Siren clients and the FL server.

To overcome these shortcomings, we propose a new proactive attack-agnostic defense system for FL, named Siren. Fig. 1 presents the structure of Siren that there are two processes on the client end, the training process and the alarm process. Siren preserves a small partition of the local dataset as local test data on each client. In contrast, in the standard FL system and the systems using other aggregation rules, the client only has a training process. The training process in Siren is the same as that in the standard FL, responsible for the local training using local data shard. The alarm process is responsible for testing the global weights. In each communication round, the alarm process on each client checks the global weights by using local weights and the local test dataset. If a client regards the global weight as poisoned weight, it will alarm the FL server, and the FL server will start the detecting process to exclude the malicious weight updates according to the alarm status of each client.

4.2 Client End

Fig. 2 presents Siren's client-end workflow that a client executes an alarming process to verify whether the global model $\boldsymbol{g}_t$ is poisoned, and uploads an alarm status $A_t^{(i)}$ and a model weight update $\Delta\boldsymbol{g}_t^{(i)}$ to the FL server in each communication round. Siren requires each client to keep a local test dataset and a copy of the local model weights generated in the previous round.

The alarming process compares the accuracy between the local model and the global model over the local test dataset to justify whether the global model is trusty so that the client can use it for the next round of local training. For simplicity, we use the client i to represent a general participating client, which could be either malicious or benign. A detailed description of the client-end algorithm is as follows:

Step 1: When the (t+1)-th communication round begins, the client i receives the global model weight $\boldsymbol{g}_t$ aggregated by the FL server in the previous (*i.e.*, the t-th) communication round.

Algorithm 1 Training and Alarming on clients.

function ClientUpdate(i, $\boldsymbol{g}_t$): ▷ *training process*
  $A_t^{(i)} \leftarrow$ **Alarm**($\boldsymbol{g}_t$, $\boldsymbol{g}_t^{(i)}$)
  if $A_t^{(i)}$ is 0 **then**
    $\boldsymbol{g}^{(i)} \leftarrow \boldsymbol{g}_t$
  else
    $\boldsymbol{g}^{(i)} \leftarrow \boldsymbol{g}_t^{(i)}$
  end if
  for each epoch $e = 1, \cdots, E$ **do**
    train the model $\boldsymbol{g}^{(i)}$ on the local dataset $D^{(i)}$,
    and obtain $\boldsymbol{g}_{t+1}^{(i)}$
    $\Delta \boldsymbol{g}_{t+1}^{(i)} \leftarrow \boldsymbol{g}_{t+1}^{(i)} - \boldsymbol{g}_t$ ▷ calculate model updates
  end for
  return $\Delta \boldsymbol{g}_{t+1}^{(i)}$
end function

function Alarm($\boldsymbol{g}_t$, $\boldsymbol{g}_t^{(i)}$): ▷ *alarming process*
  $\omega_t \leftarrow$ testing $\boldsymbol{g}_t$ on the local test dataset $D_0^{(i)}$
  $\omega_t^{(i)} \leftarrow$ testing $\boldsymbol{g}_t^{(i)}$ on the local test dataset $D_0^{(i)}$
  if $\omega_t - \omega_t^{(i)} \geq C_c$ **then**
    $A_t^{(i)} \leftarrow 0$ ▷ *the global model is normal*
  else
    $A_t^{(i)} \leftarrow 1$ ▷ *the global model is abnormal*
  end if
  send $A_t^{(i)}$ in a secure tunnel to the FL server
  return $A_t^{(i)}$
end function

Step 2: Unlike the default FedAvg algorithm that directly starts local training with the global model $\boldsymbol{g}_t$, each Siren client first launches the alarm process to evaluate both the global model $\boldsymbol{g}_t$ and the local model $\boldsymbol{g}_t^{(i)}$ trained in the previous communication round. The global model $\boldsymbol{g}_t$'s accuracy is ω_t, and the local model $\boldsymbol{g}_t^{(i)}$'s accuracy is $\omega_t^{(i)}$.

Step 3: To justify whether the global model $\boldsymbol{g}_t$ is poisoned, the alarm process further compares the accuracy ω_t and $\omega_t^{(i)}$. If the global model $\boldsymbol{g}_t$ is more accurate than the local model $\boldsymbol{g}_t^{(i)}$, *i.e.*, $\omega_t - \omega_t^{(i)} \geq C_c$, where C_c is a pre-defined positive threshold, the client i initializes the local model $\boldsymbol{g}^{(i)}$ with $\boldsymbol{g}_t$ in the $(t+1)$-th communication round training. Besides, the client i sets the alarm status $A_t^{(i)}$ as 0. In contrast, if $\omega_t - \omega_t^{(i)} < C_c$, then client i initializes the local model $\boldsymbol{g}^{(i)}$ with $\boldsymbol{g}_t^{(i)}$ instead of $\boldsymbol{g}_t$, due to the global model's abnormal performance. Correspondingly, the client i sets the alarm status $A_t^{(i)}$ to 1.

Step 4: The client i sends the alarm status $A_t^{(i)}$ to the FL server in a secure tunnel (*e.g.*, an IPsec tunnel based on the Diffie-Hellman algorithm), which prevents the alarm status from being tampered in network transmission, even when the client i is malicious and generates a false alarm.

The client i obtains a new model $\boldsymbol{g}_{t+1}^{(i)}$ by training the model $\boldsymbol{g}^{(i)}$ on its local data, where the alarm status $A_t^{(i)}$ determines $\boldsymbol{g}^{(i)}$ to be either $\boldsymbol{g}_t^{(i)}$ or $\boldsymbol{g}_t$ in Step 3. Then, the client i calculates and sends the local weight update $\Delta \boldsymbol{g}_{t+1}^{(i)} = \boldsymbol{g}_{t+1}^{(i)} - \boldsymbol{g}_t$ to the FL server and stores the local model $\boldsymbol{g}_{t+1}^{(i)}$ for the next round.

Algorithm 1 presents the pseudo code of the above client-end alarming and training processes. This client-end alarming mechanism guarantees that *a poisoned global model always triggers benign clients to alarm*. It should also be noted that malicious clients can deliberately fake alarms to delude the FL server even when the model is not poisoned. Siren recognizes such delusive alarms from malicious clients at the FL server end.

4.3 FL Server End

The FL server trusts neither local model updates nor alarms from any participating clients due to the inherent vulnerability of federated learning. Before aggregating local model updates and updating the global model as FedAvg does, Siren's FL server first launches a detecting process that analyzes the alarm statuses and evaluates local model weights to identify potential attacks.

In a communication round t, the FL server performs a *two-phase* detection: 1) Examining whether the global model generated in the previous round aggregation (*i.e.*, $\boldsymbol{g}_t$) is poisoned. 2) Testing whether the client model updates collected in the current round (*i.e.*, $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in K\}$) are poisoned. The following steps illustrate the two-phase detection process of the FL server in each communication round:

Step 1: In the t-th communication round, the FL server retrieves alarm status $A_t^{(i)}$ from all participating clients through secure tunnels and collects client model weight updates $\Delta \boldsymbol{g}_{t+1}^{(i)}$, where $i \in K$.

Step 2: The FL server analyzes all client alarms following the decision process illustrated in Fig. 3. If no client alarms, the FL server directly aggregates model weight updates from clients and updates the global model. However, if there are any alarms, the FL server further evaluates the model updates $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in S_a\}$ from the clients with activated alarms $A_t^{(i)} = 1$, where $S_a \subseteq K$ and S_a is the set of alarming clients.

Step 3: The FL server recovers client model weights using $\boldsymbol{g}_{t+1}^{(i)} = \Delta \boldsymbol{g}_{t+1}^{(i)} + \boldsymbol{g}_t$ and evaluates the client model with the root test dataset to justify whether the client model $\boldsymbol{g}_{t+1}^{(i)}$ is poisonous and whether client i is malicious, where $i \in S_a$. If no malicious clients are identified in S_a, the FL server

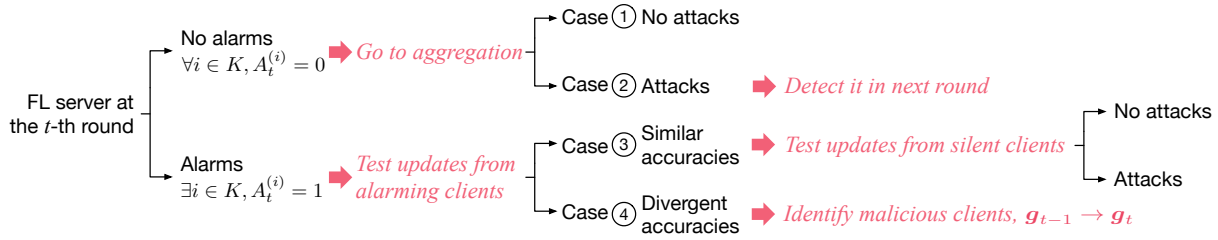


Figure 3: The FL server's decision process. The decisions made by the FL server are highlighted in red and *italic*.

will extend the model weight evaluation to all participating clients.

Step 4: The FL server filters out the client model updates identified as poisonous when aggregating model weight updates to update the global model $\boldsymbol{g}_{t-1}$ from the (t-1)-th communication round, rather than $\boldsymbol{g}_t$, since which is identified as poisoned. Therefore, the global model is updated as $\boldsymbol{g}_{t+1} = \boldsymbol{g}_{t-1} + \sum_{i \in S_b} \alpha^{(i)} \Delta \boldsymbol{g}_{t+1}^{(i)}$, where S_b is the set of clients identified as benign.

Step 5: The FL server drops $\boldsymbol{g}_t$ and copies the model weights $\boldsymbol{g}_{t-1}$ to $\boldsymbol{g}_t$. The (t+1)-th communication round begins after the global model $\boldsymbol{g}_t$ is pushed to all clients—we also craft a black list (Sec. 4.6) on the FL server—to exclude the clients identified as malicious for certain times from participating in training.

Algorithm 2 presents the pseudo-code of the detecting and aggregating processes at the FL server. The weight analysis used in Algorithm 2 is introduced in Sec. 4.5.

4.4 Decision Process and Security Analysis

We further analyze the detailed decisions made by the FL server and present the corresponding reasoning. If the FL server receives zero activated alarms at t-th round, there are two possible cases as shown in Fig. 3: Case ① $\boldsymbol{g}_t$ is not poisoned, and $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in K\}$ are all benign updates. Case ② $\boldsymbol{g}_{t-1}$ is not poisoned, but there are poisoned model updates in $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in K\}$. If $\boldsymbol{g}_t$ is poisoned, the client-end alarming mechanism will guarantee to activate alarms as long as one benign client exists. Besides, Case ② only happens when malicious clients poison the global model $\boldsymbol{g}_1$ at the first attacked communication round. Benign clients will detect such poisoned updates by comparing accuracy of the global model and the local model in the upcoming communication round (Sec. 4.2 Step 3). Thus, the FL server chooses to directly aggregate model updates when there is no alarm.

However, when there are activated alarms, the FL server first tests the accuracy of the models from the clients with activated alarms (*i.e.*, $i \in S_a$) and looks for the maximum accuracy $\max\{\omega_t^{(i)} | i \in S_a\}$ among these clients. We use a user-defined threshold C_s to measure the difference between the maximum accuracy and each alarming client's accuracy. The alarming clients either share a similar accuracy as the

Algorithm 2 Detection and aggregation on the FL server.

for each communication round $t = 1, 2, \cdots, T$ **do**
  for each client $i \in K$ **in parallel do**
    $A_t^{(i)} \leftarrow$ sent back by **Alarm**$(\boldsymbol{g}_t, \boldsymbol{g}_t^{(i)})$
    $\Delta \boldsymbol{g}_{t+1}^{(i)} \leftarrow$ **ClientUpdate**$(i, \boldsymbol{g}_t)$
  end for
  if $\forall i \in K, A_t^{(i)} = 0$ **then** ▷ *no alarms: Case ①②*
    $\boldsymbol{g}_{t+1} \leftarrow \boldsymbol{g}_t + \sum_{i \in K} \alpha^{(i)} \Delta \boldsymbol{g}_{t+1}^{(i)}$
  else ▷ *there are alarms*
    if $\forall i \in S_a, \max\{\omega_t^{(i)} | i \in S_a\} - \omega_t^{(i)} < C_s$ and $\omega_t^{(i)}$ passes the weight analysis **then**
      ▷ *similar accuracies: Case ③*
      if $\max\{\omega_t^{(i)} | i \in S_a\} - \max\{\omega_t^{(i)} | i \in S_s\} \leq C_s$ **then** ▷ *false alarms*
        $S_b \leftarrow$ detect and add benign silent clients
        $\boldsymbol{g}_{t+1} \leftarrow \boldsymbol{g}_t + \sum_{i \in S_b} \alpha^{(i)} \Delta \boldsymbol{g}_{t+1}^{(i)}$
      else
        $S_b \leftarrow$ all the alarming clients
        $\boldsymbol{g}_{t+1} \leftarrow \boldsymbol{g}_{t-1} + \sum_{i \in S_b} \alpha^{(i)} \Delta \boldsymbol{g}_{t+1}^{(i)}$
      end if
    else ▷ *divergent accuracies: Case ④*
      $S_b \leftarrow$ detect and add benign alarming clients
      $\boldsymbol{g}_{t+1} \leftarrow \boldsymbol{g}_{t-1} + \sum_{i \in S_b} \alpha^{(i)} \Delta \boldsymbol{g}_{t+1}^{(i)}$
    end if
  end if
end for

Case ③ that $\forall i \in S_a, \max\{\omega_t^{(i)} | i \in S_a\} - \omega_t^{(i)} < C_s$, or have divergent accuracies that $\exists i \in S_a, \max\{\omega_t^{(i)} | i \in S_a\} - \omega_t^{(i)} \geq C_s$ as the Case ④.

For Case ③, if there is no attack—neither the global model $\boldsymbol{g}_{t-1}$ nor client updates $\{\Delta \boldsymbol{g}_t^{(i)} | i \in K\}$ are not poisoned, the activated alarms must be *false alarms* deliberately generated by malicious clients. If there are attacks and the alarming clients' model updates have similar accuracies, then we should test the silent clients' model updates to further verify whether the alarming clients' model updates are all poisoned or all benign. Thus, for Case ③ we should always test all the silent clients' model updates $\{\Delta \boldsymbol{g}_t^{(i)} | i \in S_s\}$, where S_s is the set of silent clients. If the silent clients' highest accuracy is close to or even better than the alarming clients' highest

accuracy:

$$\max\{\omega_t^{(i)}|i \in S_a\} - \max\{\omega_t^{(i)}|i \in S_s\} \leq C_s,$$

the FL server can assure that the benign clients are silent and thus, all alarming clients' updates are poisoned. If the accuracy of a silent client's updates is close to the maximum accuracy of all silent clients, we believe this client is benign. So we add a silent client to the benign client set S_b, when its accuracy matches $\max\{\omega_t^{(i)}|i \in S_s\} - \omega_t^{(i)} < C_s$, where $i \in S_s$. Since all benign clients are silent, the alarms are generated by malicious clients as *false alarms*, and the global model from last round $\boldsymbol{g}_t$ is not poisoned.

Contrarily, for Case ③, if the maximum accuracy of silent clients is lower than the maximum accuracy of the alarming clients: $\max\{\omega_t^{(i)}|i \in S_a\} - \max\{\omega_t^{(i)}|i \in S_s\} > C_s$, then all silent client's model updates are poisoned, and all alarming clients are benign due to their similar accuracies.

For Case ④, the divergent accuracies of the alarming clients indicate that both benign and malicious clients are alarming. Again, we use the maximum accuracy of all alarming clients to filter out the alarming malicious clients. If an alarming client's accuracy satisfies $\max\{\omega_t^{(i)}|i \in S_a\} - \omega_t^{(i)} < C_s$, then we add it to the benign client set S_b. Since benign clients always alarm when detecting a poisonous $\boldsymbol{g}_t$, there are no benign clients among the silent clients. Thus, we will ignore all silent clients in this case.

Besides, we jointly apply accuracy checking and weight analysis to recognize malicious clients and achieve better detection (Sec. 4.5).

4.5 Weight Analysis

Since most of the attacks aim to generate malicious weight updates which can have reverse impacts on the global model compared with benign weight updates, these malicious weight updates usually represent reverse changing directions of the model compared with benign updates. With this intuition similar to FLTrust [7], weight analysis is added into Siren. However, unlike FLTrust using an auxiliary model trained by the data of the server, which means that the system has a predefined expectation of the global model, Siren only uses the information from clients. With weight analysis in Siren, the server not only compares the accuracy between the update with max accuracy and other updates but also compares the angles between these updates. If the angle between an update ω_i and the update with max accuracy $\max\{\omega_t^{(i)}|i \in S_a\}$ is greater than $\frac{\pi}{2}$, then ω_i will be regarded as a malicious update by the server. Otherwise, ω_i is regarded as a benign update and can be calculated into the global model. Via weight analysis, the server can check the updates from clients through another perspective while keeping its objectivity.

4.6 Auxiliary Mechanisms

To further improve the capability of Siren, some auxiliary mechanisms are also applied to Siren's architecture which is introduced in Sec. 4.2 and Sec. 4.3. All these auxiliary mechanisms are used only by the server so that clients do not have any extra computational burdens. And the server can flexibly determine whether to use these auxiliary mechanisms according to the computational resources on the server as well as the demand for better security and performance.

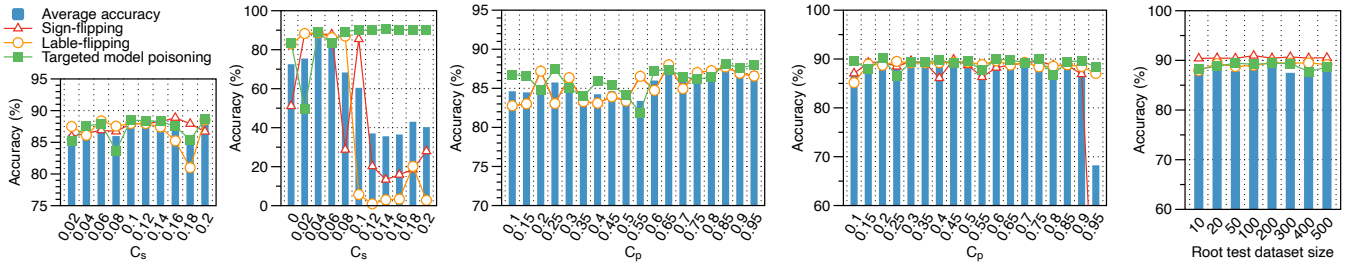
Penalty Mechanism: We design a penalty mechanism to improve the stability of Siren. Since malicious clients can attack the server consistently and the corresponding consistently checks waste a huge amount of computational resources. With the penalty mechanism, the server records the times of each client being regarded as a malicious client. If this count of a client is greater than a threshold C_p, then the server will not accept the update from this client without checking anymore since at this time, this client is regarded as a malicious client by default. With this method, the server can effectively save the computational overheads and improve the stability of the system.

Award Mechanism: The penalty mechanism may misjudge benign clients to be malicious clients because of the variance of the data on each client. Thus, the server exploits an award mechanism to rejoin a banned client into the training with a probability. In a communication round, if a banned client is regarded as a benign client by the server, the penalty count of this client will reduce C_a by the award mechanism. If the penalty count of this banned client is less than C_p, then this client could participate in the training process again. With this mechanism, the server can alleviate the side effect of the penalty mechanism.

5 EVALUATION

We have implemented a prototype of Siren based on TensorFlow [1] with more than 2,000 lines of Python code. We use the `multiprocessing` library to launch multiple processes to simulate multiple clients. Siren will be open-sourced after the review.

We verify the effectiveness of Siren by running practical FL tasks on two public benchmark datasets: Fashion-MNIST [25] and CIFAR-10 [14]. We evaluate Siren with three attacking methods, sign-flipping attack, label-flipping attack, and targeted model poisoning, and compare the two prevailing Byzantine-robust methods, Krum and coordinate-wise median. We also explore the capability of Siren to defend different proportions of malicious clients with and without weight analysis in our experiments. All of our evaluation experiments run on an NVIDIA DGX-2 virtual instance with six vCores and one NVIDIA Tesla V100 GPU.



(a) C_s on the server. (b) C_c on clients. (c) C_p on the FL server with only the penalty mechanism. (d) C_p on the FL server with both the penalty and award mechanisms. (e) Root dataset size on the FL server.

Figure 4: Searching the configuration of important hyper-parameters.

Our experimental results show that Siren can outperform Krum and coordinate-wise median even without knowing the types of attacks in advance. When the training data is IID, the global model trained by Siren can reach 90.94% and 90.64% respectively under sign-flipping and label-flipping attacks when the proportion of malicious clients is 40% and $|K| = 10$, while the FL baseline is 91.46%. In contrast, the model trained by Krum gets 88.18% and 87.81%, while the model trained coordinate-wise median obtains 68.8% and 79.09%. When the training data is non-IID, these accuracy differences become more obvious. The model trained by Siren can reach 88.88% and 89.89% when the FL baseline is 90.53%, while the model trained by Krum can only reach 84.66% and 85.14% and the model trained by coordinate-wise median has accuracy less than 80%. Besides, experiments with about 50 clients also show such differences. When the proportion of malicious clients reaches 80%, the model trained by Siren can always achieve 84% or higher in accuracy, while the other two kinds of Byzantine-robust defense methods cannot work in most cases.

5.1 Experiment Settings

Table 2 summarizes the experiment settings for FL on Fashion-MNIST. For experiments on CIFAR-10, we reuse the settings of Fashion-MNIST but change the local batch size to 32.

Model and Datasets: We train a CNN model with two CNN layers and two dense layers on the Fashion-MNIST dataset. For the CIFAR-10 dataset, we reuse this CNN model.

IID and Non-IID Data: For IID training data, we randomly split the whole training data into $|K|$ shards and allocate these data shards to $|K|$ clients directly. For non-IID training data, we introduce the non-IID degree p. By using p, a training data with label l is distributed into lth group of clients with possibility p. In this case, a higher p indicates a higher degree of non-IID. We set $p = 0.5$ in all the experiments over non-IID training data.

Table 2: Default settings of main parameters.

	Description	IID	Non-IID
B	Local batch-size	64	64
T	Communication rounds	40	40
E	Local training epochs	5	5
p	Non-IID degree	0	0.5
C_c	Client identification threshold	4%	4%
C_s	Server identification threshold	10%	10%
C_p	Penalty mechanism threshold	$0.45 \cdot \|K\|$	$0.45 \cdot \|K\|$
C_a	Award mechanism parameter	0.5	0.5

Metrics: We mainly compare the accuracy of the models and the robustness of the system (by using different proportions of malicious clients) under the attacks from different proportions of malicious clients using various types of attacks.

The root test dataset on the server: Siren uses a root test dataset on the server to recognize potentially malicious clients. We randomly pick 100 instances (Sec. 5.2) from the training dataset and exclude them from the training dataset, then distribute the remaining data to each client, as the server's root test dataset should be collected by the server instead of being derived from clients. This small root test dataset shares the same data distribution with the overall training data.

5.2 Parameter Selection

Figure 4 presents our exploration of Siren's optimal hyper-parameters for the FL server and clients. We use $|K| = 10$ over IID data to test Siren and calculate the average accuracy of the global models under these three attacks. Figure 4(b) and Figure 4(a) illustrate the result of C_c and C_s, respectively. According to the result, a larger threshold on clients can significantly reduce the performance, while a larger threshold on the server cannot influence the performance of the global model obviously. Figure 4(c) and Figure 4(d) show the result of C_p using the system with only the penalty mechanism and with both the penalty and the award mechanisms. With only the penalty mechanism, Siren needs a large C_p, which is around $0.8 \cdot K$ to reach the best performance. However, with the help of the award mechanism, Siren can easily reach a similar performance even using a quite smaller C_p. Besides, we also test the influence of different sizes of the root dataset, as shown in Figure 4(e). We pick a root dataset size that can promise the stable performance of Siren. All the detailed numbers of these hyper-parameters are shown in Table 2.

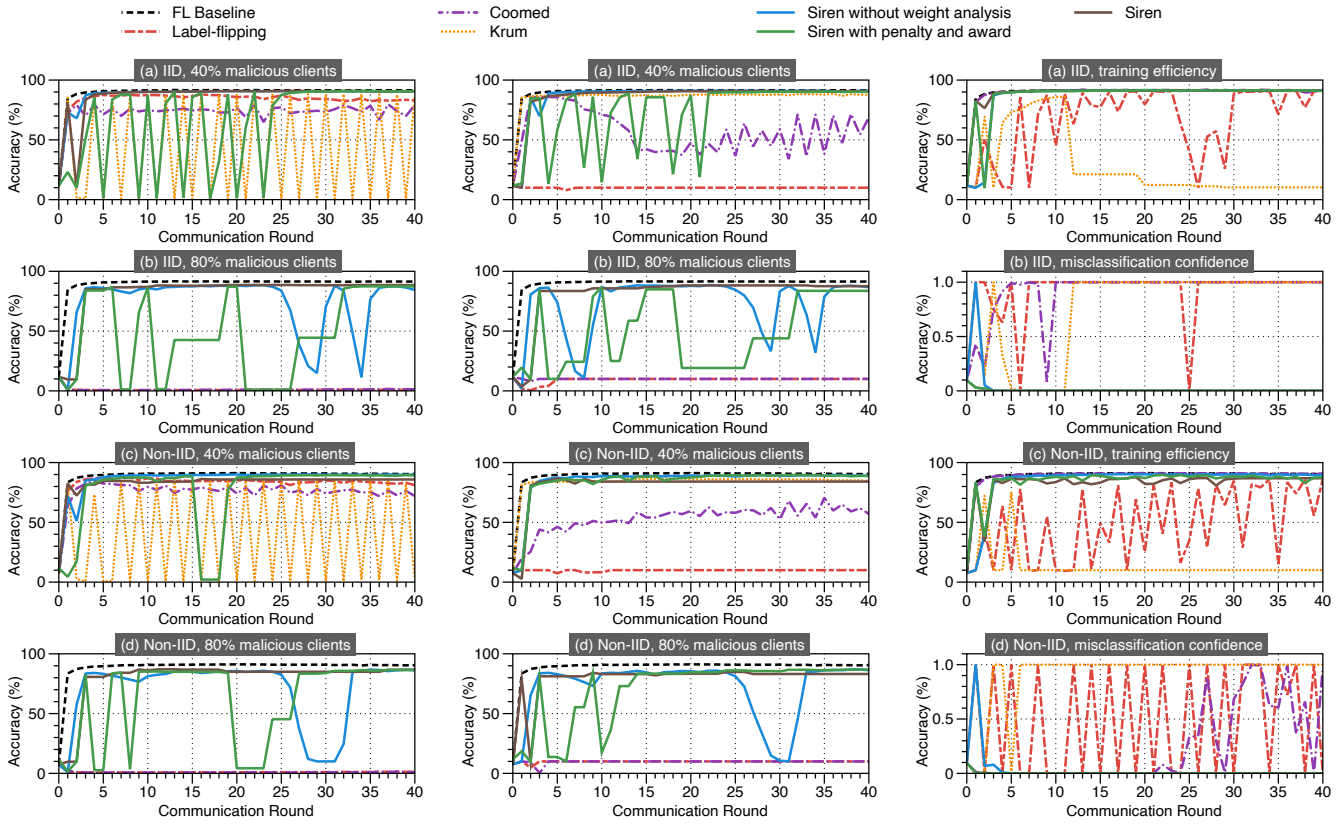


Figure 5: Training efficiency under label-flipping attack when $|K| = 10$.

Figure 6: Training efficiency under sign-flipping attack when $|K| = 10$.

Figure 7: Training efficiency and misclassification confidence under targeted model poisoning, $|K| = 10$.

5.3 Defending Sign-flipping Attack

Figure 6 shows the training efficiency when the FL system is protected by SIREN, Krum, and coordinate-wise median under sign-flipping attacks when $|K| = 10$. We use boosting factor -4 to boost the malicious weight updates when implementing the sign-flipping attack. Figure 6(a), Figure 6(c), Figure 6(b), and 6(d) describe the accuracy of FL with 40% and 80% of the clients as malicious, respectively. Krum cannot be initiated with 80% of clients as malicious so that Figure 6(b) and 6(d) omit it.

With 40% malicious clients in the system, both SIREN and Krum successfully defend the system on IID and non-IID training data. However, the coordinate-wise median is influenced by the attack, especially on non-IID data. With 80% malicious clients in the system, both Krum and coordinate-wise median fail to protect the FL system. However, Figure 6(b) and Figure 6(d) show that SIREN successfully protects the global model against 80% malicious clients. When the global model is aggregated from model weights of the only two benign clients, its accuracy evaluated on the test dataset drops as in Figure 6(b) and 6(d) because of the limited training data on the two benign clients. However, such reductions in the accuracy of SIREN are quite small and acceptable.

Figure 6(b) and Figure 6(d) compare SIREN with and without weight analysis. The global model's accuracy curves of SIREN without weight analysis drop suddenly, while the global model's accuracy curves of SIREN with weight analysis do not have such problems. Besides, Figure 6 shows that the accuracy of all the defense methods drops with the increasing of the malicious proportion since fewer training data samples are available when the malicious proportion increases.

Figure 11 visualizes each client's malicious index, which is maintained by the server of SIREN with the penalty mechanism to determine whether SIREN can detect malicious clients. A higher malicious index means that this client is more likely to be malicious since it has been regarded as a malicious client by the server more times. Figure 11 shows that malicious clients have higher malicious indexes. The difference between benign and malicious updates is negligible—some benign clients are regarded as malicious clients by the server—lead to global model performance degradation. Figure 13 shows that the award mechanism enhances the difference between benign updates and malicious updates and reduces the server's misjudgments.

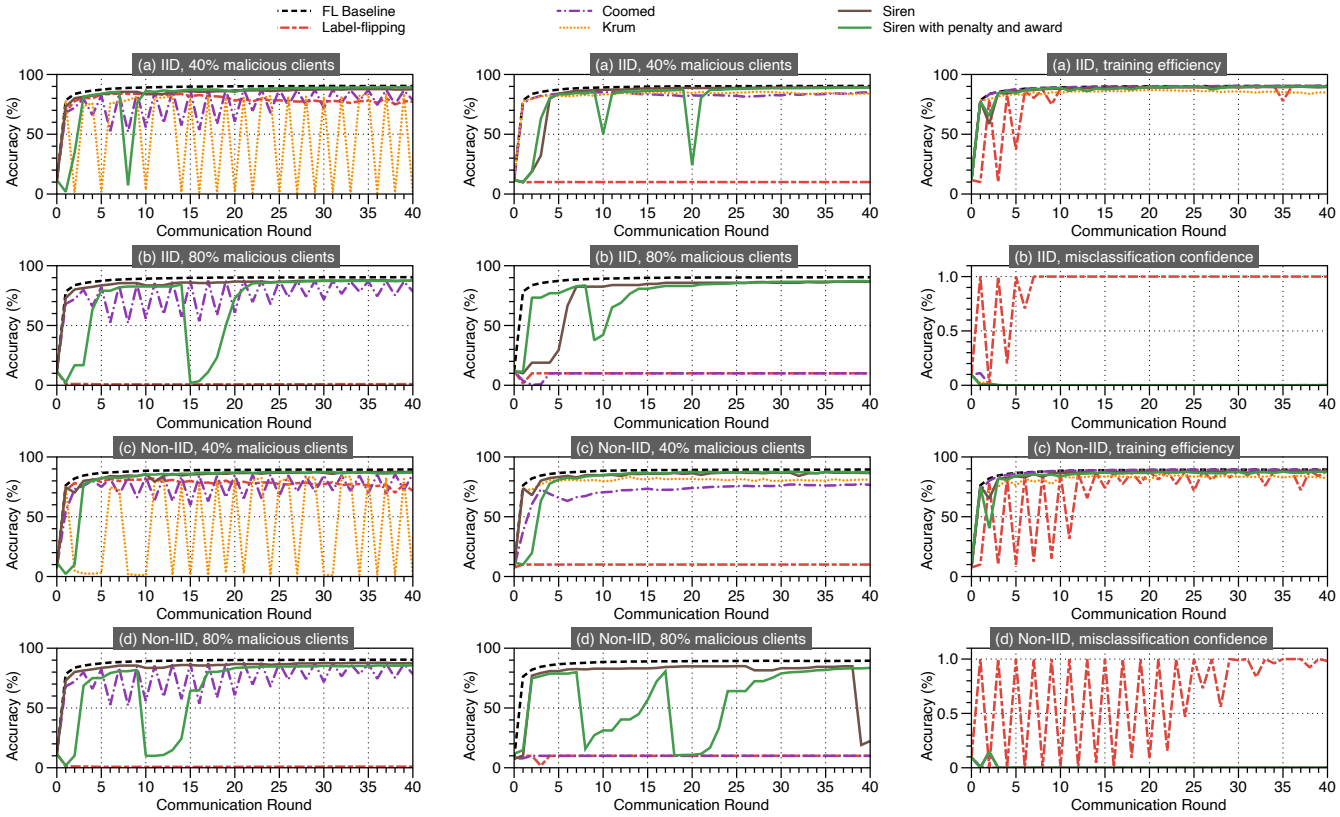


Figure 8: Training efficiency under label-flipping attack when $|K| = 50$.

Figure 9: Training efficiency under sign-flipping attack when $|K| = 50$.

Figure 10: Training efficiency and misclassification confidence under targeted model poisoning, $|K| = 50$.

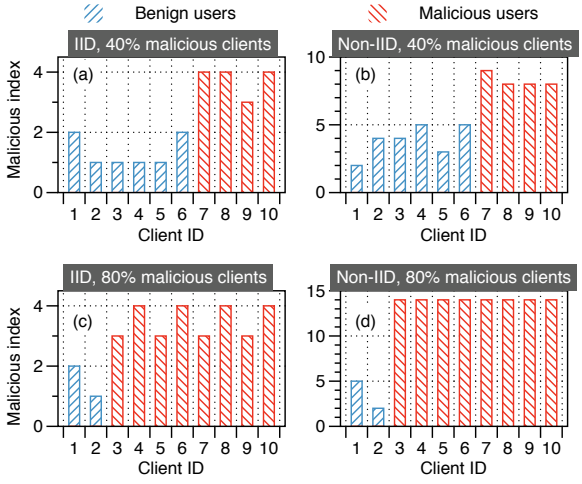


Figure 11: Malicious index of each client on the server under sign-flipping attack when $|K| = 10$ using SIREN.

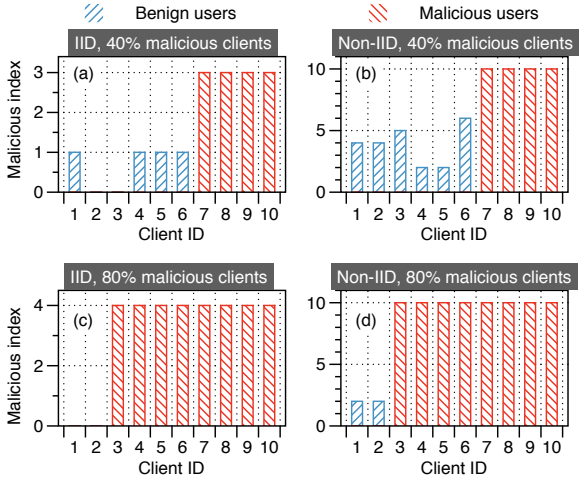


Figure 12: Malicious index of each client on the server under label-flipping attack when $|K| = 10$ using SIREN.

5.4 Defending Label-flipping Attack

Figure 5 shows the accuracy of FL under the label-flipping attack with 40% and 80% of clients in the system as malicious clients when $|K| = 10$, respectively. Since we do not use a boosting factor (which is -4 in the experiments about the sign-flipping attack) to boost the malicious weight updates, it is harder for the defense methods to detect malicious clients, especially for those methods only based on weight analysis. Similar to Section 5.3, the accuracy drops with an increasing proportion of malicious clients in the system, and Krum does not work when the proportion of malicious clients reaches 80%. As shown in Figure 5, unlike the results in Section 5.3,

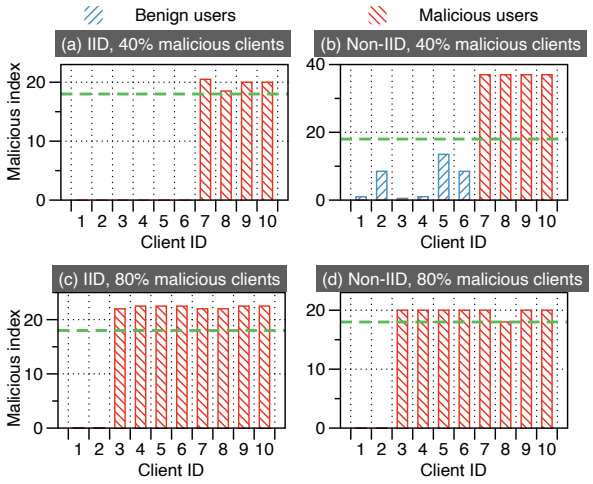


Figure 13: Malicious index of each client on the server under sign-flipping attack when $|K| = 10$ using SIREN with the penalty and the award mechanism (The green dash line represents C_p).

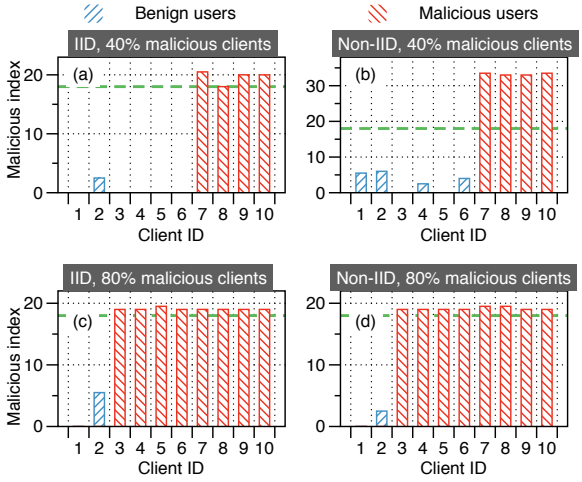


Figure 14: Malicious index of each client on server under label-flipping attack when $|K| = 10$ using SIREN with the penalty and the award mechanism (The green dash line represents C_p).

both Krum and coordinate-wise median fail to effectively defend the label-flipping attack when the proportion of malicious clients is 40% on both IID and non-IID data. However, SIREN protects the global model and keeps the training going free of attacks. With the proportion of malicious clients approaching 80%, the accuracy of the global model trained by coordinate-wise median is reaching 0 while the global model trained by SIREN steadily keeps an accuracy of more than 85%. Compared with SIREN without weight analysis, a complete SIREN is more stable like the results in Section 5.3.

We also visualize the malicious indexes of each client in Figure 12. Similar to the results in Figure 11, SIREN can successfully distinguish malicious clients from all the clients when the system is under label-flipping attack since the

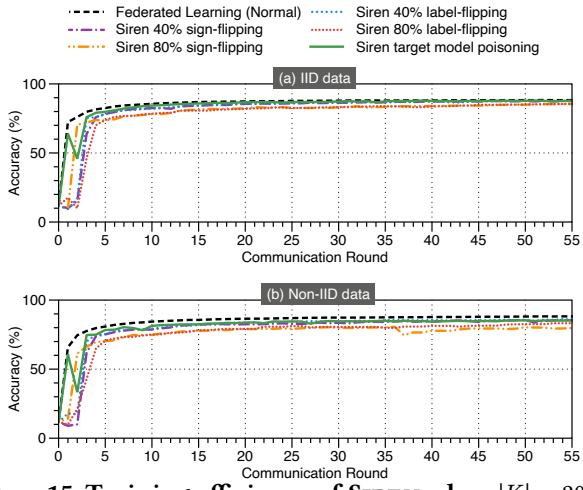


Figure 15: Training efficiency of SIREN when $|K| = 200$.

malicious clients' indexes are much higher than the indexes of benign clients. Figure 14 shows that the award mechanism also improves SIREN's performance.

5.5 Defending Targeted Model Poisoning

Targeted model poisoning is a targeted attacking method. A poisoned model only twists specific predictions, making it much harder to detect than the sign-flipping attack and label-flipping attack. In this case, malicious clients can perform as similar as benign clients do. Figure 7(a) and Figure 7(b) show the accuracy curves of our method, Krum and coordinate-wise median when the training data is IID and non-IID when $|K| = 10$. Both coordinate-wise median and SIREN perform well, and Krum fails, aligned to the original paper of targeted model poisoning [5]. Since targeted model poisoning can attack the global model while letting it perform normally, we continue to analyze the misclassification confidence of the global model, illustrated in Figure 7(c) and Figure 7(d).

5.6 Scalability: Experiments with 50 and 200 Clients

After testing SIREN over IID and non-IID training data with 10 clients in total, we increase the total number of clients to 50 and explore the scalability of SIREN on both IID and non-IID training data.

We first compare the accuracy of FL using SIREN, Krum, and coordinate-wise median under sign-flipping attack with 40% and 80% malicious clients, as illustrated in Figure 9. With 40% malicious clients, all the three defense methods successfully protect the training process, and SIREN achieves the best performance, close to the FL baseline. However, when the proportion of malicious clients is 80%, only SIREN can defend the global model. Neither of Krum and coordinate-wise median works effectively.

Then, we compare the accuracy of FL using these three methods under label-flipping, which is shown in Figure 8.

With 40% malicious clients, Krum and coordinate-wise median fail to defend the attack, while Siren still works. When the proportion of malicious clients is 80%, only our method can protect the system successfully.

Figure 10 presents the result of the comparison under targeted model poisoning. As the results of the experiments shown in Section 5.5, Siren can also defend the targeted model poisoning with a larger number of clients.

Figure 15 presents the experiments with 200 clients. The number of clients in the system reaches 200, and the amount of each client's data is already quite small, though Siren can still defend the system from various attacks and maintain an ideal performance close to the baseline.

5.7 Generality: Experiments on CIFAR-10 Dataset

We further explore the performance of Siren when FL is performed over CIFAR-10 [14] dataset. We compare Siren with Krum and coordinate-wise median under sign-flipping attack and label-flipping attack while using ten clients over IID distributed training data. Table 3 shows that no matter the attack type and the proportion of malicious clients, Siren can always get the best performance compared with Krum and coordinate-wise median. The overall low accuracy of all cases is because the model we used is quite simple.

We also evaluate targeted model poisoning using the CIFAR-10 dataset with a more powerful three-CNN-layer model with more kernels. According to the result, both Krum and coordinate-wise median fail to protect the system since the performance of the model degrades severely with Krum and the misclassification confidence converges to 1. Though the model trained by coordinate-wise median can reach the accuracy of 67.69%, the misclassification confidence keeps equalling to 1. However, Siren defends the attacks and trains the global model to achieve an accuracy of 65.47%.

Table 3: Training efficiency over CIFAR-10 using IID data distribution when $|K| = 10$.

Attack Type	Defense Methods	Malicious Proportion	Accuracy
None	None	0%	55.33%
Sign-flipping	None	40%	10.01%
		80%	10.01%
	Krum[1]	40%	41.53%
	Coomed	40%	34.25%
		80%	9.99%
	Siren	40%	**51.58%**
		80%	**45.50%**
Label-flipping	None	40%	34.70%
		80%	11.68%
	Krum[1]	40%	44.72%
	Coomed	40%	37.21%
		80%	9.60%
	Siren	40%	**49.82%**
		80%	**43.52%**

[1] Krum cannot work properly when malicious clients' proportion reaches 80%.

5.8 Efficiency Analysis

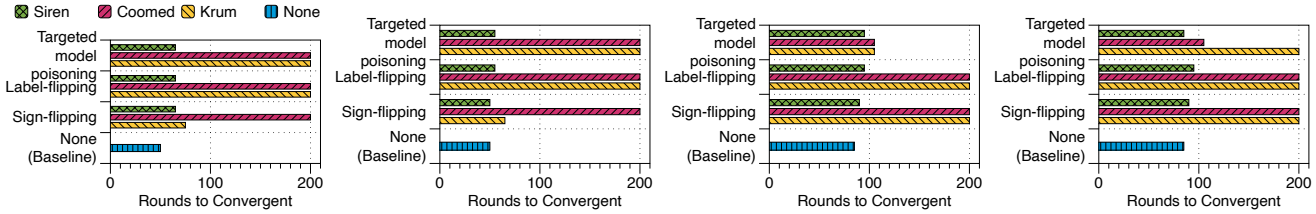
Compared with FedAvg, in each round, Siren causes extra local testing on clients, while since the local test dataset is quite small, clients do not need to spend too much time on it. Besides, Siren also requires clients to send alarm states to the server in each round. Since the alarm state is only one digit (or several digits when encrypted), the communication overheads between clients and the FL server won't increase much. On the server, Siren needs extra storage on the FL server for tracking global models and alarms. In each round, if the FL server receives an alarm, it will spend more time and resources to filter malicious clients by using accuracy checking and weight analysis. Besides, Figure 16 shows the rounds needed by each method to let the global model be convergent over the Fashion-MNIST dataset. The results show that compared with Krum and coordinate-wise median, Siren can let the global model be convergent using fewer communication rounds. And the rounds required by Siren are similar to FedAvg, which also means that Siren do not incur much more communication overheads between clients and the FL server.

5.9 Recovering the Training from Attacks

Unlike other prevailing defensive methods, Siren has a unique new feature that Siren can restore the training process and the global model, even though the server has been compromised successfully. To illustrate this, we design several experiments under a scenario that in the first 10 rounds, the server does not use any defensive methods. After the 10^{th} round, the server starts to use defensive methods. According to the results shown in Figure 17, only Siren can successfully restore the global model and finally gets a normal global model.

5.10 Adaptability to More Attacks

According to the experimental results, Siren can defend the FL system against various attacks, no matter how these attacks camouflage themselves. However, due to the space limitation, there are still several attacks not included in the experiments, such as adaptive attacks [12], and Siren also has the ability to defend the FL system against them. To poison the global model, current attacks design various malicious local updates. Specifically, these modifications on local updates can be reflected by either the property of the updates or the performance of the updated models. Current attacks can hardly camouflage themselves from both aspects at the



(a) $|K| = 10$, IID data. (b) $|K| = 10$, Non-IID data. (c) $|K| = 50$, IID data. (d) $|K| = 50$, Non-IID data.

Figure 16: Communication rounds needed for being convergent under different scenarios. If the rounds reach 200, it means that the global model is attacked successfully or cannot achieve accuracy of 85%.

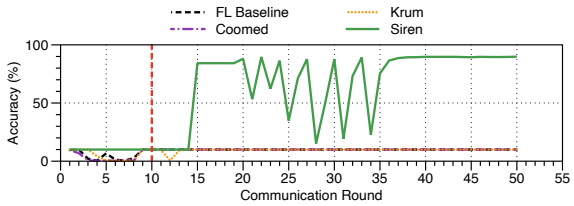


Figure 17: Training efficiency under sign-flipping attack when $|K| = 10$, 40% malicious clients and the defense delay is 10 rounds.

same time. Based on this observation, SIREN is designed to monitor both potential modifications. Therefore, SIREN can defend potential new attacks as long as the attacks attempt to degrade the global model's performance.

6 RELATED WORK

FL is motivated by data privacy concerns. However, due to its distributed nature, FL is rather vulnerable and can be attacked by different kinds of methods. Many studies on attacks and defenses of FL have been emerging in recent years.

Attacks: Compromised clients aim to mislead the global model by sending malicious model updates to the FL server. Most existing attacks focus on degrading the whole performance of the global model for testing dataset [12, 17, 27], referred as untargeted attack. Another type of Byzantine attack is known as targeted attack [3, 5, 21, 23], seeking to poison the global model on some specific data examples in a targeted manner while maintaining good performance on the global model for the rest of the data. Recently, adaptive attacks [12] are proposed to update the malicious weight to the inverse direction of correct global model updates and apply optimization to seduce the FL server to pick the malicious weight updates, making it challenging for existing Byzantine-robust frameworks to defend. However, since adaptive attacks extensively reduce the global model accuracy, and the malicious weight updates' directions are very different from correct one's, SIREN clients can firstly detect such attacks by using the accuracy comparison, and the server can also easily recognize malicious updates by using accuracy checking and weight analysis.

Defenses: FL Byzantine-robust methods mainly focus on proposing new aggregation mechanisms that alleviate the negative effects of Byzantine attacks. Since the median is a robust estimator to be well applied to defend the Byzantine attacks, many studies have adopted and improved this strategy [6, 8, 10, 27, 32, 33]. However, existing Byzantine-robust methods suffer from a few practical issues, such as false alarms triggered by non-IID data and vulnerabilities to a large proportion of malicious clients. Zeno [29] effectively addresses weaknesses of previous majority-based methods using a stochastic first-order oracle to grade each client's update and aggregate updates with high scores. However, it also only uses weight analysis to defend the system. FLTrust [7] calculates a trust score for each client model updates based on the cosine similarity between the server's model updates and clients' updates. Then, the FL server uses the trust scores and clients' weight updates to update the global model. Compared with FLTrust that only uses weight updates to recognize malicious clients, SIREN jointly applies accuracy checking and weight analysis on the server. Besides, SIREN crafts a proactive alarming mechanism that orchestrates all participating clients and the FL server to defend themselves from attacks.

7 CONCLUSION

This paper proposes SIREN, a proactive attack-agnostic defense system for federated learning. Unlike existing defense systems based on the analysis of weight updates, SIREN is based on accuracy checking and can defend against crafted attacks that are hard to detect for current Byzantine-robust aggregation rules. SIREN creatively distributes detecting processes to client ends, making it possible to defend FL from all types of attacks in real-world scenarios, *e.g.*, a large portion of clients is malicious. Extensive experiments with different attack methods on IID and non-IID data prove the effectiveness of SIREN, compared with other state-of-the-art defense methods, such as Krum and coordinate-wise median.

8 ACKNOWLEDGEMENTS

This work is partially funded by the National Natural Science Foundation of China (NO. 61872234, 61732010), Shanghai Key Laboratory of Scalable Computing and Systems. Ruhui Ma is the corresponding author.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. 2018. Byzantine Stochastic gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to Backdoor Federated Learning. In *International Conference on Artificial Intelligence and Statistics (ICAIS)*.

[4] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2018. Model Poisoning Attacks in Federated Learning. In *NeurIPS Workshop on Security in Machine Learning (SecML)*.

[5] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin B. Calo. 2019. Analyzing Federated Learning through an Adversarial Lens. In *International Conference on Machine Learning (ICML)*.

[6] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[7] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2020. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. *arXiv preprint arXiv:2012.13995* (2020).

[8] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. Provably Secure Federated Learning against Malicious Clients. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[9] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. 2018. DRACO: Byzantine-resilient Distributed Training via Redundant Gradients. In *International Conference on Machine Learning (ICML)*.

[10] Yudong Chen, Lili Su, and Jiaming Xu. 2017. Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*.

[11] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. 2018. The Hidden Vulnerability of Distributed Learning in Byzantium. In *International Conference on Machine Learning (ICML)*.

[12] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2020. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *USENIX Security Symposium (USENIX Security)*.

[13] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, et al. 2016. Federated Learning: Strategies for Improving Communication Efficiency. In *NeurIPS Workshop on Private Multi-Party Machine Learning (NeurIPS Workshop)*.

[14] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning Multiple Layers of Features from Tiny Images. *Tech Report* (2009).

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TPLS)* 4, 3 (1982), 382–401.

[16] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. 2017. Speeding Up Distributed Machine Learning Using Codes. *IEEE Transactions on Information Theory (TIT)* 64, 3 (2017), 1514–1529.

[17] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. 2019. Rsa: Byzantine-robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[18] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, et al. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[19] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. 2020. Learning to Detect Malicious Clients for Robust Federated Learning. *arXiv preprint arXiv:2002.00211* (2020).

[20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine (SPM)* 37, 3 (2020), 50–60.

[21] Lingjuan Lyu, Han Yu, and Qiang Yang. 2020. Threats to Federated Learning: A Survey. *arXiv preprint arXiv:2003.02133* (2020).

[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics (ICAIS)*.

[23] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. 2019. Can You Really Backdoor Federated Learning? *arXiv preprint arXiv:1911.07963* (2019).

[24] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data Poisoning Attacks Against Federated Learning Systems. *arXiv preprint arXiv:2007.08432* (2020).

[25] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[26] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2019. DBA: Distributed Backdoor Attacks against Federated Learning. In *International Conference on Learning Representations (ICLR)*.

[27] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2018. Generalized Byzantine-tolerant SGD. *arXiv preprint arXiv:1802.10116* (2018).

[28] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2020. Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation. In *Uncertainty in Artificial Intelligence (UAI)*.

[29] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance. In *International Conference on Machine Learning (ICML)*.

[30] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, et al. 2015. Petuum: A New Platform for Distributed Machine Learning on Big Data. *IEEE Transactions on Big Data (TBD)* 1, 2 (2015), 49–67.

[31] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[32] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2019. Defending Against Saddle Point Attack in Byzantine-robust Distributed Learning. In *International Conference on Machine Learning (ICML)*.

[33] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter L. Bartlett. 2018. Byzantine-robust Distributed Learning: Towards Optimal Statistical Rates. In *International Conference on Machine Learning (ICML)*.

[34] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, et al. 2018. Federated Learning with Non-IID Data. *arXiv preprint arXiv:1806.00582* (2018).