# BoBa: Boosting Backdoor Detection through Data Distribution Inference in Federated Learning

**Zhengyuan Jiang[a,1], Xingyu Lyu[b,1], Shanghao Shi[c], Yang Xiao[d], Yimin Chen[b], Y. Thomas Hou[c], Wenjing Lou[c] and Ning Wang[a,*]**

[a]University of South Florida, USA
[b]University of Massachusetts, Lowell, USA
[c]Virginia Polytechnic Institute and State University, USA
[d]University of Kentucky, USA

**Abstract.** Federated learning, while being a promising approach for collaborative model training, is susceptible to backdoor attacks due to its decentralized nature. Backdoor attacks have shown remarkable stealthiness, as they compromise model predictions only when inputs contain specific triggers. As a countermeasure, anomaly detection is widely used to filter out backdoor attacks in FL. However, the non-independent and identically distributed (non-IID) data distribution nature of FL clients presents substantial challenges in backdoor attack detection, as the data variety introduces variance among benign models, making them indistinguishable from malicious ones.

In this work, we propose a novel distribution-aware backdoor detection mechanism, BoBa, to address this problem. To differentiate outliers arising from data variety versus backdoor attacks, we propose to break down the problem into two steps: clustering clients utilizing their data distribution, and followed by a voting-based detection. We propose a novel data distribution inference mechanism for accurate data distribution estimation. To improve detection robustness, we introduce an overlapping clustering method, where each client is associated with multiple clusters, ensuring that the trustworthiness of a model update is assessed collectively by multiple clusters rather than a single cluster. Through extensive evaluations, we demonstrate that BoBa can reduce the attack success rate to lower than 0.001 while maintaining high main task accuracy across various attack strategies and experimental settings.

## 1 Introduction

Federated learning (FL) is gaining popularity for its privacy advantage for users over traditional centralized machine learning [20, 25, 19], with applications including Android Gboard for next-word prediction [17] and WeBank's credit risk control [37]. While the distributed nature preserves the data privacy of individual clients, it creates opportunities for malicious clients to *backdoor* the global FL model [4, 39, 35]. Backdoor attacks aim to direct the model to make a desired prediction on targeted inputs while maintaining a relatively high accuracy on non-targeted inputs.

***The Non-IID Data Challenge.*** The non-IID data in real-world FL systems further complicates the backdoor detection task. Most current defense mechanisms designed for IID data heavily rely on the assumption that a distinction exists between malicious model updates and benign model updates in a certain feature space [31, 12, 24]. Such solutions analyze and extract the most distinguishable features, e.g., the angular distance of gradient updates [12], low-dimensional embeddings of model parameters [24], model accuracy on generated data [43], etc. However, as illustrated in the top left figure (labeled as original) of Figure 1, we observe that malicious models are not differentiable from benign models in non-IID scenarios, which is similarly found in [28, 2, 6]. After applying BoBa, clients are divided into multiple clusters. Within each cluster, malicious clients are obviously distinct from benign models.

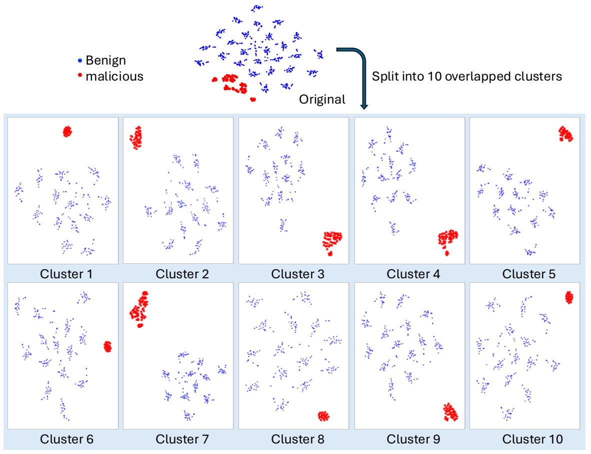

**Figure 1.** TSNE of client models in non-iid data scenario without clustering and with clustering (BoBa).

Clustering method has been explored by multiple existing works. Ghosh et al. [13] propose clustering clients based on their empirical risk value and then using trimmed mean [41] within each cluster to filter out possible Byzantine nodes. Rieger et al. [28] suggest employing an ensemble of clustering algorithms to effectively cluster model updates with similar training data. [29] proposes a stacked clustering method, that selects the most representative voting vector from all submissions. Existing works utilize model weights for clustering without considering the data distribution differences. To cleanse the received model updates, the server will inevitably remove novel benign clusters from the final aggregation. It is a sacrifice of efficiency/utility for security. Our method aims to better balance utility

* Corresponding Author. Email: ningw@usf.edu.
[1] Equal contribution.

and security by reducing the probability of removing novel benign clients.

To achieve the goal, we propose `BoBa`, a **Bo**oster for **Ba**ckdoor attack detection. `BoBa` introduces a novel clustering strategy based on a data distribution inference module, dubbed DDIG—*Data Distribution Inference from Gradients*. DDIG can accurately estimate the relationship between gradients and label distribution theoretically and empirically. Based on the label distribution, we cluster the clients into multiple groups. In each cluster, clients hold similar data distributions. We successfully transformed the challenging backdoor detection in a non-IID scenario into an IID case. Further, we identify that a naive clustering method is vulnerable to malicious collisions where multiple attackers obtain the same distribution and dominate one cluster. We propose two constraints, *uniform cluster inclusion* and *balanced cluster*, to decrease the impact of collusion attack. Our system is a booster as we can attach the proposed clustering method to various traditional backdoor detection mechanisms designed for IID scenarios.

In summary, we make the following contributions:

- We propose `BoBa` to address the challenging problem of backdoor detection for FL systems in non-IID data scenarios. To distinguish backdoored models from benign ones in non-IID data scenarios, we've developed a two-step detection method involving client clustering by data distribution and subsequent detection of malicious model updates within each cluster.
- We find out that the lack of knowledge of client data distribution is a key reason for the low accuracy of backdoor detection in non-IID scenarios. We propose a novel data distribution inference model–DDIG, to address this challenge, which has significantly improved the detection performance.
- We propose an overlapping clustering method to improve the robustness of backdoor attack detection. We model clustering as an optimization problem and address these challenges by incorporating balanced cluster and uniform cluster inclusion into the objective function.
- Our evaluation showcases the superiority of `BoBa` over other baseline methods, as it consistently achieves a lower attack success rate (ASR) across various attack strategies and non-IID levels on multiple datasets.

## 2 Background and Related Work

### *2.1 Federated Learning*

In FL systems, there are two entities, one parameter server (PS), and $n$ clients (we define $[n] := \{1, 2, ..., n\}$). Each client manages a local dataset $\mathcal{D}_i$ following non-identical distributions. We use $m$ to represent the total number of data classes. We assume that each client manages a local model, and the model parameter of client $i$ is denoted by $\theta_i \in \mathcal{W} \subseteq \mathbb{R}^d$, wherein $\mathcal{W}$ is the parameter space and $d$ is the presumed model dimensionality. The global model parameter is denoted by $\Theta \in \mathcal{W}$. We denote the model update from client $i$ as $\delta_i = \theta_i - \Theta$. PS acts as the model distributor and aggregator on the cloud side. The entropy-based loss value for a local model is $\mathcal{L}(\theta_i, \mathcal{D}_i)$. The total loss function $\mathcal{F}(\Theta)$ is calculated using the loss of the $k$ selected clients, which can be formulated as

$$\mathcal{F}(\Theta) := \sum_{i=1}^{k} \mathcal{L}(\theta_i, \mathcal{D}_i) \tag{1}$$

The goal of the FL system is to jointly minimize the loss $\mathcal{F}(\Theta)$ by optimizing the global model parameters $\Theta$.

### *2.2 Poisoning Attacks in FL*

In FL, since attackers are able to manipulate both data and models, a poisoning attack in FL is referred to as a model poisoning attack (MPA) [8, 33, 16, 10, 4]. Attackers carefully manipulate model parameters, with the aim of gradually degrading the FL model efficacy without being detected. Based on the goals of attackers, MPA can be categorized into two classes: untargeted attacks that aim at increasing the overall prediction error [10] and backdoor attack/targeted attacks that manipulate the prediction on targeted inputs [4, 3]. We focus on the far more stealthy backdoor attacks in this work. Backdoor attacks have advanced their performance by enhancing stealthiness and sophistication [39, 34, 4, 3].

**BadNet**[15] serves as a foundational attack, illustrating straightforward trigger injection without model-level manipulations. **Alternate**[4] alternates between optimizing classification accuracy and minimizing deviation from benign models to achieve stealthy yet effective backdoor insertion. **DBA**[39] decomposes and distributes global triggers across clients to enhance persistence and concealment. **Sybil** [12] leverages multiple colluding malicious clients sharing the same crafted update to amplify the backdoor's influence. **Neurotoxin** [42] exploits parameters with minimal updates during training to maintain backdoor persistence. **IBA** [27] advances further by embedding stealthy and irreversible backdoors, offering strong resistance to existing defenses. These attacks span simple injection, optimization-based stealth, distributed triggering, multi-client collusion, minimal-update persistence, and irreversible embedding, providing a rigorous testbed for assessing defense mechanisms in adversarial federated learning environments.

### *2.3 Defense against Poisoning Attack*

In the literature, the initial defenses, e.g., *Krum* [5], *Median*, *Trim* [41], and *Bulyan* [16], typically leverage outlier-robust measures to compute the center of updates to filter out Byzantine updates. These mechanisms provide provable resilience against poisoning attacks to some extent. [12] utilizes angular distance of gradient updates for detection. Low-dimensional embeddings extracted from model parameter update are utilized in [24]. [29, 36, 43] explicitly detects and filters abnormal model updates using model representations/accuracy.

Later, trust-based approaches [7, 2, 1] and clustering-based approaches [21, 11, 29] are proposed to further enhance the defense landscape. **FLTrust** [7] trains a server model using an auxiliary dataset at PS in each iteration. PS bootstraps a trust score for each client based on its directional deviation from the server model update. **CONTRA** [2] detects malicious clients by evaluating the alignment of any two clients' gradients. Malicious client pairs have larger alignments than other pairs. CONTRA implements a cosine similarity-based measure to determine the credibility of local model parameters. **MESAS** [21] used a multi-faceted detection strategy, statistical tests, clustering, pruning, and filtering for backdoor detection. **FreqFed** [11] tackled backdoor attacks in a unique way where it transforms model updates into the frequency domain and identify unique components from malicious model updates. **AGSD** [1] utilized a trust index history with FGSM optimization to resist backdoor attacks. Other types of defenses mitigate poisoning attacks by suppressing or perturbing model updates. Xie et al. [40] propose to apply clip and smoothing on model parameters to control the global model smoothness, which yields robustness certification on backdoors. FLAME [26] adds noise to eliminate backdoors based on the concept of differential privacy. To the best of our knowledge, we are the first to transform the challenging backdoor mitigation in a non-IID scenario into an IID case.

## 3 System Setting and Threat Model

In the FL system, at the system onset, PS initializes $\Theta$. Then each training iteration works as follows: (1) PS first selects multiple clients and sends $\Theta$ to them. (2) Each of the selected clients, initializes $\theta_i = \Theta$ and trains the model with its local data and provides its model update $\delta_i$ to PS. PS infers an abstract data distribution for each client, followed by overlapping clustering, feature extraction, and vote-based trust estimation. (3) PS aggregates local model updates weighted by their trust scores and updates the global model.

We consider the data among clients to be non-IID. While the meaning of IID is generally clear, data can be non-IID in many ways. We consider differences in the data label distribution on each client, *label distribution skew*, which is mostly discussed in FL[2, 10, 30, 19]. For example, when clients are tied to particular geo-regions, the distribution of labels varies across clients — kangaroos are only in Australia or zoos; a person's face is only in a few locations worldwide; for mobile device keyboards, certain emoji are used by one demographic but not others [19].

**Threat Model:** Attackers' primary goal is to lead the global model to classify inputs as the target labels, while simultaneously maintaining a relatively high overall accuracy to avoid being detected. Attackers have access to a dataset to train their local model. We assume they can arbitrarily manipulate their local data, which may involve injecting triggers into input data and altering labels. Additionally, they can also manipulate their model parameter updates sent to PS directly. As legitimate clients, attackers have white-box access to the global model but not to other client models. We assume PS is trusted and the defense mechanism against the poisoning attack is deployed at PS.

## 4 Technical Design of BoBa

### 4.1 Challenges Identification

In non-IID scenarios, discrepancies in model updates can arise from either inherent variations in data distributions or malicious manipulations. Determining the root cause of these model anomalies presents a formidable challenge. To tackle this problem, we propose a two-step solution. The first step involves clustering clients based on their data distribution, and the second step focuses on applying a voting-based detection methodology within each cluster. We have identified two challenges:

- **Data Distribution Inference**: With only the knowledge of model updates $\delta_i$ provided by clients, it is challenging for the defender to estimate the data distribution of clients.
- **Detection Challenge**: Traditional clustering methods cannot defend against collisions where attackers obtain the same dataset to be clustered in one cluster.

### 4.2 DDIG: Data Distribution Inference via Gradients

To solve the first challenge, we propose DDIG to extract label distributions from gradients/updates.

We consider a classification task with the cross-entropy loss. Without loss of generality, we denote the gradient produced by a client in its local training process as $\nabla\theta = \frac{1}{|\mathcal{D}|}\sum_j \nabla_\theta \mathcal{L}(x_j, y_j)$, where $|\mathcal{D}|$ is the number of data samples in $\mathcal{D}$ and $x_j, y_j \in \mathcal{D}$ are sample/label pairs.

**Theorem 1.** *For each input pair $x_j, y_j$, the gradient of the logits layer $\mathbf{z} \subseteq \mathbb{R}^m$ (pre-softmax layer) is $\nabla_{\mathbf{z}}\mathcal{L}(x_j, y_j) = \mathbf{p}_j - \mathbf{y}_j$, where $\mathbf{p}_j \subseteq \mathbb{R}^m$ is a post-softmax probability vector and $\mathbf{y}_j \subseteq \mathbb{R}^m$ is a binary label vector with only the correct label class values 1. As all $p_j^s \in \mathbf{p}_j$ are within $[0,1]$, $\nabla_{\mathbf{z}}\mathcal{L}(x_j, y_j)$ will have a negative value only on the "correct label class" element.*

The parameter server has no access to $\nabla_{\mathbf{z}}\mathcal{L}(x_j, y_j)$ but only the gradient. For the last linear layer $W \subseteq \mathbb{R}^{m\times r}$, where $r$ is the input feature number (i.e., the output dimension of the previous layer), the gradient of its element $W_{s,t}$ is:

$$\nabla W_{s,t} = \sum_{x_j, y_j} \frac{\partial \mathcal{L}(x_j, y_j)}{\partial z_s} \frac{\partial z_s}{\partial W_{s,t}} = \sum_j (p_j^s - y_j^s) o_j^t \qquad (2)$$

where $o_j^t \in \mathbf{o}_j \subseteq \mathbb{R}^r$ are the inputs to the final layer, and $\mathbf{o}_j$ is non-negative when ReLU activation function is used in the penultimate layer. Therefore, for each $x_j, y_j$, $(p_j^s - y_j^s)o_j^t < 0$ still holds if and only if $s$ corresponds to the correct label class. We then define an indicator $I_s \subseteq \mathbb{R}^m$ for each label class ($s \in [m]$):

$$I_s = -\sum_{t=1}^{r} \nabla W_{s,t} = \sum_j (y_j^s - p_j^s) \sum_{t=1}^{r} o_j^t \qquad (3)$$

where $\sum_{t=1}^{r} o_j^t$ does not change with respect to $s$. Therefore, the value of $I_s$ is mainly determined by the number of samples for class $s$ in $\mathcal{D}_i$. And peaks in vector $\mathbf{I} = [I_1, I_2, ..., I_m]$ indicates more samples in the corresponding class index.

The previous method also applies for an accumulated gradient. If we assume the clients uses the SGD optimizer to train the local model for $H$ rounds before sending the updates, the indicator vector $\mathbf{u}$ accumulates with respect to training rounds. The previous property — the larger the elements in $\mathbf{I}$, the larger the number of samples the corresponding classes still holds.

### 4.3 Data Distribution Representation

We introduce a definition of **Abstract Data Distribution** to represent data distribution. We use $\mathcal{A}_{ij} \in \{0, 1\}$ as an indicator for relative data sufficiency. $\mathcal{A}_{ij} \leftarrow 1$ indicates that client $j$ has *sufficient* [2] data for class $i$. A matrix $\mathcal{A} \in \{0,1\}^{m\times n}$ is used to represent the abstract data distribution information of all clients.

For each data class, a portion of clients have sufficient data due to clients' various functional environments. The number of clients having sufficient data for class $i \in [m]$ is denoted by $n_i$. Further, each client $j \in [n]$ has a subset of the $m$ classes, and their class numbers are denoted by $m_j$ as shown in Figure 2. The value satisfies $(n_i \le n), \forall i \in [m]$ and $(m_j \le m), \forall j \in [n]$.

### 4.4 Client Clustering

#### 4.4.1 Naive Clustering

A traditional clustering method, e.g., k-means, takes each column of $\mathcal{A}$ (e.g., one client) as input and outputs multiple groups. The results heavily depend on the selection of the number of clusters. A naive way to eliminate the dependence of an adaptively selecting a cluster

[2] Data sufficiency is dataset specific and needs to be tuned based on the task complexity.

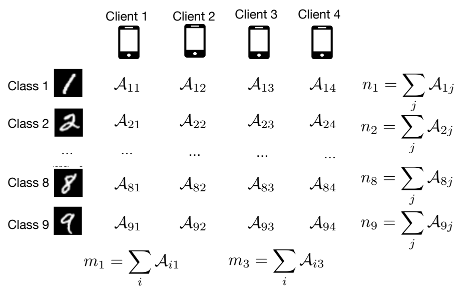
Client 1
Client 2
Client 3
Client 4
Class 1
Class 2
Class 8
Class 9
$\mathcal{A}_{11}$ $\mathcal{A}_{12}$ $\mathcal{A}_{13}$ $\mathcal{A}_{14}$ $n_1 = \sum_j \mathcal{A}_{1j}$
$\mathcal{A}_{21}$ $\mathcal{A}_{22}$ $\mathcal{A}_{23}$ $\mathcal{A}_{24}$ $n_2 = \sum_j \mathcal{A}_{2j}$
$\mathcal{A}_{81}$ $\mathcal{A}_{82}$ $\mathcal{A}_{83}$ $\mathcal{A}_{84}$ $n_8 = \sum_j \mathcal{A}_{8j}$
$\mathcal{A}_{91}$ $\mathcal{A}_{92}$ $\mathcal{A}_{93}$ $\mathcal{A}_{94}$ $n_9 = \sum_j \mathcal{A}_{9j}$
$m_1 = \sum_i \mathcal{A}_{i1}$
$m_3 = \sum_i \mathcal{A}_{i3}$

**Figure 2.** $\mathcal{A}_{ij}$ is an indicator of client $j$'s data sufficiency in class $i$: 1 for sufficient data and 0 for non-sufficient data.

number is to use the number of data classes $m$. In the naive clustering, we assign clients who possess sufficient data for data class $i \in [m]$ to cluster $i$ by default. It is worth noting that a client may belong to multiple clusters if they have sufficient data for multiple classes. But the naive clustering cannot deal with collusion, where malicious clients try to be clustered in one large group to evade detection (Neither can traditional one). To solve this problem, we introduced balanced cluster and uniform cluster inclusion.

#### *4.4.2 Balanced Cluster & Uniform Cluster Inclusion*

We propose a balanced clustering scheme that features a random selection of clients in each data class, instead of including all clients with sufficient data in the cluster. The randomness can decrease the likelihood that malicious clients to form a cluster and dominate the detection. But we have not fully solved the problem. Malicious clients can collude in multiple clusters by showing sufficient data in each of the clusters. To solve this problem, we proposed a uniform cluster inclusion, meaning that, each client can participate in an identical number of clusters.

#### *4.4.3 Detailed Clustering Method*

We use a matrix $\mathbf{x} \in \{0,1\}^{m\times n}$ to represent the clustering result, and $\mathbf{x}_{ij}$ represent client $j$ is in the cluster $i$. We set $\mathbf{x}_{ij} = 1$ if client $j$ is selected to cluster $i$ and $\mathbf{x}_{ij} = 0$ otherwise. We initialize the clustering as the naive cluster method — forming a cluster with all clients having sufficient data for a specific class. In other words, the initialized $\mathbf{x} \in \{0,1\}^{m\times n}$ is equal to data sufficiency matrix $\mathcal{A}_{ij} = 1$.

To meet uniform cluster inclusion and balanced cluster, we then remove some clients from each cluster. The selection can be modeled as converting some elements with value one in matrix $\mathcal{X}$ into zeros and making no change for zero elements. The objective is to maximize the summation of elements in $x_{ij}$ (or minimize 1-$x_{ij}$) while guaranteeing uniform cluster inclusion and balanced clusters. We formulate the problem as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n}(1-\mathbf{x}_{ij}) * \mathcal{A}_{ij}, \\ \text{subject to} \quad & \mathbf{x}_{ij} \in \{0,1\}, \text{ and } x_{ij} \le \mathcal{A}_{ij}, \qquad (4) \\ & \sum_{i=1}^{m}\mathbf{x}_{ij} = m_{th}, \forall j; \text{ and } \sum_{j=1}^{n}\mathbf{x}_{ij} = n_{th}, \forall i, \end{aligned}$$

where $m_{th}$ denotes the maximum allowed cluster size, $n_{th}$ represents the maximum number of clusters one client can participate in. The details about how the two thresholds are decided are in the Appendix. The **first condition** ensures we only select clients with sufficient data. The second line of condition corresponds to the goal of uniform cluster inclusion and balanced cluster discussed above. The optimization is a non-convex optimization problem and by definition is NP-hard [18]. It does not have a polynomial-time closed-form solution for finding the optimal point. Therefore, we use a search-based optimization method to iteratively approximate the best solution.

Our approach begins with initializing the cluster matrix $\mathbf{x}$ as $\mathcal{A}$, as described in Algorithm 1. In the initialization step, each cluster $i$ contains all clients with sufficient data for class $i$. We create balanced clusters by removing clients from each cluster. We proceed by iterating over each cluster (i.e., each row) in the matrix $\mathcal{A}$. If a cluster contains more clients than the specified threshold $n_{th}$, we employ a removal process to balance the cluster. The removal process involves sorting the clients in the cluster based on their data class numbers in descending order (**Lines 5 to 10**). We then remove the client with the highest data class number by changing the corresponding 1 to 0 in matrix $\mathbf{x}$. Following this, we repeat a similar procedure by iterating over each column in $\mathbf{x}$ (**Lines 11 to 16**). This process of iterating and applying removal actions continues until the condition any$(RowCount) > n_{th}$ or any$(ColCount) > m_{th}$ is no longer satisfied.

It is important to note that the greedy algorithm may result in a sub-optimal solution, meaning that the result may not perfectly satisfy perfectly balanced clusters. However, in the context of backdoor attack detection, a sub-optimal solution is sufficient for achieving effective and reliable results. Thus, the proposed greedy algorithm strikes a practical balance between computational efficiency and detection accuracy, making it a valuable component of our overall approach.

### *4.5 Summarized Workflow*

On the **client side**, following the traditional FL process, clients receive global model $\Theta$ from PS and use it to initialize their local model. Each client continues to train the global model using their local data and uploads model update $\delta_i$ to PS.

On the **server side**, the workflow is shown as Figure 3:

- PS first performs DDIG to infer the data distributions of each client.
- PS clusters the clients into overlapped clusters.
- In the evaluation part, we utilized client-wise cosine similarity as a detection method. Instead of obtaining a binary detection result in this step, we obtain the nearest neighbor of each client. We use $K_i$ to represent the number of times that Client $i$ is selected as the nearest neighbor of others. We can also select an existing backdoor detection method (e.g., Krum, FLDetect) and apply it to each of the clusters.
- *Voting-based trust estimation:* The final trust of user $i$ is calculated by $T_i = \frac{exp(K_i)}{\sum_{i=1}^{n} exp(K_i)}$ using its voting counts $K_i$.
- PS aggregated all the received model parameters weighted by their accumulated trust score, which can be represented as $\Theta = \Theta - \lambda \sum_{i=1}^{n} T_i \frac{\delta_i}{\|\delta_i\|}$ where $\lambda$ represents the learning rate, and $\frac{\delta_i}{\|\delta_i\|}$ denotes the normalized gradient of client $i$. Another iteration starts.

## 5 Evaluations

We implement BoBa on the TensorFlow platform and run the experiments on a server equipped with an Intel Core i7-8700K CPU

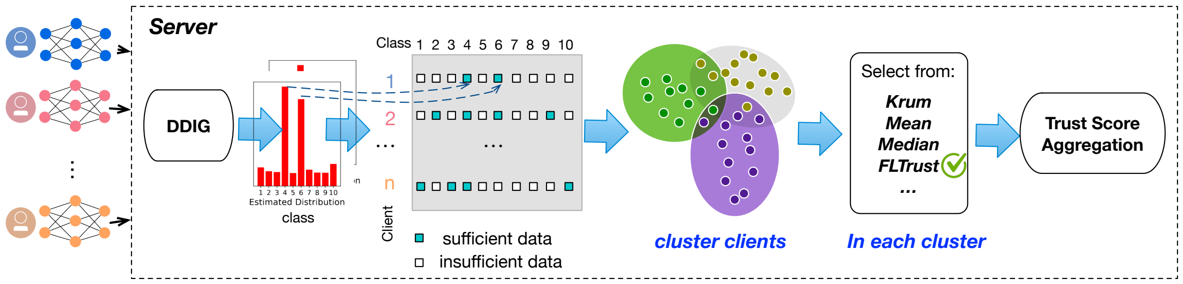

**Figure 3.** Overview of `BoBa`.

3.70GHz×12, a GeForce RTX 2080 Ti GPU, and Ubuntu 18.04.3 LTS.

## 5.1 *Experimental Setting*

**Federated learning settings:** In the studied FL system, we set the client number as $n = 50$ and the per-round selection ratio as 0.2, i.e., 10 clients will be selected in each FL round. To simulate the real-world scenario, we randomize the malicious clients ratio from 0% to 50% of the total selected clients (with an average of 25%) in each round. Each client manages a local model and trains the local model using an Adam optimizer with a learning rate of 0.001. A client trains its local model for five epochs before submitting the model updates. The number of total FL iterations is $T = 60$. We run each experiment *three* times and show the average performance.

**Datasets:** We evaluate `BoBa` on four datasets including three image datasets—fMNIST dataset [38], CIFAR-10 dataset [22], MNIST dataset [9], and one text dataset—Sentiment-140. The model architectures for the three datasets are a plain CNN, VGGNet [32]), AlxeNet [23], and LSTM, respectively.

**MNIST** [9] is a dataset of handwritten digits. It consists of 60,000 training records and 10,000 testing records, each is a 28 × 28 grayscale image. **fMNIST** consists of a training set of 60,000 records and a test set of 10,000 records. Each data record is a 28 × 28 grayscale image, associated with a label from 10 classes, including T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. **CIFAR-10** [22] consists of 60000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10000 test images. Each image is from one of the ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. **Sentiment-140** [14] consists of 1.6 million tweets, annotated with positive or negative sentiments. We adopt the same attack settings on the Sentiment-140 dataset as described in [35]. We synthetically generated the non-IID dataset following the methodology in [25]. Details of this setting are shown in Sec 5.6.

**Backdoor Settings:** We assume malicious clients possess clean data records and poisoned data records. Each client has a dataset of 1,200 (1,000) local training data records in MNIST and fMNIST datasets (CIFAR dataset). For each malicious client, we vary the number of backdoor data used for local training from 1 to 500 (i.e., malicious ratio 0 to around 0.3) to maintain a trade-off between stealthy and attack effectiveness.

## 5.2 *Evaluation Metrics and Baselines*

**Main task accuracy** is evaluated on the whole test dataset, just as an attack-free scenario. **Attack success rate (ASR)** refers to the percentage of samples with trigger patterns that are successfully classified as the target label.

We selected six attacks including BadNet [15], Alternate [4], DBA [39], Sybil [12], Neurotoxin [42], and IBA [27], reflecting a comprehensive spectrum of poisoning strategies in FL. We selected eight defenses, including Krum [5], Median [41], Trim [41], FLTrust [7], CONTRA [2], AGSD [1], MESAS [21], and FreqFed [11] as our core baselines based on their widespread adoption[12, 24, 41, 16].

**Algorithm 1** A Greedy Algorithm for Clustering

**Input**: Abstract of data distribution $\mathcal{A} \in \{0,1\}^{m\times n}$, $m$ indicates the number of classes, $n$ indicates the number of clients.
**Output**: Cluster matrix $\mathbf{x} \in \{0,1\}^{m\times n}$.

```
 1: Set the cluster size $n_{th} = \min(n_1, n_2, ...., n_m)$;
 2: $m_{th} = \text{mean}(m_1, m_2, ..., m_n)$ # one client can participate in
    $m_{th}$ clusters.
 3: $\mathbf{x} \leftarrow \mathcal{A}$ # Initialization
 4: while any$(RowCount) > n_{th}$ or any$(ColCount) > m_{th}$ do
 5:   for $1 \leq i \leq m$ do
 6:     if $RowCount_i > n_{th}$ then
 7:       $MaxColID \leftarrow \text{argmax}(ColCounts)$
 8:       $\mathbf{x}_{i,MaxColID} \leftarrow 0$
 9:     end if
10:   end for
11:   for $1 \leq j \leq n$ do
12:     if $ColCount_j > m_{th}$ then
13:       $MaxRowID \leftarrow \text{argmax}(RowCounts)$
14:       $\mathbf{x}_{MaxRowID,j} \leftarrow 0$
15:     end if
16:   end for
17: end while
```

## 5.3 *Data Distribution and Inference Accuracy*

Figure 4 provides a visual representation of the non-iid data distributions among clients. In the left subfigure, different colors are utilized to indicate distinct data classes, and the height of each color bar represents the corresponding data amount. As observed, clients have **5-7** data classes (**out of 10**) and different data amounts per class.

In Figure 5, we demonstrate the data distribution inference performance. We plot the distribution of the target client with 1200 samples, along with the estimated distribution from the server. The x-axis of the figures refers to different classes, and the y-axis refers to sample numbers. We can observe that the estimated data distribution is almost the same as the real one.

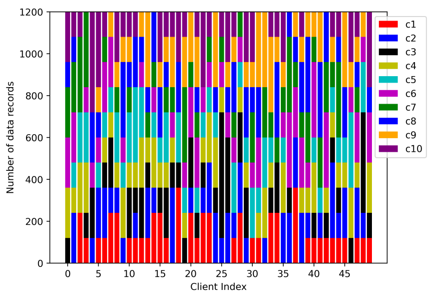

**Figure 4.** The illustration of data distribution. Each client has a subset of the 10 data classes (c1-c10).

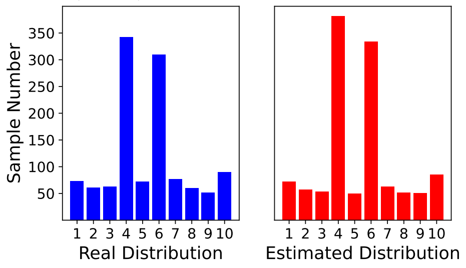

**Figure 5.** Data distribution inference result.

## 5.4 *Overall Performance under Non-IID Data*

TABLE 1 shows the ASRs of six attacks under ten defenses. From the table, we can see that BoBa outperforms other defenses by achieving the lowest ASR. We utilized heatmap color to show the differences among defenses, where green color indicates a better performance. We further present the main task accuracy of the final global model in TABLE 2. We can see that BoBa achieves higher or comparable accuracy when compared with other baselines. In summary, BoBa not only reduces the ASR but also helps maintain a better model accuracy, indicating its superiority in securing FL systems in non-IID scenarios.

To further illustrate the detection performance, we present the trust score distributions of malicious clients in Figure 6. We scale the trust scores for all selected clients to make sure they sum up to one. For decrease the impact of backdoor attacks, ideally, we want to reduce the trust score for malicious clients as close as possible to zero. As we can see from Figure 6, BoBa achieves the lowest average trust scores compared with the other two trust-based defenses—FLTrust and CONTRA, indicating that BoBa has a better detection performance against backdoor attacks.

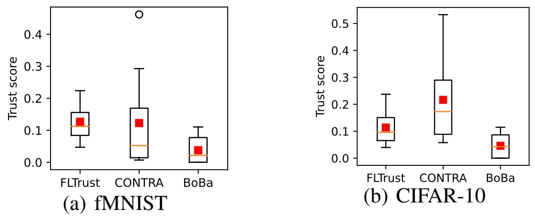

**Figure 6.** Trust scores of **MALICIOUS** clients (the lower the better). Orange lines represent the median and red squares represent the mean.

## 5.5 *Performance under Adaptive Attack*

An adaptive attack refers to an attack that is specifically designed to defeat the proposed defense based on knowledge of the defense strategy. We initialize an adaptive attack against the proposed clustering module. We assume that an adaptive attacker can successfully mislead DDIG by manipulating their gradients. For collusion purposes, they may present data covering all categories to increase their chances of being included in clusters. For better illustrate the effectiveness of adaptive attacks, we record the number of times malicious clients are selected throughout the learning rounds. From Figure 7, we observe that, compared to the base attack (Alternate attack), the adaptive attacker does not increase their chance of being selected. Besides the selected times, we also evaluated the final ASR of the adaptive attack and base attack. Adaptive attack achieves 0.0004 ASR, which shows no obvious increase compared to the non-adaptive attack (ASR = 0.00015). This resilience is due to uniform cluster inclusion, which ensures that each client participates in the same—or a comparable—number of clusters, regardless of their data sufficiency. We acknowledge that there are many other possible adaptive attack strategies, that can be explored in future research.

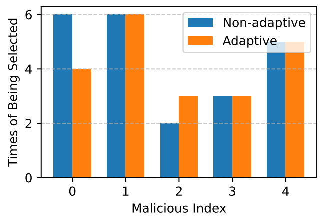

**Figure 7.** Times of malicious client being selected under adaptive attack.

## 5.6 *Robustness against Various Settings*

We first evaluate the performance of BoBa under various malicious client fractions. As shown in Figure 8, the system maintains a high detection performance when the malicious client number is less than 50%. We see the performance degrades promptly when the malicious ratio is larger than 50% which is not shown in this figure. We note that we do not focus on the part where the malicious ratio is larger than 50% as we assume benign clients are the majority.

We evaluate the performance under different non-IID degree and plot the results in Figure 8. As we can see from the figure, ASR is not sensitive to non-IID degree. Here, a value from 0 to 1 is used to indicate the non-IID degree, where 0 indicates IID data and 1 indicates fully non-IID data. Suppose we use $p$ to indicate the non-IID degree, and the total number of data records is $N$. In experiments, we first assign the $(1-p)$ portion of the dataset uniformly to all clients (i.e., IID data). In the second step, we sort the remaining $p * N$ data by digit label, divide it into 1000 shards, and assign each of the 50 clients 20 shards. This is a pathological non-IID partition of the data. We can see that BoBa also achieves a low ASR in the IID data setting and a fully non-IID scenario.

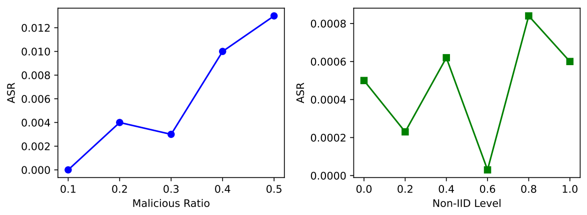

**Figure 8.** ASR under various malicious client ratios and non-IID degrees.

## 5.7 *Computation Overhead*

Compared to traditional FL systems, the server has two additional tasks in BoBa—DDIG, client clustering. DDIG is a deterministic method by directly summing the last layer gradients as shownin

**Table 1.** Backdoor **attack success rate** under defenses (↓).

| Dataset | Attack | FedAvg | Krum | Median | Trim | FLTrust | CONTRA | MESAS | FreqFed | AGSD | `BoBa` |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MNIST | Alternate | 0.122 | 0.645 | 0.047 | 0.103 | 0.201 | 0.363 | 0.700 | 0.257 | 0.172 | 0 |
| | BadNet | 0.494 | 0 | 0.023 | 0.452 | 0.205 | 0.343 | 0.379 | 0.317 | 0.452 | 0 |
| | DBA | 0.020 | 0.135 | 0.002 | 0.018 | 0.021 | 0.170 | 0.158 | 0.100 | 0.008 | 0 |
| | Sybil | 0.117 | 0.666 | 0.048 | 0.125 | 0.232 | 0.107 | 0.009 | 0.048 | 0.031 | 0 |
| | Neurotoxin | 0.255 | 0.210 | 0.420 | 0.783 | 0.230 | 0.300 | 0.057 | 0.120 | 0.115 | 0.005 |
| | IBA | 0.206 | 0.520 | 0.100 | 0.189 | 0.202 | 0.998 | 0.283 | 0.240 | 0.191 | 0.002 |
| fMNIST | Alternate | 0.667 | 0.953 | 0.786 | 0.714 | 0.915 | 0.154 | 0.359 | 0.019 | 0.004 | 0.001 |
| | BadNet | 0.828 | 0.680 | 0.811 | 0.804 | 0.836 | 0.566 | 0.557 | 0.470 | 0.037 | 0 |
| | DBA | 0.452 | 0.071 | 0.428 | 0.539 | 0.507 | 0.342 | 0.017 | 0.032 | 0.010 | 0 |
| | Sybil | 0.678 | 0.971 | 0.813 | 0.736 | 0.448 | 0.039 | 0.026 | 0.103 | 0.053 | 0.001 |
| | Neurotoxin | 0.114 | 0.250 | 0.190 | 0.319 | 0.169 | 0.104 | 0.081 | 0.076 | 0.097 | 0.003 |
| | IBA | 0.355 | 0.662 | 0.566 | 0.195 | 0.480 | 0.212 | 0.103 | 0.110 | 0.170 | 0.001 |
| CIFAR-10 | Alternate | 0.692 | 0.994 | 0.597 | 0.734 | 0.593 | 0.249 | 0.071 | 0.063 | 0.054 | 0.002 |
| | BadNet | 0.960 | 1.00 | 0.951 | 0.802 | 0.309 | 0.379 | 0.062 | 0.018 | 0.023 | 0.003 |
| | DBA | 0.178 | 0.406 | 0.200 | 0.260 | 0.216 | 0.354 | 0.129 | 0.083 | 0.015 | 0.009 |
| | Sybil | 0.627 | 0.880 | 0.531 | 0.655 | 0.574 | 0.560 | 0.074 | 0.094 | 0.016 | 0.004 |
| | Neurotoxin | 0.981 | 0.867 | 0.130 | 0.235 | 0.182 | 0.158 | 0.364 | 0.130 | 0.100 | 0.001 |
| | IBA | 0.104 | 0.137 | 0.288 | 0.352 | 0.267 | 0.145 | 0.0082 | 0.019 | 0.112 | 0 |
| Sentiment-140 | Alternate | 0.127 | 0.260 | 0.103 | 0.209 | 0.042 | 0.087 | 0.106 | 0.087 | 0.051 | 0.002 |
| | BadNet | 0.595 | 0.283 | 0.180 | 0.077 | 0.062 | 0.079 | 0.034 | 0.153 | 0.012 | 0.001 |
| | DBA | 0.965 | 0.710 | 0.763 | 0.751 | 0.265 | 0.271 | 0.197 | 0.072 | 0.080 | 0.003 |
| | Sybil | 0.210 | 0.653 | 0.305 | 0.655 | 0.259 | 0.272 | 0.073 | 0.090 | 0.057 | 0.001 |
| | Neurotoxin | 0.979 | 0.846 | 0.335 | 0.300 | 0.185 | 0.098 | 0.107 | 0.100 | 0.066 | 0.007 |
| | IBA | 0.533 | 0.347 | 0.521 | 0.136 | 0.143 | 0.109 | 0.121 | 0.020 | 0.045 | 0.003 |

Heatmap is used to indicate performance level, where green color represents better performance, yellow to red represents worse performance.

**Table 2.** Main task **accuracy** (↑).

| Dataset | Attack | FedAvg | Krum | Median | Trim | FLTrust | CONTRA | MESAS | FreqFed | AGSD | `BoBa` |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MNIST | Attack-free | 0.992 | 0.991 | 0.992 | 0.992 | 0.992 | 0.990 | 0.991 | 0.990 | **0.993** | 0.991 |
| | Alternate | 0.991 | 0.991 | 0.990 | 0.991 | 0.991 | 0.989 | **0.992** | 0.983 | 0.990 | 0.991 |
| | BadNet | 0.991 | 0.991 | 0.990 | 0.991 | 0.989 | 0.988 | 0.985 | **0.992** | 0.990 | 0.991 |
| | DBA | **0.992** | 0.989 | **0.992** | **0.992** | 0.991 | 0.866 | 0.991 | 0.989 | 0.991 | 0.990 |
| | Sybil | **0.991** | 0.990 | **0.991** | **0.991** | **0.991** | 0.987 | 0.985 | **0.991** | 0.990 | **0.991** |
| | Neurotoxin | 0.990 | 0.989 | 0.991 | 0.990 | 0.988 | 0.990 | 0.992 | 0.991 | **0.993** | 0.992 |
| | IBA | 0.986 | 0.988 | 0.987 | 0.989 | 0.985 | 0.990 | 0.991 | **0.992** | 0.990 | 0.991 |
| fMNIST | Attack-free | **0.911** | 0.901 | 0.909 | 0.910 | 0.905 | 0.890 | 0.899 | 0.897 | 0.900 | 0.902 |
| | Alternate | **0.908** | 0.895 | 0.904 | 0.906 | 0.897 | 0.886 | 0.893 | 0.898 | 0.890 | 0.897 |
| | BadNet | **0.903** | 0.610 | 0.893 | **0.903** | 0.887 | 0.612 | 0.891 | 0.899 | 0.893 | **0.903** |
| | DBA | 0.894 | 0.881 | 0.897 | 0.895 | 0.876 | 0.613 | 0.889 | 0.891 | 0.887 | **0.900** |
| | Sybil | **0.907** | 0.895 | 0.904 | 0.906 | 0.906 | 0.889 | 0.894 | 0.898 | 0.895 | 0.903 |
| | Neurotoxin | 0.901 | 0.894 | 0.899 | 0.902 | 0.896 | 0.885 | 0.895 | 0.892 | 0.896 | **0.904** |
| | IBA | 0.896 | 0.890 | 0.893 | 0.895 | 0.887 | 0.889 | 0.891 | 0.894 | 0.890 | **0.900** |
| CIFAR-10 | Attack-free | 0.711 | 0.691 | 0.709 | 0.702 | 0.721 | **0.722** | 0.698 | 0.710 | 0.703 | 0.720 |
| | Alternate | 0.687 | 0.653 | 0.681 | 0.681 | 0.705 | 0.676 | 0.682 | 0.688 | 0.691 | **0.709** |
| | BadNet | 0.683 | 0.662 | 0.675 | 0.679 | 0.495 | 0.586 | 0.683 | 0.690 | 0.695 | **0.716** |
| | DBA | 0.677 | 0.638 | 0.668 | 0.667 | 0.675 | 0.466 | 0.673 | 0.681 | 0.687 | **0.703** |
| | Sybil | 0.693 | 0.668 | 0.679 | 0.685 | 0.681 | 0.592 | 0.686 | 0.689 | 0.696 | **0.717** |
| | Neurotoxin | 0.690 | 0.665 | 0.674 | 0.678 | 0.672 | 0.650 | 0.679 | 0.684 | 0.688 | **0.698** |
| | IBA | 0.672 | 0.661 | 0.670 | 0.675 | 0.668 | 0.659 | 0.676 | 0.681 | 0.685 | **0.693** |
| Sentiment-140 | Attack-free | 0.768 | 0.759 | 0.763 | 0.770 | 0.752 | 0.760 | 0.765 | 0.769 | 0.772 | **0.781** |
| | Alternate | 0.782 | 0.774 | 0.779 | **0.783** | 0.770 | 0.768 | 0.775 | 0.780 | 0.781 | 0.781 |
| | BadNet | 0.776 | 0.763 | 0.772 | 0.771 | 0.765 | 0.752 | 0.770 | 0.773 | 0.778 | **0.785** |
| | DBA | 0.765 | 0.754 | 0.760 | 0.759 | 0.750 | 0.742 | 0.758 | 0.765 | 0.769 | **0.780** |
| | Sybil | 0.780 | 0.768 | 0.775 | 0.777 | 0.774 | 0.760 | 0.773 | 0.776 | 0.779 | **0.786** |
| | Neurotoxin | 0.762 | 0.745 | 0.751 | 0.757 | 0.746 | **0.780** | 0.750 | 0.776 | 0.763 | 0.779 |
| | IBA | 0.749 | 0.735 | 0.742 | 0.748 | 0.739 | 0.732 | 0.745 | 0.751 | 0.755 | **0.775** |

Equation 3. The time complexity is negligible. The time complexity of the proposed greedy algorithm for clustering is shown in Figure 3. We further report the full end-to-end runtime (including DDIG computation, clustering, voting, and aggregation) in comparison to standard aggregation-based defenses on the MNIST dataset with 200 clients under the BadNet attack. Across 50 rounds, BoBa achieves an average runtime of 0.0523 seconds/round, which is approximately twice that of FedAvg (no defense) with a runtime of 0.0231 seconds/round.

**Table 3.** Computation Time of Clustering.

| Client Num | 10 | 20 | 30 | 40 | 50 | 100 | 200 |
|---|---|---|---|---|---|---|---|
| Runtime (ms) | 0.83 | 1.21 | 1.57 | 1.87 | 2.20 | 2.62 | 3.75 |

## *5.8 Discussions*

BoBa performs best when some clients have distinct label distributions, as this supports effective clustering. In cases where client distributions are overlapping, DDIG's signal will weaken, but in such cases, the method will work as in IID scenarios, where BoBa has been shown to generalize well(Figure 8). If malicious clients attempt to mimic diverse distributions, the defense may degrade. Nonetheless, uniform cluster inclusion ensures that even boundary clients do not exert disproportionate influence across multiple clusters.

## 6 Conclusions

We proposed `BoBa`, an effective solution for mitigating backdoor attacks in FL with non-IID data. By introducing overlapping clustering with the DDIG module, we establish `BoBa` as a powerful performance booster for detection mechanisms against various backdoor attack scenarios. We propose two constraints — Balanced Clustering and Uniform Inclusion in the clustering algorithm to solve the collusion problem. Our extensive evaluation showcases the superiority of `BoBa` over other baseline methods, as it consistently achieves lower ASRs across various attack strategies and non-IID levels on multiple datasets. Moreover, the extensive evaluation results confirm the robustness of `BoBa`, positioning it as a promising safeguard for FL applications.

## Acknowledgements

This work was supported in part by the Office of Naval Research under grants N00014-24-1-2730, the US National Science Foundation under grants 2154929, 2247560, 2247561, 2235232, and 2312447.

## References

[1] H. Ali, S. Nepal, S. S. Kanhere, and S. Jha. Adversarially guided stateful defense against backdoor attacks in federated deep learning. In *2024 Annual Computer Security Applications Conference (ACSAC)*, pages 794–809. IEEE, 2024.

[2] S. Awan, B. Luo, and F. Li. Contra: Defending against poisoning attacks in federated learning. In *26th European Symposium on Research in Computer Security*, pages 455–475. Springer, 2021.

[3] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[4] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[5] P. Blanchard, R. Guerraoui, J. Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NeurIPS*, 2017.

[6] C. Briggs, Z. Fan, and P. Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[7] X. Cao, M. Fang, J. Liu, and N. Z. Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *Network and Distributed Systems Security Symposium NDSS*, 2021.

[8] Y. Chen, L. Su, and J. Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25, 2017.

[9] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[10] M. Fang, X. Cao, J. Jia, and N. Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th {USENIX} Security Symposium*, pages 1605–1622, 2020.

[11] H. Fereidooni, A. Pegoraro, P. Rieger, A. Dmitrienko, and A.-R. Sadeghi. Freqfed: A frequency analysis-based approach for mitigating poisoning attacks in federated learning. In *Network and Distributed Systems Security Symposium NDSS*, 2024.

[12] C. Fung, C. J. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[13] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv:1906.06629*, 2019.

[14] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision, 2009. URL http://help.sentiment140.com/. CS224N Project Report, Stanford.

[15] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[16] R. Guerraoui, S. Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In *International Conference on Machine Learning*, pages 3521–3530. PMLR, 2018.

[17] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[18] P. Jain, P. Kar, et al. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–363, 2017.

[19] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

[20] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[21] T. Krauß and A. Dmitrienko. Mesas: Poisoning defense for federated learning resilient against adaptive attackers. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1526–1540, 2023.

[22] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[24] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[26] T. D. Nguyen, P. Rieger, R. De Viti, H. Chen, B. B. Brandenburg, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, et al. Flame: Taming backdoors in federated learning. In *31st USENIX Security Symposium*, pages 1415–1432, 2022.

[27] T. D. Nguyen, T. A. Nguyen, A. Tran, K. D. Doan, and K.-S. Wong. Iba: Towards irreversible backdoor attacks in federated learning. *in NeurIPS*, 2024.

[28] P. Rieger, T. D. Nguyen, M. Miettinen, and A.-R. Sadeghi. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. In *Network and Distributed Systems Security Symposium*, 2022.

[29] P. Rieger, T. Krauß, M. Miettinen, A. Dmitrienko, and A.-R. Sadeghi. Crowdguard: Federated backdoor detection in federated learning. In *Network and Distributed Systems Security Symposium NDSS*, 2024.

[30] V. Shejwalkar and A. Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.

[31] S. Shen, S. Tople, and P. Saxena. Auror: Defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 508–519, 2016.

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[33] J. So, B. Güler, and A. S. Avestimehr. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[34] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[35] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee, and D. Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33:16070–16084, 2020.

[36] N. Wang, Y. Xiao, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou. Flare: defending federated learning against model poisoning attacks via latent space representations. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 946–958, 2022.

[37] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[38] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[39] C. Xie, K. Huang, P.-Y. Chen, and B. Li. Dba: Distributed backdoor attacks against federated learning. In *ICLR*, 2019.

[40] C. Xie, M. Chen, P.-Y. Chen, and B. Li. Crfl: Certifiably robust federated learning against backdoor attacks. In *ICML*, pages 11372–11382. PMLR, 2021.

[41] D. Yin, Y. Chen, R. Kannan, and P. Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International conference on machine learning*, pages 5650–5659. Pmlr, 2018.

[42] Z. Zhang, A. Panda, L. Song, Y. Yang, M. Mahoney, P. Mittal, R. Kannan, and J. Gonzalez. Neurotoxin: Durable backdoors in federated learning. In *ICML*, 2022.

[43] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, and S. Yu. Pdgan: A novel poisoning defense method in federated learning using generative adversarial network. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 595–609. Springer, 2019.