FedCmp

Byzantine-robust federated learning through clustering model update parameters

Wang, Hua; Wang, Shaoxiong; Wang, Rui; Wang, Pengxiang

DOI
10.1016/j.ins.2025.122377

Publication date
2025

Document Version
Final published version

Published in
Information Sciences

Information Sciences 718 (2025) 122377

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

FedCmp: Byzantine-robust federated learning through clustering model update parameters

Hua Wang [a,*], Shaoxiong Wang [a], Rui Wang [b], Pengxiang Wang [c]

[a] School of Computer Science, Qufu Normal University, Rizhao, 276500, China
[b] EEMCS, Delft University of Technology, Delft, 2628XE, Netherlands
[c] Jinan Foreign Language School, Jinan, 250000, China

ARTICLE INFO

Keywords:
Federated learning
Distributed machine learning
Byzantine attack
Robust
Security

ABSTRACT

Federated Learning (FL) is a distributed machine learning paradigm that enables multiple clients to collaboratively train a model without sharing their private data. However, its distributed nature makes FL vulnerable to Byzantine attacks. Most existing Byzantine-robust FL schemes have limitations, such as ineffective defense against well-crafted malicious updates or degraded performance in non-independent and identically distributed (non-IID) data scenarios. To address these challenges, we propose FedCmp, a robust FL framework with an anomaly detection mechanism. Our approach identifies malicious updates by leveraging a significant disparity in vote counts between benign and compromised clients. We first propose a clustering strategy for update parameters, followed by the implementation of a multi-round voting mechanism to accelerate vote accumulation for benign or compromised clients based on parameter diversity. Finally, following the majority principle, malicious updates are accurately filtered out without compromising the contributions of benign clients. Experimental results demonstrate that FedCmp outperforms existing robust FL schemes and maintains high accuracy even in highly non-IID data scenarios.

1. Introduction

In federated learning (FL) [1,2], training data (e.g., hospital records [3] or text snippets [4]) stays on local devices, including smartphones, sensors, and other edge devices [5]. These devices participate in training a global model by uploading only model updates, thereby preserving user data privacy. A well-known FL algorithm, FedAvg [6], enables each client to train a local model for multiple epochs using its own data, after which the server aggregates all local models by averaging to construct the global model.

Although FL has several advantages, it is vulnerable to Byzantine attacks. Adversaries can manipulate model updates or train local models on poisoned data, either degrading the global model's accuracy [7,8] or altering its outputs [9–11], potentially leading to severe consequences. Therefore, developing effective Byzantine-robust FL schemes is essential for mitigating such threats.

Many studies have focused on enhancing the Byzantine robustness of FL. Some approaches [12,13] rely on auxiliary data collected at the server to detect and adjust model updates uploaded by clients. This data can be sourced from either the Internet or directly from clients. However, in highly non-independent and identically distributed (non-IID) FL scenarios, Internet-sourced data often has a distribution that significantly differs from clients' data. As a result, it becomes ineffective. For instance, in FL-based mobile keyboard

* Corresponding author.
E-mail address: wanghua@qfnu.edu.cn (H. Wang).

https://doi.org/10.1016/j.ins.2025.122377
Received 14 April 2024; Received in revised form 16 May 2025; Accepted 29 May 2025

Available online 3 June 2025

prediction [4], users from different regions exhibit distinct language styles due to cultural differences. Therefore, using text snippets with a specific language style to evaluate model updates is unreasonable. On the other hand, collecting data directly from clients may violate their privacy, contradicting the fundamental principle of FL.

Instead of collecting data, other robust schemes employ descriptive statistical methods like mean aggregation [14,15] and median-based filtering [16,17], or distance metrics such as cosine similarity [18] and Euclidean distance [19], to identify outliers. However, recent attacks [20–22] are able to bypass these defenses by carefully crafting malicious model updates. The key reason for their success is that these updates are crafted to blend seamlessly with benign ones, making them undetectable by conventional outlier-based defenses.

Another category of defense mechanisms [23,24] treats model updates with high mutual similarity as potential malicious updates and either limits or discards them. However, this approach fails to defend against adversarial updates containing random perturbations. These updates may not exhibit high similarity but can still degrade the global model's performance.

Furthermore, some schemes [16,17] retain only updates or parameters that appear benign. However, these approaches often do not align well with real-world FL scenarios. In highly non-IID settings, model updates from different clients often differ significantly. Consequently, these schemes may unintentionally penalize benign clients, leading to suboptimal performance in such FL environments.

To address these challenges, we propose FedCmp, a novel Byzantine-robust FL scheme. Previous work has focused on mitigating the impact of malicious parameters on model convergence through anomaly detection, but they cannot fundamentally identify malicious updates. In contrast, we analyze the distributional differences between malicious and benign parameters, enabling FedCmp to precisely identify malicious updates. Specifically, FedCmp leverages the high variance and disorder of hidden layer parameters in model updates near the output layer under non-IID data scenarios. By leveraging multiple rounds of voting, FedCmp accurately detects and filters out malicious model updates. In each training iteration, the server constructs multiple parameter vectors, each representing parameters from a specific dimension of all model updates. These vectors are then clustered, and clients with parameters in the same cluster receive a vote. Finally, malicious updates are identified and removed based on clustered voting results. High variance and disorder intuitively increase the positional diversity of benign update parameters within the vector. In contrast, although malicious parameters may be numerically similar to benign ones, their positions remain relatively fixed. Consequently, one cluster consistently contains the same malicious parameters along with varying benign ones. This creates a clear distinction between the votes received by compromised clients and benign clients. Leveraging this pattern, FedCmp accurately identifies all compromised clients while aggregating updates from benign ones. By effectively utilizing knowledge from all benign updates, FedCmp maintains strong performance, even in highly non-IID FL scenarios. In summary, the main contributions of this work are as follows:

- To the best of our knowledge, FedCmp is the first Byzantine-robust FL scheme capable of accurately identifying well-crafted malicious updates that resemble and blend in with benign ones.
- We preserve the updates of all benign clients without penalizing their parameters, allowing the Byzantine-robust scheme to perform effectively in FL scenarios with highly non-IID client data.
- We design a detection method that leverages parameter diversity to identify malicious updates. It is highly scalable and can be seamlessly integrated into any FL framework without modifying the protocol.
- We evaluate FedCmp against five model poisoning attacks on three benchmark datasets and compare it with four state-of-the-art Byzantine-robust algorithms. Experimental results demonstrate its superior performance.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 provides background knowledge on FL and elaborates on our threat model. Section 4 presents our proposed method in detail. Section 6 summarizes and analyzes the experimental results. Finally, section 8 concludes the paper.

2. Related work

In some Byzantine-robust schemes, the server verifies model updates using a collected dataset. FLTrust [27] computes the cosine similarity between each client's model update and the update derived from the server's dataset, discards those with lower similarity and assigns weights to the remaining updates for aggregation. However, due to the discrepancy between the server's dataset distribution and the highly non-IID client data, FLTrust performs poorly in such scenarios. DiverseFL [12] pre-trains a model for each client using their respective data before the FL iteration and uses these models to monitor subsequent training updates. Error Rate-based Rejection (ERR) and Loss Function-based Rejection (LFR) [29] eliminate local updates that significantly degrade the global model's error rate or loss, respectively. A common limitation of these methods is their reliance on a external dataset to detect malicious updates, which not only risks violating user privacy but also contradicts the core principle of FL—ensuring that training data remains on client devices.

Other researchers have adopted outlier detection methods to identify malicious model updates, avoiding reliance on datasets. Krum [25] computes the sum of Euclidean distances between each model update and its nearest $n - f - 2$ updates, where n is the total number of clients and f is the number of compromised clients. The update with the smallest sum of distances is selected to update the global model. To better utilize client knowledge, MultiKrum extends Krum by selecting c updates such that $n - c > 2f + 2$, and then aggregates them to update the global model. Trimmed-mean [14] removes the top and bottom f extreme values for each parameter dimension across all model updates, and then averages the remaining values. Bulyan [26] first selects $c (c \leq n - 2f)$ updates

Table 1
Comparison of Byzantine-robust FL schemes.

Scheme	Strategy	Limitation	Time complexity	Model accuracy (%)
[14]	Mean	Requires a certain number of malicious updates	$O(ndlogn)$	97.69
[25]	Euclidean distance	$m < (N-2)/2$; Ineffective in non-IID scenarios; Ineffective against well-crafted malicious updates	$O(n^2d)$	93.98
[26]	Euclidean distance; Mean	$m < N/2$; Ineffective against well-crafted malicious updates	$O(n^2logn + n^2d)$	97.77
[23]	Cosine similarity	Ineffective against single malicious updates	$O(n^2d)$	97.81
[27]	Trusted dataset; Cosine similarity	Requires a trusted dataset; Ineffective in non-IID settings	$O(nd)$	96.91
[15]	Clustering; Euclidean distance; Median; Clipping	Ineffective against well-crafted malicious updates	$O(nlogn + n^2d)$	96.60
[28]	Norms; Clipping	Ineffective against well-crafted malicious updates	$O(nlogn + nd)$	97.68
[19]	Euclidean Distance; Mean	Ineffective against well-crafted malicious updates	$O(nd)$	97.69
Our	Clustering	$m < N/2$	$O(knlogn + nd)$	97.61

via MultiKrum, and then applies Trimmed-mean for aggregation. However, recent studies [29,20,21] have shown that attackers can craft malicious updates with geometric characteristics similar to benign ones, allowing them to bypass these defenses.

Some methods also leverage similarity metrics to detect anomalous updates. AFA [18] computes a weighted average of all model updates and measures the cosine similarity between each individual update and this average, discarding those with abnormal similarity. However, the weighted average itself can be influenced by malicious updates, thereby reducing its robustness. FoolsGold [23] assumes that malicious model updates exhibit higher similarity to each other than to benign updates, and dynamically adjusts clients' learning rates based on update similarity. Nevertheless, FoolsGold is less effective against adaptive attacks [30] or malicious updates that incorporate random perturbations. FPD [24] integrates four types of defense mechanisms to iteratively cleanse and correct local model updates. However, its high computational complexity significantly slows down the FL iteration process.

We summarize previous works in Table 1. To address these limitations, we propose FedCmp, as detailed in section 4.

3. System overview

3.1. Federated learning (FL)

Consider a common classification task in a real-world FL scenario. We assume that the system consists of a central server and N clients. Each client $n (n = 1, ..., N)$ possesses a local training dataset $D_n\{(x_i^n, y_i^n)\}_{i=1}^{|D_n|}$, where x_i^n represents the ith data sample, and $y_i^n \in (1, ..., M)$ is its corresponding label from M classes. The goal of the FL system is to collaboratively train a single global model θ by solving the following optimization problem:

$$\theta = \underset{\theta}{argmin} G(F_1(\theta), ..., F_N(\theta)) \tag{1}$$

where $F_n(\theta)$ represents the local optimization objective for client n, and is defined as follows:

$$F_n(\theta) = E_{(x,y)\sim D_n}[f_n(\theta; x, y)] \tag{2}$$

where f_n represents the empirical loss function on the training dataset D_n.

$G(\bullet)$ is a specific aggregation rule. For example, in FedAvg [6], the aggregation function is defined as:

$$G(\bullet) = \sum_{n=1}^{N} \frac{w_n}{w} F_n(\theta) \tag{3}$$

where w_n and w represent the number of training samples of client n and the total number of training samples across all clients, respectively. Specifically, in each training iteration, the FL training process can be divided into the following three steps:

- Step I: The server randomly selects a subset of clients and distributes the current global θ^t, where t represents the current iteration.
- Step II: Each selected client n utilizes its local training dataset to train θ^t using mini-batch stochastic gradient descent (SGD [31]), yielding an updated local model θ_n^{t+1}. Then, the client uploads the model update $g_n^{t+1} = \theta_n^{t+1} - \theta^t$ back to the server.
- Step III: The server aggregates the received updates and updates the global model $\theta^{t+1} = \theta^t - \alpha \bullet G(g_1^{t+1}, ..., g_N^{t+1})$, where α is the learning rate, and $G(\bullet)$ represents the aggregation function defined previously.

3.2. Threat model

3.2.1. Adversary setup

Attacker's knowledge and capability. We consider an attack model that closely resembles a realistic scenario. Specifically, the attacker can compromise any m clients out of the N clients selected by the server in each iteration, but cannot compromise the server itself. For the compromised clients, the attacker has full knowledge of their states and can control them to send arbitrary model updates

to the server, whether these updates are carefully crafted, randomly generated, or trained with poisoned data. Regarding the benign clients, the attacker is aware of the updates they upload, which enables more potent attacks. The attacker's goal is to disrupt the convergence of the global model as much as possible and degrade its accuracy.

Number of compromised clients. Some Byzantine-robust schemes [25,26] rely on prior knowledge of the number of compromised clients to guarantee algorithm convergence; however, obtaining such information is challenging in real-world FL scenarios. Other schemes [24,18] impose no assumptions on the number of compromised clients, instead focusing on guiding the update direction of the global model. However, these methods may not be suitable for highly non-IID data scenarios. FedCmp makes only a mild assumption: the number of compromised clients does not exceed $N/2$, where N represents the total number of clients selected to participate in training during each iteration.

3.2.2. Defense model

Some previous studies [12,13] require the central server to maintain a dataset. However, implementing this requirement in practice is challenging. Collecting data from clients may be hindered by users' reluctance to upload their private information to the server. Although the Internet can also serve as an alternative data collection channel, in practical FL scenarios, clients' training data is highly non-IID, rendering the collected data of limited reference value. In contrast, FedCmp requires the server to access only the model updates uploaded by clients in order to identify malicious ones. Beyond this, the server does not require any auxiliary datasets or additional information about the clients.

4. Details of FedCmp

4.1. Overview

- **Step I: Constructing parameter vectors.** The server constructs multiple parameter vectors, each consisting of all parameters in a specific dimension extracted from the local model updates uploaded by clients.
- **Step II: Voting for clients.** Each parameter vector is clustered into two groups, and all clients associated with the parameters in one of the clusters receive a vote. After clustering all parameter vectors, the final voting result is obtained.
- **Step III: Clustering the voting result.** The voting results are further clustered into two groups. Clients in the larger cluster are identified as benign, and their model updates are aggregated to update the global model.

4.2. Constructing parameter vectors

FedCmp selects parameters from the lth layer of model updates to construct parameter vectors (as detail in subsection 4.3), where l is a hyperparameter that we set. Specifically, the server first generates a parameter set W^n for the model update uploaded by each client n:

$$W^n = \{w_1^n, ..., w_K^n\} \tag{4}$$

where w_k^n represents the weight parameter randomly selected for the kth time without repetition. Then, the parameter matrix W^* is constructed using all sets $\{W^n\}_{n=1}^N$:

$$W^* = \begin{bmatrix} w_1^1 & w_2^1 & \dots & w_K^1 \\ w_1^2 & w_2^2 & \dots & w_K^2 \\ \dots & \dots & \dots & \dots \\ w_1^N & w_2^N & \dots & w_K^N \end{bmatrix} \tag{5}$$

where each column is a parameter vector v_k to be clustered in subsection 4.3.

4.3. Voting for clients

In a general neural network [32], the output layer processes complex features from the preceding layer and flexibly adjusts parameter values based on specific task objectives, such as category distinction in classification tasks or numerical prediction in regression tasks, to generate the final output. Consequently, from the perspective of parameter distribution, hidden layer parameters close to the output layer tend to be more complex and diverse, exhibiting a wider range and greater variance. Furthermore, in non-IID FL scenarios, the data distribution varies across clients, leading to significant differences in the distribution of these parameters within local models. In summary, within the matrix W^*, the values of benign parameters exhibit a high degree of randomness. For attackers, the goal is to generate malicious parameters that resemble benign ones to evade detection. Since malicious updates are not learned from data but are instead generated through specific computations, these parameters lack the same distribution characteristics. Based on this observation, we design a voting mechanism to identify compromised clients.

The K-Means clustering algorithm [33] partitions data points by computing their distances to cluster centers, ensuring that points within the same cluster are as similar as possible while those in different clusters are as distinct as possible. This approach effectively captures differences in data point distributions. Given that distinguishing between benign and malicious parameter distributions is

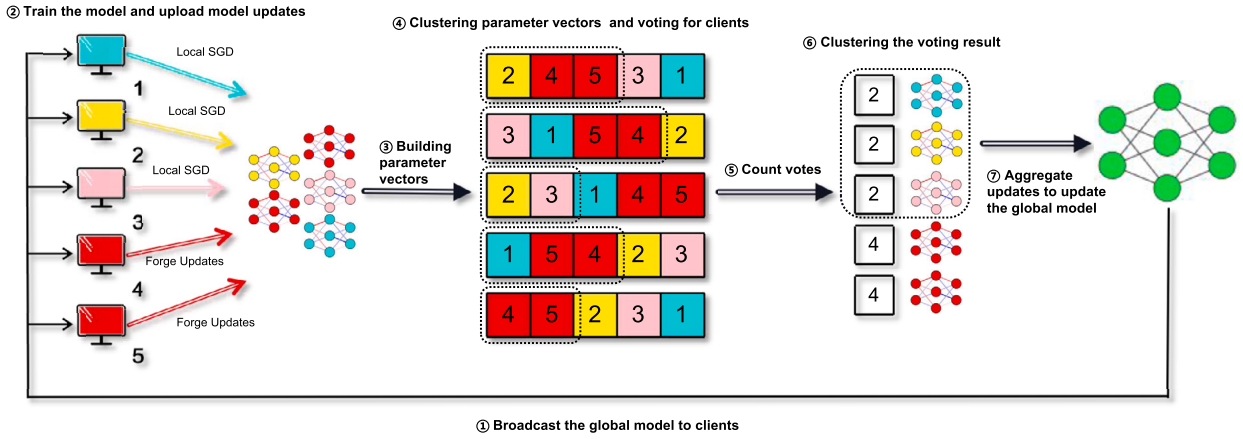


Fig. 1. The overall framework of FedCmp.

crucial, we adopt K-Means as our core clustering algorithm. Each column vector in matrix W^* is first clustered into γ groups, which are subsequently merged into two clusters, with γ being a tunable hyperparameter. Clients associated with parameters in the first cluster receive a vote. The intuition behind this approach is to mitigate the impact of outliers on the clustering algorithm. In the presence of extreme outliers, directly clustering into two groups may result in one cluster containing only a few outlier parameter values, thereby undermining the effectiveness of subsequent voting. Conversely, using too many clusters may prevent the clustering algorithm from effectively capturing the distributional differences between benign and malicious parameters, thus blurring the boundary in the final voting results. Finally, merging into two clusters aims to balance the accumulation of votes while leveraging the distributional differences between malicious and benign parameters. By applying clustering algorithms to multiple parameter vectors, we rapidly accumulate votes for one group, enabling effective differentiation. Fig. 2 illustrates the clustering of parameter vectors. In Fig. 2a, when malicious parameters closely resemble benign ones, the first cluster (represented by circles in the figure) contains both types. However, due to the diversity of benign parameters, they are not consistently clustered into the first cluster across different vectors. From an overall perspective, this results in benign clients receiving a more evenly distributed share of votes, while compromised clients rapidly accumulate votes. In Fig. 2b, when malicious parameters are randomly generated and differ significantly from benign ones, the first cluster contains fewer benign parameters, causing malicious ones to appear more frequently in the first cluster across different vectors. Therefore, a clear boundary will eventually emerge in the number of votes between benign and malicious clients.

Specifically, in each iteration, the server first clusters each parameter vector in matrix W^* into $\{C_1, C_2, ..., C_\gamma\}$, which are then further merged into C_α and C_β in order of incremental parameters. The merging strategy is to keep the number of clients in C_α and C_β remains as balanced as possible. Finally, the clients in cluster C_α receive a vote. After clustering all vectors, the voting result is obtained, *i.e.*, $R = \{r_n | n = 1, ..., N\}$.

It is important to note that the number of votes a client receives does not directly indicate whether it is malicious, but is solely used to differentiate between clients. Therefore, when counting votes based on cluster assignments, benign and malicious clients tend to receive votes in a mutually exclusive manner. Nonetheless, both voting patterns contribute to distinguishing malicious clients from benign ones.

4.4. Clustering the voting result

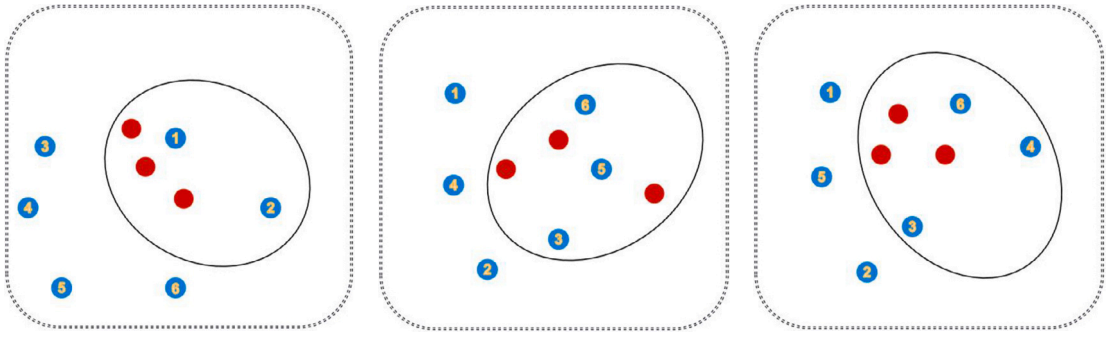
The server first clusters the voting result $R = \{r_n | n = 1, ..., N\}$ into two groups, G_1 and G_2, where $|G_1| + |G_2| = N$. If $|G_1| \leq |G_2|$, the model updates from all clients in G_2 are aggregated to update the global model; otherwise, the updates from G_1 are used instead.

An overview of FedCmp is illustrated in Fig. 1, and its pseudocode is provided in Algorithm 1.

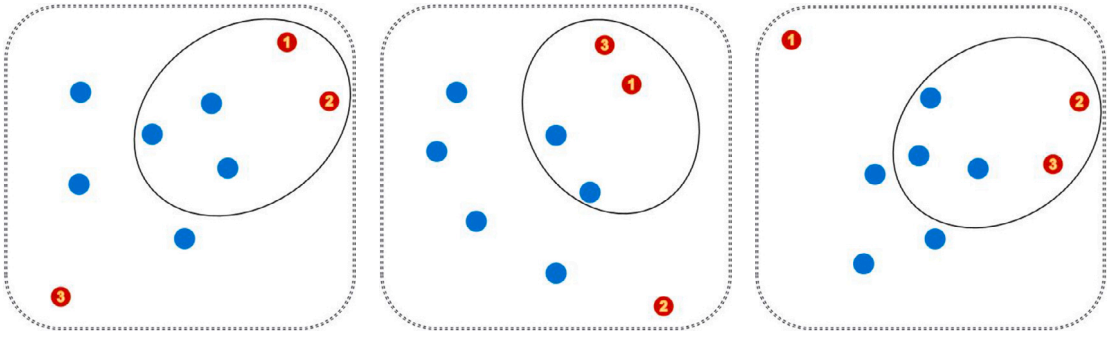
5. Performance analysis

Given that FedCmp relies on clustering algorithms, analyzing its time complexity is essential. Let n, d and k represent the number of model updates, the number of parameter dimensions, and the number of parameter vectors, respectively.

1. Constructing Parameter Vectors: This step involves traversing and slicing operations. Its time complexity is $O(nk)$.
2. Clustering Parameter Vectors: All vectors are traversed and clustered. The time complexity is $O(k(nlogn + ln))$, where l represents the maximum number of K-Means iterations.
3. Calculating Client Votes: To determine the number of votes each client receives, the clustering results must be traversed. The time complexity is $O(n)$.
4. Clustering Voting Results: Clustering the voting results requires $O(nlogn + ln)$ time.
5. Merging and Averaging Updates: Finally, merging and averaging the updates has a time complexity of $O(nd)$.



(a) Malicious parameters are similar to benign parameters.



(b) Malicious parameters are very different from benign parameters.

Fig. 2. An illustration of different parameter distributions when clustering vectors. The blue dots and red dots represent benign parameters and malicious parameters respectively, and the parallel subgraphs represent different parameter vectors.

Algorithm 1 Byzantine-robust federated learning through clustering model update parameters (FedCmp).

Input: Total number of clients N; Total number of iterations T; Total number of parameter vectors K;
Output: Global model θ^T;

for $t \leftarrow 0$ to T **do**
  Initialize an empty parameter matrix W^*
  Initialize a voting result R with all zeros
  for $n \leftarrow 0$ to N **do**
    $\theta_n^{t+1} = SGD(D_n, \theta^t)$
    Randomly select K distinct parameters from the lth layer of θ_n^{t+1} and put them in the nth row of the matrix W^*
  end for
  for $v_k \in W^*$ **do**
    Cluster v_k into γ groups and then merge them into clusters C_α and C_β
    for $n \in C_\alpha$ **do**
      R[n] += 1
    end for
  end for
  $G_1, G_2 = 2 - Means(R)$
  if $|G_1| \leq |G_2|$ **then**
    $\theta^{t+1} = \theta^t - \alpha \sum_{n \in G_1} \frac{|D_n|}{\sum_{n=1}^{N} |D_n|} \theta_n^{t+1}$
  else
    $\theta^{t+1} = \theta^t - \alpha \sum_{n \in G_2} \frac{|D_n|}{\sum_{n=1}^{N} |D_n|} \theta_n^{t+1}$
  end if
end for

In conclusion, the overall time complexity of FedCmp is $O(knlogn + nd)$. In most cases, the time complexity is dominated by the $O(knlogn)$ term.

n is a fundamental parameter in the FL system. In FedCmp, clustering, sorting, and voting operations all depend on n. A larger n results in a significant increase in the computational overhead of clustering. To mitigate this, optimization strategies such as Mini-Batch K-Means or distributed training can be employed to reduce computational costs.

The value of k directly influences the number of K-Means clustering iterations. As k increases, the number of K-Means executions grows, leading to a linear increase in overall complexity. Empirically, the value of k influences the disparity in the number of votes received by benign and malicious clients. The optimal choice of k depends on the specific attack algorithms employed. If k is too small, the clustering algorithm may fail to effectively distinguish between the two groups, while an excessively large k increases

Table 2
The default hyperparameters setting.

Hyperparameter	MNIST	Fashion-MNIST	CIFAR10
The number of iterations (t)	100	100	1000
The number of clients (n)	50	50	50
The number of clients selected in each iteration (s)	20	20	20
The number of compromised Clients (m)	8	8	8
Batch size (b)	64	64	128
The number of parameter vectors (k)	40	40	40
The number of layers selected for parameter selection (l)	6	12	30
Number of clusters before merging (c)	3	3	3
Learning Rate (α)	1	0.001	0.01
Degree of non-IID (p)	1	1	1

computational overhead. Therefore, k can be tuned as a hyperparameter to strike an optimal balance between algorithm performance and computational efficiency.

6. Experimental evaluation

6.1. Experimental setup

6.1.1. Datasets

- **MNIST [31].** MNIST is a handwritten digit recognition dataset that is widely used in the field of machine learning. It consists of grayscale images of handwritten digits from 0 to 9, each of size 28×28 pixels. The dataset includes 60,000 training samples for model training and 10,000 test samples for performance evaluation.
- **Fashion-MNIST [34].** Fashion-MNIST is an image dataset similar to MNIST, but it features images of 10 different fashion items instead of handwritten digits.
- **CIFAR10 [35].** CIFAR-10 is a widely used dataset for image classification tasks. It consists of 60,000 color images across 10 categories, with each category containing 6,000 images. The dataset includes 50,000 training samples and 10,000 test samples, with each image being 32×32 pixels in size and comprising three channels: red, green, and blue.

6.1.2. Data distribution

We use the Dirichlet distribution [36] to partition the dataset. In real-world scenarios, data is often non-IID. The Dirichlet distribution helps simulate scenarios in which different clients possess different data categories, making the experiment more realistic. By adjusting the distribution parameter α, we can control the balance of category distribution across sub-datasets. A larger α (e.g., 10) results in a nearly uniform distribution, resulting in a more balanced category distribution across clients. Conversely, a smaller α (e.g., 0.1) leads to a more skewed distribution, where each client may only have data from a few categories.

6.1.3. Application scenarios

FL is typically applied in two scenarios: cross-device (e.g., keyboard next-word prediction [3]) and cross-silo (e.g., healthcare prediction [4]). In cross-device FL, the number of clients is large, with varying hardware capabilities, data distributions, and data quality. During each training iteration, the server randomly selects a subset of available clients to participate in model training. In contrast, cross-silo FL involves large institutions (e.g., hospitals [37], banks [38]) as clients. These institutions possess vast amounts of data but are reluctant to share it due to privacy and commercial concerns. In our experiments, we simulated a cross-device FL scenario.

6.1.4. Model structure

We employ a 6-layer convolutional neural network as the global model for MNIST, and a 12-layer network for Fashion-MNIST. For CIFAR-10, we adopt the ResNet-18 architecture [39] as the global model.

6.1.5. Hyperparameters setting

Table 2 shows the default parameter settings for the FL training process. Unless otherwise specified, all experiments use these default settings.

6.1.6. Evaluation metrics

- **Model accuracy (MA).** MA refers to the percentage of samples for which the global model correctly predicts the true labels.
- **False negative rate (FNR).** FNR represents the proportion of benign updates mistakenly classified as malicious by the algorithm. A higher FNR means more benign updates are incorrectly rejected.
- **False positive rate (FPR).** FPR represents the proportion of malicious updates incorrectly classified as benign by the algorithm. A higher FPR means more malicious updates are mistakenly accepted.

Table 3
The performance of FedCmp under multiple attacks when selecting parameters from different layers in LeNet. The results are in the form of "FNR (%) / FPR (%) / Highest MA (%)".

Attack	The number of layers selected for parameter selection			
	1	2	3	4
Fang	1/1/97.58	5.42/5/97.31	0/0/97.48	0.12/0/97.64
LIE	0/0/97.42	1.25/1/97.46	0/0/97.47	0/0/97.45
AGR-tailored	6.83/7/97.58	12.42/16/97.75	7.75/10/97.43	0/0/97.45
AGR-agnostic	37.42/65/29.63	20.75/33/88.51	18.08/28/96.65	0.83/1/97.56
Gaussian	0/0/97.47	0/0/97.63	0/0/97.45	0/0/97.48

6.1.7. Evaluated attacks

- **Fang attack [29].** The Fang attack maximizes the directed deviation between the aggregated updates of benign clients and those of all clients, including the compromised ones.
- **LIE attack [21].** The LIE attack evades detection by robust algorithms by adding carefully crafted noise to model updates.
- **AGR-tailored attack [20].** The AGR-tailored attack maximizes the Euclidean distance, in contrast to the directed deviation used in the Fang attack.
- **AGR-agnostic attack.** The AGR-agnostic attack computes the maximum distance among all benign model updates and optimizes malicious updates to ensure that their distance from all benign updates does not exceed this maximum value.
- **Gaussian attack.** The Gaussian attack samples noise from a Gaussian distribution with a mean of 0 and a variance of 1, using the sampled noise as malicious parameters to construct model updates.

6.1.8. Evaluated defenses

- **FLAME [15].** FLAME first calculates the median Euclidean distance between model updates and clips all model updates to this value. HDBSCAN [40] is then used to cluster the model updates based on cosine similarity to obtain a set of benign model updates. Finally, these updates are aggregated and noise is added to update the global model.
- **FLTrust [27].** FLTrust trains a model on the server to evaluate model updates uploaded by clients. Any model update that is not similar to the server update is removed, and the remaining model updates are weighted and aggregated to update the global model.
- **MoNNA [19].** MoNNA computes the distance between each vector and a reference vector, selects the k closest vectors, and returns their mean.
- **ARC [28].** ARC first calculates the norms of all updates and sorts them in ascending order. Then, f updates with the largest norms are clipped so that their norms do not exceed a threshold, where f represents the number of malicious clients. Finally, the mean of the clipped updates is returned.

Additionally, we implemented the widely used FedAvg algorithm [6] as a non-robust baseline for comparison with the robust algorithms.

6.2. Experimental results

6.2.1. The impact of selecting parameters from different layers

Table 3 illustrates the impact of selecting parameters from different layers on the model performance of FedCmp when defending against various attacks. The model consists of four layers, with the first two being convolutional layers and the last two fully connected layers. The fourth layer—the final fully connected layer—contains more stochastic parameters because it is responsible for extracting high-dimensional features from the input samples. As a result, when its parameters are used to construct parameter vectors, FedCmp is able to accurately identify and eliminate all malicious updates across a range of attack scenarios.

6.2.2. The impact of proportions of compromised clients

Table 4 shows the performance of FedCmp with varying percentages of compromised clients. When the proportion of compromised clients is below 50%, FedCmp accurately distinguishes between malicious and benign updates, resulting in both the FNR and FPR being close to 0. When the proportion of compromised clients reaches 50% and 100%, FedCmp's FNR and FPR rise to approximately 50% and 100%, respectively. In fact, FedCmp remains effective in distinguishing between updates when the proportion of compromised clients is 50% or higher. However, since FedCmp operates without prior knowledge of the attack method, it cannot confidently determine which merged clusters contain benign updates. As a solution, FedCmp randomly selects one cluster or chooses the one with the majority of updates for global model updating.

6.2.3. The impact of the number of clusters before merging

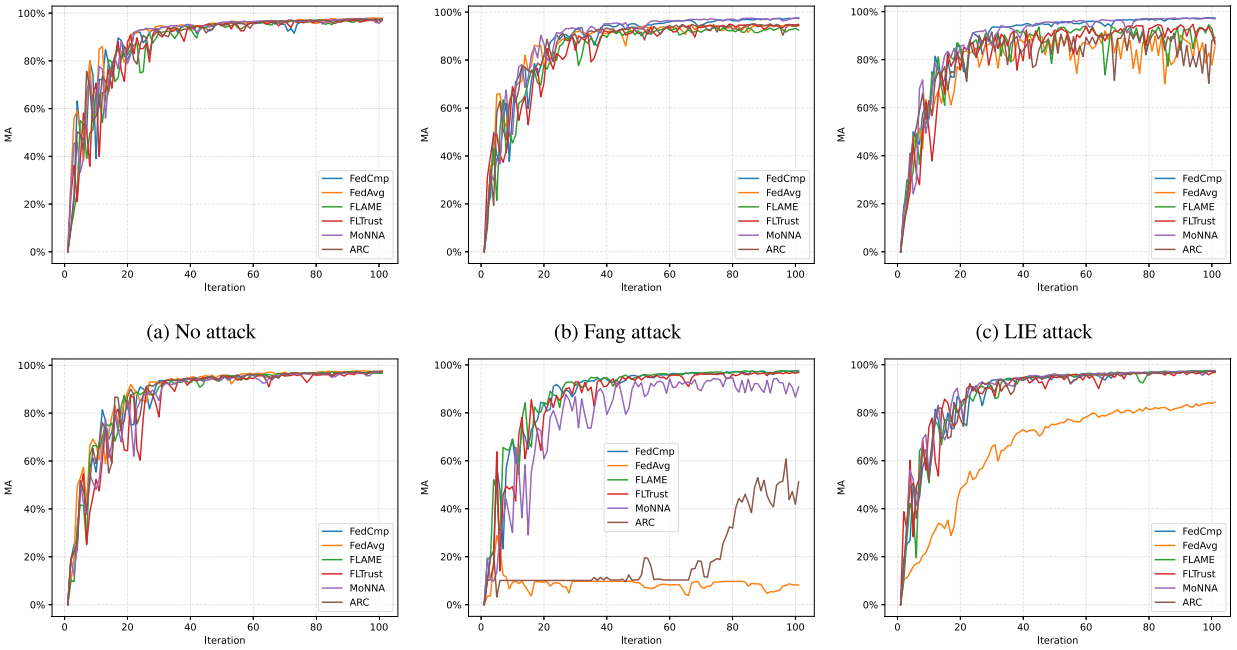
In Table 5, we examine multiple cases with different numbers of clusters. A value of 2 means that the parameter vectors are clustered into two groups without further merging, while a value of 20 means the parameter vectors are clustered into 20 clusters, ensuring that after merging, the two resulting clusters contain an equal number of clients. The model achieves the best performance

Table 4
The performance of FedCmp under multiple attacks with different proportions of compromised clients. The results are in the form of "FNR (%) / FPR (%) / Highest MA (%)".

Attack	The proportion of compromised clients				
	5%	20%	40%	50%	65%
Fang	1.26/1/97.47	0.13/0/97.85	0.12/0/97.64	60/60/93.99	100/100/18.06
LIE	0/0/97.55	0/0/97.59	0/0/97.45	54/54/89.89	100/100/74.80
AGR-tailored	0/0/97.63	0/0/97.64	0/0/97.45	47/47/97.45	100/100/97.42
AGR-agnostic	8.58/4/97.54	0.69/1/97.43	0.83/1/97.56	54.6/57/54.62	96.29/100/10.09
Gaussian	0/0/97.57	0/0/97.85	0/0/97.48	54/54/79.55	100/100/11.53

Table 5
The performance of FedCmp under multiple attacks when using a different number of clusters before merging. The results are in the form of "FNR (%) / FPR (%) / Highest MA (%)".

Attack	The number of clusters before merging				
	2	3	5	10	20
Fang	0/0/97.49	0.12/0/97.64	0/0/97.68	3/2/97.66	1.5/2/97.50
LIE	0/0/97.71	0/0/97.45	0/0/97.51	0/0/97.64	0/0/97.53
AGR-tailored	43.25/86/97.78	0/0/97.45	41.17/78/97.28	37.75/66/97.42	36.33/60/97.30
AGR-agnostic	9.08/13/97.67	0.83/1/97.56	1.42/2/97.68	0.83/1/97.53	0/0/97.48
Gaussian	0.42/53.13/87.48	0/0/97.48	0.5/5.25/91.16	43.33/73.38/77.46	40.08/76.13/76.48



(a) No attack (b) Fang attack (c) LIE attack

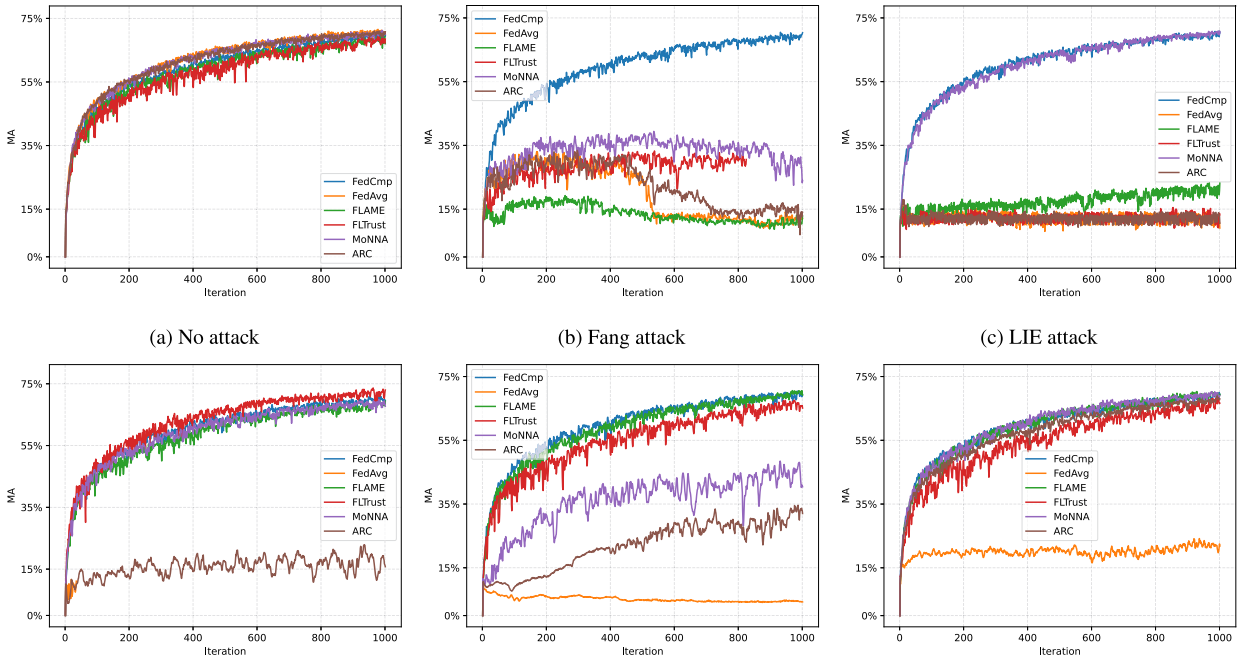
(d) AGR-tailored attack (e) AGR-agnostic attack (f) Gaussian attack

Fig. 3. MA of different schemes in defending against various attacks on MNIST.

when the number of clusters is 3. This is because 3 represents an optimal balance, allowing the clustering algorithm to capture the distributional features inherent in the parameter vectors while remaining unaffected by outliers.

6.2.4. Performance comparison against various attacks

Fig. 3 and Fig. 4 present the comparison of MA among various robust schemes when defending against multiple types of attacks on MNIST and CIFAR10, respectively. Since MNIST is relatively simple in terms of recognition difficulty, it provides limited opportunities for attacks to cause substantial degradation. As a result, the differences in MA among the robust schemes are less pronounced and most of them are able to maintain high performance. In contrast, CIFAR10 presents a more complex and challenging learning task, making distinctions between the capabilities of different schemes become more evident. The increased difficulty of the dataset amplifies the



(a) No attack (b) Fang attack (c) LIE attack

(d) AGR-tailored attack (e) AGR-agnostic attack (f) Gaussian attack

Fig. 4. MA of different schemes in defending against various attacks on CIFAR10.

Table 6
Highest MA of different schemes in defending against various attacks on Fashion-MNIST.

Attack	Byzantine-robust scheme					
	FedCmp	FedAvg	FLAME	FLTrust	MoNNA	ARC
No attack	85.73	86.87	86.50	85.06	87.63	87.23
Fang	86.62	73.19	76.16	76.91	84.62	74.12
LIE	86.01	67.13	78.21	70.69	78.08	68.41
AGR-tailored	86.94	10.00	81.60	84.17	86.58	10.51
AGR-agnostic	87.13	77.38	85.04	82.95	80.25	78.06
Gaussian	86.61	72.45	85.42	84.23	86.46	86.64

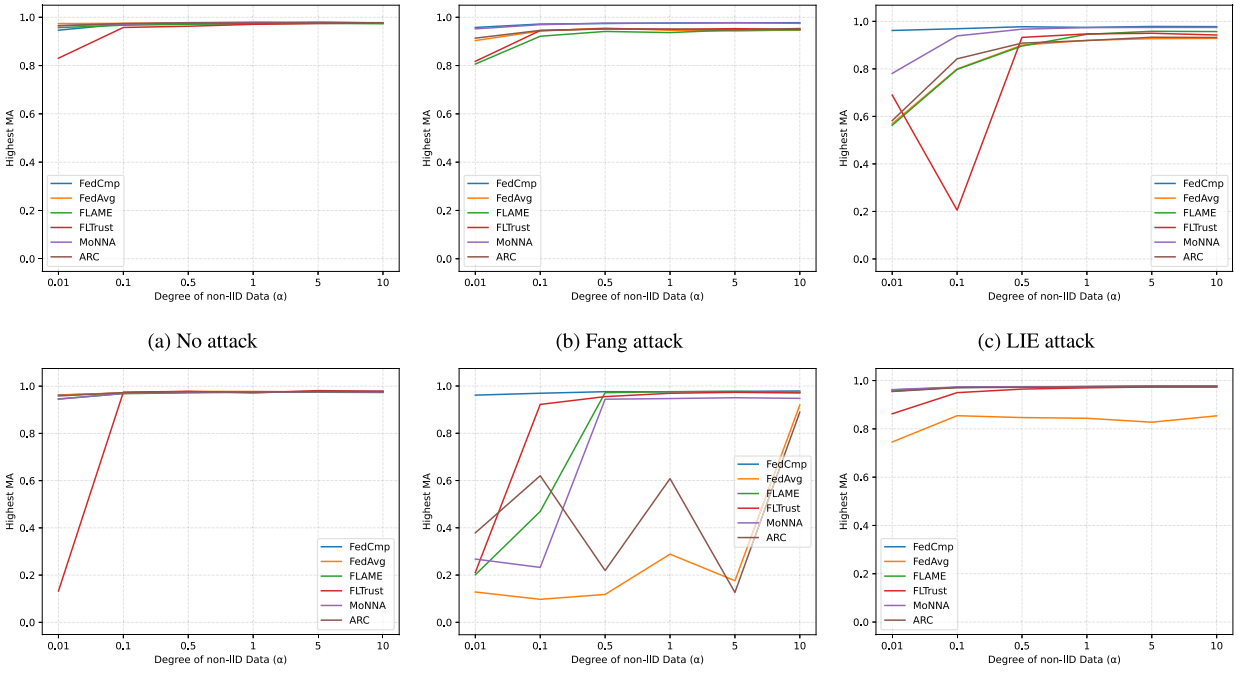
effects of both the attack strategies and the effectiveness of the defense mechanisms, thereby exposing the limitations or strengths of each scheme more clearly.

In the no-attack setting, FedCmp exhibits slightly lower MA compared to other schemes. This is primarily because FedCmp still removes a small number of benign model updates as a precautionary measure, even when no malicious updates are present. While this conservative approach enhances its robustness and general resilience, it inevitably leads to a slight reduction in accuracy due to the exclusion of useful information contributed by benign clients. Interestingly, in the presence of attacks, FedCmp's MA often exceeds that in the no-attack scenario. This is because the presence of malicious updates enables FedCmp to retain almost all of the remaining benign updates. As a result, the global model is less affected by adversarial manipulation and can generalize better, leading to improved accuracy under attack conditions. Table 6 supplements the experiments on Fashion-MNIST, and the results are largely consistent with our analysis.

Overall, in a wide range of attack scenarios, FedCmp consistently outperforms other robust schemes with respect to MA. This consistent superiority highlights its strong capability in mitigating the effects of model poisoning attacks and preserving the global model's integrity and effectiveness across varying levels of task complexity.

6.2.5. Performance comparison across different data distributions

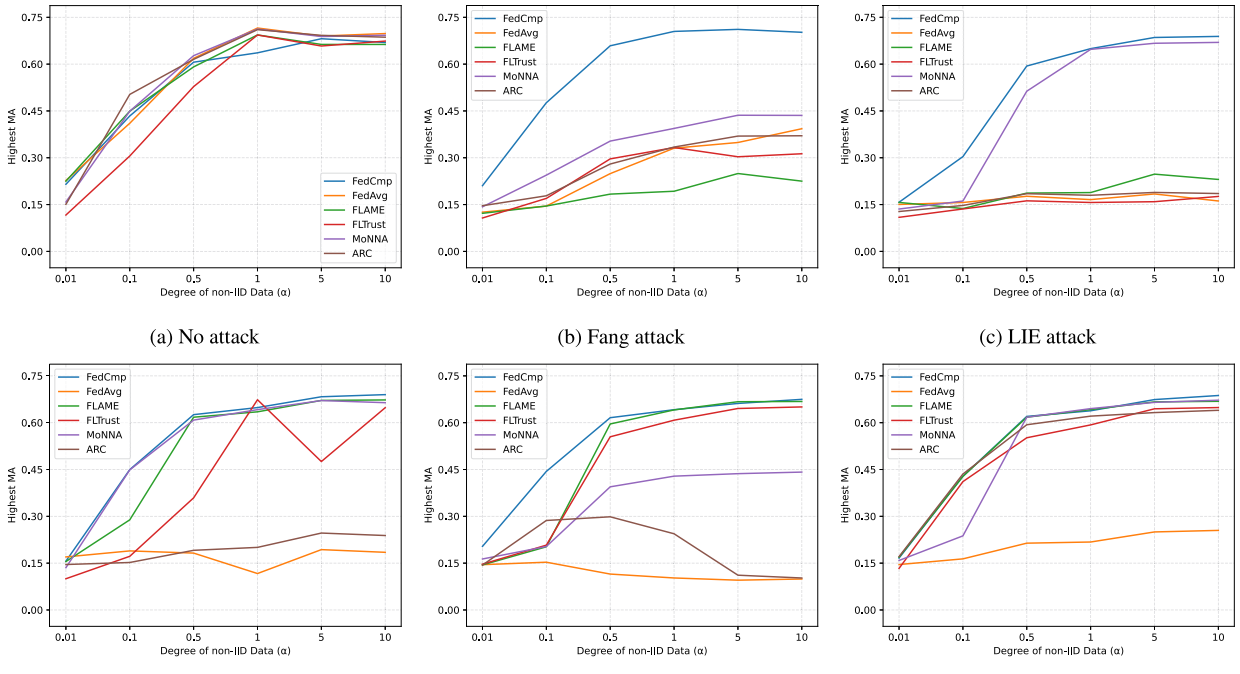
Fig. 5 and Fig. 6 illustrate the variation in the highest MA achieved at different values of α for different robust schemes defending against various attacks on MNIST and CIFAR10, respectively. We consider six values of α: 0.01, 0.1, 0.5, 1, 5, and 10. Here, $\alpha = 0.01$ corresponds to an extremely non-IID data distribution, where most clients possess samples from only a single class and exhibit significant data imbalance. $\alpha = 1$ is a commonly used setting, where clients' data exhibits a moderate degree of diversity. When $\alpha = 10$, the data across clients becomes nearly IID, with both the sample counts and label distributions approximately uniform.



(a) No attack (b) Fang attack (c) LIE attack

(d) AGR-tailored attack (e) AGR-agnostic attack (f) Gaussian attack

Fig. 5. Variation in highest MA across different schemes defending against various attacks with varying α on MNIST.



(a) No attack (b) Fang attack (c) LIE attack

(d) AGR-tailored attack (e) AGR-agnostic attack (f) Gaussian attack

Fig. 6. Variation in highest MA across different schemes defending against various attacks with varying α on CIFAR10.

Since MNIST is relatively simple, variations in α have minimal impacts on the highest MA, and the performance differences among different robust schemes are mainly due to the nature and intensity of the attacks themselves. Under these conditions, FedCmp effectively defends against multiple attacks, maintaining a stable and high level of MA. When applied to the more challenging CIFAR10, significant performance variation is observed. In particular, when $\alpha = 0.01$, the global model exhibits severe overfitting.

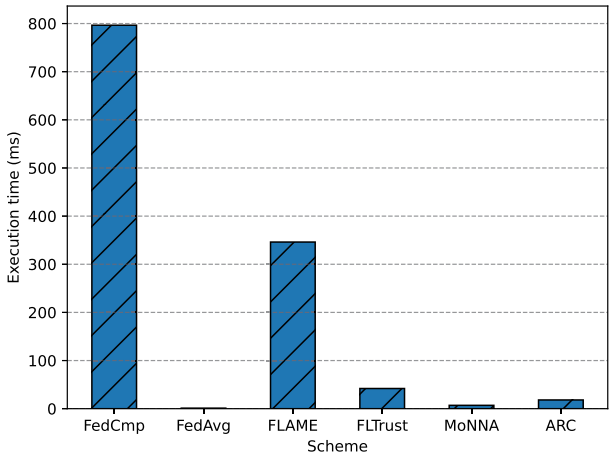


Fig. 7. Comparison of computational efficiency.

This overfitting results from the fact that the global model is updated based on highly biased local data, making it difficult for the model to generalize across the entire data space. As a result, all schemes, including FedCmp, suffer a noticeable drop in MA under this setting. As the value of α increases, the performance of FedCmp improves significantly. With moderate to large α values, FedCmp not only recovers from performance degradation but also frequently outperforms other robust schemes, even under different attacks. This is because FedCmp is able to accurately identify and exclude malicious client updates, thereby preserving the integrity of the global model.

Compared to other robust schemes, FedCmp shows clear advantages across all distribution settings. It consistently achieves higher highest MA and demonstrates better resilience to both adversarial behavior and data heterogeneity, making it a strong candidate for real-world FL applications.

6.2.6. Computational efficiency

Fig. 7 illustrates the average time consumed per training round by different robust schemes when defending against Gaussian attack throughout the entire training process. Among them, FedCmp demonstrates relatively lower computational efficiency. This is primarily due to its frequent use of K-Means clustering algorithm: in each training round, FedCmp must perform clustering on all parameter vectors and determine the voting result. These additional steps introduce overhead, increasing the per-round time cost compared to schemes with simpler strategies, such as FedAvg, which only performs averaging. However, this also endows FedCmp with enhanced robustness against various attacks.

7. Limitations

Although FedCmp effectively removes malicious updates and maintains high model accuracy against carefully crafted model poisoning attacks, it still has certain limitations. First, in the absence of attackers, FedCmp may misclassify some benign updates as malicious, leading to a slight decline in accuracy. Second, while effective against model poisoning, its defense capabilities remain limited when faced with broader threats such as data poisoning and backdoor attacks. Additionally, FedCmp assumes that compromised clients constitute less than 50% of the total client population. Finally, since FedCmp relies on clustering algorithms, its computational overhead can be substantial in large-scale FL scenarios (i.e., with a large number of clients), which may affect overall system efficiency. In future work, we aim to address these limitations to further enhance its robustness and practicality.

8. Conclusion

In this work, we propose FedCmp, a Byzantine-robust federated learning (FL) scheme designed to defend against model poisoning attacks. FedCmp leverages the diversity of model update parameters to filter out malicious updates, regardless of whether the malicious updates are carefully crafted or randomly generated. Specifically, FedCmp first clusters model parameters, then assigns votes to clients based on the clustering results, and finally identifies and removes malicious updates by clustering the voting outcome. We conduct a comprehensive evaluation of FedCmp's performance. Experimental results demonstrate that FedCmp effectively eliminates malicious updates and achieves higher model accuracy compared to existing approaches. Moreover, FedCmp maintains stable performance even in FL scenarios with highly non-IID data.

CRediT authorship contribution statement

Hua Wang: Writing – review & editing, Validation, Investigation. **Shaoxiong Wang:** Writing – review & editing, Writing – original draft, Software, Methodology, Conceptualization. **Rui Wang:** Writing – review & editing, Validation, Project administration, Investigation. **Pengxiang Wang:** Writing – review & editing, Supervision, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Qiang Yang, Yang Liu, Tianjian Chen, Yongxin Tong, Federated machine learning: concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19.
[2] Jingjing Guo, Zhiquan Liu, Siyi Tian, Feiran Huang, Jiaxing Li, Xinghua Li, Kostromitin Konstantin Igorevich, Jianfeng Ma, Tfl-dt: a trust evaluation scheme for federated learning in digital twin for mobile networks, IEEE J. Sel. Areas Commun. 41 (11) (2023) 3548–3560.
[3] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, Daniel Ramage, Federated learning for mobile keyboard prediction, arXiv preprint, arXiv:1811.03604, 2018.
[4] Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, Wei Shi, Federated learning of predictive models from federated electronic health records, Int. J. Med. Inform. 112 (2018) 59–67.
[5] Latif U. Khan, Walid Saad, Zhu Han, Ekram Hossain, Choong Seon Hong, Federated learning for Internet of things: recent advances, taxonomy, and open challenges, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1759–1799.
[6] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Aguera y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.
[7] Marco Barreno, Blaine Nelson, Anthony D. Joseph, J. Doug Tygar, The security of machine learning, Mach. Learn. 81 (2010) 121–148.
[8] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, Bo Li, Manipulating machine learning: poisoning attacks and countermeasures for regression learning, in: 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 19–35.
[9] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, Vitaly Shmatikov, How to backdoor federated learning, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 2938–2948.
[10] Chulin Xie, Keli Huang, Pin-Yu Chen, Bo Li, Dba: distributed backdoor attacks against federated learning, in: International Conference on Learning Representations, 2019.
[11] Yinbin Miao, Rongpeng Xie, Xinghua Li, Zhiquan Liu, Kim-Kwang Raymond Choo, Robert H. Deng, Efficient and secure federated learning against backdoor attacks, IEEE Trans. Dependable Secure Comput. 21 (5) (2024) 4619–4636.
[12] Saurav Prakash, Amir Salman Avestimehr, Mitigating Byzantine attacks in federated learning, arXiv preprint, arXiv:2010.07541, 2020.
[13] Cong Xie, Sanmi Koyejo, Indranil Gupta, Zeno: distributed stochastic gradient descent with suspicion-based fault-tolerance, in: International Conference on Machine Learning, PMLR, 2019, pp. 6893–6901.
[14] Dong Yin, Yudong Chen, Ramchandran Kannan, Peter Bartlett, Byzantine-robust distributed learning: towards optimal statistical rates, in: International Conference on Machine Learning, PMLR, 2018, pp. 5650–5659.
[15] Thien Duc Nguyen, Phillip Rieger, Roberta De Viti, Huili Chen, Björn B. Brandenburg, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, et al., {FLAME}: taming backdoors in federated learning, in: 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 1415–1432.
[16] Yudong Chen, Lili Su, Jiaming Xu, Distributed statistical machine learning in adversarial settings: Byzantine gradient descent, Proc. ACM Meas. Anal. Comput. Syst. 1 (2) (2017) 1–25.
[17] Haibo Yang, Xin Zhang, Minghong Fang, Jia Liu, Byzantine-resilient stochastic gradient descent for distributed learning: a Lipschitz-inspired coordinate-wise median approach, in: 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, 2019, pp. 5832–5837.
[18] Luis Muñoz-González, Kenneth T. Co, Emil C. Lupu, Byzantine-robust federated machine learning through adaptive model averaging, arXiv preprint, arXiv: 1909.05125, 2019.
[19] Sadegh Farhadkhani, Rachid Guerraoui, Nirupam Gupta, Lê-Nguyên Hoang, Rafael Pinot, John Stephan, Robust collaborative learning with linear gradient overhead, in: International Conference on Machine Learning, PMLR, 2023, pp. 9761–9813.
[20] Virat Shejwalkar, Amir Houmansadr, Manipulating the Byzantine: optimizing model poisoning attacks and defenses for federated learning, in: NDSS, 2021.
[21] Gilad Baruch, Moran Baruch, Yoav Goldberg, A little is enough: circumventing defenses for distributed learning, Adv. Neural Inf. Process. Syst. 32 (2019).
[22] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, Seraphin Calo, Analyzing federated learning through an adversarial lens, in: International Conference on Machine Learning, PMLR, 2019, pp. 634–643.
[23] Clement Fung, Chris JM Yoon, Ivan Beschastnikh, The limitations of federated learning in sybil settings, in: 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), 2020, pp. 301–316.
[24] Wei Wan, Shengshan Hu, Minghui Li, Jianrong Lu, Longling Zhang, Leo Yu Zhang, Hai Jin, A four-pronged defense against Byzantine attacks in federated learning, in: Proceedings of the 31st ACM International Conference on Multimedia, 2023, pp. 7394–7402.
[25] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, Julien Stainer, Machine learning with adversaries: Byzantine tolerant gradient descent, Adv. Neural Inf. Process. Syst. 30 (2017).
[26] Rachid Guerraoui, Sébastien Rouault, et al., The hidden vulnerability of distributed learning in byzantium, in: International Conference on Machine Learning, PMLR, 2018, pp. 3521–3530.
[27] Xiaoyu Cao, Minghong Fang, Jia Liu, Neil Zhenqiang Gong, Fltrust: Byzantine-robust federated learning via trust bootstrapping, arXiv preprint, arXiv:2012.13995, 2020.
[28] Youssef Allouah, Rachid Guerraoui, Nirupam Gupta, Ahmed Jellouli, Geovani Rizk, John Stephan, The vital role of gradient clipping in Byzantine-resilient distributed learning, arXiv preprint, arXiv:2405.14432, 2024.
[29] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, Neil Gong, Local model poisoning attacks to {Byzantine-Robust} federated learning, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 1605–1622.

[30] Cong Xie, Oluwasanmi Koyejo, Indranil Gupta, Fall of empires: breaking Byzantine-tolerant sgd by inner product manipulation, in: Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 261–270.
[31] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[32] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[33] James MacQueen, et al., Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.
[34] Han Xiao, Kashif Rasul, Roland Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint, arXiv:1708.07747, 2017.
[35] Alex Krizhevsky, Geoffrey Hinton, et al., Learning multiple layers of features from tiny images, 2009.
[36] David M. Blei, Andrew Y. Ng, Michael I. Jordan, Latent Dirichlet allocation, J. Mach. Learn. Res. 3 (Jan 2003) 993–1022.
[37] Jie Xu, Benjamin S. Glicksberg, Chang Su, Peter Walker, Jiang Bian, Fei Wang, Federated learning for healthcare informatics, J. Healthc. Inform. Res. 5 (2021) 1–19.
[38] Guodong Long, Yue Tan, Jing Jiang, Chengqi Zhang, Federated learning for open banking, in: Federated Learning: Privacy and Incentive, Springer, 2020, pp. 240–254.
[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[40] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, Jörg Sander, Hierarchical density estimates for data clustering, visualization, and outlier detection, ACM Trans. Knowl. Discov. Data 10 (1) (2015) 1–51.