A Client Detection and Parameter Correction Algorithm for Clustering Defense in Clustered Federated Learning

Junyu Ye, Lei Shi, Hao Xu, Sinan Pan, Juan Xu
School of Computer Science and Information Engineering, Hefei University of Technology
Hefei, China

Abstract

As a new federated learning(FL) paradigm, clustered federated learning (CFL) could effectively address the issue of model training accuracy loss due to different data distribution in FL. However, the introduction of the clustering process also brings new risks. Adversaries can implement model poisoning by adding crafted perturbations with clients' model parameters, potentially resulting in overall clustering failure. To tackle this problem, we propose a client detection and parameter correction framework in this paper. Our approach aims to identify malicious clients by analyzing the difference in vector parameter density distribution between malicious and benign clients. We precisely locate malicious perturbations in the parameters and recover them, enabling the server to effectively utilize benign updates for normal clustering and training within the CFL framework. Experiment results show that our defense algorithm outperforms others, consistently improving training accuracy by an average of 30% under various kinds of attacks.

Keywords

Clustered federated learning, Malicious detection, Robust learning.

1 Introduction

Federated learning (FL) [7] is a well-known framework of the distributed machine learning. FL enables clients to train a global model collaboratively under the coordination that clients do not need to share their original training data[3]. However, in the real-world scenarios, data generated by different clients often have varying distributions[13], which can result in Non-IID and unbalanced data, reducing training accuracy and efficiency. Fortunately, Clustered Federated Learning (CFL) addresses data heterogeneity by grouping clients into clusters based on data features and training separate models for each cluster. However, the extra clustering process may lead extra security risks. Malicious data or poisoning attacks from some clients may destroy the accuracy of the CFL model[4]. To tackle the problem, researchers have done some work on this field. For example, in [8], to address the defect of adversarial clients inferring private information by exploiting the similarity of data distribution, authors proposed a CFL secure aggregation framework that simultaneously aggregates local gradients from multiple user clusters without knowing any information about cluster identities or local gradients. In [10], authors stated that CFL uses the geometric properties of the FL loss surface to identify cluster structures. It filters out adversarial clients in a relatively small number of communication rounds.

Previous research mainly focuses on system robustness based on final model training effectiveness, overlooking the impact of clustering failures, caused by malicious clients. The goal of CFL is personalization at the group level, while incorrect clustering will reduce the training efficiency, and will cause the leakage of model parameters [8]. To address this, we propose a resilient defense frame with two components: a malicious client detection module and a parameter correction module. This is the first known study to focus on addressing CFL clustering failures. We summarize our contributions below:

1) We are the first to propose a flexible model poison attack that can disable the clustering function in CFL. This attack is capable of poisoning model parameters stealthily, evading detection by the server.
2) We introduce a novel detection module based on parameter density distribution. This module can accurately identify and flag potential malicious clients, maintaining a high recognition rate even as the number of malicious clients increases.

ACM MobiCom '24, November 18–22, 2024, Washington D.C., DC, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0489-5/24/11
https://doi.org/10.1145/3636534.3698247

3) We propose our defense algorithm with residual-based iterative confidence weight, which recovers the poisoned parameters and removes the impact of the attack on the server side.

The remainder of the article is structured as follows: Section II introduces our problem scenario and proposes the dynamic attack algorithm for CFL. Section III designs and analyzes the detection method of malicious client and the poisoning parameter correction algorithm. Section IV shows the performance of our algorithm under various datasets and attacks. In section VI we conclude our work.

2 Problem Statement and Threat Model

2.1 Problem Scope

Consider a clustered federated learning scenario with several clients participating in the training process and a central server responsible for parameter aggregation and client clustering [9]. After completing a round of local training, each client will upload its model parameters to the server and the server will calculate these parameters by the cosine similarity to decide client clusters. This process will be iteratively executed until the CFL converges, and in each round, a cluster which has been decided in the last round may be further bipartitioned. Suppose there is an adversary who has taken control of one or more clients through trojan horses, or physical intrusion and induces those clients to become malicious clients. Suppose the adversary has no knowledge of the target model of a particular client or its training data, but it can modify some of the uploaded model parameters. The adversary wants to disable CFL's cluster process by modifying these model parameters. We further suppose that the server has some ability to discover attacks. If the server detects suspicious parameters, it can discard them and exclude the client for several rounds. The adversary's goal is to disrupt CFL's clustering by strategically altering model parameters.

2.2 Threat Model

Denote $c_i (1 \leq i \leq M)$ as one client, and denote $C_i (1 \leq i \leq T)$ as one cluster. According to the previous work [11], the adversary can add noise to the original parameters instead of replacing the parameters entirely. We define the malicious parameter as $\theta^*_{i,n,k} = \theta_{i,n,k} + \gamma \nabla p$, where $\theta_{i,n,k}$ represents the parameter of the k_{th} convolutional layer of benign client at the n_{th} round of training, and poisoned entity represented by $\theta^*_{i,n,k}$. γ is the disturbance factor and ∇p is the disturbance vector. From [11] we know two methods for adding perturbation are commonly discussed, the inverse unit vector and the inverse sign, which can be represented as the following.

Table 1: Notations

Symbol	Explanation
N	Training iteration round
M	Number of clients
M^*	Number of malicious clients
K	Number of parameters for a single model
D_i	Datasets of client i
$C \in \mathbb{C}$	One cluster in the set of clusters
$Med(\cdot)$	Median estimate
p	Perturbation vector
$\theta_{i,n,k}$	Layer k parameters of the client i for the round n of training
$\theta^*_{i,n,k}$	Parameters which under attack
$\delta_{i,n,k}$	Indexes of $\theta_{i,n,k}$
β_1	Federated learning stopping criterion in CFL
β_2	Clustering stopping criterion in CFL
$\mu_{i,n,k}$	Normalised residuals
$z_{i,n,k}$	Model parameter confidence
$\Delta\theta_C$	Global models within clusters

- Inverse unit vector:

$$\nabla^p_{uv}(\theta_i) = \frac{-\theta_i}{\|\theta_i\|_2}. \quad (1)$$

- Inverse sign:

$$\nabla^p_{sgn}(\theta_i) = -sign(f_{avg}(\theta_i)). \quad (2)$$

Define the data of c_i as D_i, and D is the sum of data from all clients. Define the empirical risk function of c_i as $r_i(\theta)$. Define β_1 and β_2 as two constants that determine whether the clustering process should be stopped. According to [9], we have

$$0 < \|\sum_{i=1,\ldots,M} \frac{D_i}{|D|} \nabla_\theta r_i(\theta)\| < \beta_1 \leq \beta_2 < \max_{i=1,\ldots,M} \|\nabla_\theta r_i(\theta)\|, \quad (3)$$

which can ensure that the CFL jointly optimizes local risk functions for all clients in different clusters and converge towards the target.

The adversary's goal is to change the clustering process without excessively affecting training accuracy. Hence, the perturbation factor can be defined by

$$\nabla p \leftarrow \nabla^p(\arg\min(\|\sum_{i=1,\ldots,M} \frac{D_i}{|D|} \nabla_\theta r_i(\theta)\| > \beta_1)). \quad (4)$$

From this equation we can get the ideal range of the perturbation vector.

After the perturbation vector is determined, the adversary will use a perturbation factor $\gamma (\gamma > 1)$ with the perturbation vector to determine how to modify origin model parameters. In Algorithm 1, we describe whole steps for the adversary's attack.

3 Detection and Correction Algorithm

In previous section, we detail the steps of the adversary's attack. Here, we present our defense algorithm. Since the attack is dynamic and employs weak perturbations, it is challenging for the server to detect.[12]. To address this, we design a

Algorithm 1 Attack Algorithm

```
Input: $\gamma_{init}$, $\theta_{i,n,k}$.
Output: $\theta^*_{i,n,k}$.
    for each training iteration $n$ in $[1, N]$ do
2:      Client :
        for each client i in $[1, M]$ in parallel do
4:          Client receive updated model parameters
            // For malicious client
6:          Compute perturbation $\nabla p$ by Equation (4)
            function computeGamma($\gamma_{init}$)
8:          step $\leftarrow \gamma_{init}/2$, $\gamma \leftarrow \gamma_{init}$
            while $\gamma > 1$ and training iteration $n < N$ do
10:             if the server selected this client then
                    Inject optimized perturbation $\gamma \nabla p$ and training
12:                 if $\gamma \neq \gamma_{init}$ then
                        $\gamma \leftarrow (\gamma + step)$
14:                 end if
                else
16:                 $\gamma \leftarrow (\gamma - step)$
                    if the client trained last round then
18:                     return $\gamma$
                    end if
20:             end if
                $step = step/2$
22:         end while
            end function
24:         $\theta^*_{i,n,k} = \theta_{i,n,k} + \gamma \nabla p$
        end for
26: end for
```

two-step defense algorithm. Firstly, the server will build an adaptive classification algorithm to identify malicious clients. Secondly, the server will locate the poisoning parameters. Fig. 1 provides an overview of our defense algorithm, which we will discuss further in the following sections.

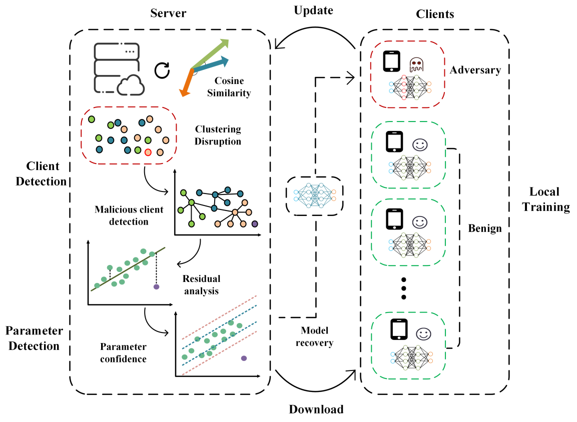


Figure 1: Overview of the our defense in CFL.

3.1 Malicious Client Detection Algorithm

The server is unable to differentiate between malicious and benign clients solely based on the vector norm of the model

Algorithm 2 Client Detection and Parameter Correction Algorithm

```
Input: Iteration round $N$, clients number $M$, set of clusters $\mathbb{C}$, set of model
    parameters $\Theta$.
Output: Malicious client labels $M^*$, global models within clusters $\Delta\theta^n_C$.
1:  for each training iteration $n$ in $[1, N]$ do
2:      for each client in $[1, M]$ in parallel do
3:          $\theta \leftarrow \theta + \Delta\theta_c$
4:          $\Delta\theta_c \leftarrow SGD_n(\theta, D) - \theta$
5:      end for
6:      for C $\in \mathbb{C}$ do
7:          Detecting malicious clients:
8:          $(\Delta\omega_1 \ldots \Delta\omega_M) \leftarrow$ FLATTEN$(\{\theta_i\}_{i=1}^{\Theta})$
9:          $(e_{(1,1)} \ldots e_{(M,M)}) \leftarrow$ COSINESIMILARITY$(\Delta\omega_1 \ldots \Delta\omega_M)$
10:         Compute edges weight:$d_{mr}(\omega_p, \omega_q)$
11:         Create minimum spanning tree: $Prim(d_{mr}(\omega_p, \omega_q))$
12:         Compress tree nodes to get noise labels $L_{noise}$ foreach client
13:         if COUNT $L_{noise} > \Gamma$ then
14:             Parameter Correction:
15:             Estimated linear regression model:
16:             $\hat{B}_{n,k} = Med^2\{(\theta_{j,n,k} - \theta_{i,n,k})/(\delta_{j,n,k} - \delta_{i,n,k})\}$
17:             $\hat{A}_{n,k} = Med^2\{(\delta_{j,n,k}\theta_{i,n,k} - \delta_{i,n,k}\theta_{j,n,k})/(\delta_{j,n,k} - \delta_{i,n,k})\}$
18:             Calculate parameters confidence: $\mu_{i,n,k}$
19:             Conduct parameters recovery: $Recovery(\theta^*_{i,n,k})$
20:             Parameters transmission: $\theta^*_{i,n,k} \leftarrow Recovery(\theta^*_{i,n,k})$
21:         end if
22:         Continue training for clustering results: $(C_1 \ldots C_T) \in \mathbb{C}$
23:     end for
24:     for C $\in \mathbb{C}$ do
25:         $\Delta\theta^n_C \leftarrow \frac{1}{|C|}\sum_{i \in C}\Delta\theta_{i,n}$
26:     end for
27: end for
```

parameters uploaded by the clients. Therefore, we propose using a Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)-based algorithm[1] for malicious client detection in clustered federated learning scenario. To identify malicious clients without accessing their data, we embed the HDBSCAN clustering method in the recognition algorithm, through which we are able to further identify the variability among the parameters of different clients by vector direction.

The server executes the CFL algorithm upon receiving the model parameters uploaded by the client. The perturbations introduced by malicious clients hinder the server's ability to perform effective clustering and allocate distinct model parameters. If the clustering step does not satisfy condition(3) due to the existence of malicious clients, HDBSCAN-based malicious client detection algorithm will be executed. The specific steps are as follows:

(1) To increase the algorithm processing speed, we serialize the weight parameters uploaded by the client $(\Delta\omega_1 \ldots \Delta\omega_M) \leftarrow$ FLATTEN$(\{\theta_i\}_{i=1}^{\Theta})$, and denote $\Delta\omega_i$ as flattened model parameters of the client i.

(2) Calculate the cosine distances $e_{i,j}$ between these vectors to determine mutual reachable distances. These distances measure the true density distribution of the model

parameters for each client. Benign clients typically have a higher parameter density, whereas malicious clients exhibit a lower density.

$$e_{i,j} = \frac{\langle \Delta\omega_i, \Delta\omega_j \rangle}{\|\Delta\omega_i\|\|\Delta\omega_j\|} \ \forall i, j \in c, \tag{5}$$

$$d_{mr}(\omega_p, \omega_q) = \max\{d_{core}(\omega_p), d_{core}(\omega_q), (e_{(1,1)} \dots e_{(M,M)})\}. \tag{6}$$

According to[6], $d_{core}(x)$ is the distance from x to its m_pts-nearest neighbor.

(3) We construct minimum spanning tree, which forms a weighted graph based on the parameters of all the clients. Note that the weight of the edges between any two points is equal to the $d_{mr}(\omega_p, \omega_q)$ of these points.

(4) We sort the edges of the tree in increasing order based on the distance. Then, we iterate through the sorted edges, compressing the tree nodes to normalize the parameter regions assigned to each client. The parameter uses the weights of the edges and the number of coverage points to calculate the stability value. This stability value is utilized by HDBSCAN to identify cluster candidates with the most homogeneous density and to specify the number of malicious clients.

The above steps allow us to detect and mark malicious clients to separate them from other clients.

3.2 Parameter Correction Algorithm

In general CFL, there is a high probability that malicious client will "honestly" train a model, but then only add perturbations to 10 % of the model's parameters[2]. This allows the adversary's model to still achieve no less than 90% confidence in the parameters. To address this, we introduce a residual-based iterative confidence weight repair algorithm, which utilize benign parameters and repair the poisoned model effectively. Fig. 4 shows our algorithm in detail.

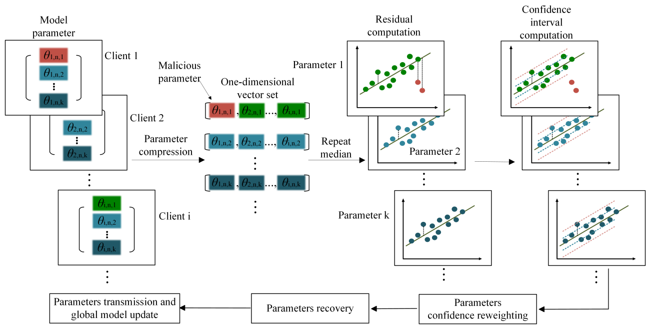


Figure 2: Overview of the our Parameter Correction Algorithm in CFL.

We are inspired by the Iteratively Reweighted Least Squares (IRLS) algorithm in M-estimation[5], which is commonly used in deblurring algorithms in image processing to address the effects of non-normally distributed noise in images. Similarly, we adapt this idea to our attack detection algorithm to tackle the model poisoning problem by malicious clients. As Fig. 2 shows, we assume that a total of M clients participate in the training, and the model parameters uploaded by each client are denoted as $\theta_{i,n,k}$, where $i \in \{1, 2, \dots, M\}$. Whenever our algorithm is executed, the server extract the model parameters with the same labels of each client and merges them into the sets θ_k respectively (k is the index of model parameters type), and the subscripts corresponding to $\theta_{i,n,k}$ are $\delta_{i,n,k}$, each parameter in θ_k with the same labels is passed through the server to generate a linear regression estimator. For the binary linear regression model $Y = A + BX + \varepsilon$, the slope $\hat{B}$ and intercept $\hat{A}$ are estimated using the repeated median strategy:

$$\hat{B}_{n,k} = Med^2\{(\theta_{j,n,k} - \theta_{i,n,k})/(\delta_{j,n,k} - \delta_{i,n,k})\}, \tag{7}$$

$$\hat{A}_{n,k} = Med^2\{(\delta_{j,n,k}\theta_{i,n,k} - \delta_{i,n,k}\theta_{j,n,k})/(\delta_{j,n,k} - \delta_{i,n,k})\}, \tag{8}$$

where $j \in \{1, 2, \dots, M\}$. The residuals of the same model parameters for different clients are obtained by estimating the two parameters above:

$$\varepsilon_{n,k} = \theta_{n,k} - \hat{B}_{n,k}\theta_{n,k} - \hat{A}_{n,k}. \tag{9}$$

Because of the different data ranges for the different parameters, the standardised residuals $\sigma_{i,n,k}$ were obtained by normalising $\epsilon_{n,k}$:

$$\sigma i, n, k = \frac{\varepsilon_{i,n,k}}{\tau_{n,k}}, \tag{10}$$

$$\tau_{n,k} = Med\{\kappa|\varepsilon_{n,k}|(1 + \frac{M}{2(M-1)})\}. \tag{11}$$

The confidence level of the kth model parameter for the ith client under the nth round is further obtained from the obtained normalised residuals:

$$\mu_{i,n,k} \leftarrow \frac{\sqrt{1 - diag(H_{n,k})}}{\sigma_{i,n,k}} \Psi(\frac{\sigma_{i,n,k}}{\sqrt{1 - diag(H_{n,k})}}). \tag{12}$$

The hat matrix is:

$$H_{n,k} = \delta^*_{n,k}\{(\delta^*_{n,k})^T \delta^*_{n,k}\}^{-1}(\delta^*_{n,k})^T, \tag{13}$$

$$\delta^*_{n,k} = [\delta^{(1)}_{n,k}\,\delta^{(2)}_{n,k} \dots \delta^{(M)}_{n,k}]^T. \tag{14}$$

Ψ here denotes the confidence interval, where the confidence level of the model parameters within the confidence interval is set to 1, satisfying:

$$\Psi(x) = max\{-\lambda\sqrt{2/M}, min(\lambda\sqrt{2/M}, x)\}, \tag{15}$$

The hyper-parameter λ can be adjusted to indirectly increase the server's trust in benign clients. To account for the inherent model heterogeneity among these clients, we introduce a confidence decay interval based on the original confidence interval and set a threshold v for smooth transitions of parameter confidence between the confidence and non-confidence intervals. The model parameter confidence is then governed by the following equation.

$$z_{i,n,k} = \begin{cases} 1 & \text{, if } \left|\sigma_{i,n,k}\right| \le \frac{\sqrt{2}\lambda}{\sqrt{K}} \text{ ;} \\ \left|\frac{\sqrt{2}\lambda}{\sqrt{K}\sigma_{i,n,k}}\right| & \text{, if } \frac{\sqrt{2}\lambda}{\sqrt{K}} < \left|\sigma_{i,n,k}\right| \le \left|\frac{\sqrt{2}\lambda v}{\sqrt{K}}\right|. \\ 0 & \text{, otherwise.} \end{cases} \tag{16}$$

Therefore, the final model weights that can be obtained after our iterative confidence reweighting algorithm can be expressed as:

$$\vartheta_{i,n,k} = \theta_{i,n,k}\mu_{i,n,k}z_{i,n,k}. \tag{17}$$

Obviously, the tensor of $\vartheta_{i,n,k}$ is probably all 0, it is represented as the kth model parameter of this client having parameter confidence of 0, in this case we affirm that the model parameter has been tampered.

We reassign the value of the parameter k that was poisoned in this client i in the nth round of training, and transfer it to the target client when the server sends the parameters:

$$Recovery(\theta^*_{i,n,k}) = \sum_{i=M/2}^{M-M^*} \frac{1}{M-M^*}\vartheta_{i,n,k}. \tag{18}$$

4 Experiments

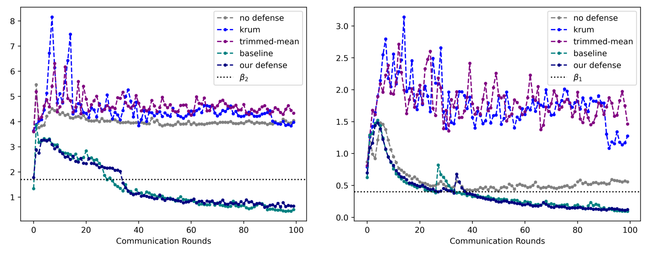
In this section, we give simulation results. The objectives of our experimental evaluation are as follows: (a) the performance of our frame relative to other existing defense methods, (b) our frame performance of successful defense and clustering, (c) model robustness of using our defense frame against different attacks.

4.1 Experiment Setup

To begin with, we implement CFL framework based on Pytorch, and conduct our experiments on a server equipped with NVIDIA RTX 2070 SUPER with 8GB RAM. We perform our experiments on the well-known EMNIST, Fashion-MNIST and CIFAR-10 datasets. All the model training tasks are performed using Pytorch. We also adopt three known robust methods: Krum, Trimmed-mean,and Fedavg for comparison with our approach.

4.2 Experiment Evaluation

As shown in Fig. 3, we compare the robustness of the clustering function of our defense method concerning existing methods, and we select two indicators: the maximum inter-cluster distance and the average intra-cluster distance, they indicate whether we can successfully defend and clustering.



(a) Maximum inter-cluster distance (b) Mean intra-cluster distances

Figure 3: Maximum inter-cluster distance and mean intra-cluster distances under different defense methods.

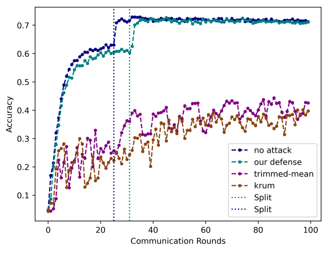
In this set of experiments, the hyper parameters $\beta_1 = 0.4$ and $\beta_2 = 1.7$, the number of clients M is 10, in which there exists one malicious client, and the number of training rounds N is 100, we follow IRLS set κ to 1.48, set Γ to 3.0. According to the setting in CFL, the system starts clustering when the maximum inter-cluster distance is greater than β_2 and the average intra-cluster distance is less than β_1 satisfied. In Fig.3-(a), our defense is below β_2 and converges after 30 rounds, similar to the CFL without attacks, indicating successful clustering before 30 rounds. In Fig.3-(b), our defense clearly outperforms the other methods, achieving convergence and satisfying the clustering condition 2.

Table 2: Simulation result of training accuracy(%)

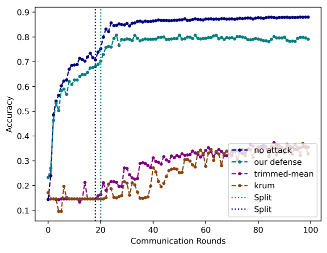
		sign attack			uv attack		
		5%	10%	20%	5%	10%	20%
EMNIST	No defense	73.16	33.31	32.18	66.79	52.61	26.89
	Krum	52.03	41.26	41.89	52.40	41.47	39.14
	Trimmed-mean	47.47	41.59	41.92	52.06	40.84	39.50
	Fedavg	64.11	23.63	15.96	64.84	46.80	20.89
	Our defense	71.89	70.02	70.72	72.45	71.36	64.26
	CFL	72.66					
Fashion MNIST	No defense	83.58	34.86	26.56	72.81	42.30	41.28
	Krum	48.43	52.93	49.30	63.47	57.08	47.96
	Trimmed-mean	64.22	57.30	48.38	64.38	70.55	52.95
	Fedavg	83.89	84.79	24.29	84.48	85.83	27.19
	Our defense	87.77	85.50	79.49	88.25	85.44	80.08
	CFL	88.06					
CIFAR 10	No defense	41.66	20.87	19.09	43.99	35.80	46.27
	Krum	22.76	17.05	12.89	35.15	20.43	20.42
	Trimmed-mean	20.67	17.26	15.33	24.97	19.25	19.03
	Fedavg	25.43	23.60	18.45	29.48	26.64	25.13
	Our defense	48.03	63.93	55.44	49.34	60.55	57.68
	CFL	70.52					

We evaluate the effects of different defense algorithms in CFL scenarios with 5%, 10%, and 20% malicious clients in Table 2. Our method achieves training accuracy close to that of models without attacks and outperforms others.

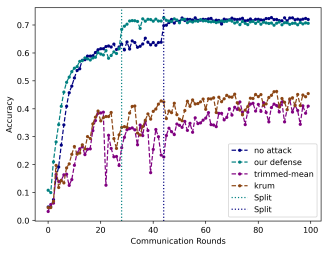
As shown in Fig. 4, we compare training accuracy and clustering results across various defense methods and datasets. The dotted line represents the clustering process in Fig. 4-(a). Green points represent our defense's clustering, blue



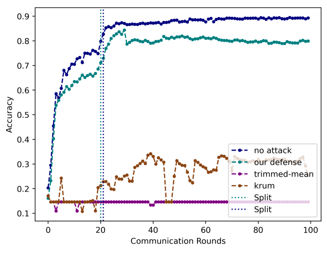
(a) Training with EMNIST under uv attack



(b) Training with Fashion-MNIST under sign attack



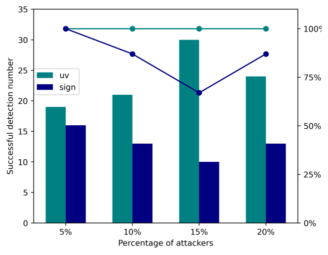
(c) Training with EMNIST under uv attack



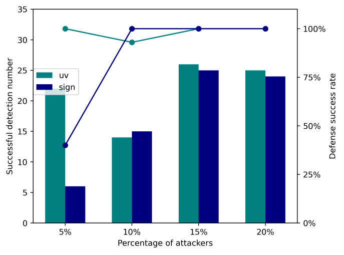
(d) Training with Fashion-MNIST under sign attack

Figure 4: Comparison of CFL and poisoned CFL under different defense methods with various datasets and models.

points indicate no-attack scenarios. In contrast, Krum and Trimmed-mean fail to improve clustering accuracy. Our algorithm effectively restores clustering after attacks.



(a) attack conv1



(b) attack conv2

Figure 5: Comparison of different model parameters identified by our defense method after attack(First 30 rounds).

Fig. 5 compares the effectiveness of our defense algorithm in identifying malicious clients during the first 30 rounds. We tested with 5%, 10%, 15%, and 20%, malicious clients and evaluated the identification results under various attack methods. The bar graph shows successful identifications, while the line graph reflects the defense success rate post-initiation.

5 Conclusion

This paper presents a defense framework for CFL clustering failure. The method includes a HDBSCAN-based malicious client detection algorithm and a residuals based weight reweighting algorithm. Our algorithm realizes the detection and the location of poisoning parameters. Experimental results show that this method can effectively defend against adversary attacks and significantly improve the robustness of CFL. Compared with the existing defense methods, it has a stronger adaptive model recovery ability, thus guaranteeing the performance of the global model.

References

[1] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-Based Clustering Based on Hierarchical Density Estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining.*

[2] Shuhao Fu, Chulin Xie, Bo Li, and Qifeng Chen. 2019. Attack-Resistant Federated Learning with Residual-based Reweighting. (2019). arXiv:1912.11464

[3] Lina Ge, Haiao Li, Xiao Wang, and Zhe Wang. 2023. A review of secure federated learning: Privacy leakage threats, protection technologies, challenges and future directions. *Neurocomputing* 561 (2023), 126897.

[4] Xuechao He, Jiaojiao Zhang, and Qing Ling. 2023. Byzantine-Robust and Communication-Efficient Personalized Federated Learning. In *ICASSP 2023.* 1–5.

[5] Paul W. Holland and Roy E. Welsch. 1977. Robust regression using iteratively reweighted least-squares. *Communications in Statistics - Theory and Methods* 6, 9 (1977), 813–827.

[6] Jingya Liu, Shiteng Sun, and Chen Chen. 2021. Big data Analysis of Regional Meteorological Observation Based : on Hierarchical Density Clustering Algorithm HDBSCAN. In *AINIT.* 111–116.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Vol. 54. 1273–1282.

[8] Hasin Us Sami and Başak Güler. 2023. Secure Aggregation for Clustered Federated Learning. In *2023 IEEE International Symposium on Information Theory (ISIT).* 186–191.

[9] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. 2021. Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints. *IEEE Transactions on Neural Networks and Learning Systems* 32, 8 (2021), 3710–3722.

[10] Felix Sattler, Klaus-Robert Müller, Thomas Wiegand, and Wojciech Samek. 2020. On the Byzantine Robustness of Clustered Federated Learning. In *ICASSP 2020.* 8861–8865.

[11] Virat Shejwalkar and Amir Houmansadr. 2021. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. In *28th Annual Network and Distributed System Security Symposium.*

[12] Geming Xia, Jian Chen, Chaodong Yu, and Jun Ma. 2023. Poisoning Attacks in Federated Learning: A Survey. *IEEE Access* 11 (2023), 10708–10722.

[13] Zuobin Xiong, Zhipeng Cai, Daniel Takabi, and Wei Li. 2022. Privacy Threat and Defense for Federated Learning With Non-i.i.d. Data in AIoT. *IEEE Transactions on Industrial Informatics* 18, 2 (2022), 1310–1321.