SECURE FEDERATED LEARNING AGAINST MODEL POISONING ATTACKS VIA CLIENT FILTERING

Duygu Nur Yaldiz* , **Tuo Zhang***, **Salman Avestimehr**
Department of Electrical and Computer Engineering
University of Southern California
yaldiz@usc.edu, tuozhang@usc.edu, avestime@usc.edu

ABSTRACT

Given the distributed nature, detecting and defending against the backdoor attack under federated learning (FL) systems is challenging. In this paper, we observe that the cosine similarity of the last layer's weight between the global model and each local update could be used effectively as an indicator of malicious model updates. Therefore, we propose `CosDefense`, a cosine-similarity-based attacker detection algorithm. Specifically, under `CosDefense`, the server calculates the cosine similarity score of the last layer's weight between the global model and each client update, labels malicious clients whose score is much higher than the average, and filters them out of the model aggregation in each round. Compared to existing defense schemes, `CosDefense` does not require any extra information besides the received model updates to operate and is compatible with client sampling. Experiment results on three real-world datasets demonstrate that `CosDefense` could provide robust performance under the state-of-the-art FL poisoning attack.

1 INTRODUCTION

The gist of Federated Learning (FL) is to train a model coordinated by a server while preserving the clients' data privacy Zhang et al. (2021). However, this substantial property introduces new challenges. Since the server does not have access to the client data due to privacy concerns, FL is vulnerable to data or model poisoning attacks, in which the attacker send corrupted updates and contaminates the global model. Given the distributed nature of FL, it is challenging to detect and correct these failures under the vanilla FL framework McMahan et al. (2016); Zhang et al. (2022).

Several solutions have been proposed to defend the server from model poisoning attacks to relax the security challenge for the FL framework. Server-side robust aggregation approaches aim to detect outliers by inspecting the client updates, and filtering the malicious updates before model aggregation such as Blanchard et al. (2017). Besides completely filtering out before model aggregation, approaches proposed by Xu & Lyu (2021); Cao et al. (2021); Regatti et al. (2021); Fung et al. (2020); Prakash et al. (2020) diminish the aggregation coefficients of the clients that are likely to be malicious. However, existing approaches have some critical drawbacks or unfeasible assumptions as we summarize in Table 1.

In this work, we propose `CosDetect`, a cosine similarity based outlier detection algorithm, to tackle the fundamental issues of existing defense methods. We provide an intriguing finding that the weight inside the last layer of the local model update is more sensitive to the local data distribution than other layers. Based on this crucial observation, we propose that the last layer of local updates from the malicious clients should be outliers compared to the ones from the benign clients. By calculating the cosine similarity of the last layer between each collected model update and the last global model, it is possible to filter the poisoning updates before model aggregation.

As shown in Table 1, the proposed cosine similarity based outlier detection scheme, though simple, has equipped `CosDetect` with multiform merits compared to prior strategies: (1) `CosDetect` does not require representative benign data at the server to distinguish the malicious updates. It is only built on accessing the global model parameters and the clients' updates. (2) `CosDetect`

*The first two authors contributed equally.

Table 1: Comparison of `CosDefense` with existing backdoor defense FL methods.

	Compatibility with Client Sampling	**Validation Data at the Server**	**Information of Attacker Number**	**Client Score Maintenance at the Server**
Krum Blanchard et al. (2017)	√	×	√	×
Multi-Krum Blanchard et al. (2017)	√	×	√	×
Median Yin et al. (2018)	√	×	×	×
RFFL Xu & Lyu (2021)	×	×	×	√
FoolsGold Fung et al. (2020)	√	×	×	√
FLTrust Cao et al. (2021)	√	√	×	×
ByGars Regatti et al. (2021)	√	√	×	√
SageFlow Park et al. (2021)	√	√	×	×
CosDefense (Our Method)	✔	✖	✖	✖

does not need the precise number of malicious clients per communication round in advance for robust performance. (3) `CosDetect` does not rely on the previous clients' records to cluster the malicious clients. Instead, `CosDetect` eliminates the outliers only based on the current round information, which makes our algorithm compatible with client selection and brings more flexibility for the implementation.

2 Approach Overview

2.1 Layer-wise Cosine Similarity for Model Weights

We first focus on a critical question: *how would the attack be reflected on the server side during FL training?* Under the vanilla FL setting McMahan et al. (2016), the only information the server holds during the training are the collected model updates and the following aggregated model. A previous study Zhao et al. (2020) notes that label information for the training data can be computed analytically from the gradients of the last layer inside the machine learning model under the centralized setting. Following this direction, one intriguing finding from our empirical study is that compared to other layers, the last layer's weight is more sensitive to the input data distribution. We quantify the similarity of model weights by cosine similarity, which is the dot product of the two vectors divided by the product of individual norms as $cos(\alpha) = \frac{<x,y>}{||x||\cdot||y||}$ where α is the angle between vectors x and y. Figure 1 shows the average cosine similarity for each layer in the model across independent clients. In this experiment, ten clients train a four-layer CNN-based model independently without any model synchronization on the MNIST dataset, which has been non-iid partitioned (See Section 3.1 for details). We observe that with the increment of the iteration, the input-side layers show higher similarity than the output-side layers, and the last layer has the lowest similarity score, because the local data distribution among all clients mainly varies on the label distribution. These observations provide critical insight that the local data label distribution could be efficiently reflected in the last layer's weight compared to the other layers.

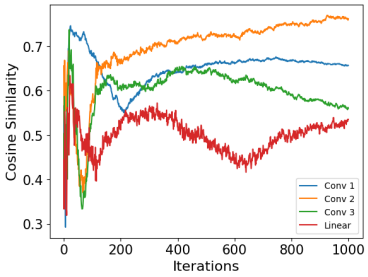

Figure 1: The average cosine similarity of model weight in various layers between one client and the other nine clients. All 10 models are trained independently for 1000 iterations without synchronization.

2.2 CosDefense: Filtering the Malicious Model Update via Cosine Similarity

Based on the crucial observation we described in the previous section, we propose that the last layer of the local models from the attackers should be outliers compared to the ones from the benign models. Therefore, it is possible to filter the malicious model update on the server by calculating the cosine similarity between each collected model update and the last global model. When calculating

the cosine similarity between global model weights and local model update, we have the following:

$$cos(\alpha_i) = \frac{< \theta_t, g^i_{t+1} >}{||\theta_t|| \cdot ||g^i_{t+1}||} = \frac{< \theta_t, \theta^i_{t+1} - \theta_t >}{||\theta_t|| \cdot ||\theta^i_{t+1} - \theta_t||} = \frac{< \theta_t, \theta^i_{t+1} > - < \theta_t, \theta_t >}{||\theta_t|| \cdot ||\theta^i_{t+1} - \theta_t||} \quad (1)$$

where α_i denotes the angle between global model weights (θ_t) and local model update of client i (g^i_{t+1}). If all the participated clients are benign nodes, as the communication round t goes to infinity, the global model should be converged to an optimal point based on the received local model update. Therefore, the cosine similarity between global model weights and the local model update tends to be decreasing during the training, as the difference between θ_t and θ^i_{t+1} (local model for client i) goes smaller for each client.

However, the cosine similarity scores from malicious clients do not follow this phenomenon. As their aim is to prevent convergence, their local optimization direction would be different compared to the benign client models or the global model. As a result, the difference between θ_t and θ^i_{t+1} from malicious nodes becomes more extensive than the benign nodes, making the cosine similarity score between the malicious client update and the global weights tend to be increasing during the training.

To demonstrate our thoughts, we trace the average cosine score of the last layer local updates and that of the global model under benign and poisoning attack cases (Figure 2). Under both cases, the total client number is 100, and the sampling rate is 0.1 in each round. For the poisoning attack case, we launch the IPM attack Xie et al. (2019) at round 200, with 30% of the participants being malicious clients. For a better illustration, Figure 2 is smoothed by the moving averaged filter with window size 40. The average cosine similarity increases sharply as the attack happens and is much higher than the scores under the all-benign case, which confirms our speculation.

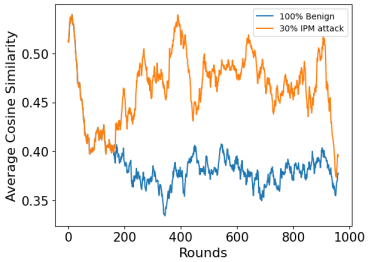

Figure 2: The trace of the average cosine similarity score of the last layer's weight between all received local updates and global model.

We observed that cosine similarity between global model parameters and the benign local model updates tends to be smaller during the training, which indicates that the direction of benign updates is less than perpendicular to the global model, making it move toward a different direction in each round. However, the average cosine similarity scores increase after the attack, indicating that cosine similarity scores from malicious clients are much larger than the scores from benign clients. The large cosine similarity values represent that malicious client updates are aligned with the global model parameters and do not update the global model in different directions but keep it still. As a result, if there are enough malicious clients in the system, their updates considerably diminish the benefit of benign client updates, preventing the convergence of the global model.

Following these directions, we propose `CosDefense`, a server-side clustering-based defense method as shown in the Appendix C. In each round t, after the server receives all the updates from the sampled clients, it calculates the cosine similarity between the last layer's global weights and the last layer's local updates for each client. Then, the server clusters clients based on their cosine similarity scores either as malicious or benign. If the cosine similarity score of a client is much higher than others, then the server labels that client as malicious for round t and exclude that client from aggregation. `CosDefense` only detects malicious clients before aggregation and excludes them, allowing the server to perform any aggregation method with benign client updates. Hence, `CosDefense` algorithm is compatible with any aggregation method.

3 Evaluations

3.1 Experiment Setup

We conduct the evaluations on three datasets and two models: MNIST LeCun et al. (1998) and Fashion-MNIST Xiao et al. (2017) with a four-layer CNN-based model following the previous related work Li et al. (2022), and CIFAR-10 Krizhevsky (2009) with ResNet-18 model He et al. (2015). We generate the non-iid partition for all datasets following previous FL works Li et al. (2022); Fang et al. (2019) with default q as 0.5, where the higher q represents a higher non-iid degree. We compare our

defense method with two representative robust aggregation rules: Krum Blanchard et al. (2017) and Clipping-Median. We do not include the baselines such as FLTrust Cao et al. (2021) or RFFL Xu & Lyu (2021) because they either need validation data on the server or do not compatible with the client sampling. Further details of the baseline implementation and parameter selection can be found in Appendix D

3.2 Defense Performance with the State-of-the-Art Model Poisoning Attack

To evaluate the defense performance, we evaluate the `CosDefense` on top of the state-of-the-art model poisoning attack methods, Inner Product Manipulation (IPM) Xie et al. (2019) attack. In the following experiments, we set the attacker control 30% of the participants as malicious nodes, and the attack would launch at round 200. The server randomly samples 10 clients from 100 for local model training in each communication round.

Evaluation Results: Figure 3 shows the accuracy curve over 1000 communication rounds for Krum, Clipping-Median, and `CosDefense` for MNIST, Fashion-MNIST, and CIFAR-10. The detailed numerical results and ablation study related to the non-iid degree are shown in Appendix E. We have three observations from the results. (1) As Krum and Clipping-Median use all layers of the updates, the results of `CosDefense` indicate that the similarity of the last layer between the collected model update and global model is more sensitive to attacks, which could effectively be used to filter out the malicious model updates. (2) After the IPM attack begins, all the defense strategies drop the accuracy significantly. However, the proposed `CosDefense` strategy has a much faster convergence speed and shorter recovery time to resume the learning compared to Krum and Clipping-Median. (3) For the most challenging dataset CIFAR-10, after the IPM attack happens, `CosDefense` not only recovers the best accuracy before round 200, but also has an increasing trend on the curve, while other defense baselines even could not reach the best accuracy before the attack.

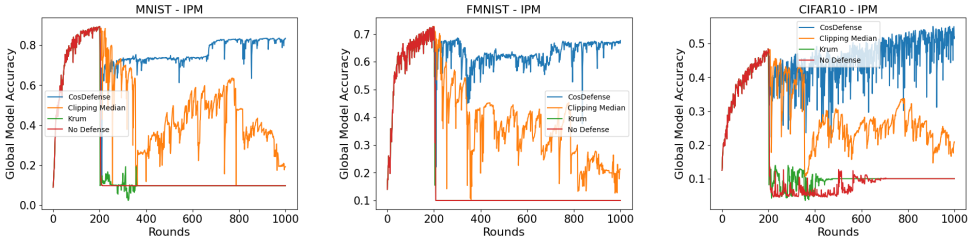

Figure 3: Accuracy performance over 1000 rounds for Krum, Clipping-Median, `CosDefense`, and FedAvg on MNIST (left), FMNIST (middle), and CIFAR-10 (right) under IPM attack.

Impact of the number of the attackers. Some previous works on untargeted model poisoning assume that there is a large fraction of attackers among participants Fang et al. (2019); Xie et al. (2019). Therefore, we also investigate the number of attackers' impact on defense performance. In this section, we conduct experiments on non-iid partitioned MNIST with $q = 0.5$, and vary the number of attackers from 10% to 40% of the total participants. Results of this study are provided in Figure 4, showing the final global model accuracy of defense methods under various settings. We observe that the two defense baselines remain robust when the ratio of attackers is less than 30%. As the ratio reaches 30%, both Krum and Clipping Median collapse on the accuracy performance and fail to defend the model convergence, while `CosDefense` still provides robust accuracy performance. One notable point is that the `CosDefense` achieves the best accuracy compared to other baselines among all different ratios of attackers. The experiment results demonstrate that `CosDefense` is robust and reliable under both small-scale and large-scale attack scenarios.

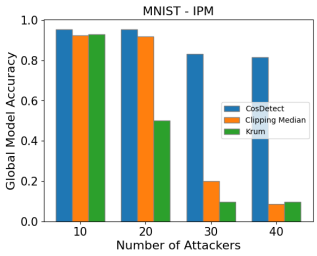

Figure 4: Accuracy performance on MNIST dataset under different numbers of attackers perform IPM attacks.

4 CONCLUSION

We presented `CosDefense`, a cluster-based defense scheme that could filter the malicious updates out of the model aggregation on the server. Experiment results on real-world datasets demonstrate that `CosDefense` can provide robust performance under the state-of-the-art FL poisoning attack. We picture the limitations and potential future work in Appendix F.

5 ACKNOWLEDGEMENT

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0156, ARO award W911NF1810400, ONR Award No. N00014-16-1-2189. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. ICML, 2012.

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. NeurIPS, 2017.

Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. NDSS, 2021.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security Symposium*, 2019.

Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2020.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278–2324, 1998.

Henger Li, Xiaolin Sun, and Zizhan Zheng. Learning to attack federated learning: A model-based reinforcement learning attack framework. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=4OHRr7gmhd4`.

H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016.

Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Toward robustness and privacy in federated learning: Experimenting with local and central differential privacy. arXiv, 2020.

Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. Sageflow: Robust federated learning against both stragglers and adversaries. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 840–851, 2021.

Saurav Prakash, Hanieh Hashemi, Yongqin Wang, Murali Annavaram, and Salman Avestimehr. Secure and fault tolerant decentralized learning. 2020.

Jayanth Regatti, Hao Chen, and Abhishek Gupta. Bygars: Byzantine sgd with arbitrary number of attackers. International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML, 2021.

Jingwei Sun, Ang Li, Louis DiValentin, Amin Hassanzadeh, Yiran Chen, and Hai Li. Fl-wbc: Enhancing robustness against model poisoning attacks in federated learning from a client perspective. NeurIPS, 2021.

Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can you really backdoor federated learning? arXiv, 2019.

Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *Computer Security – ESORICS 2020*, pp. 480–501. Springer International Publishing, 2020. ISBN 978-3-030-58951-6.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *35th Conference on Uncertainty in Artificial Intelligence*, 2019.

Xinyi Xu and Lingjuan Lyu. A reputation mechanism is all you need: Collaborative fairness and adversarial robustness in federated learning. International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML, 2021.

Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5650–5659. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/yin18a.html`.

Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and Salman Avestimehr. Federated learning for the internet of things: Applications, challenges, and opportunities. *IEEE Internet of Things Magazine*, 5:24–29, 2021.

Tuo Zhang, Tiantian Feng, Samiul Alam, Sunwoo Lee, Mi Zhang, Shrikanth S Narayanan, and Salman Avestimehr. Fedaudio: A federated learning benchmark for audio tasks. *arXiv preprint arXiv:2210.15707*, 2022.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *ArXiv*, abs/2001.02610, 2020.

Giulio Zizzo, Ambrish Rawat, Mathieu Sinn, and Beat Buesser. Fat: Federated adversarial training. NeurIPS Workshop on Scalability, Privacy, and Security in Federated Learning (SpicyFL), 2020.

A PRELIMINARY

A.1 FEDERATED LEARNING

In this paper, we consider the federated learning setting that is similar to the vanilla FedAvg McMahan et al. (2016), in which the FL system is composed of a server with $\mathcal{K}$ clients, whose data is only locally kept without sharing. The clients cooperate in training a global model $\mathcal{W}_g$ with parameters θ. We consider the following distributed optimization problem: $\min_{\theta} f(\theta)$, where $f(\theta) := \sum_{i=1}^{\mathcal{K}} p_i F_i(\theta)$. The $F_i(\cdot)$ represents the local objective of client i and p_i denotes the aggregation weight of client i satisfying $p_i \geq 0$ and $\sum_{i=1}^{\mathcal{C}} p_i = 1$.

The federated learning algorithm runs as follows: in each communication round t, the server randomly selects a subset S_t from available clients for client sampling and sends the global model parameters θ_t to the selected clients. Each selected client $i \in S_t$ initializes its local model parameters as θ_t and performs local model updates $\theta_{t+1}^i = \theta_t - \eta \frac{1}{B} \sum_{z \in b_i} \nabla_\theta \ell(\theta_t; z)$, where η is learning rate, b_i is a mini-batch of size B randomly sampled from local data $\mathcal{D}_i$. After local update finishes, each client $i \in S_t$ send the model update $g_{t+1}^i := \theta_{t+1}^i - \theta_t$ to the server. Lastly, the server performs the aggregation function over the received model updates and obtains a new global model such that $\theta_{t+1} \leftarrow \theta_t - G_t$, where $G_t \leftarrow Aggr(\{g_{t+1}^i\}_{i \in S_t})$. The newly updated model parameters θ_{t+1} is then used to perform the next round of FL training. The FL training is ended when the round number reaches $\mathcal{T}$.

A.2 THREAT MODEL

We assume that the attacker's goal is to diminish global model accuracy regardless of specific inputs or classes, in which malicious clients perform untargeted attacks. In this work, we assume that p percentage of the total number of clients is malicious, where $0 \leq p \leq 1$. Let the malicious clients be denoted by $\mathcal{M}$ and the set of malicious clients as $\mathcal{A}$. All malicious clients share the same purpose and perform the same type of attack. They are either coordinated by an external attacker or by a leader client. During the FL training, the malicious attackers send crafted local updates $\{\widetilde{g}_k^t\}_{k \in \mathcal{A}}$ to the server to purposely maximize the global optimization function $f(\theta)$. The crafted local updates are generated from their local data distributions, which differ from the benign clients' local distributions. The illustration of the threat model is shown in Figure 5.

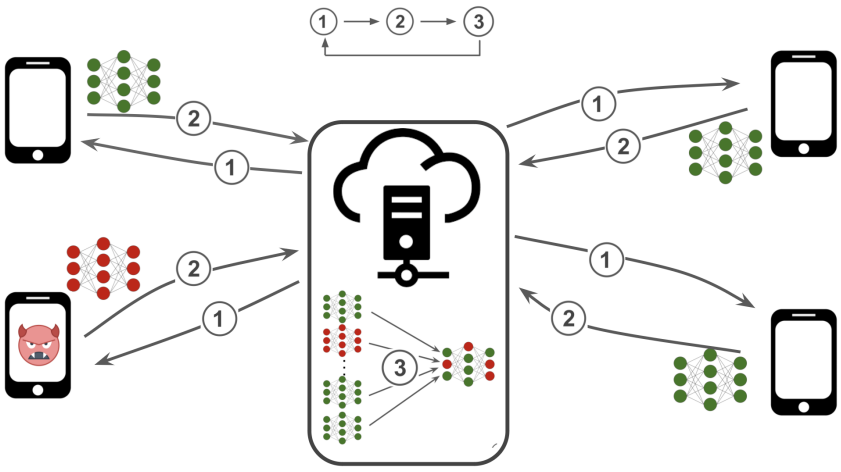

Figure 5: Threat model of the data poisoning attack in FL. After receiving the global model from the server, the malicious clients purposely upload contaminated model updates to poison the model.

B RELATED WORKS

Poisoning Attacks in FL. In federated learning, the attackers may control a fraction of the clients and manipulate the local data or training process to purposely poison the global model performance by injecting the wrong local model update. This kind of attack is called the model poisoning attack, in which the attacker contaminates the global model. Poisoning attacks are mainly categorized into targeted and untargeted attacks according to the attacker's aim. Targeted attacks aim to keep global model performance high, while the model considerably misbehaves on a specific set of inputs. This can be done either by injecting backdoor triggers to the inputs regardless of the original class or flipping the labels of only one class to a similar label (e.g., flipping 1 to 7 in MNIST or 'cat' to 'dog' in CIFAR10). On the other hand, the purpose of untargeted attacks is to decrease the global model's performance significantly. Label-flip Biggio et al. (2012), and gradient modification (e.g., sending random noise to the server instead of real update) Blanchard et al. (2017) attacks are examples of this type. Moreover, Li et al. (2022) utilizes reinforcement learning to create an optimal untargeted attack, however, building the framework to generate online attack is costly in terms of both time and computation. Another state-of-the-art untargeted model poisoning attack is Inner Product Manipulation (IPM) proposed by Xie et al. (2019). The aim of the IPM is to modify the local updates of the malicious clients such that the inner product between the true gradient and the aggregated updates is negative.

Defenses for Poisoning Attacks in FL. Several solutions have been proposed to mitigate the effect of the attacks, so that model performance is not negatively affected. Client-side defenses modify the local training algorithm and rely on benign clients Zizzo et al. (2020); Sun et al. (2021). For example, Sun et al. (2021) finds the model parameters that the attack is hidden and perturbs those parameters during local training of benign clients. Some server-side defense methods decrease the aggregation weights of the malicious client updates Regatti et al. (2021); Cao et al. (2021); Xu & Lyu (2021). While others perform clustering based on different techniques and aggregate only benign clients Blanchard et al. (2017); Tolpegin et al. (2020). For instance, Multi-Krum Blanchard et al. (2017) calculates the total Euclidean distance from the $n - f - 2$ nearest neighbors for each client update, where n is the number of clients and f is the estimated malicious client number. Then, only the $n - f$ updates with the minimum distance is aggregated. Another work Sun et al. (2019) applies norm-clipping to the client updates before the aggregation, based on the observation that norm of malicious updates are large. Also, differential privacy methods (e.g., adding noise to updates) are shown to be mitigating model poisoning attacks Sun et al. (2019); Naseri et al. (2020).

However, some current solutions have critical drawbacks in terms of real-world feasibility. For instance, algorithms proposed by Regatti et al. (2021); Cao et al. (2021); Park et al. (2021) rely on representative benign data on the server, which is hard to accomplish in actual FL deployment, where the server seldom stores the ground-true validation data. Moreover, Xu & Lyu (2021) is incompatible with client selection, an inevitable necessity for cross-device FL where thousands of clients participate. Also, Krum and Multi-Krum Blanchard et al. (2017) require the knowledge of the number of malicious clients in the FL setting , which is not feasible. Lastly, methods proposed by Xu & Lyu (2021); Fung et al. (2020); Regatti et al. (2021) maintains reputation scores for each client on the server side, which is a serious overhead since thousands of clients may participate in the FL system. Overall, there is a gap for a defense solution in the FL setting that does not require server data, is compatible with client selection, and is self-sufficient for detecting malicious clients.

C ALGORITHM

`CosDefense` algorithm is presented in the Algorithm 1. More specifically, after calculating the cosine similarity scores of each client, the server takes their absolute value before and performs min-max normalization, which is beneficial for separating the actual malicious and benign clients better. After these two adjustments, we define the threshold value as the mean of current scores and cluster clients accordingly. If the client score exceeds the mean value, it is labeled malicious. Otherwise, the server considers it benign and aggregates its weights for the next global model.

Algorithm 1 `CosDefense`

Input: Global model parameters θ_t, local updates g^i_{t+1} of sampled clients $i \in S_t$, and aggregation method $Aggr(\cdot)$
Initialize: Benign client set $B = \{\}$ß
for each client i in S_t **do**
  $cs_i \leftarrow |cosine\ similarity(\theta_{t,L}, g^i_{t+1,L})|$
for each client i in cs_i **do**
  $cs_i \leftarrow (cs_i - min(cs))/(max(cs) - min(cs))$
threshold $\leftarrow mean(cs)$
for each client i in S_t **do**
  if $cs_i <$ threshold **then**
    $B \leftarrow B \cup \{i\}$
$G_g \leftarrow Aggr(\{g^i_{t+1}\}_{i \in B})$
$\theta_{t+1} \leftarrow \theta_t - G_t$
Output: θ_{t+1}

D Experimental Details

Datasets and partitioning. Dataset statistics are provided in Table 2. We generate the non-iid partition for all datasets following previous FL works Li et al. (2022); Fang et al. (2019). Specifically, suppose there are C various classes in a dataset. We evenly break all the clients into C groups, in which each group is assigned $1/C$ of training samples as follows. A training point with label c would be assigned to c-th group with probability $q \geq 1/C$ and to each of the rest groups with probability $(1-q)/(1-C)$. The attacking clients would be evenly distributed across C groups. A higher q value stands for a higher non-iid degree. In the experiments above, we set the default q equals to 0.5 and partition the datasets into 100 clusters.

Table 2: Dataset statistics.

Dataset	Train Clients	Train Examples	Test Examples	Input Size
MNIST	100	60,000	10,000	1x28x28
Fashion-MNIST	100	60,000	10,000	1x28x28
CIFAR-10	100	50,000	10,000	3x32x32

Baseline implementations and parameter selections. We follow the same implementation details for Krum and Clipping-Median as in the previous work Li et al. (2022). For Krum, we set the number of the attacker parameter f to its real value in each round for the best performance. For the Clipping-Median, we first apply norm-clipping Sun et al. (2019) then perform coordinate-wise median Yin et al. (2018), since it is shown to be more powerful than the vanilla median Li et al. (2022). Moreover, we provide the results for no defense case, where the server only performs FedAvg aggregation. For all experiments, the communication rounds fix to 1000, and the local training iteration number is 1 with batch size 128. The aggregation function is FedAvg McMahan et al. (2016). The default local learning rate is 0.01 for all experiments.

Configurations. All experiments are conducted by CPU/GPU simulation. The simulation experiments are conducted on a computing server with eight GPUs. The server is equipped with AMD EPYC 7502 32-Core Processor and 1024G memory. The GPU is NVIDIA RTX A4000.

E Experimental Results

The final global model accuracy of each method is provided in the table 3.

Impact of the non-iid degree. In order to investigate the impact of data heterogeneity on `CosDefense`, we partition the MNIST dataset with q equals 0.1, 0.3, and 0.5, respectively, and run all three defense strategies on top of the IPM attack for the evaluation. We set 30% of the devices as malicious clients, the same as in the previous section. As shown in Figure 6, we observe that `CosDefense` is stable with the change of the non-iid degree, and outperforms the other two defense

Table 3: Global model accuracy comparison between No Defense, Krum, Clipping-Median, and `CosDefense`.

	No Defense	Krun	Clipping-Median	`CosDefense`
MNIST	9.8%	9.8%	19.29%	**83.37%**
F-MNIST	10%	10%	21.24%	**67.63%**
CIFAR-10	10%	10%	20.89%	**52.02%**

baselines on accuracy performance among all scenarios. The results indicate that the `CosDefense` is resistant to the impact of data heterogeneity compared to the other defense baselines.

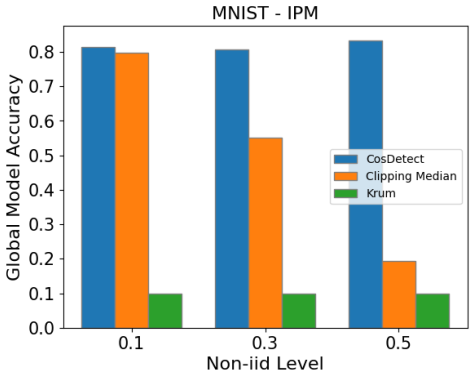

Figure 6: Accuracy performance on MNIST dataset under different non-iid partition levels.

F LIMITATIONS AND FUTURE WORKS.

One of the directions for future work is the combination of `CosDefense` with the existing secure aggregation frameworks. The challenging point is that under the secure aggregation frameworks, each model update is locally encrypted and uninspectable by a server. Instead, only the sum of the model update would be revealed to the server when sufficient updates have been received, which impedes the implementation of `CosDefense` on top of it.