DeMAC: Towards Detecting Model Poisoning Attacks in Federated Learning System

Han Yang[a,*], Dongbing Gu[a] and Jianhua He[a]

[a]School of Computer Science and Electronic Engineering, University of Essex, UK, Colchester, United Kingdom

ARTICLE INFO

Keywords:
Deep Learning (DL)
Federated Learning
Backdoor attacks
Model Poisoning

ABSTRACT

Federated learning (FL) is an efficient distributed machine learning paradigm for the collaborative training of neural network models by many clients with the assistance of a central server. Currently, the main challenging issue is that malicious clients can send poisoned model updates to the central server, making FL vulnerable to model poisoning attacks. In this paper, we propose a new system named DeMAC to improve the detection and defence against model poisoning attacks by malicious clients. The main idea behind the new system is based on an observation that, as malicious clients need to reduce the poisoning task learning loss, there will be obvious increases in the norm of gradients. We define a metric called GradScore to measure this norm of clients. It is shown through experiments that the GradScores of malicious and benign clients are distinguishable in all training stages. Therefore, DeMAC can detect malicious clients by measuring the GradScore of clients. Furthermore, a historical record of the contributed global model updates is utilized to enhance the DeMAC system, which can spontaneously detect malicious behaviours without requiring manual settings. Experiment results over two benchmark datasets show that DeMAC can reduce the attack success rate under various attack strategies. In addition, DeMAC can eliminate model poisoning attacks under heterogeneous environments.

1. Introduction

Federated learning is a distributed machine learning paradigm [1][2][3]. Each client trains a model locally, and then all local model updates are aggregated by a central server to derive a global model. This process is repeated multiple times, and the accuracy of the global model on the main task is gradually improved. FL offers efficiency and scalability compared to centralised training since many clients execute the training in parallel [4]. In particular, FL provides client's privacy as they can keep their training dataset locally [1] rather than sharing it with other participants. Such a secure aggregation mechanism complies with General Data Protection Regulation (GDPR) [5] and protects clients against privacy leakage attacks. Due to FL's potential functions of privacy protection, it has been deployed in the real world. For example, Android Gboard [6] has been installed with FL for next-word prediction. In finance, WeBank [7] has applied FL for credit risk prediction. FL has been widely used in pharmaceutical companies for drug discovery in MELLODDY project [8].

A major challenge faced by FL is that it leaves the door open for malicious clients. A FL system is vulnerable to model poisoning, especially *backdoor attack* that may insert backdoors into the trained global models [9]. The backdoors can make the global machine-learning model misclassify a small set of samples with chosen triggers into targeted labels, while the backdoored global model can show good performance on both main and backdoor tasks.

Existing defences such as [10][11][12] propose Byzantine-tolerant aggregation rules and remove statistical outliers by comparing client local model updates. However, these previous works make some assumptions, such as the data distribution should be IID (Independent and Identically Distributed), which is not valid in non-IID setting [10]. [12] relies on the aggregation of updates using the geometric median, not the standard arithmetic mean aggregation. However, our work shows that this method can be bypassed by malicious clients who carefully design their poisoned updates.

In view of the above drawbacks of the existing works, we propose a system (called DeMAC) to detect and defend against model poisoning attacks from malicious clients. Our design relies on the key finding that genuine clients train their model updates following the main federated training task and their local benign datasets, while malicious

*Corresponding author
hy20497@essex.ac.uk (H. Yang); dgu@essex.ac.uk (D. Gu); j.he@essex.ac.uk (J. He)
ORCID(s):

clients will craft their local model updates trained on the poisoning task and poisoned datasets. To succeed in the poisoning attacks, the adversaries should increase the number of poisoned samples to decrease the training loss of the poisoning task. Therefore, the $L2$-norm of gradients of the poisoned local model updates will be increased. Although the adversaries can reduce the number of poisoned samples and decrease the deviations from benign models' norm of gradients, this may cause the poisoning task to fail. Hence, to measure the $L2$-norm of gradients and capture the abnormal changes, we define a new metric which is called GradScore as the $L2$-norm of gradients in the last layer of the client model updates after the first local epoch training. Using GradScore DeMAC can effectively detect potential malicious clients with abnormally large GradScore values. When the global model training converges, the loss and the norm of gradients of the main federated training task will be small. If there are poisoning attacks in this training stage, the difference between model GradScores of genuine clients and malicious clients will be more obvious. Therefore, DeMAC can easily detect and mitigate malicious clients. As for poisoning attacks starting from an early stage, our experiments also show that DeMAC can work effectively.

Furthermore, to improve the defence performance, a historical record of the contributed global model updates is utilised in DeMAC to help spontaneously estimate the convergence trend of the global model and determine the time to start detecting malicious behaviours. This historical record will store a list of variables within a flexible look-back window size. The variables are the absolute values of two adjacent accuracy values of the global model on the validation set. Once the maximum value among this historical record is below the default threshold, DeMAC would be triggered and start to detect.

We evaluate DeMAC on two benchmark datasets and model-replacement attack [9], distributed attack [13], scaling-scale attack and muti-poisoning attack [14]. Experiment results show that DeMAC can effectively mitigate model-replacement, distributed, and scaling-scale attacks. When malicious clients participate in every training iteration and insert perturbations, the Attack Success Rate (ASR) of the baseline algorithms increases gradually while the ASR of DeMAC keeps at a low level. Therefore, DeMAC can effectively suppress the propagation errors. In brief, our contributions include: 1) We propose DeMAC, a defence system to detect malicious clients and defend against model poisoning attacks via checking the abnormal model updates from potential malicious clients. 2) We utilize the historical record in DeMAC for defence against malicious attacks, which can spontaneously detect malicious clients without manual settings; 3) We extensively evaluate DeMAC by experiments against multiple model poisoning attacks and backdoor attacks on benchmark datasets, which shows high efficiency in defence against malicious attacks in both early and late training stages and significant performance improvement over the existing baseline methods.

The remainder of our paper is organized as follows. Section 2 discusses the research related to poisoning attack and defence. In section 3, we introduce the system and threat models with specific descriptions of adversaries, objectives and requirements for attacks and defences. In Section 4 and 5, we present our novel DeMAC system to defend against model poisoning attacks. We present the evaluation setup in Section 6. In Section 7, the evaluation results of DeMAC are presented. Section 8 concludes the paper and presents our future work.

2. Related Work

Defense mechanisms (against backdoor attacks) in the literature can be roughly classified into two categories: *Byzantine-robust federated learning methods* [10][11][15][16][17][18][19][12], and *anomaly detection-based methods* [20][21][22][23][24]. Byzantine-robust federated learning methods aim to tolerate Byzantine clients failures. In contrast, anomaly detection-based methods attempt to filter potential malicious clients.

2.1. Byzantine-robust federated learning methods

The principle of existing Byzantine-robust defences [10][11] is to train a global model with high performance, even if there are some malicious clients.

Krum [10] tries to find a representative model update as the aggregated model update. Suppose there are n local clients in every iteration. And f among these local clients is malicious. The score for the ith client is calculated as $s_i = \sum_{\mathbf{w}_j \in \Gamma_{i,n-m-2}} \|\mathbf{w}_j - \mathbf{w}_i\|_2^2$, where $\Gamma_{i,n-m-2}$ is the set of $n-m-2$ local clients that have the smallest Euclidean distance to $\mathbf{w}_i$. So the representative model update is the one that has the smallest score. This representative model update will be the global model for the next iteration. Krum attempts to limit the iteration between poisoned and clean models in a single iteration. However, it does not consider compound propagation errors [25]. Therefore, the iterative nature of learning ensures that small deviations at the start of training compound exponentially.

Median [11] is a coordinate-wise aggregation rule. The coordinate-wise median of sorted local models is selected as the aggregated global model update. Instead of using the mean value among local clients, this aggregation rule considers the coordinate median value of the parameters as the corresponding parameter in the global model for the next iteration.

Trimmed-mean [11] is another coordinate-wise aggregation rule. Given the trim parameter $k < \frac{n}{2}$, the server removes the k maximum and minimum values of the coordinates in client model updates and then computes the mean of the remaining $n-2k$ values as the corresponding parameter in the global model for the next iteration. Trimmed-mean relies on the assumption that the coordinate of the attacker would either be the minimum or the maximum value of the corresponding parameters. However, this assumption does not hold for model poisoning attacks [25]. Therefore, even a single attacker can compromise Trimmed-mean.

RFA [12] is a robust aggregation rule based on similarity metrics. RFA aggregates the model updates and appears robust to outliers by replacing the weighted arithmetic mean in the aggregation process with an approximate geometric median. Model-replacement attack [9] is more easily detected by RFA due to the scaling operation [12]. However, by strictly controlling the total weights of the outliers with only a few attackers poisoning a small set in every batch, the attacker model updates can have lower distances and can be assigned higher aggregation weights [13]. By doing so, the attackers can bypass RFA and perform a successful backdoor attack.

2.2. Anomaly detection-based methods

Many existing defences [20][21][22][23][24] follow an anomaly-detection-based strategy and exclude anomalous model updates. FoolsGold [20] defines indicative features. By measuring the cosine similarity on the indicative features and checking the Sybil clones, Sybil attacks can be detected in no-IID data scenarios as Sybils have highly similar updates. However, FoolsGold shows poor performance on one Sybil attack scenario. FLAME [21] uses similar detecting strategies by calculating the angular differences between all model updates. Rather than comparing the probabilities of global models, DeepSight [24] compares the local model updates with the previous global model. However, it does not work in no-IID scenarios. Auror [23] defines indicative features and finds that all the indicative features come from the final layer. Auror assumes that the indicative features from benign clients would have a similar distribution while the indicative features from malicious clients would have an anomalous distribution. However, it does not work in no-IID scenarios.

3. Background

In this section, we give some background knowledge of federated learning and attack strategies against federated learning systems.

3.1. Definition of symbols and corresponding descriptions

The overall definition of symbols and corresponding descriptions is listed in Tab.1.

3.2. Preliminaries

Here, $D = \{(x_i, y_i)\}_{i=1}^{N}$ denotes the training set on local devices, with input vectors $x \in \mathbb{R}^d$ and $y \in \{0,1\}^K$ encoding labels. Each training sample in D is drawn from the unknown distribution $\mathbb{Z}$. Local clients are assumed to have the same architecture neural network model in federated learning. For a chosen neural network model on clients, $p(\mathbf{w}, x) = \sigma(f(\mathbf{w}, x))$ denotes the probability vector of the neural network with activation function σ and weights $\mathbf{w} \in \mathbb{R}^D$. For any probability vector p, let $\ell(p, y)$ denote the loss function.

For any local client, let $\mathbf{w}^0, \mathbf{w}^1, \mathbf{w}^2, ..., \mathbf{w}^t$ be the client updates at iterations of SGD (Stochastic Gradient Descent). $S_0, S_1, .., S_{n-1} \subseteq S$ of size M are mini-batches at one iteration. Here we have

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \sum_{(x,y)\in S} g^t(x, y) \tag{1}$$

$g^t(x, y) = \nabla_{\mathbf{w}^t} \ell(p(\mathbf{w}^t, x), y)$ is the gradient of the loss for a training sample (x, y).

Symbol	Descriptions
$D = \{(x_i, y_i)\}_{i=1}^{N}$	The training set on local devices
N_D	the number of samples in training set D
$p(\mathbf{w}, x)$	The probability vector
$\sigma(f(\mathbf{w}, x))$	Activation function
$\ell(p, y)$	The loss function
$\ell_B(p, \tau)$	The poisoning task loss function
$\mathbf{w}^t$	Weights at local iteration t
$S_0, S_1, .., S_{n-1} \subseteq S$	Mini-batches
$g(x, y)$	The gradient of the loss function
G^t	Global model at global round t
$\{C_1, ...C_n\}$	n clients chosen at global round t
PDR	Poisoned Data Rate
PMR	Poisoned Malicious Clients Rate
m	The number of compromised clients
$\mathbf{w}^{'}$	Compromised client weights
$G^{'}$	Compromised global model
D^P	Poisoned data set on local devices
N_{D^P}	the number of samples in D^P
τ	Targeted label
y	Genuine label
$x^{'}$	Poisoned data
x	Genuine data
η	the local learning rate
γ	The scaling factor
β	The Scaling-Coefficient parameter
p	The pruning rate
σ	The validation threshold
l	The size of sliding window
S	Central server

Table 1
Symbols and Descriptions

3.3. System Setting

We assume that n clients train their local models before sending local updates to the central server. The central server combines these updates by using FedAvg [1]. In addition, all the clients keep their data secret, and no client can intercept training or testing data. The optimization problem of FL is $\min_{G} F(G)$, where $F(G) = \mathbb{E}_{D\sim\mathbb{Z}}[\ell(p(G, x), y)]$ is the expectation of the empirical loss $\ell(p(G, x), y)$ on the local training set D [26].

One iteration of FL training is shown below (see the left part in Fig.1):

- **Step 1:** Synchronizing the global model with local clients: The server sends the current global model G^t to the chosen clients.
- **Step 2:** Training local models: Each client initializes its local model as the global model G^t and trains a local model using its training set $D = \{(x_i, y_i)\}_{i=1}^{N}$. The optimization problem of clients is minimizing $\ell(p(\mathbf{w}, x), y)$, where $\mathbf{w}$ is the local model. By using SGD, the client updates the local model as described in Eq. (1). Then the client sends its updates $\mathbf{w}_i^{t+1}$ to the server.
- **Step 3:** Aggregation: The server the updated global model via aggregating the local updates by some aggregation rules. The FedAvg[1] is given as: $G^{t+1} = G^t + \frac{\eta}{n}\sum_{i=1}^{n}(\mathbf{w}_i^{t+1} - G^t)$

To simulate a non-IID distribution, we assign data to clients according to the Dirichlet distribution [27].

3.4. Attack Strategies

In this paper, we will focus on targeted model poisoning attacks. The adversary manipulates the local models $\mathbf{w}$ to obtain the compromised clients model $\mathbf{w}^{'}$ before being aggregated into the global model G^{t+1}. The adversary wants

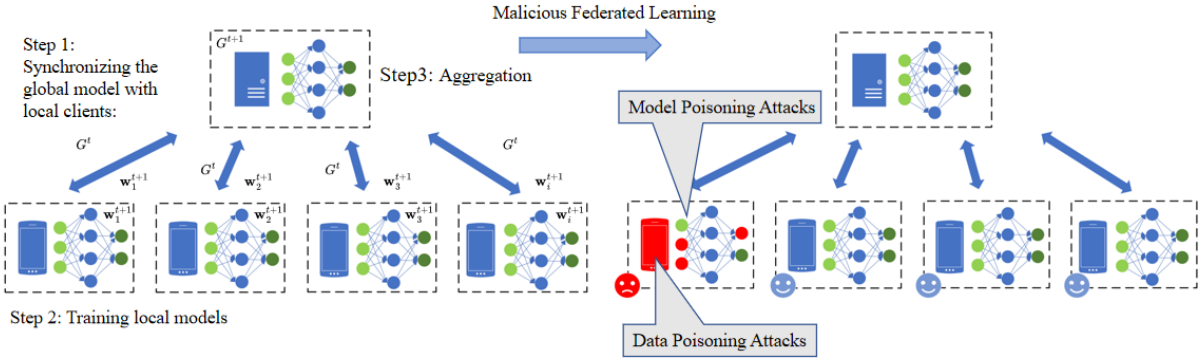


Figure 1: On the left are the three steps of Federated Learning. On the right is the malicious Federated Learning

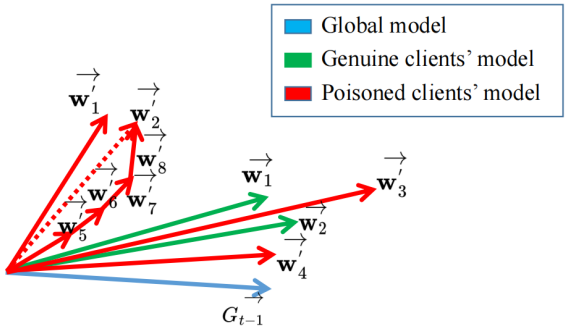


Figure 2: Weight vectors of genuine and poisoned models

the compromised global model $G^{'}$ to behave normally on all samples except for poisoned samples $x^{'} \in D^{P}$. Once the adversary attacks successfully, the compromised global model $G^{'}$ would misclassify poisoned samples into the attacker-chosen label τ rather than the genuine label y.

3.4.1. Data poisoning

In this attack strategy (see the right part in Fig.1), the adversary can only manipulate the training set on local clients by adding triggers into data samples or by changing the labels of a group of attacker-chosen data samples. By varying the Poisoned Data Rate (PDR), i.e., $PDR = \frac{N_{DP}}{N_D}$, the attacker can make a trade-off between attack impact and attack stealthiness.

3.4.2. Model Poisoning

In this attack strategy (see the right part in Fig.1), the adversary can fully control a subset of the clients. Here, we denote the fraction of compromised clients as Poisoned Malicious Clients Rate (PMR), i.e., $PMR = \frac{m}{n}$. To increase the attack's impact on the aggregated model, the adversary can deliberately modify the model updates before submitting them to the aggregator. This is done by (1) turning up the scaling factor γ to increase attack impact (e.g., model-replacement attack [9]) and (2) constraining the training process by setting the scaling-coefficient parameters β to evade anomaly detection (e.g., constrain-and-scale [9]). In this attack strategy, the adversary can create multi-objective optimization $(\beta\ell(p, y) + (1-\beta)\ell_B(p, \tau))$. By tuning the scaling-coefficient parameter β, the adversary can attack more stealthily.

3.4.3. Propagation Error

[25] firstly introduces the *propagation error*. Suppose clients conduct a protocol $f = (\varrho, A, \eta)$ at global iteration $t \in [T]$. Here, $\varrho(G^t, D, t)$ is a gradient oracle that inputs the tth global model G^t, local dataset D and outputs the updated weights $\mathbf{w}^t$. Malicious clients conduct a poisoned protocol $f^* = (\varrho^*, A, \eta)$ with $\varrho^*(G^t, D^P, t)$. For any round t and any global model G^t and any dataset D, we have $\varrho^*(G^t, D^P, t) = \varrho(G^t, D, t) + \epsilon$ with $\|\epsilon\|_1 \leq \rho$. At each iteration t, the upper bound ρ on ϵ gives the *additive error* introduced by poisoning. Small additive errors introduced at early iterations can build upon each other and create large divergences. This is referred to as *propagation error*.

In this work, we design a multi-poisoning attack to instantiate such propagation errors. In this attack strategy, the adversarial clients perform model poisoning or data poisoning attacks at every iteration. The adversarial clients can vary the PDR or the PMR. The upper bound ρ would vary with different PDR or PMR.

3.5. Characterization of Model Poisoning Attacks

To illustrate various model poisoning attacks more visually, we use a two-dimensional representation of the weight vectors of models. Then each model can be characterized by two factors: direction and magnitude. The cosine distance of the weight vectors can measure the direction between the two given models. The L_2 norm of the distance between the weight vectors can measure their magnitude difference. Fig.2 shows several types of poisoned models. The first poisoned client model $\overrightarrow{\mathbf{w}'_1}$ is trained by adding a large fraction of poisoned data D^P into genuine dataset D. $\overrightarrow{\mathbf{w}'_1}$ has an obvious direction deviation from the benign client models. The second type $\overrightarrow{\mathbf{w}'_2}$ consist of four small vectors ($\overrightarrow{\mathbf{w}'_5}$, $\overrightarrow{\mathbf{w}'_6}$, $\overrightarrow{\mathbf{w}'_7}$ and $\overrightarrow{\mathbf{w}'_8}$). This type of attack is achieved by four distributed attacks (Distributed Backdoor Attack (DBA) [13]). Poisoned models trained by distributed attacks are more undetectable in direction and magnitude than centralised attacks. The next type is $\overrightarrow{\mathbf{w}'_3}$, which has a less direction deviation but a larger magnitude difference. Such poisoned client models can be obtained by boosting the poisoned models with a large scaling factor γ (model-replacement attack [9]). The last type $\overrightarrow{\mathbf{w}'_4}$ has similar representations with genuine models. It is more stealthy compared to the first three types. This poisoned model can be crafted by constrain-and-scale attacks [9].

4. DeMAC Design Principle and key observation

In this section, we introduce our proposed approach, DeMAC. First of all, we give a novel scoring method, GradScore. We analyze that the PDR directly impacts the value of GradScore of a poisoned model. Then we describe how to detect malicious clients by evaluating the corresponding GradScore value in federated learning. Meanwhile, we analyze that based on GradScore, DeMAC can detect model poisoning attacks no matter the data distribution among clients. Finally, we give a security analysis that the proposed scoring method is not affected no matter how the adversary scales its model updates.

4.1. GradScore and analysis

Now, we give the definition of GradScore and analyze why and how this scoring method can be used for detecting poisoning attacks in FL.

Definition 4.1. *The GradScore of training set* $D = \{(x_i, y_i)\}_{i=1}^N$ *on a local client at global iteration* t *is* $GradScore(C_i^t) = \|g^t(\{(x_i, y_i)\})\|_2$.

It is approximated that the training dynamics are in continuous time. For a labelled example (x, y) from local data set $D = \{(x_i, y_i)\}_{i=1}^N$, the time derivative of the loss on this labelled sample is $\Delta_t((x, y), S_i) = -\frac{d\ell(f(\mathbf{w}^t, x), y)}{dt}$ at time t. By the chain rule,

$$\Delta_t((x, y), S_i) = g^t(x, y)\frac{d\mathbf{w}^t}{dt} \tag{2}$$

The instantaneous rate of change in $\mathbf{w}^t$ at time t, $d\mathbf{w}^t \approx \mathbf{w}^{t+1} - \mathbf{w}^t = -\eta \sum_{(x,y)\in S_t} g^t(x, y)$. The goal is to understand how poisoned samples from minibatch S_i affect the time derivative of the loss for any samples (x^*, y^*) from the same minibatch.

Lemma 4.1. *Let $S_{\neg j} = S \backslash (x_j, y_j)$ denotes minibatch excluding sample (x_j, y_j). Then for all rest samples (x^*, y^*), there exists c such that*

$$\|\Delta_t((x^*, y^*), S) - \Delta_t((x^*, y^*), S_{\neg j})\| = c\|g^t(x_j, y_j)\|. \tag{3}$$

Proof. For a given example (x^*, y^*), the chain rule yields $\Delta_t((x^*, y^*), S) = g^t(x^*, y^*)\frac{d\mathbf{w}^t}{dt}$. Therefore, for the left part of Eq. (3),

$$\begin{aligned}
&\|\Delta_t((x^*, y^*), S) - \Delta_t((x^*, y^*), S_{\neg j})\| \\
&= \|\frac{d\ell(f_t(x^*, y^*))}{d\mathbf{w}^t}(-\eta \sum_{S_i} g^t(x, y)) \\
&- \frac{d\ell(f_t(x^*, y^*))}{d\mathbf{w}^t}(-\eta \sum_{S_{\neg j,t}} g^t(x^*, y^*))\| \\
&= \|\frac{d\ell(f_t(x^*, y^*))}{d\mathbf{w}^t}(-\eta g^t(x_j, y_j))\| \\
&= \eta\|\frac{d\ell(f_t(x^*, y^*))}{d\mathbf{w}^t} g^t(x_j, y_j)\|
\end{aligned} \tag{4}$$

Let $c = \eta\|\frac{d\ell(f_t(x^*, y^*))}{d\mathbf{w}^t}\|$, we can get the right part of Eq. (3). □

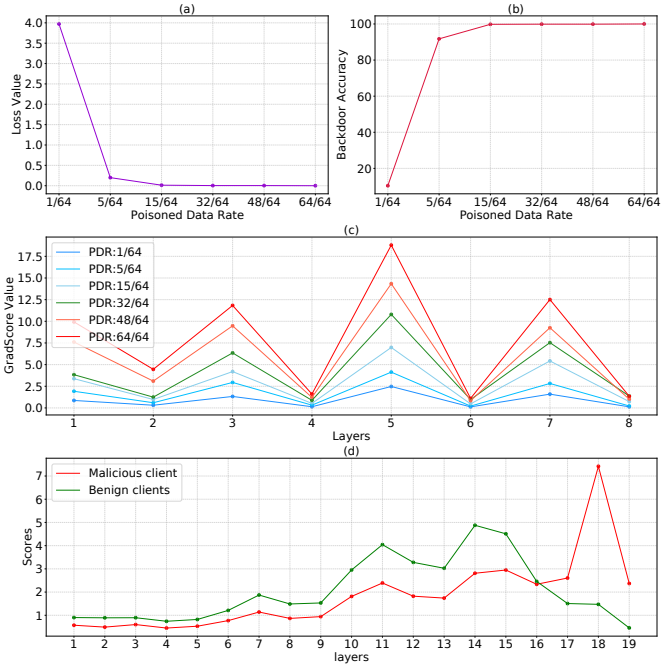


Figure 3: Impact of the PDR on loss value (a), Backdoor Accuracy (b) and GradScore value (c), the GradScore of the last layer gradients of the malicious updates and benign updates (d)

It can be seen from above that the contribution of a training sample (x_j, y_j) to the decrease of loss on other samples from the same minibatch can be quantified by Eq. (3). The value of $\|g^t(x_j, y_j)\|$ is the GradScore of a sample (x_j, y_j). Samples with large GradScore have a strong influence on learning. For poisoning training, poisoned samples have a stronger influence. To reduce the poisoning task training loss $\ell_B(p, \tau)$, malicious clients should increase the PDR. Therefore, the sum of the GradScore of samples is larger with higher PDR on malicious client datasets. In Fig.3 (a)(b)(c), we evaluated this inference, running backdoor training on MNIST dataset with a minibatch of 64

samples. With the same pre-trained model, the model trained with a higher poisoned data rate causes an obvious decrease in the backdoor training loss and a higher GradScore value.

Now we analyze how this scoring method can detect the model poisoning attacks in federated learning. No matter what the data distribution is among clients, the deviations between local models and the global model start to cancel out, i.e.,$\forall \mathbf{w} \in \{\mathbf{w}_i\}_{i=1}^{m}, \mathbf{w}_i^{t+1} - G^t \approx 0$ [9], in the benign setting, as the global model converges. Therefore, the updates of benign local models, $d\mathbf{w} \approx \mathbf{w}^{t+1} - \mathbf{w}^t$ is bounded. Therefore, $\|g^t(x_j, y_j)\|$ of one example from the benign dataset is small. The second observation is that when the global model starts to converge, poisoning behaviours on the malicious client will deviate the malicious updates from the current iteration global model[21] to reduce the training loss on the poisoning task. The GradScore of benign clients is small, while the GradScore of malicious clients is larger. Fig.3 (d) shows that the GradScore of the last layer gradients of the malicious client model is larger than benign clients. Therefore, malicious clients can be detected by comparing the GradScore of the last layer of local models.

To avoid detection, the adversary can try weak model poisoning attacks by limiting scaling up the poisoned model. The Theorem 4.2 shows that GradScore is unaffected if the adversary scales its poisoned model.

Theorem 4.2. *The GradScore is not affected by scaling or clipping the model update.*

Let G^t denote the global model of round t. An arbitrary local model update is $\boldsymbol{u}_i^{t+1}$, with $\boldsymbol{u}_i^{t+1} = \boldsymbol{w}_i^{t+1} - G^t$. Let γ denote the scaling factor. Let $\boldsymbol{u}_i^{t+1,*}$ denote the scaled model. $\boldsymbol{u}_i^{t+1,*} = \gamma \boldsymbol{u}_i^{t+1} = \gamma(\boldsymbol{w}_i^{t+1} - G^t)$ holds. Let C_i denote client i and analogously for the scaled client C_i^*

Then for $\forall \gamma \in \mathbb{R} \backslash \{0\}$: GradScore(C_i)=GradScore(C_i^*)

Proof. ←

$$\begin{aligned} GradScore(C_i^*) &= \|g(\{(x_i, y_i)\})\|_2 = \| \sum_{(x,y)\in S^t} g^t(x, y)\|_2 \\ &= \| \sum_{(x,y)\in S^t} \nabla_w \ell(p(\mathbf{w}_i^{t+1}, x), y)\|_2 \end{aligned} \tag{5}$$

$$\begin{aligned} GradScore(C_i) &= \|g(\{(x_i, y_i)\})\|_2 = \| \sum_{(x,y)\in S^t} g^t(x, y)\|_2 \\ &= \| \sum_{(x,y)\in S^t} \nabla_w \ell(p(\mathbf{w}_i^{t+1}, x), y)\|_2 \end{aligned} \tag{6}$$

Therefore,

$$GradScore(C_i^*) = GradScore(C_i) \tag{7}$$

holds. □

5. Overview and Design of DeMAC

In this section, we instantiate DeMAC for deep inspection and analysis of model updates to discover model poisoning attacks. We describe the design details of DeMAC below.

5.1. DeMAC Design

Fig.4 shows the main components and the workflow of DeMAC during global iteration t. It follows a deterministic algorithm and does not know the attack strategies or data distributions. DeMAC is deployed during the training session before the testing phase. Firstly, it should identify malicious behaviours in federated learning systems. Here comes a design challenge. At the beginning of federated learning training, benign local models should update their parameters continually to make the global model converge to the global minimum. So how can we perceive the convergent trend? To solve this problem, we design a historical global model update record with a flexible look-back window size l. This history record records the continuous variation of model accuracy on the validation set. When the maximum value among this history record is lower than a threshold value α, the defence approach can start to process.

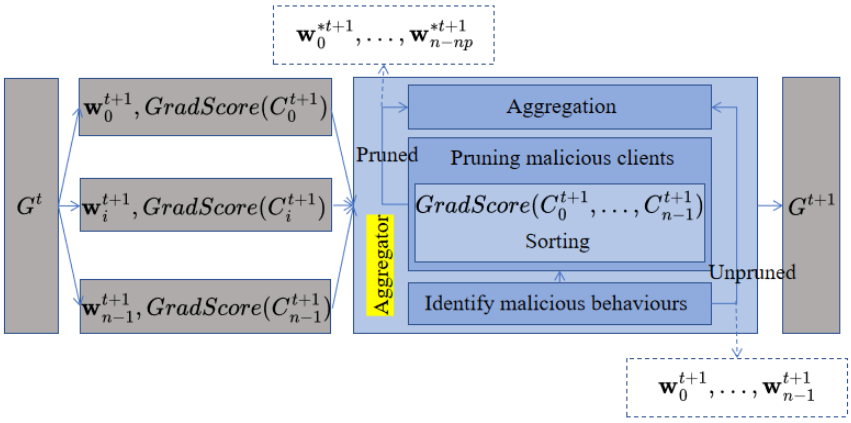


Figure 4: Illustration of DeMAC's workflow in global iteration t

After malicious behaviours are identified, the GradScore values of corresponding clients would be sorted in ascending order to detect anomalous clients. The top p with the highest scores can be pruned and excluded from benign clients. The value of p depends on the number of malicious clients in one global iteration. Finally, DeMAC trains the global model, excluding the updates sent by malicious clients. DeMAC automatically generates a clean global model given the training algorithm, which means the proposed technique, DeMAC, is an efficient defence for federated learning systems.

In the rest of this section, we demonstrate a detailed description of every main component of DeMAC. Algorithm 1 outlines the procedure of DeMAC.

5.2. Identifying malicious behaviours

In designing DeMAC, the first step is identifying and measuring malicious behaviours in the federated learning system.

$D = \{(x_i, y_i)\}_{i=1}^{N}$ denotes the training dataset on client C_i. If the $GradScore$ value for a client is significantly higher than other client $GradScore$ values in the same global iteration round, it indicates that malicious behaviours might happen on this client. The step of calculating $GradScore$ is shown in line 7 of Algorithm 1.

To avoid benign clients being misidentified when the global model is not yet converging, we demonstrate a validation phase to monitor the convergent trend. This validation phase consists of recording a group of previous global models and measuring the distance between the validation accuracies of two neighbouring global models. In the first step, we design a historical global model record, $history(G^0, ..., G^l)$, where $(G^0, ..., G^l)$ refers to a list of previous global models, and l is the size of the sliding window. In the second step, we define the distance between neighbouring global model validation accuracies as:

$$v(G^{i-1}, G^i) = |Acc(G^i) - Acc(G^{i-1})| \tag{8}$$

Where $Acc(G^i)$ is the accuracy of global model G^i on the validation dataset. We use a list $V(G^0), ..., V(G^{l-1})$, to contain neighboring validation variations of a historical record, $history(G^0, ..., G^l)$. The related steps are shown in lines 2-3 of Algorithm 1. If the maximal $V(G^i) \in [V(G^0), ..., V(G^{l-1})]$ is below the threshold σ, the global model can be regarded as convergence.

When the aggregator detects unusually large $GradScore$ values sent by some clients and the global model converges, malicious behaviours can be identified in the federated system.

5.3. Pruning and excluding malicious clients

After malicious behaviours are identified, the next step in DeMAC is to identify and exclude anomalous clients based on corresponding $GradScore$ values. First, the $GradScores$ to corresponding clients are sorted in ascending order. The top p clients with the highest scores are pruned and excluded from the benign client list. The parameter

Algorithm 1: Design of DeMAC

Input: n, G^0, T

// n is the number of clients in one iteration, G^0 is the initial global model, T is the number of global iterations

Output: G^T

// G^T is the updated global model after T iterations

1 **for** $t \in [1, .., T]$ **do**

2 $\quad historyRecord \leftarrow (G^{t-l}, ..., G^t)$;

$\quad$ // l is the look-back size

3 $\quad [V(G^{t-l}), ..., V(G^t)] \leftarrow VALIDATE(historyRecord)$;

4 $\quad$ **if** $max[V(G^{t-l}), ..., V(G^t)] < \sigma$ // σ is the convergence threshold; Malicious behaviours exist

5 $\quad$ **then**

6 $\quad\quad$ **for** $i \in [C_0^{t+1}, ..., C_{n-1}^{t+1}]$ **do**

7 $\quad\quad\quad GradScore(C_i^{t+1}) = \|g\{(x, y)\}\|_2$ // $\|g\{(x, y)\}\|_2$ is the L_2-norm of gradients of parameters in final layer of models

8 $\quad\quad$ **end**

9 $\quad\quad SCORE \leftarrow [GradScore(C_0^{t+1}), ..., GradScore(C_{n-1}^{t+1})]$;

10 $\quad\quad Sort(SCORE)$;

11 $\quad\quad Pruned(\mathbf{w}_0^{*t+1}, ..., \mathbf{w}_{n-np}^{*t+1}) \leftarrow Pruning_{p\%}([\mathbf{w}_0^{t+1}, ..., \mathbf{w}_{n-1}^{t+1}])$ // $p\%$ is the pruning rate

12 $\quad\quad SendPruned(\mathbf{w}_0^{*t+1}, ..., \mathbf{w}_{n-np}^{*t+1}) \rightarrow Aggregator$

13 $\quad$ **else**

14 $\quad\quad SendUnpruned(\mathbf{w}_0^{t+1}, ..., \mathbf{w}_{n-1}^{t+1}) \rightarrow Aggregator$ // Malicious behaviours do not exist

15 $\quad$ **end**

16 $\quad G^{t+1} \leftarrow G^t + \frac{\eta}{L-1} \sum_0^L (\mathbf{w}_i^{t+1} - G^t)$ // Global Aggregating, η is the global learning rate

17 **end**

p depends on the number of anomalous clients in one global iteration. Only one malicious client should be excluded when the adversary takes model-replacement attack strategies. However, malicious clients may collude and strengthen the impact of poisoning in one iteration. Considering the real-world federated learning deployments, it is unrealistic to assume that the fraction of malicious clients is above the range ($0 < m < n/2$). In an application scenario like Gboard [6], over 50% malicious clients mean the adversary should control at least 500 million Android devices. That is incredible [28]. So we only consider below 50% cases. Generally, p is set to 0.5. In this work, we assume the server has a knowledge of the number of malicious clients at one iteration. Hence, the server can decide the value of p. The sorting and pruning step is shown in lines 9-11 of Algorithm 1.

The aggregator excludes the updates sent by malicious users in the current iteration and trains the global model on the remaining model updates (line 16 of Algorithm 1). The global training algorithm varies based on the underlying training algorithm used in the application. We use FedAvg [1] to train the global model in this proposed work.

6. Evaluation Setup

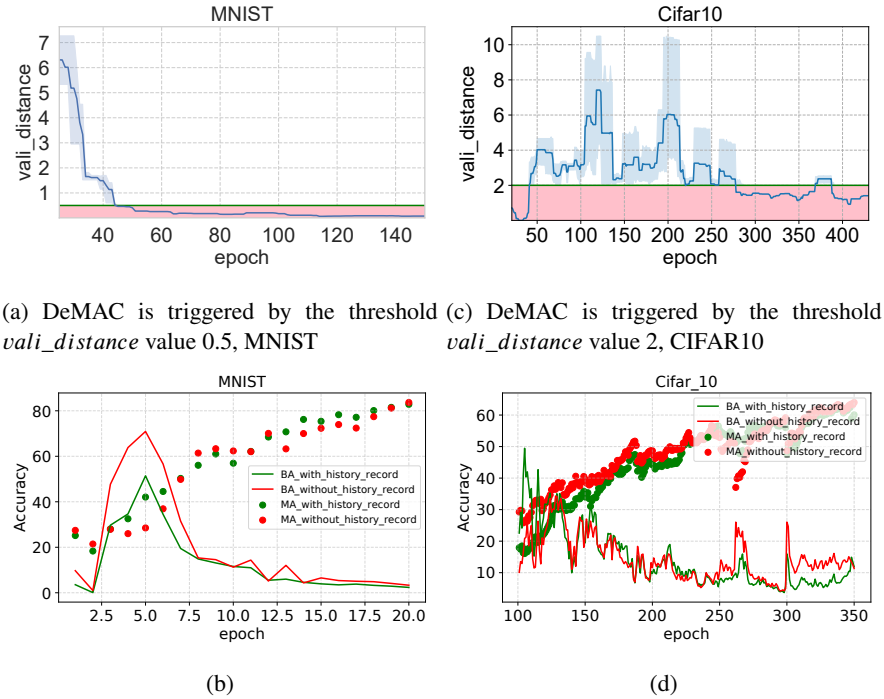
In this section, we give the details of the experimental setup and evaluation metrics used in this work for evaluating the effectiveness of DeMAC.

6.1. Experimental Setup

Datasets and global-model settings: In this work, two well-known benchmark datasets MNIST [29] and CIFAR10 [30] are considered to evaluate DeMAC. MNIST dataset is a ten-class-balanced image classification task with 70000 grey-scale images. And CIFAR10 dataset is a ten-class image classification task with 60000 RGB images. It is assumed there are 100 clients for global training. To simulate non-IID distribution, data is assigned to clients

Layer	Size
Input	$28 \times 28 \times 1$
Convolutional + ReLU	$3 \times 3 \times 30$
Max Pooling	2×2
Convolutional + ReLU	$3 \times 3 \times 5$
Max Pooling	2×2
Fully Connected + ReLU	100
Softmax	10

Table 2
the CNN Network Architecture



(a) DeMAC is triggered by the threshold *vali_distance* value 0.5, MNIST

(c) DeMAC is triggered by the threshold *vali_distance* value 2, CIFAR10

(b)

(d)

Figure 5: Measuring the maximum $vali\_distance\ V(G^i)$ enables DeMAC to detect convergence of the global model (a)(c). DeMAC with history global model record vs DeMAC without history global model record (b)(d)

according to Dirichlet distribution with concentration parameter α, so client datasets are unbalanced to classes. The distribution is more concentrated when the value of α is smaller. On the contrary, the distribution tends to be more uniform. Without Dirichlet sampling, data are uniformly distributed (IID) to clients. Unless otherwise mentioned, the concentration parameter α is set to 0.5. For MNIST dataset, a four-layer Convolutional Neural Network (see Tab.2) is used. For CIFAR10, ResNet18 [31] architecture is considered the global model.

Federated Learning settings: FedAvg [1] is considered as the FL method. In each global round, 10 of 100 clients are randomly selected. Considering the different characteristics of the datasets, we adopt the following parameter settings for federated training: for MNIST, clients train for 1 local epoch with a local learning rate of 0.1. For CIFAR10, clients train for 2 local epochs with a local learning rate of 0.1.

Attack strategy. We consider four targeted model poisoning attacks, single-shot model-replacement attacks [9], constrain-and-scale [9], and DBA [13] and multi-poisoning attacks.

(1) Model-replacement attacks, constrain-and-scale and DBA. In the case of MNIST, we modify the pixels of the digital image at training time, causing the images with pixel-pattern to be classified towards a target class. On CIFAR10 dataset, we apply the same attack strategy as on MNIST dataset. The attackers can set the scaling parameter γ for single-shot and DBA to control the impact of model poisoning. Unless otherwise mentioned, we set γ to 30, and PDR is set to $30/64$ with a local batch size of 64. For single-shot model replacement attacks and constrain-and-scale

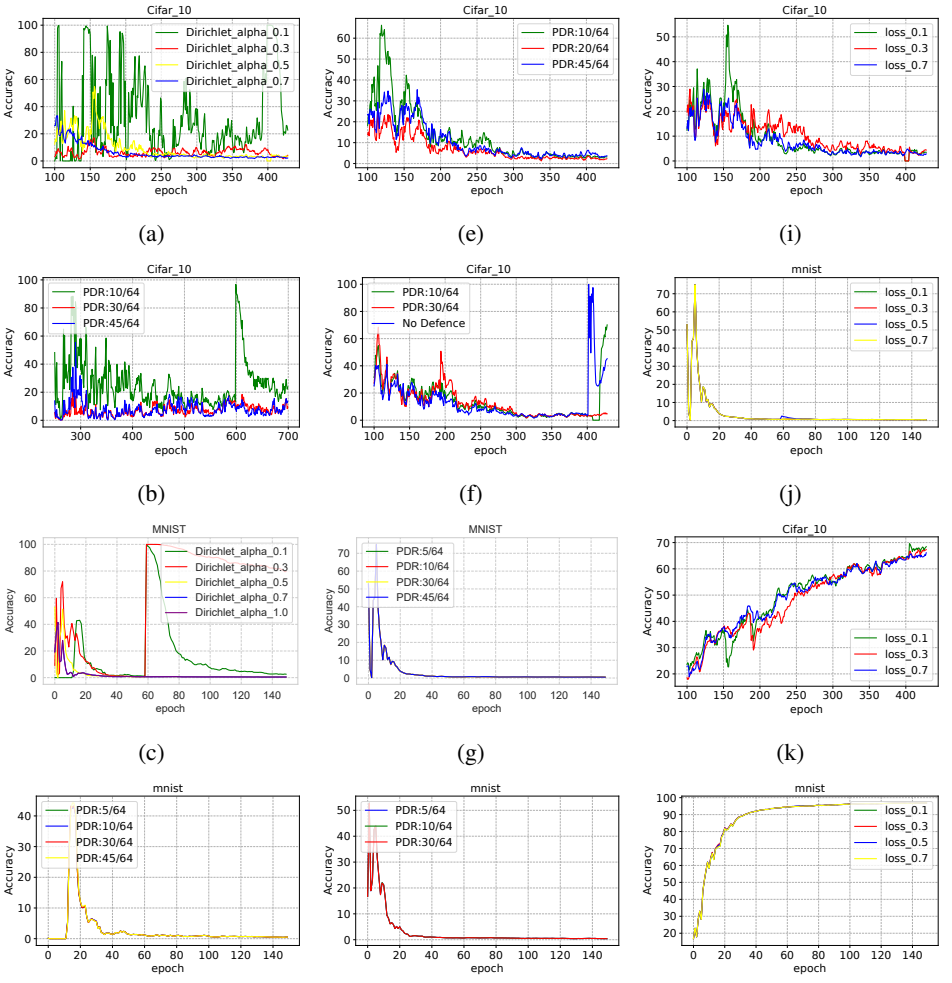


Figure 6: The ASR for DeMAC against model-replacement attack under different non-IID settings (a)(c). The ASR for DeMAC against model-replacement attack, where 0.1 of concentration parameter α, threshold σ value 2 for CIFAR10, and σ value 0.6 for MNIST are used (b)(d). Impact of the poisoned data rate on DeMAC against model-replacement attack (e)(g) and distributed backdoor attack (f)(h). MA and BA of the global model under the protection of DeMAC against constrain-and-scale attack with different β values (i)(j)(k)(l).

attacks, we assume that attackers perform attacks after 60 rounds for MNIST and 400 rounds for CIFAR10. For DBA, as attackers split triggers into four equal parts, it is assumed that malicious clients attack at round 62, 64, 66, and 68 for MNIST and at round 402, 404, 406, and 408 for CIFAR10.

(2) Multi-poisoning attacks. In the case of MNIST, adversarial clients perform the multi-poisoning attack at round 10 and 20, respectively. In the case of CIFAR10, adversarial clients perform the multi-poisoning attack at round 80 and 300, respectively. Unlike the three types of attacks described above, the multi-poisoning attack executes every round after being performed. We set γ to 1, and the PDR is set to $30/64$.

Detecting time.

(1) Single-shot model-replacement attack, constrain-and-scale and DBA. Unlike existing works [22][14] manually setting detecting time, DeMAC can spontaneously detect malicious clients according to the information provided by the historical record. We set the sliding window size l to 15 for MNIST and 20 for CIFAR10. DeMAC will be triggered when the maximum distance between neighbouring global model accuracies on the validation set $V(G^i)$ is below the predefined threshold σ. We choose $\sigma = 0.5$ for MNIST and $\sigma = 2$ for CIFAR10.

(2) Multi-poisoning attacks. Our experiments show that the increase in global model accuracy has an obvious oscillation rather than increasing monotonically during the early training stage. Hence, the predefined threshold σ is looser than the above setting in this case.

6.2. Evaluation Metrics

We consider two evaluation metrics for evaluating the accuracy and efficiency of DeMAC. *Main task accuracy (MA)* is used to evaluate the accuracy of the global model on the main task. MA is the ratio of testing examples that are correctly classified. *Backdoor Accuracy (BA) or Attack Success Rate (ASR)* is the ratio of poisoned examples that are classified as target labels by the global model. We define an evaluation metric for measuring the performance of DeMAC. *Computation cost per round (CCR)* measures the computation cost at one round in the Byzantine-robust FL system.

7. Evaluation Results

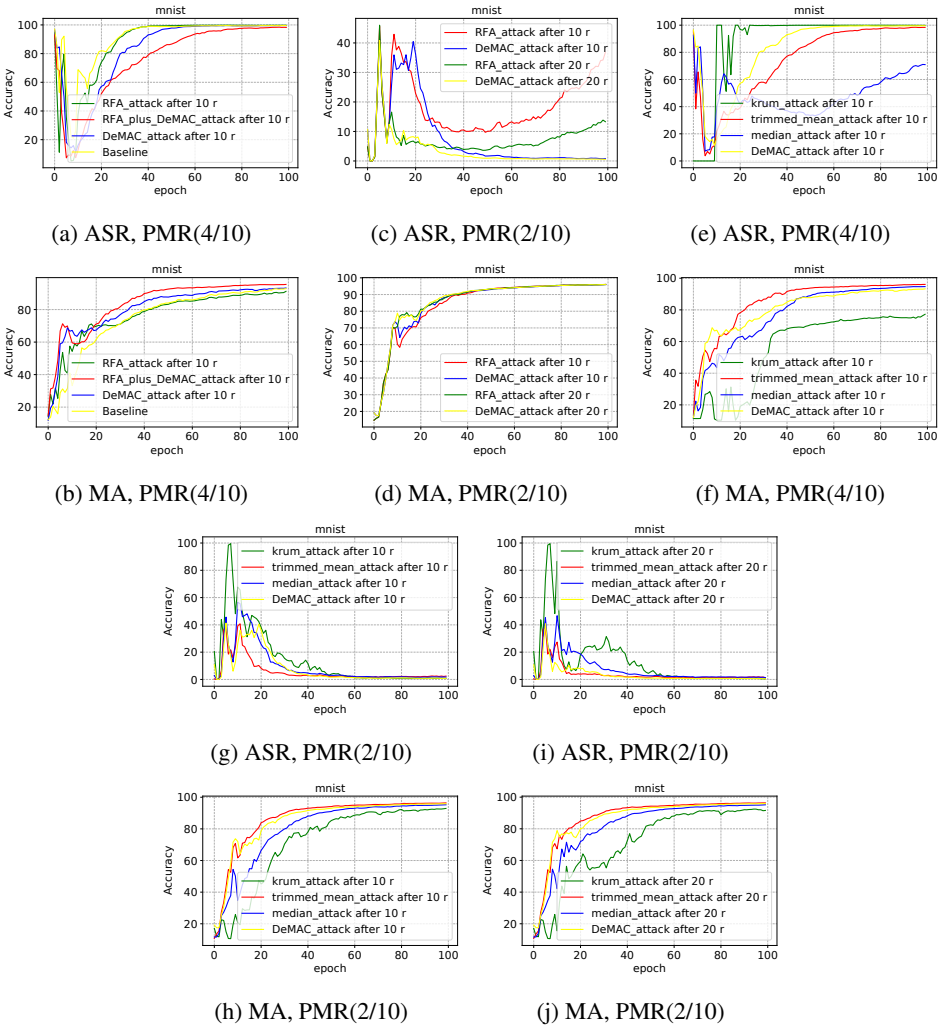
Efficiency of history record. Detecting/Attacking timing is rarely discussed in previous defence works. [22][9] discussed that the impact of model-replacement attacks in early rounds is not durable as the ASR decreases sharply within several rounds. The poisoning impact tends to stay long in the later training rounds. The simple way is to detect when the global model starts training. However, it is not cost-friendly to start detecting from train-from-scratch to defend against poisoning attacks, such as model-replacement attacks. To solve this problem, we combine DeMAC with a historical record. By applying this historical global model record, DeMAC can track the convergence of global training. Fig.5(a)(b) shows that DeMAC will be triggered when the maximum $vali\_distanceV(G^i)$ is below the predefined threshold (pink area in Fig.5(a)(b)). The DeMAC defence is performed only when the global model starts to stabilize. Fig.5(b)(d) shows the comparison between DeMAC with a historical global model record and DeMAC without a historical global model. It is not difficult to see that enabling DeMAC in early rounds may cause a delay in the convergence of the global model. It might be a drawback in federated learning deployments. In Fig.5(b) for MNIST, in the initial 20 rounds, DeMAC without a historical global model record shows a higher error rate than DeMAC with a historical global model record. The same result is shown in Fig.5(d).

Impact of the degree of non-IID: Fig.6(a)(b)(c)(d) shows the impact of the non-IID degree on DeMAC. First, from Fig.6(a)(c), we observe that with PDR $(30/64)$ and threshold σ fixed, the ASR can be reduced to nearly 0% when the non-IID degree is larger than some threshold. When α is set to 0.1 for CIFAR10 or α is set below 0.3 for MNIST, DeMAC cannot detect malicious clients, which causes a high ASR on the global model. In Fig.6(b), we postpone the attacking time at round 600, when the global model stabilises. We observe that at the same PDR $(30/64)$, threshold σ value (2) and non-IID setting ($\alpha = 0.1$), DeMAC can mitigate poisoned updates and reduce backdoor accuracy to a low level compared with the failure of detection in Fig.6(a). In Fig.6(d), we set threshold σ value to 0.6 rather than the default value 0.5 with non-IID setting($\alpha = 0.1$). So, DeMAC would be triggered and start to detect malicious clients earlier. BA of the global model can be reduced to nearly 0% under all PDR settings. From the above analysis, it is not difficult to see that the success of model poisoning attacks is highly rated to the convergent trend of the global model.

Impact of Poisoned Data Rate (PDR): Fig.6(e)(f)(g)(h) shows the impact of the poisoned data rate on DeMAC. In Fig.6(e)(g), DeMAC can mitigate malicious client updates and reduce the ASR to a low level for both two datasets. In Fig.6(f)(h), we evaluate the efficiency of DeMAC against distributed backdoor attacks. Fig.6(f) shows that DeMAC cannot mitigate the last split backdoor attack when PDR is low. One possible reason is that compared with the central backdoor attack, distributed property of DBA makes attack behaviour more stealthy. In Fig.6(h), DeMAC can decrease the attack impact under all the attack strategies.

Defending Anomaly-Evasion Attack: As discussed in section 3, attackers can balance the impact and stealth of attack by varying the scaling-coefficient parameter β. $\ell_B(p, \tau)$ is calculated as the L_2 norm between the current poisoned and round global models. Figure 6(i)(j)(k)(l) shows that DeMAC can successfully mitigate attack impact for both datasets and different β values. We also add some tables for corresponding to the results in the appendix 9.

Detecting multi-poisoning attacks: Fig.7 and Fig.8 show the comparison results on two datasets for detection methods, different attack timing, and different numbers of malicious clients. From our results, it is not difficult to see that malicious perturbations in every iteration can gradually compromise baseline Byzantine-robust FL algorithms and cause high ASR. DeMAC can effectively suppress such propagation errors. Here are several observations. Firstly, DeMAC can mitigate attack impact and reduce the ASR to a low level in most cases, except in the case (MNIST dataset, PMR(4/10), attack after ten rounds, Fig.7(a)(b)). All the Byzantine robust methods fail in this case (Fig.7(a)(b)). The



(a) ASR, PMR(4/10) (c) ASR, PMR(2/10) (e) ASR, PMR(4/10)

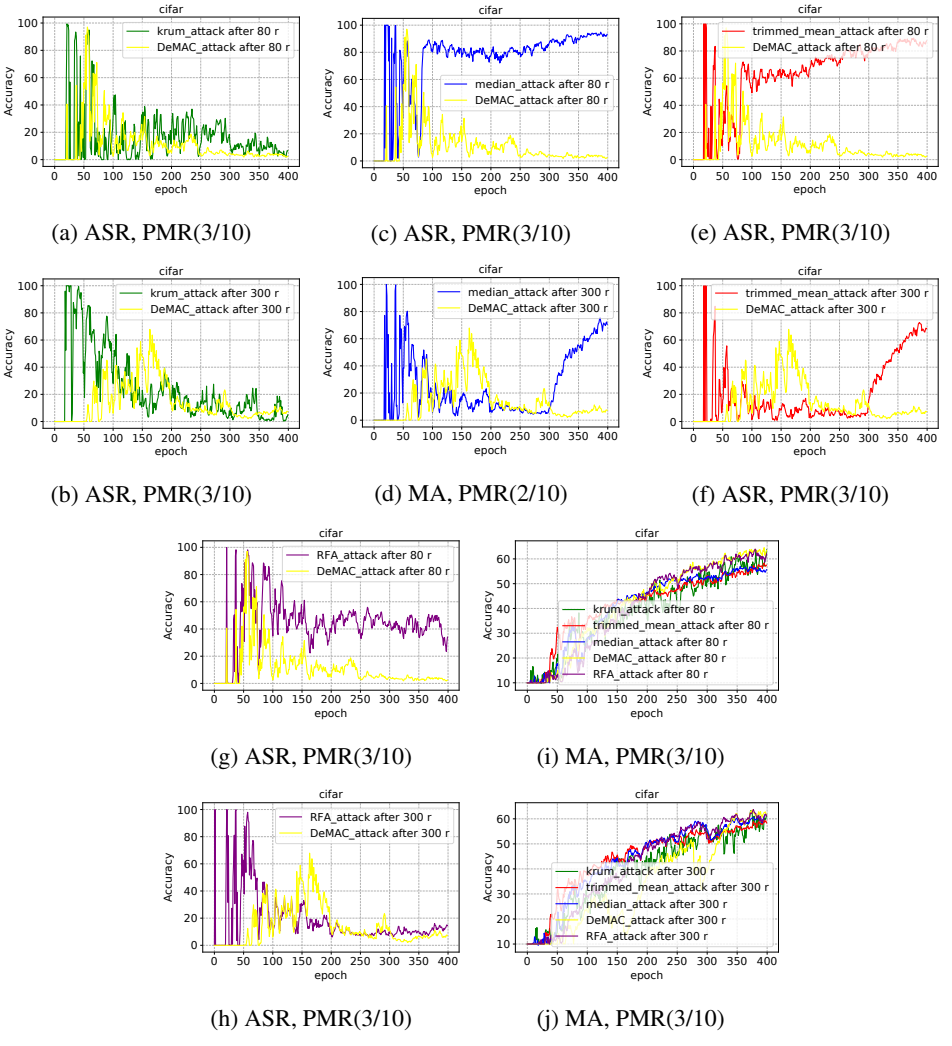
(b) MA, PMR(4/10) (d) MA, PMR(2/10) (f) MA, PMR(4/10)

(g) ASR, PMR(2/10) (i) ASR, PMR(2/10)

(h) MA, PMR(2/10) (j) MA, PMR(2/10)

Figure 7: ASR and MA of malicious detection for different detection methods. Concentration parameter $\alpha(0.5)$, MNIST dataset and scaling parameter $\gamma(1)$ are used.

main reason could be that the detection methods are too hard to distinguish benign clients from many malicious clients, as the global model is unstable in the first ten rounds. Second, in the case (MNIST dataset, PMR(2/10), attack after ten rounds) and case (MNIST dataset, PMR(2/10), attack after 20 rounds), all the defending methods except RFA [12] can mitigate attack impact. In subsection 2.1, we discuss that by carefully setting the scaling factor γ and then controlling the total weights of the outliers, the attacker can bypass RFA. We set the scaling factor γ as 1 and PDR as $30/64$. RFA fails to detect malicious behaviours. This conclusion is in line with conclusions from previous work [13]. DeMAC, trimmed_mean [11], and median achieve comparable main accuracy and outperform Krum [10]. The main reason could be that Krum selects one client update to represent the global model. Therefore, due to the heterogeneous data distribution, these chosen model updates cannot achieve the same performance as on the global test dataset. This conclusion is in line with conclusions from previous work [25]. Third, in cases (CIFAR10 dataset, attack after 80 rounds), all the defending methods except DeMAC fail to eliminate attack impact. In cases (CIFAR10 dataset, attack after 300 rounds), DeMAC, Krum, and RFA can reduce the ASR to a low level, but median and trimmed-mean still cannot defend attack behaviours. As we discuss in subsection 2.1, the assumption of trimmed-mean does not hold for model poisoning attacks. Therefore, this observation is in line with the discussion from prior sections. Fourth, in the case (CIFAR10 dataset, attack after 300 rounds), other defending methods can achieve comparable main accuracy as DeMAC. However, in the case (CIFAR10 dataset, attack after 80 rounds), the main accuracy of DeMAC outperforms other defending methods after 400 rounds.



(a) ASR, PMR(3/10) (c) ASR, PMR(3/10) (e) ASR, PMR(3/10)

(b) ASR, PMR(3/10) (d) MA, PMR(2/10) (f) ASR, PMR(3/10)

(g) ASR, PMR(3/10) (i) MA, PMR(3/10)

(h) ASR, PMR(3/10) (j) MA, PMR(3/10)

Figure 8: ASR and MA of malicious detection for different detection methods. Concentration parameter $\alpha(0.5)$, MNIST dataset and scaling parameter $\gamma(1)$ are used.

Defense methods	**Before Attack** (sec)	**After Attack** (sec)
Krum	62.648	127.47
Median	44.867	105.304
RFA	61.892	141.256
Trimmed-mean	43.933	108.236
DeMAC	58.688	126.37

Table 3
The comparison of the effectiveness of DeMAC with other Byzantine-robust methods on CIFAR set

Performance Comparison: In this work, we define the CCR as the time for one iteration of training in the FL system equipped with the chosen defence method. We use two tables to show the comparison of the effectiveness of DeMAC with other Byzantine-robust methods on two different datasets. In these experiments, we take the multi-poisoning attack strategy. The details of this attack strategy are described in section 6.1. Experimental Setup. In Table 3, RFA [12] is the most time-consuming method. Median [11] and trimmed-mean [11] are the most time-saving methods. DeMAC and Krum [10] show similar performance.

Defense methods	Before Attack (sec)	After Attack (sec)
Krum [10]	19.837	59.609
Median [11]	19.652	61.403
RFA [12]	18.086	59.699
Trimmed-mean [11]	18.700	62.448
DeMAC	21.473	67.260

Table 4
The comparison of the effectiveness of DeMAC with other byzantine-robust methods on MNIST set

8. Conclusion

Backdoor attacks and, more specifically, model poisoning attacks are a big challenge faced by federated learning. To address the shortcomings of the existing defence approaches, we proposed a novel defence system, which is called DeMAC to defend against malicious attacks by measuring the difference in the contribution of benign clients and malicious clients to the global model. We defined a new metric GradScore, to compute the L2-norm of the gradients of the last layer of contributed model updates, which is shown to be effective in detecting updates from malicious clients. Furthermore, we utilized the history records of the contributed model updates to enhance the malicious client detection performance. We evaluated and compared DeMAC with state-of-the-art defence techniques over various attack strategies and datasets. Experiment results show that DeMAC can effectively mitigate the model poisoning attacks without sacrificing the performance of the main task and significantly outperforms the existing defence approaches. Future research directions include extending the proposed method to defend against adaptive attacks based on well-known non-targeted model poisoning frameworks. This proposed method can also be combined with Byzantine-robust aggregation rules. We leave it for future work.

Acknowledgement

This work was funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 824019 and No 101022280, Horizon Europe MSCA programme under grant agreement number 101086228, and EPSRC with RC Grant reference EP/Y027787/1. For the purpose of open access, the author has applied a CC BY licence to any Author Accepted Manuscript (AAM) arising from this submission.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017, pp. 1273–1282.
[2] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, arXiv preprint arXiv:1610.05492 (2016).
[3] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, et al., Large scale distributed deep networks, Advances in neural information processing systems 25 (2012).
[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan, et al., Towards federated learning at scale: System design, Proceedings of Machine Learning and Systems 1 (2019) 374–388.
[5] P. Regulation, General data protection regulation, Intouch 25 (2018).
[6] Federated learning: Collaborative machine learning without centralized training data. [online]., https://ai.googleblog.com/2017/04/federated-learning-collaborative.html (2017).
[7] Utilization of fate in risk management of credit in small and micro enterprises. [online]., https://www.fedai.org/cases/utilization-of-fate-in-risk-management-of-credit-in-small-and-micro-enterprises/.
[8] Machine learning ledger orchestration for drug discovery (melloddy). [online]., https://www.melloddy.eu/ (2017).
[9] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, V. Shmatikov, How to backdoor federated learning, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 2938–2948.
[10] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, J. Stainer, Machine learning with adversaries: Byzantine tolerant gradient descent, Advances in Neural Information Processing Systems 30 (2017).
[11] D. Yin, Y. Chen, R. Kannan, P. Bartlett, Byzantine-robust distributed learning: Towards optimal statistical rates, in: International Conference on Machine Learning, PMLR, 2018, pp. 5650–5659.
[12] K. Pillutla, S. M. Kakade, Z. Harchaoui, Robust aggregation for federated learning, IEEE Transactions on Signal Processing 70 (2022) 1142–1154.
[13] C. Xie, K. Huang, P.-Y. Chen, B. Li, Dba: Distributed backdoor attacks against federated learning, in: International Conference on Learning Representations, 2019.

[14] Z. Zhang, X. Cao, J. Jia, N. Z. Gong, Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients, in: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022, pp. 2545–2555.
[15] Y. Chen, L. Su, J. Xu, Distributed statistical machine learning in adversarial settings: Byzantine gradient descent, Proceedings of the ACM on Measurement and Analysis of Computing Systems 1 (2) (2017) 1–25.
[16] R. Guerraoui, S. Rouault, et al., The hidden vulnerability of distributed learning in byzantium, in: International Conference on Machine Learning, PMLR, 2018, pp. 3521–3530.
[17] S. Rajput, H. Wang, Z. Charles, D. Papailiopoulos, Detox: A redundancy-based framework for faster and more robust gradient aggregation, Advances in Neural Information Processing Systems 32 (2019).
[18] C. Xie, S. Koyejo, I. Gupta, Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance, in: International Conference on Machine Learning, PMLR, 2019, pp. 6893–6901.
[19] Z. Yang, W. U. Bajwa, Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning, IEEE Transactions on Signal and Information Processing over Networks 5 (4) (2019) 611–627.
[20] C. Fung, C. J. Yoon, I. 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.
[21] T. D. Nguyen, P. Rieger, R. De Viti, H. Chen, B. B. Brandenburg, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, et al., {FLAME}: Taming backdoors in federated learning, in: 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 1415–1432.
[22] S. Andreina, G. A. Marson, H. Möllering, G. Karame, Baffle: Backdoor detection via feedback-based federated learning, in: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), IEEE, 2021, pp. 852–863.
[23] S. Shen, S. Tople, P. Saxena, Auror: Defending against poisoning attacks in collaborative deep learning systems, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, 2016, pp. 508–519.
[24] P. Rieger, T. D. Nguyen, M. Miettinen, A.-R. Sadeghi, Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection, arXiv preprint arXiv:2201.00763 (2022).
[25] A. Panda, S. Mahloujifar, A. N. Bhagoji, S. Chakraborty, P. Mittal, Sparsefed: Mitigating model poisoning attacks in federated learning with sparsification, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2022, pp. 7587–7624.
[26] X. Cao, M. Fang, J. Liu, N. Z. Gong, Fltrust: Byzantine-robust federated learning via trust bootstrapping, arXiv preprint arXiv:2012.13995 (2020).
[27] T. Minka, Estimating a dirichlet distribution (2000).
[28] V. Shejwalkar, A. Houmansadr, P. Kairouz, D. Ramage, Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning, in: 2022 IEEE Symposium on Security and Privacy (SP), IEEE, 2022, pp. 1354–1371.
[29] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[30] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).
[31] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

9. Appendix

9.1. Tables corresponding to Evaluation Results

Impact of the degree of non-IID: Tables 5, 6 show the impact of the non-IID degree on DeMAC. From Table 5, with other hyperparameters fixed, DeMAC can decrease the ASR to a very low level when α is larger than 0.1. When α is set to 0.1 and the attacking time is postponed to 600 round, DeMAC can mitigate the poisoned updates. Table 6 shows similar results. When threshold σ is set from 0.5 up to 0.6, DeMAC is able to reduce the ASR to a low level. From above analysis, we can see that the success of model poisoning attacks is closely related to the convergence of FL training.

Dirichlet (α)	**PDR**	σ	**Round**	**ASR** (%)
0.1	30/64	2	400	99.98
0.1	30/64	2	600	2.411
0.3	30/64	2	400	7.888
0.5	30/64	2	400	5.133
0.7	30/64	2	400	3.033

Table 5: Impact of the degree of non-IID on CIFAR10

Dirichlet (α)	**PDR**	σ	**Round**	**ASR** (%)
0.1	30/64	0.6	60	1.260
0.1	30/64	0.5	60	99.77
0.3	30/64	0.5	60	99.97
0.5	30/64	0.5	60	0.657
0.7	30/64	0.6	60	0.680

Table 6: Impact of the degree of non-IID on MNIST

Impact of Poisoned Data Rate (PDR): Tables 7, 8, 9, 10 show the impact of PDR on DeMAC. From Tables 7, 8, DeMAC can effectively mitigate the malicious behaviours. From Table 9, DeMAC cannot work well when PDR is set to 10/64. The possible reason is that with few data samples being poisoned, DBA is too stealthy to be detected.

PDR	σ	**Round**	**ASR** (%)
10/64	2	400	3.022
20/64	2	400	2.211
30/64	2	400	4.722

Table 7: Impact of Poisoned data rate (PDR) on CIFAR

PDR	σ	**Round**	**ASR** (%)
5/64	0.5	60	0.635
10/64	0.5	60	0.646
30/64	0.5	60	0.635
45/64	0.5	60	0.644

Table 8: Impact of Poisoned data rate (PDR) on MNIST

PDR	σ	**Round**	**ASR** (%)
10/64	2	400	34.42
30/64	2	400	3.488

Table 9: Impact of Poisoned data rate (PDR) on CIFAR for defending DBA

PDR	σ	**Round**	**ASR** (%)
5/64	0.5	60	0.657
10/64	0.5	60	0.747
30/64	0.5	60	0.724

Table 10: Impact of Poisoned data rate (PDR) on MNIST for defending DBA

Defending Anomaly-Evasion Attack: Tables 11, 12 show DeMAC can successfully mitigate attack impact for two datasets.

β	σ	**Round**	**ASR** (%)	**MA** (%)
0.1	2	400	5.133	64.25
0.3	2	400	5.6	66.65
0.7	2	400	3.133	63.43

Table 11: Defending Anomaly-Evasion Attack on CIFAR

β	σ	**Round**	**ASR** (%)	**MA** (%)
0.1	0.5	60	0.635	94.4
0.3	0.5	60	0.646	94.45
0.5	0.5	60	2.531	94.07
0.7	0.5	60	0.635	94.42

Table 12: Defending Anomaly-Evasion Attack on MNIST