CLUSTER-BASED SCORING FOR MALICIOUS MODEL DETECTION IN FEDERATED LEARNING

by

Cem Çağlayan

B.S., Electrical and Electronics Engineering, Bahcesehir University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2023

ACKNOWLEDGEMENTS

This thesis and the paper could not have been finished without the help of my supervisor and my family members. Additionally, I want to thank my jury members for reviewing my work.

ABSTRACT

CLUSTER-BASED SCORING FOR MALICIOUS MODEL DETECTION IN FEDERATED LEARNING

Federated learning is a distributed machine learning technique aggregating every client model on a server to obtain a global model. However, some clients may harm the system by poisoning their model or data to make the global model irrelevant to its objective.

This thesis introduces an approach for the server to detect adversarial models by coordinate-based statistical comparison and eliminate them from the system prior to aggregation. A new attack type, layer poisoning, where the malicious nodes prefer poisoning selected small size layers of the model to deceive the detection system, is also introduced. Adaptive thresholding is adopted for preserving the robustness of the detection mechanism for various network against different attack types. A simulation framework is developed to benchmark and realize tests as a distributed system. Experiments that use random sampling of independent and identically distributed (iid) datasets with different batch sizes have been carried out to show that the proposed method can identify the malicious nodes successfully even if some of the clients learn slower than others or send quantized model weights due to energy limitations. The proposed approach is extensively tested with malicious-benign client ratios, model types, and datasets to present its versatility. The results show that the proposed system successfully eliminates the malicious models when their generating clients constitute at most 45% of the network. Comparison with the approaches from the literature shows that the proposed method performs the same as or better than the state of art solutions.

ÖZET

FEDERE ÖĞRENMEDE ZARARLI MODELLERİN TESPİTİ İÇİN KÜMELEME TABANLI SKORLAMA

Federe öğrenme, küresel bir model elde etmek için her istemci modelini bir sunucuda toplayan dağıtılmış bir makine öğrenimi tekniğidir. Ancak, bazı istemciler modellerini veya verilerini zehirleyerek küresel modeli istenenden alakasız hale getirebilir ve sisteme zarar verebilir.

Bu tez, sunucu için koordinat tabanlı istatistiksel karşılaştırma yoluyla zehirli modelleri tespit etmesi ve bunları birleştirme öncesinde sistemden uzaklaştırması için bir yaklaşım sunmaktadır. Koruma mekanizmasından kaçınmak için zararlı istemcilerin modelin daha az bir kısmını zehirlediği saldırı türü olan Katman Zehirlemesi de tanıtılmıştır. Farklı saldırı türlerine ve ağlara karşı koruma mekanizmasının sağlamlığını teyit etmek için uyarlanabilir eşik değeri uygulanmıştır. Önerilen yöntemin başarımını ölçmek için dağınık bir test ortamı tasarlanmıştır. Bazı istemciler, diğerlerinden daha yavaş öğrense veya enerji sınırlamaları nedeniyle nicelenmiş model parametreleri gönderse bile kötü niyetli cihazları başarılı bir şekilde belirleyebildiğini göstermek için farklı küme boyutlarına sahip, rastgele örneklenmiş bağlantılı veri kümelerini kullanan deneyler yapılmıştır. Ayrıca, koruma sisteminin detaylı analizi için farklı kötü niyetli cihaz sayıları, model türleri ve veri kümeleri ile denenmiştir. Sonuç olarak, sunulan korumanın, zararlı cihazların tüm istemcilere oranı %45'e kadar olduğunda zararlı modelleri tespit edebildiği ve sistemden uzaklaştırabilmeyi başarabildiği görülmüştür. Sunulan çözümün, literatürden alınmış diğer koruma mekanizmalarıyla kıyaslaması yapılmış; bu sistemler ile aynı veya daha fazla koruma sağladığı gözlemlenmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C	Number of total clients
$d_{Oi,Oj}$	Average score difference between centroids i and j
L	Dimension value of the centroid
l_i	$i'th$ layer of a model
M	Number of malicious clients
$Model_{poisoned}$	Poisoned model of a client
N	Number of total clients
$\vec{\mathbf{O}}_i$	Centroid of cluster i
P_{rate}	Poisoning rate for poisoning calculation
S_c	Score of client c
$S_{c,L}$	$L'th$ data-point of client $c's$ score
t_i^+	Upper filter value of weight collection
t_i^-	Lower filter value of weight collection
w	Weight parameter of the model
w_i	$i'th$ weight parameter of the model
$w_{i,c}$	$i'th$ weight parameter of client $c's$ model
x	Boolean parameter of controlled weight point
$x_{i,c}$	$i'th$ Boolean parameter of client $c's$ model
α	Hyper parameter in the poisoning calculation
γ	Threshold for average score difference d
θ	Scaling parameter for the threshold mechanism
μ	Mean of weight parameters
μ_i	$i'th$ weight parameter of mean
σ	Standard deviation of weight parameters
σ_i	$i'th$ weight parameter of standard deviation

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
FedAVG	Federated Average
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
GRU	Gated Recurrent Units
iid	Independent and Identically Distributed
IoT	Internet of Things
KPCA	Kernel Principal Component Analysis
lb	Local Batch
L-BFGS	Limited-Memory Broyden–Fletcher–Goldfarb–Shanno
MSE	Mean Square Error
PCA	Principal Component Analysis
REST	Representational State Transfer

1. INTRODUCTION

The rise of the Internet of Things (IoT) has led to the emergence and active use of billions of devices connected to the internet. Consequently, a massive amount of data is generated for producing actionable insights. Privacy is a major concern when data have to travel across several devices on the internet. A solution comes with federated learning where each device does its training locally with its data before transmitting its model to a server. The server aggregates the collected client models with specific algorithms such as federated average (FedAVG) [1] which averages the models. Then, it transmits the aggregated model to each client so that training can continue on the aggregated model. This communication between the server and the clients continues until an acceptable global model accuracy is obtained for all participants of the learning system.

Due to privacy issues, the server never knows the intention of a client in federated learning. This is an opportunity for malicious devices to harm the system by sending poisoned models. They may poison their data to collapse the aggregated model by changing their labels or spoiling the training data. They may also send an entirely different or statistically poisoned model instead of a trained model. Hence, current research focuses on server-side methods to prevent model breakdown from malicious users. There are two different approaches to the detection of adversaries. The first one is to find malicious devices in the network [2–4] where the traffic activities of the devices are collected as training data to detect unusual traffic. However, this method requires pretraining to discover the regular traffic flow. The second approach is to detect adversarial models among the regular ones [5, 6]. The server uses filtering and classification methods to categorize the poisoned models in the detection system.

This thesis proposes an adversarial model detection system in deep neural networks by combining statistical methods with clustering algorithms. Clustering algorithms are widely used in detecting anomalies in federated learning systems. This work

differs from the existing studies in the sense that collected client models are analyzed iteratively and statistically to generate multidimensional data points for each client to be used in the clustering mechanism. The data structure used in clustering has shown to be robust in extracting malicious client models when the adversary rate is not greater than 45% of the total clients' count. The performance of the method is also evaluated with slow learning devices and devices that can send only quantized weights due to energy constraints. Thus, the proposed detection mechanism allows some devices to be different from the rest of the clients, which is not analyzed deeply in the literature. Moreover, a targeted attack, the layer poisoning is introduced and analyzed. A layer poisoning attack can generate vulnerabilities in the detection system since the malicious devices hide their existences with small but effective poisoning. The test cases are divergent due to the heterogeneity of the clients. A framework has also been developed for this purpose to generate various attacks from the attack server.

The rest of the thesis is organized as follows. The next chapter presents related studies and compares them with this work. It also includes the background to introduce the attack types. In Chapter 3, the proposed method is explained. The framework is introduced in Chapter 4, and the experiments are presented in Chapter 5. The final chapter concludes the thesis.

2. RELATED WORK

Recent research for adversary detection on federated learning systems can be studied with respect to network activities and model parameters. The former targets the detection of malicious activities on the network. The solutions in this category relies on differentiation of normal and attack traffic with the federated learning. They mostly employ autoencoders, where encoders compress the input and decoders try to reconstruct it with a model. Autoencoders have to be trained with benign traffic data in order to discover abnormal network activity. When traffic is normal, reconstruction error is low. Abnormal traffic causes a high reconstruction error. Based on this fact, various studies exist with different thresholding techniques. In [2], the auto-encoders are selected as the defense mechanism for the unsupervised models where the clients can access only their benign traffic data. The mean square error (MSE) is used for calculating the threshold. Binary classification is carried out with multi-layer perceptrons in the supervised model, where the clients can access all their labeled data. In [7] and [8], threshold is computed as the sum of the mean and weighted standard deviation of MSE. In [9], Gated Recurrent Units (GRUs) are used to find maliciousness in traffic flow by giving "occurrence probability" with a pre-trained GRU model to the structured network packets. Similarly, unexpected packets have a small probability of occurrence since benign clients do not have these traffic patterns; thus, the GRU model gives attacker packets a low occurrence probability.

In the second category, the detection systems are implemented to find an anomaly in the deep network itself that is learned by the attendants of the federated learning system. The attackers infect the network with their local models. Consequently, the parameters of models are directly or indirectly affected by this poisoning. Thus, the detection system needs to check the validity of every client model according to its parameters/gradients/weights. In [5], the dimension of gradients is firstly reduced with principal component analysis (PCA). Then, euclidean distances between each client are calculated to eliminate the malicious clients with a threshold. Finally, the k-means

algorithm clusters the gradients to suspend unsatisfied models; then, measures cosine similarities between remainder client models for finding optimal gradients. Another study that uses cosine similarity to detect adversarial weight updates computes the possible update direction to eliminate the bad clients from the system [6]. Their solution relies on the fact that the benign clients have to exceed malicious ones as is the case for our work. In [10], an autoencoder uses MSE as the reconstruction error. Their solution requires all clients to be benign during the training phase of the autoencoder so that the decoder will be able to identify the adversaries in the upcoming rounds. This approach surely fails in case that malicious clients exist during the initial training phases of the federated learning system. There also exist several studies that use model classification with generative adversarial networks (GAN) [11], ensemble global model prediction [12], and Isolated Trees with the scored clients [13]. As can be seen, there is no single way to apply a defense mechanism in the literature.

Clustering methods are widely used in anomaly detection. Nevertheless, clustering could not be the only defense mechanism due to the excessive number of weight parameters to compute. Thus, the defense mechanisms ensemble a frontline guard mechanism to feed the clustering system. The clustering mechanism is the decision-maker of the anomaly detection system in light of the filtered calculations. Existing approaches mainly differentiate between malicious and benign clients. In [14], the k-means algorithm is adopted in the adversary detection system as an additional layer to kernel principal component analysis (KPCA) to compare and improve the results. Another detection system [15] uses the same algorithm to cluster pair-wise L2-Norm distances within two clusters to detect malicious models. Similarly, in [16], the authors use a mechanism to cluster the masked features of the DNN model learned collaboratively. It also employs an error-rate tolerance metric on clusters to distinguish the adversary. In [17], k-means is used to cluster model scores which are calculated with the help of the Limited-Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimization algorithm.

Our work falls in the second category where adversary model detection is aimed. Similar to existing studies in the literature, the detection mechanism is based on non-complex statistical scoring and threshold computation with the utilization of model weights to detect the malicious models. However, in our approach, several scoring levels are generated, feeding the k-means clustering algorithm to reduce the number of data points and analyze the models deeply and iteratively. Similar to the proposal in [15], the cluster with the higher number of attendants is considered benign as long as the distance between the clusters exceeds a predefined threshold. In addition, clients may have different computation resources and power budgets. They can be subject to non-identical data sets. Also, as explained in Chapter 3, there is no need to pre-train our detection system, which must be done in auto-encoders.

2.1. Attack Types

There are many attack types to damage the federated learning system, such as distributed backdoor [17], sign flipping [18], and canary-gradient attack [19]. However, they are variants or similar to the model and data poisoning attacks.

In model poisoning attacks, the attacker aims to destruct the global model by adjusting the trained parameters. Thus, the federated learning system is typically defenseless to the model poisoning attacks [20]. The gradient factor attack is an example [2] where the attacker transmits an adjusted model to make the global model's update -1 instead of 1. The malicious clients achieve this aim by multiplying their models by a hyperparameter α which is calculated by the count of total clients. The gradient factor attack can be written as

$$\frac{1}{C}.(C - M + \alpha.M) = -1, \tag{2.1}$$

where C is the number of clients and M is the malicious client count. Model canceling [2] is another example to poison the global model. The objective is to make the global model parameter equal to zero instead of one; thus, it can be expressed as

$$C - M + \alpha.M = 0. \tag{2.2}$$

To control the hyperparameter calculation method in divergent scenarios, there exists a specialized attack server that knows C and organizes M of them as adversaries by informing them. In addition to these hyperparameters, the attack has a poisoning rate parameter P_{rate} to decrease the rate of attacker identification in the detection stage. The model is multiplied with α and P_{rate} as the poisoned model as

$$Model_{poisoned} = Model \cdot P_{rate} \cdot \alpha. \tag{2.3}$$

Even though it has not been studied in the literature, model poisoning attacks can also be targeted to a specific part of the deep learning model. In the literature, deep learning with the targeted attacks are analyzed [16]. These attacks are based on data, and the attackers have an attack objective. This thesis proposes targeting model layers instead of data. Suppose there are k layers in a single model as $\{l_1, l_2, \ldots, l_k\}$. The malicious client can apply model poisoning attacks to any layer l_i. Also, the attack can be applied to a single or more than one layers. With this scheme, the malicious clients can collapse the global model with a single layer without being noticed since the rest of the layers will be untouched. Within the scope of this thesis, we will call this type of attack as layer poisoning attacks. In the experiments, the gradient factor attack is applied to the layers for the layer poisoning attacks.

Data poisoning attack is realized by malicious codes running on the clients. In label flipping attack, the malicious code flips the labels of training data to make the local model useless [2, 6, 13, 14]. For example in [6], the attack is realized by turning every data label to "0". Training data can also be remotely changed by injecting undesired patterns [5].

In this thesis, label flipping and gradient factor attacks are targeted. Moreover, the layer poisoning attack is utilized as a special case of the gradient factor attack to benchmark our detection mechanism.

2.2. Quantized Models and Slow Learners

Most of the IoT end devices are resource-constrained. Especially, power consumption is a major concern for nodes running on batteries. Since federated learning requires multiple data transfers between the clients and the server, model quantization exists as a solution for these devices. Some studies utilize quantized parameters during training [21, 22]. Another study realizes anomaly detection with gradient compression [23].

Slow learners represent the clients with insufficient computing and communication power to exist in a regular training round. Thus, they train fewer data in a round or skip them. They may have smaller batch-size or local batch training. Federated rounds can be reduced for some devices, which have to use their limited power budgets effectively. Hence, they may occasionally attend in federated rounds [21].

Problematic clients represent the misconfigured devices. They try to make the training the same as regular clients; however, they cannot complete this operation and encounter problems. For instance, it can be a device that is programmed to train its data with a GPU though it lacks a GPU. In normal circumstances, the network should treat them as slow learners or occasional participants [21] of the federated learning network instead of regular clients.

In this thesis, a post-training quantization technique is used. Float16 quantization transforms the variables from float32 to float16 [24]. Moreover, maliciousness analysis is also carried out on federated learning systems including quantized nodes, slow learners, problematic clients as well as full capacity devices.

2.3. Benchmarking

The literature has no direct experiment standards since the environments and test types vary in every paper. However, there are test types that consider malicious-

benign client ratio [13, 14, 17, 18], training iterations (training and attacks start from round one) [13, 17, 18] and parameter settings of the detection system [12]. These experiments are widely used to show that they can successfully separate malicious and benign clients by up to 50% malicious-benign client ratio. Some approaches adjust the occurrence probability of data labels to each client differently via the degree of non independent and identically distributed (non-iid) [17] or change the privacy level in a blockchain-based application [25].

Non-iid dataset means that the data distribution over clients is unequal. For instance, in MNIST [26] which contains several handwritten digits, some devices could have intensively handwritten "one"s in their datasets; others could have handwritten "four"s or "five"s simultaneously. Thus, each client or group trains different kinds of data, resulting in distinct client models. With the degree of non-iid, the simulation framework can make this distribution more unequal by decreasing the joint data labels. In that case, clients' data are more divergent.

In this thesis, the detection mechanism has also been experimented with malicious-benign client ratio, training iteration, and parameter adjustments of the system. In addition, several experiments are devised to analyze aspects of the detection system such as layer analysis, differentiation of slow, problematic and quantized devices, and poisoning rate. On the other hand, the thresholding mechanism is analyzed deeply in our work since our detection system is based on adaptively determined threshold values instead of a single threshold value [2].

3. METHOD

Let C be the set of clients in a federated learning system. At any time, there can be some malicious clients. Let M represent the set of malicious clients at a specific round of aggregation. The server sends the same model to all clients. Let $\{w_1, w_2, \ldots, w_i, \ldots, w_N\}$ denote the parameters of the model to be trained. At a specific round of training, the model at each client $c \in C$ can be viewed as $\{w_{1,c}, w_{2,c}, \ldots, w_{i,c}, \ldots, w_{N,c}\}$ (Figure 3.1 lines 1-5).

Since the server computes each model weight by aggregating the parameters from all clients, w_i can be regarded as a statistical variable with a mean μ_i and variance σ_i (Figure 3.1 lines 6-7). When there are no malicious nodes in the system, the variance is small. The existence of malicious nodes causes a bigger variance. Hence, the server should aggregate only the client parameters that exist in some range which can be expressed as can be called as parameter existence range, $[t_i^-, t_i^+]$ around the mean (Figure 3.1 lines 8-9) as

$$t_i^- = \mu_i - \theta.\sigma_i, \tag{3.1}$$

$$t_i^+ = \mu_i + \theta.\sigma_i. \tag{3.2}$$

Since the server is unaware of the malicious nodes at the aggregation time, it cannot specify an exact value of θ. If it is not correctly set, malicious nodes can enter the aggregation or too many benign nodes can be left outside. Hence, the server has to scan the client parameters for a set of different values for $\theta = [\theta_{\min}, \theta_{\max}]$. Empirical studies show that $\theta_{\min}$ should be much less than σ_i and $\theta_{\max}$ should be at least one σ_i ahead of $\theta_{\min}$.

Let $x_{i,c}$ be the binary variable that is 1 if and only if $w_{i,c}$ is in the range as $t_i^- \leq w_{i,c} \leq t_i^+$ (Figure 3.1 lines 11-12). Then the amount of parameters of a client that can take place in the aggregation of a specific round can be computed as $V_c = \sum_{i=1}^{N} x_{i,c}$

(Figure 3.1 line 13). Based on this fact, a score can be calculated for each client as

$$S_c = \frac{V_c}{N} * 100 = \frac{\sum_{i=1}^{N} x_{i,c}}{N} * 100, \tag{3.3}$$

where it assigns a score in $[0 - 100]$ range to each client in the system for every value of θ (Figure 3.1 line 14). Assume that there are L distinct values for θ. Then there will be L scores for each client corresponding to all values of θ. The scores of a client can be collectively represented as L-dimensional data point with θ_l increasing from the first entry to the last entry (Figure 3.2 lines 1-5) as

$$\vec{\mathbf{S}}_c = \{S_{c,1}, S_{c,2}, \ldots, S_{c,l}, \ldots, S_{c,L}\}. \tag{3.4}$$

In this way, there will be $|C|$ data points fed to a k-means clustering algorithm for two clusters of uneven size so that data points corresponding to malicious and benign clients can be placed into two disjoint clusters (Figure 3.2 line 6). Let $\vec{\mathbf{O}}_1$ and $\vec{\mathbf{O}}_2$ be the centroids of these clusters. The proposed method assumes that the majority of clients are benign.

When some malicious clients exist in the federated learning system, the distance between the centroids of each cluster which is $|\vec{\mathbf{O}}_1 - \vec{\mathbf{O}}_2|$, will become large. If the system consists of all benign nodes, then $|\vec{\mathbf{O}}_1 - \vec{\mathbf{O}}_2|$ will be small. To obtain a measure independent of L, the distance between centroids of the clusters should be normalized by L as (Figure 3.2 line 7)

$$d_{O1,O2} = \frac{|\vec{\mathbf{O}}_1 - \vec{\mathbf{O}}_2|}{L}. \tag{3.5}$$

The server has to check $d_{O1,O2}$ against a threshold γ to decide whether to use both clusters or only the larger one in the aggregation. Note that $d_{O1,O2}$ can have a value in $[0, 100]$ range due to the definition of the client score Equation (3.3). Hence, setting γ to a low value around ten usually provides the necessary filtering.

Input: CModels, theta

Output: Score of each Model

1: $cmLength \leftarrow CModels.length$
2: $ScalingParameter \leftarrow 1/cmLength$
3: **for** $i \leftarrow 0$ ... $cmLength$ **do**
4: $\quad tempModels[i] \leftarrow CModels[i] * ScalingParameter$
5: **end for**
6: $meanModel \leftarrow mean(tempModels)$
7: $stdModel \leftarrow std(CModels)$

8: $tHigh \leftarrow meanModel + theta * stdModel$
9: $tLow \leftarrow meanModel - theta * stdModel$

10: **for** $i \leftarrow 0$ ... $cmLength$ **do**
11: $\quad lowScore \leftarrow isGreater(CModels[i], tLow)$
12: $\quad highScore \leftarrow isLower(CModels[i], tHigh)$
13: $\quad existScore \leftarrow inrange(lowScore, highScore)$
14: $\quad Score \leftarrow count(existScore)/W * 100$
15: **end for**

Figure 3.1: Score calculations algorithm.

When γ is set to a very low value, some of the benign nodes can be left from the aggregation if there are no malicious nodes. When γ is set to a very high value, malicious client weights appear in the aggregation if they exist in the system.

On the other hand, γ can be autonomously determined via validation accuracy. To do that, the server uses two different parameters: the accuracy values of the clients' models and model scores at the current round. The algorithm is listed in Figure 3.3.

Input: CModels, thetaMin, thetaMax, thetaStep, threshold

Output: Benign CModels

1: **for** $i \leftarrow thetaMin$ ... $thetaMax$ **do**
2: $tempScores \leftarrow calcScore(CModels, i)$
3: $Scores \leftarrow assignScore(tempScores, i)$
4: $i \leftarrow i + thetaStep$
5: **end for**

6: $clusters \leftarrow calcKMeans(Scores)$
7: $dist \leftarrow distance(clusters)$

8: **if** $dist \geq threshold$ **then**
9: $CModels \leftarrow deleteMaliciousModels(CModels)$
10: **end if**

Figure 3.2: Malicious model detection algorithm.

As can be seen, there are two initial variables for the adaptive gamma selection algorithm. These are the lastAcc and gamma values. The adaptive gamma calculation starts with accuracy calculation before the detection mechanism (Figure 3.3 line 1). After that, the algorithm runs the detection mechanism with the gamma value and collected models (Figure 3.3 line 2). Then, it executes malicious model detection algorithm (Figure 3.2) again to measure how the detection mechanism is performed (Figure 3.3 line 3). Suppose the accuracy after the detection mechanism is higher than the accuracy value before the detection mechanism is performed, or the accuracy after detection is higher than the previous round's accuracy (Figure 3.3 line 4). In that case, the accuracy increases dramatically, and we know that the non-eliminated clients after the detection mechanism are benign due to high accuracy. The expected increase in accuracy can be expressed with γ_{gap}. It should be large enough to stabilize γ after the network catches high accuracy values; and small enough to notice the accuracy

difference between the poisoned and disinfected aggregated model. γ_{gap} is selected as ten within the content of this thesis. Thus, the algorithm computes distance $d_{O1,O2}$ between benign clusters (Figure 3.3 line 5) and updates γ (Figure 3.3 line 6). The new value of γ is calculated by the distance summed with γ_{step} to prevent small fluctuations. It is selected as three within the scope of this thesis. Otherwise, γ would be changed in every round since clients' models are different in each round. Finally, the accuracy of this round is stored as "lastAcc" so that it can be used in the next round (Figure 3.3 line 7).

Input: CModels, Gamma, lastAcc

Output: Gamma

1: $beforeAcc \leftarrow computeAcc(CModels)$
2: $tempModels \leftarrow detectMalicious(CModels, gamma)$
3: $afterAcc \leftarrow computeAcc(tempModels)$

4: **if** $afterAcc \geq beforeAcc + \gamma_{gap}$ **or** $afterAcc \geq lastAcc + \gamma_{gap}$ **then**
5: $\quad dist \leftarrow distance(tempModels)$
6: $\quad gamma \leftarrow dist + \gamma_{step}$
7: **end if**

8: $lastAcc \leftarrow afterAcc$

Figure 3.3: Adaptive gamma algorithm.

Clustering can be extended to handle various types of nodes in the system. In almost every federated learning system, the server knows the learning types of the nodes: Some nodes can learn slower than others. Some nodes can provide quantized weights due to their characteristics. Hence, the server may prefer to have multiple clusters in the system. Still, in that case, the proposed method generates the clusters. It is up to the program running on the server to select the clusters in the aggregation

by setting up separate γ values for each case.

In addition to clustering models' scores, the detection mechanism can also score specific layers, generating the distance of cluster centroids $d_{O1,O2}$ for each layer. Then, the system can cluster the layers separately and act differently for each layer. This mechanism is a significant part of detecting layer poisoning attack since it controls each layer individually rather than globally. Hence, it improves the robustness of the detection mechanism.

4. SIMULATION FRAMEWORK

Federated learning is a distributed machine-learning mechanism that allows collaborative training among multiple clients without sharing the data. Instead, clients share their models' weights to generate the global model, which is created with aggregation methods based on the clients' weight parameters. Then, clients use this aggregated model as their new local model to continue the training. Training and aggregation phases continue until the aggregated model attains the targeted accuracy. During this process, some clients may attack the infrastructure using the attack methods explained in Chapter 2.

To simulate the behaviour of this system, a real-time federated learning framework is designed. The following sections introduce the requirements, modeling and implementation of the framework.

4.1. Requirements

The direct functional requirements of the federated learning mechanism include its primary abilities. These requirements are essential to test the proposed method with or without artificial or syntactical data. The first one is the training phase of each device. This requirement is the primary function of this thesis and federated learning structure. As stated, federated learning includes aggregating trained models from each device. So, the structure allows the devices to train their models separately with their data. The second ability is the aggregation of these models, which is the second phase of federated learning. The aggregation of models is the primary function of the federated learning principle; thus, the framework has to offer this ability directly.

The following requirement is the attacking ability of malicious devices. Since this thesis is related to the defense mechanism of federated learning, the structure should be tested with malicious activity from devices. Thus, it should provide a mechanism

for devices to generate attacks from the devices.

The last requirement is the detection mechanism of malicious activity. This thesis proposes a detection mechanism that prevents malicious devices from attacking the aggregated model. So, the structure should allow the server to control the devices' models in each round with complete flexibility.

Indirect functions consist of supporter abilities of the federated learning structure, such as preparing the datasets before training, a communication protocol for transmitting the model parameters and controlling the malicious devices from a centralized server. Moreover, the structure should be autonomously maintained since the federated learning phases can require time. All processes should be completed without human interactions. On the other hand, it should also provide a user interface simultaneously for detailed tests and monitoring.

4.2. Modeling

The proposed federated learning structure is based on the client and server model, where a centralized server controls the distributed clients. Moreover, there is a need for an attack control server that can inject maliciousness into selected devices and set parameters for necessary attacks.

The clients of the federated learning structure are the devices where the training phase is executed. Every client's responsibilities include training a specified deep-learning model and transmitting them to the server for the aggregation phase. Thus, every device has a dataset and deep-learning model configurations to handle the training. In addition to that, every device has a general configuration that controls the aspect of devices, such as being a slow, problematic or quantized client in the structure. Also, every device has a configuration for maliciousness level. This configuration is active only for malicious devices. If the attack control server inject maliciousness in a client, this configuration will be active and the benign client become a malicious

client. So, benign clients do not use them while training and transmitting the model. Lastly, the client has a communication method to handle the general communication between the client and the servers. Communication between the client and the attack control server is also used if the device is malicious.

The centralized server is the device that controls the federated learning structure. It collects and aggregates the models, utilizing the federated learning structure's proposed detection mechanism and automation. The server can use an aggregation methods such as the federated averaging, federated mean or trimmed mean algorithm for aggregating the models. To control the flow of simulation, it has a unique functionality as story handling. In story handling (Table 4.1), the server has a queue of primary functional jobs executed one by one in order. Thus, the user should enter a list of jobs at the beginning of the federated learning structure. The jobs are listed in the story handling table.

For instance, "0 4 3 2 1" means "Train four times (0 4). Then collect the models (3), after that, aggregate them (2) and transmit them back to the clients (1)". This story instance represents a single round and can be repeated for every round. Moreover, the detection mechanism is placed after collecting the model and before aggregation ("3-2"). Therefore, the server controls every model before the aggregation step. The sequence diagram of general structure and story timeline can be analyzed in Figure 4.1. It shows the numbers which operations the story refers to in the story handling process.

These jobs also represent the server's functionalities. Although it is not mandatory to include them in a single round, each job should be placed in every round to acquire a complete round. Thus, each functionality can be initialized with a job represented independently. Lastly, the server has a list of client information, such as their communication and device configuration details for automation operations via story handling and system maintenance.

Table 4.1: Story handling in the server.

#	**Operation**	**Description**
0	Start Training	Start the training phase in every device
1	Send Model Order	Collect models from clients
2	Run Detection Mechanism	Runs the detection mechanism with models
3	Aggregate Models	Aggregate the client models
4	Send Model	Send the aggregated model to all clients

The attack control server is the server that controls the malicious device. Some attack types require hyperparameter configuration. Since a device could not know the number of malicious devices in the structure to compute the hyperparameters, they should be provided from the attack control server. The attack control server can also select the attack type for each malicious device. Thus, malicious devices can work collaboratively with the same parameters and attack type the attack control server defines.

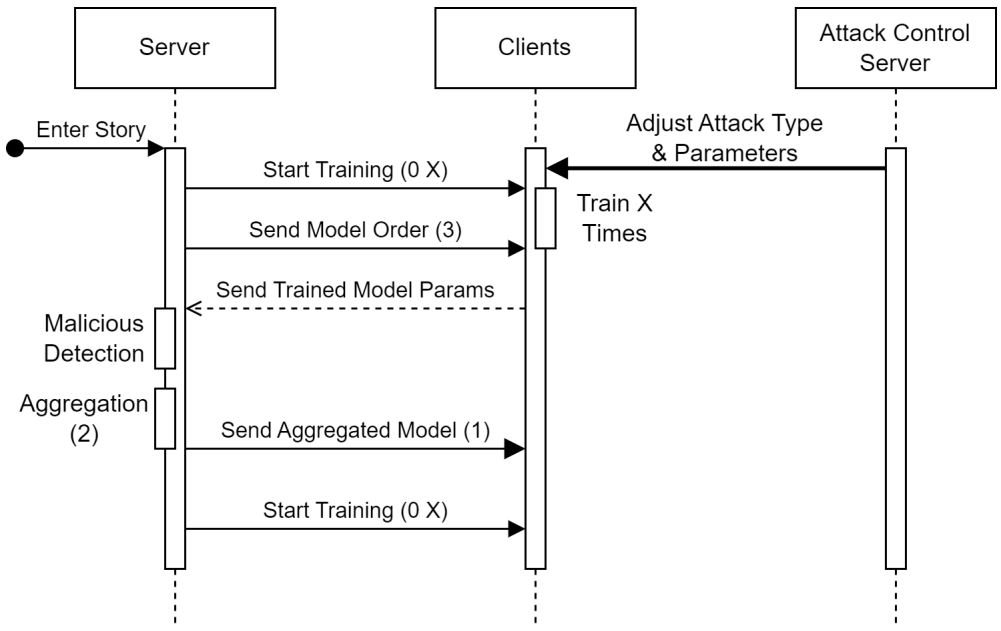


Figure 4.1: The sequence diagram of system.

4.3. Architecture

The system's architecture is based on separate devices and uses sockets to communicate data, such as model transferring between devices and automation orders which is generated from the story handling. Thus, the devices use no common object or memory resources.

Figure 4.3 and 4.2 show the UML diagram of the federated learning. Devices have unique port numbers, message queues, and message handler functions to utilize independent communication. The devices pop the message queue to handle the following message until there is no message. The "_listen()" function is prepared for listening to a specific port, "_input()" function is used for the user interface since the system should be controlled with human interactions. Similarly, the story handling works with the queues. Also, servers and clients have an additional communication method that is used for monitoring purposes which can be seen in Figure 4.4.

Several functionalities with similar duties such as attacks, quantization, and aggregations, are generated as interfaces and instances. For instance, the gradient factor and label flipping attack are instances of the Attacks interface and share standard configurations and methods. Devices use these instances since their variables and methods are bounded to the interfaces. Additional usage of the framework is explained in Appendix.

4.4. Implementation

In order to create the infrastructure for this work, Python-based scripts are used. First of all, there are two types of devices in this structure. The first type is the general federated learning devices. These devices' primary focus is to maintain the federated learning processes in a synchronized way.

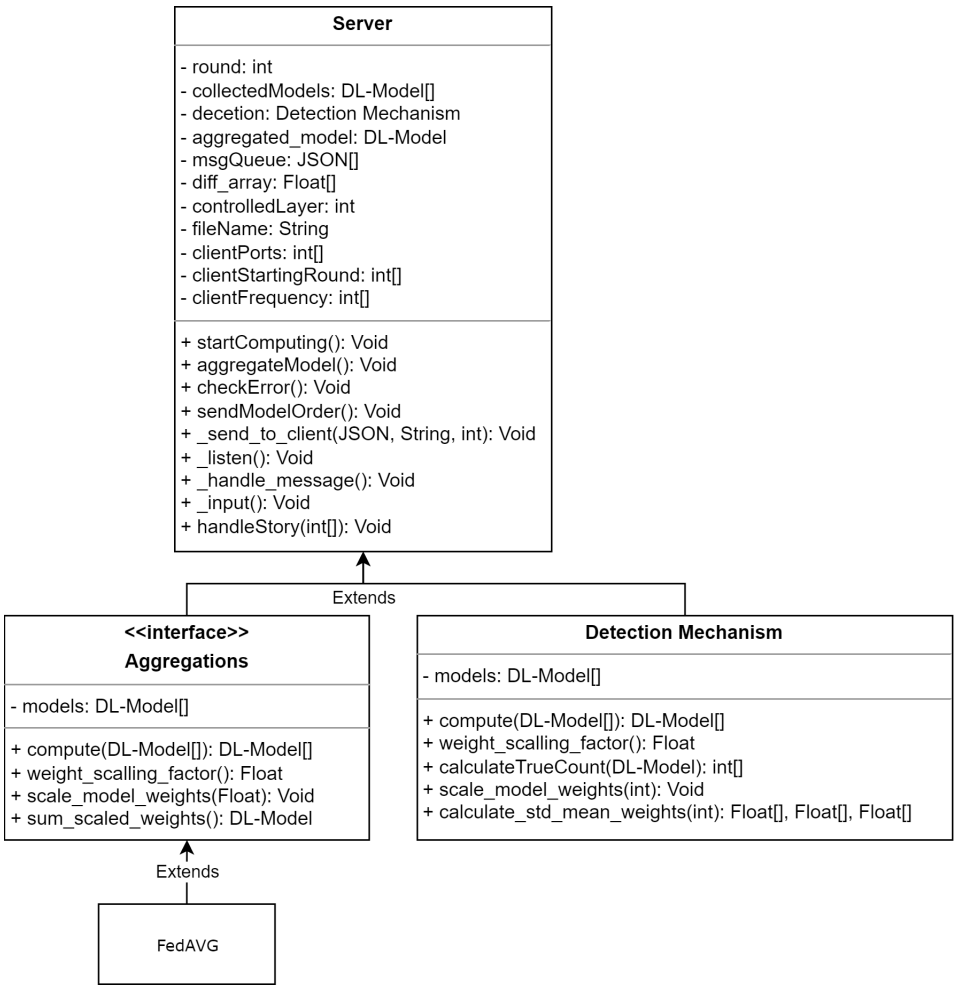


Figure 4.2: UML diagram of server.

These devices are Clients, Server, and Attack Control Server. They are generated by Python classes and run under separate bash scripts. This separation leads to separate devices with no common memory spaces or shared objects since every client has a specific Python process. Thus, every device on the system needs a standard communication protocol to communicate and maintain the processes of a federated learning structure. The Pipe Process Communication protocol is used to make this happen, and every device has a unique PORT number similar to the IP address. With this solution, every client can transport their data via sockets. The protocol and the flow of communication can be seen in Figure 4.4.

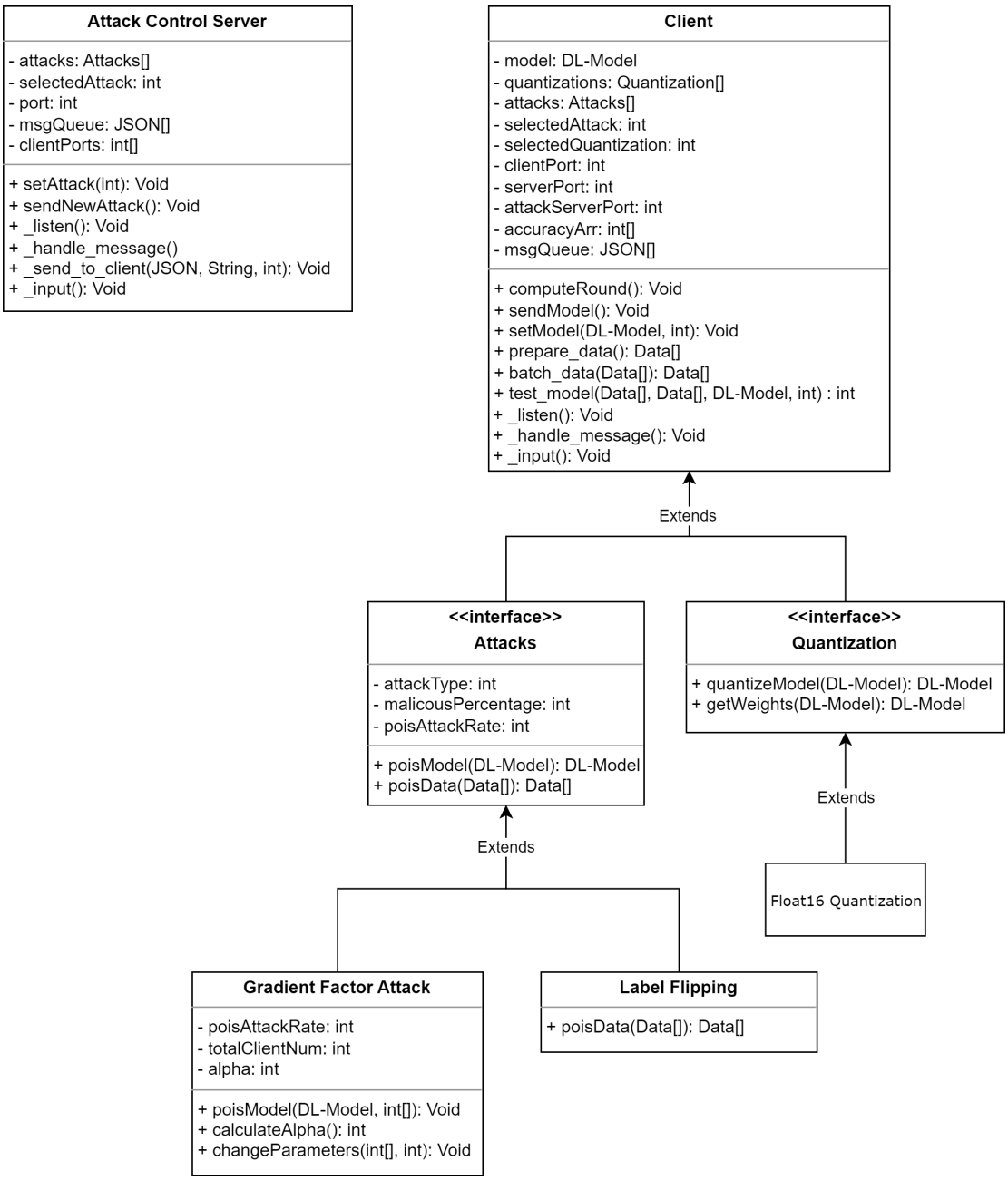


Figure 4.3: UML diagram of client.

Lastly, all clients have two major modes: Benign and Malicious. Benign mode is the default mode of a client, and the client with this mode does not poison the model or data. Nevertheless, in the malicious mode, the client starts to poison them after every training phase and is controlled by the attack control server.

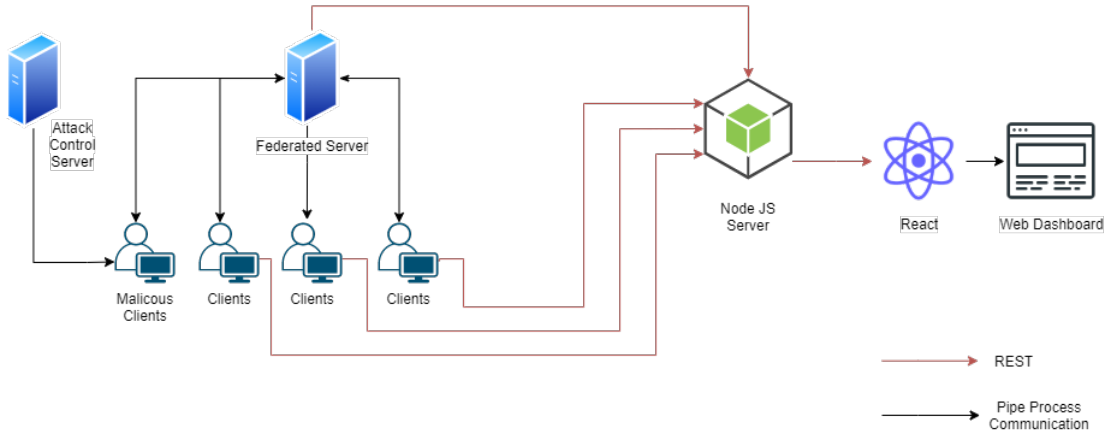


Figure 4.4: Simulation framework.

There are also submodes of each major mode, such as slow learners, a client with quantization, and problematic clients. Slow learners are triggered less frequently than other clients. Some clients may quantize their model with IEEE standard float16 format to send the quantized model. Some clients may be problematic. These devices do not try to harm the system; however, they train problematically rather than the standard client since they try to train with a GPU configuration with a disabled GPU. So, these devices encounter with optimization problems in the training phase.

The federated server controls the federated learning process between clients and handles the main tasks of the federated learning system. The server has a subscribers list that consists of client addresses (PORT numbers). If a client wants to connect to the system, the server also accepts an additional parameter: the frequency of a model's training. Suppose a client declares a slower communication frequency than the server's demand. In that case, the server identifies that client as the slow learner and skips collecting the parameters from that client at some rounds. The differentiation of slow learners among other clients is detailed in the experiments. The server's duty includes the anomaly detection system for the collected client models via the proposed method. Additionally, the server manages the structure autonomously with the job queue. The job queue is designed to make all processes autonomous and includes the periodically listed job order. The server uses this queue iteratively and controls the structure. This mechanism is called a story in our structure and consists of numbers.

The attack control server does not directly transmit data to the federated learning model. If malware takes the device's control, the client generates a connection to this server and is controlled by the Attack Control Server. The server adjusts the attacking parameters and controls the attack type. The attacking parameters depend on the number of malicious ($|M|$) and total clients ($|C|$) to collapse the global model with the gradient factor attack. Label flipping is also realized by the attack control server.

The second type of device is the monitoring device. These devices have no direct impact on the federated learning process. They collect the accuracy data from a specific benign client and plot it in a web application that is prepared with React. Additionally, a NodeJS server gathers the data with the representational state transfer (REST) application programming interface (API). In a nutshell, the selected client sends the accuracy result over the REST request to serve it to the web application as data. These devices are not mandatory in the structure since the data can also be monitored from the console logs of Python scripts.

5. EXPERIMENTS

This chapter presents the experiments carried on the framework. First, the detection mechanism is analyzed deeply with a Convolutional neural network(CNN) model by the gradient factor and label flipping attack. Then, the LeNet5 [27] architecture is tested with an increased client count and layer poisoning attack. Finally, the detection mechanism is compared with other approaches from the literature.

5.1. Basic Tests

The system is setup with ten clients. Though the aggregation method is programmable in the framework, federated average [1] is used in the experiments since harmful models are eliminated from the system with the proposed method. The scanning parameter θ is defined as follows: $\theta_{min} = 0.7, \theta_{max} = 1.6, \theta_{step} = 0.1$. Hence, ten data points constitute a score for each client at each round. The dataset consists of hand-written numbers, which is the regular MNIST [26] dataset, and the model is a CNN model that includes two convolution layers, two max-pooling layers, flatten, and softmax activation.

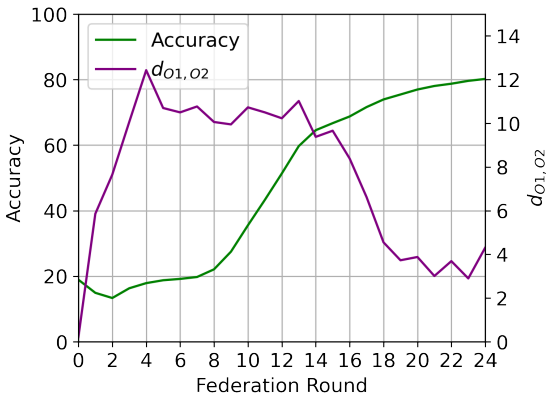


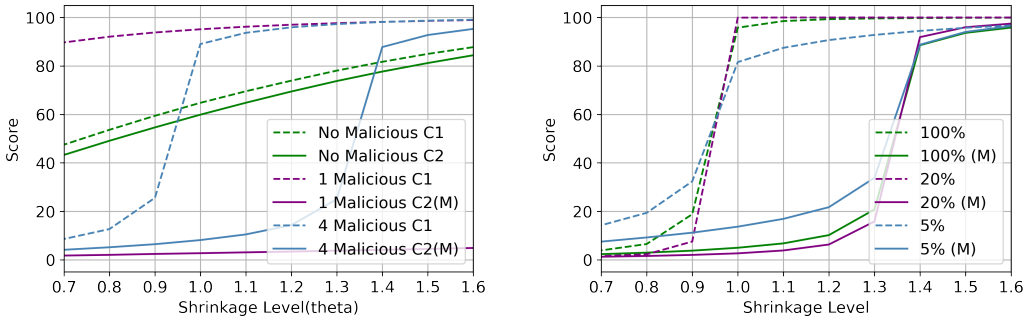
Figure 5.1: $d_{O1,O2}$ and accuracy during federated learning.

Figure 5.1 shows the accuracy versus $d_{O1,O2}$ in a federated learning network where all clients are benign, learn at full rate, and communicate at full word length. Learning is done in 25 rounds. In each round, each client takes four local batches (lb). Each local batch represents four batches of data; thus, each round means 16 batches. Batch size is 64. Dataset of clients is randomized iid with the seed of unique port numbers. The accuracy generally increases in each round. Additionally, $d_{O1,O2}$ decreases while the accuracy starts to increase. This situation can be interpreted as; lower accuracy means a higher possibility for the malicious devices to affect the system because the gradients of the clients' model have a lower similarity index in low accuracy compared to higher accuracy.

5.1.1. Gradient Factor Attack Detection

To test our malicious client detection system, some clients are transformed from benign to malicious. The gradient factor attack with 100% poisoning rate is used by the malicious clients. The scores of cluster centroids, namely $\mathbf{O_1}$ and $\mathbf{O_2}$, for each θ are shown in Figure 5.2a.

In Figure 5.2a, dashed lines represent the centroid of scores for the cluster with more clients while solid lines are used for showing the same information for the cluster with fewer clients. Recall that the smaller size cluster may imply the set of malicious nodes if $d_{O1,O2} > \gamma$. When there is one malicious client, the centroids of two clusters are far away from each other at all values of θ as shown by dashed and solid purple lines, and the distance is $d_{O1,O2} = 92.47$. When the number of malicious clients increases to four, the centroids of two clusters may come closer at some values of θ as seen with blue lines. However, the departed regions cause the overall distance to be $d_{O1,O2} = 36.92$. So, in both cases, the cluster of malicious clients can be identified and erased before the aggregation. In the same figure, green lines demonstrate a distance of 4.31 between two clusters. Since this value is smaller than γ, the server takes the clients of both clusters into aggregation.



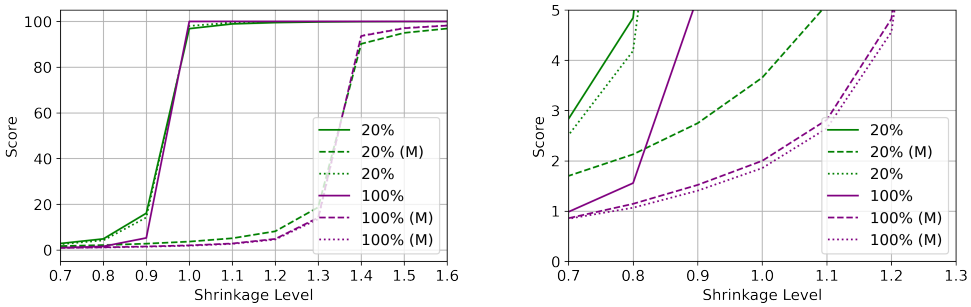
(a) (b)

Round	Poisoning Rate		
	100%	**25%**	**5%**
1st Round	60.94	51.87	15.73
10th Round	38.97	38.32	32.10
20th Round	39.13	39.41	34.73

(c)

Figure 5.2: Computationally rich devices, gradient factor attack, $|C| = 10$. (a) 100% poisoning rate, (b) various poisoning rates with $|M| = 4$, (c) $d_{O1,O2}$ at different rounds and poisoning rates.

5.1.1.1. Different Poisoning Rates. The effect of different poisoning rates is shown in Figure 5.2b. In this scenario, four malicious clients are taken into account with different poisoning rates. It can be seen that even 5% poisoning can be detected by the proposed method since $d_{O1,O2} = 31.22$ is still considerably higher than the threshold. In Figure 5.2c, the effect of malicious clients on $d_{O1,O2}$ at different rounds can be viewed. The test has been made with four malicious clients, which send their poisoned model at the end of four local batches. As can be seen, $d_{O1,O2}$ is close to the specified threshold $\gamma = 15$ at the first round with a 5% poisoning rate. Besides, the clustering algorithm classifies a benign client as malicious in addition to the other malicious ones at the tenth round at this poisoning rate. Thus, the model of this benign client will also be deleted with malicious models. This wrong classification may be neglected since the poisoning rate is low. This problem is not present when the attack is realized at a round where the global model accuracy is high even though the poisoning rate is small. For higher poisoning rates, it can be correctly detected at all rounds.



(a) (b)

Clusters	Poisoning Rate	
	100%	***20%***
Solid (O1)	$\|M\| = 0$	$\|M\| = 0$
Dashed (O2)	$\|M\| > 0$	$\|M\| > 0$
Dotted (O3)	$\|M\| > 0$	$\|M\| = 0$

(c)

Figure 5.3: Slow and problematic learners, gradient factor attack. (a) One slow learner, one problematic client, (b) zoomed in Figure 5.3a, (c) cluster maliciousness results of Figure 5.3a.

5.1.1.2. Heterogeneous Clients. Different types of clients are studied. The system is implemented with one slow learner and one problematic device in addition to eight regular devices. The slow learner represents the clients with half the computing power compared to a regular client. If a regular client makes four local batches locally in a round, the slow learner makes two local batches. Thus, they are trained with fewer data in the meantime. The problematic client represents the clients with configuration problems. They are similar to slow learners since they do not fully train their data.

The tests are made with four regular malicious devices with different poisoning rates. Since the server is aware of slow learners due to the registration parameters, it prepares the detection system for three clusters: regular, slow learners, and others. Note that the server is aware of neither malicious nor problematic devices. The results can be seen in Figure 5.3a. $d_{O1,O2}$ is 39.42, $d_{O1,O3}$ is 0.03 and $d_{O2,O3}$ is 39.38 at 20% poison rate; $d_{O1,O2}$ is 39.17, $d_{O1,O3}$ is 39.28 and $d_{O2,O3}$ is 0.11 at 100% poison rate.

In this figure, dotted lines represent the third cluster. The dashed and solid lines are used as before. When the malicious clients attack the server with 100% poisoning rate, the detection system can put every benign client in the same cluster as expected. The centroid score is very high. However, the other two clusters contain only malicious clients. The distance parameters tell the server to discard both of them. If malicious clients lower the poisoning rate to 20%, the benign clients can be successfully separated into two clusters for slow learners and regular nodes. In this case, all malicious clients are listed in a single cluster. The distance parameters tell the server to discard this cluster. The zoomed figure shows the clustering in Figure 5.3b.

When the system does not contain any malicious nodes, the score between the benign and slow learners is very small. The score difference between various slow learner counts can be seen in Figure 5.4a, 5.4b, 5.4c. The results are collected from three different random points according to the federation round time since the accuracy affects the attacks. The beginning is collected between the first and fifth rounds; the middle is collected between the tenth and fifteenth rounds; the end is collected between the twentieth and twenty-fifth rounds. In Figure 5.4a, $d_{O1,O2}$ is 7.16 for "End", 5.52 for "Middle" and 5.06 for "Beginning". In Figure 5.4b, $d_{O1,O2}$ is 7.34 for "End", 6.02 for "Middle" and 8.43 for "Beginning". In Figure 5.4c, $d_{O1,O2}$ is 9.93 for "End", 11.02 for "Middle" and 11.54 for "Beginning". The slow learners have a low impact on the scoring scheme. Slow learners can be dispatched into different clusters. However, this is not a problem for the federated learning system as long as the distance between clusters is lower than the threshold γ.

The final set of experiments for model poisoning is done with ten clients which send their model parameters after quantizing with float16. The accuracy is between 75-80% in every test. The "No Quantized" line represents a test with clients whose quantization is not applied, "10 Quantized" line represents a test with clients that quantize their models. Since there are ten clients in these tests, all clients quantize or do not quantize their models simultaneously.

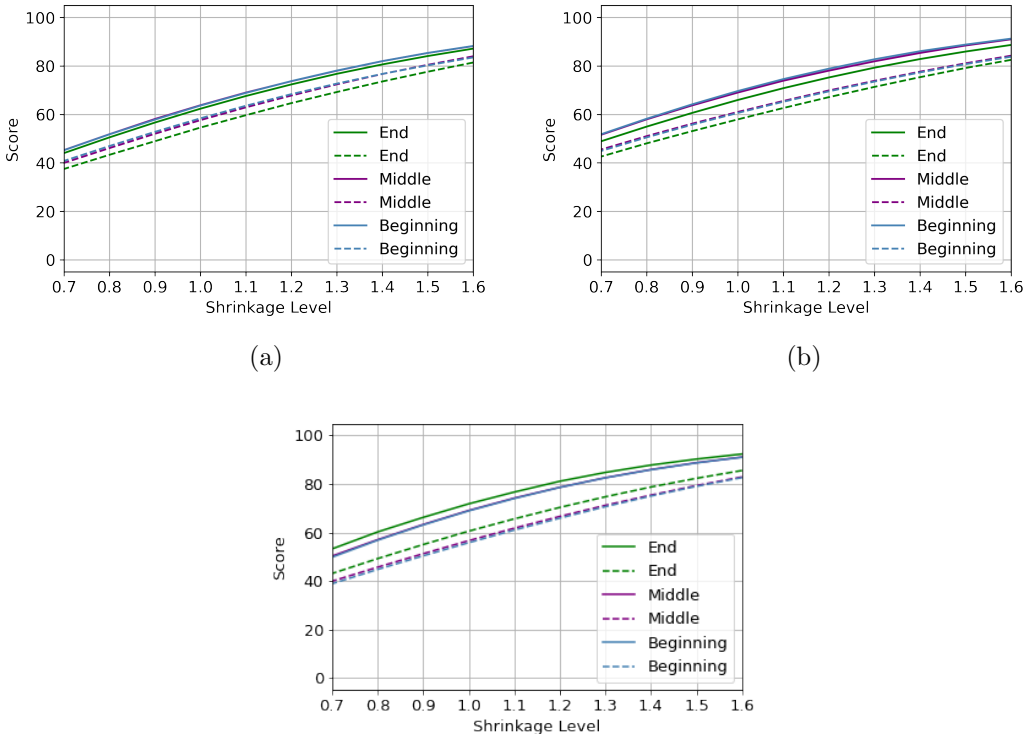


Figure 5.4: Slow and problematic learners; various amount of slow learners. (a) One slow learner, (b) two slow learners, (c) four slow learners.

Figure 5.5 shows the impact of quantization over a same model. Malicious clients attack with 100% poisoning rate. Ten quantized models with only one malicious model have a slightly lower score than ten regular benign ones as shown in Figure 5.5a. When the number of malicious nodes increases in the federated learning system, they show a performance quite similar to regular ones (Figure 5.5b and c). In Figure 5.5b, where $|M| = 3$, $d_{O1,O2}$ is 84.81 when there are no quantized clients and $d_{O1,O2}$ is 80.35 when there are ten quantized clients. In Figure 5.5c, where $|M| = 4$, $d_{O1,O2}$ is 39.18 when there are no quantized clients and $d_{O1,O2}$ is 37.53 when there are ten quantized clients.

The same system has been tested with various numbers of quantized clients. Their performance is better than the ten quantized models. Hence, quantized and regular models can be combined in a single system and still malicious nodes can be detected.

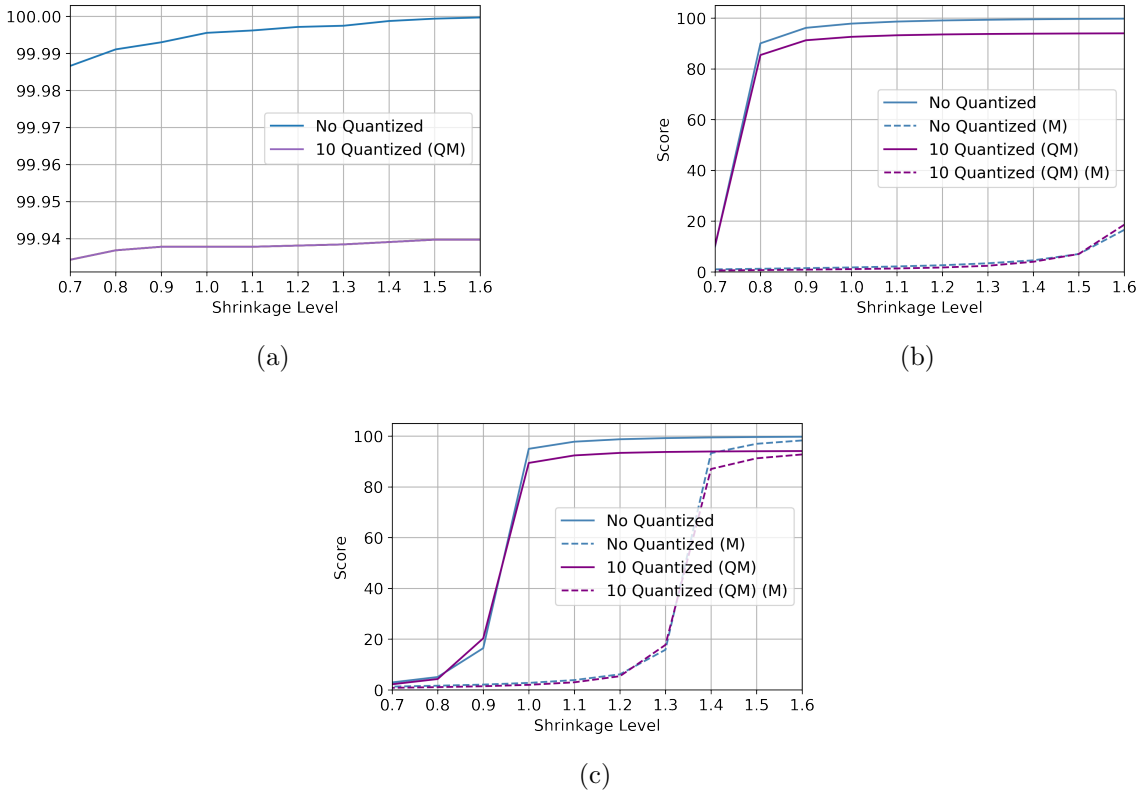
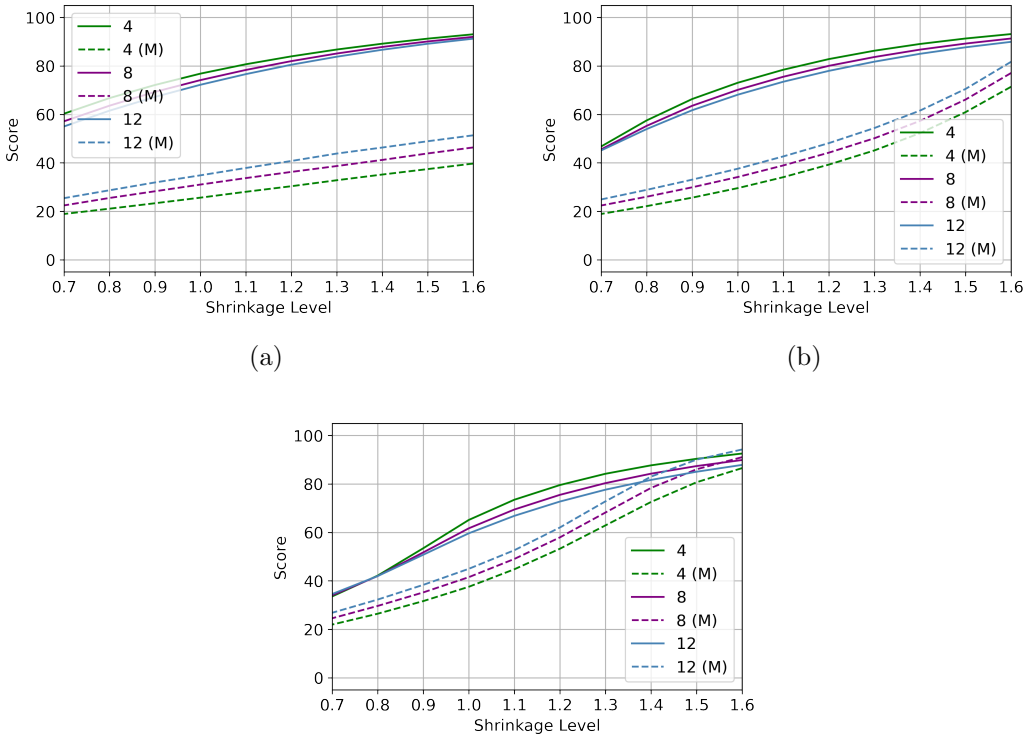


Figure 5.5: Comparison of non-quantized and ten quantized models under gradient factor attack. (a) $|M| = 1$, (b) $|M| = 3$, (c) $|M| = 4$.

5.1.2. Data Poisoning Attack Detection

For data poisoning experiments, the system is trained around 75% accuracy. Other training parameters are the same as model poisoning experiments except different local batch (lb) sizes as shown in Figure 5.6. The detection system clusters every client correctly, even at 40% malicious rate (four malicious clients among ten clients). In Figure 5.6c, where $|M| = 4$, $d_{O1,O2}$ is 18.38 for 4 lb, 11.69 for 8 lb and 8.69 for 12 lb. The other malicious rates' results showed in Figure 5.6a, where $|M| = 1$, $d_{O1,O2}$ is 50.84 for 4 lb, 43.22 for 8 lb and 37.40 for 12 lb; and in Figure 5.6b, where $|M| = 3$, $d_{O1,O2}$ is 36.54 for 4 lb, 29.47 for 8 lb and 24.14 for 12 lb. The score difference between clusters decreases if the malicious rate or local batch size increases. The effect of the malicious rate is obvious since they modify the mean μ and standard deviation σ with their population.



(a) (b)

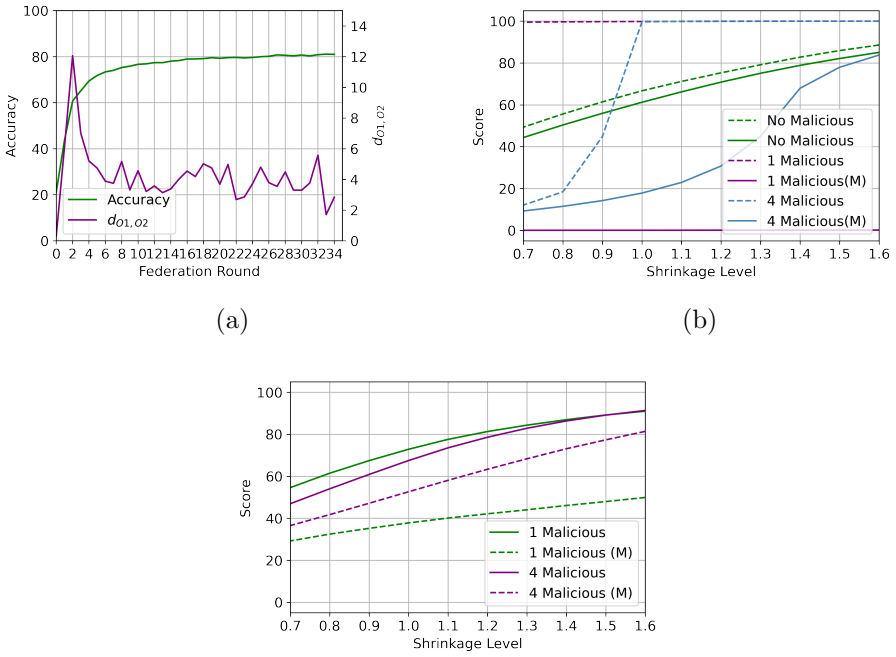
(c)

Figure 5.6: Label flipping attack with the increasing local batch (lb). $|C| = 10$. (a) $|M| = 1$, (b) $|M| = 3$, (c) $|M| = 4$.

On the other hand, increased local batch size is an unexpected aspect of the detection system since the gap between clusters decreases with the increase of malicious devices. We may explain this aspect as client models become different with larger local batch sizes since each client's dataset may not be similar to other devices' datasets.

In Figure 5.7, similar tests are made with the Fashion-MNIST [28] dataset to compare the results. In each round, each client training is done with three local batches (lb) in two epochs. Thus, the accuracy climbs up faster; however, the increase makes the system more vulnerable to data poisoning attacks since each model trains more between rounds. On the other hand, the results are quite similar to the tests with the MNIST. Figure 5.7b shows the system's performance against gradient factor attack. Here, $d_{O1,O2}$ becomes 4.59 for $|M| = 0$, 99.69 for $|M| = 1$ and 39.38 for $|M| = 4$. Figure 5.7c shows the system's performance against label flipping attack. Here, $d_{O1,O2}$

becomes 28.38 for $|M| = 1$ and 13.15 for $|M| = 4$.



(a) (b)

(c)

Figure 5.7: Comparison of clusters' centroids, Fashion-MNIST. (a) $d_{O1,O2}$ and accuracy. $|C| = 10, |M| = 0$, (b) $|C| = 10$, 100% poison rate, gradient factor attack, (c) $|C| = 10$, label flipping attack.

5.2. Experiments on LeNet5 CNN

A popular CNN network is tested with more clients in the following tests to examine the structure and defense mechanism more deeply. To do that, the structure is empowered by the LeNet5 CNN network, which has three convolution, two pooling, two dense, and one flatten layers. Similar to the basic tests' configurations, each test is made in 25 rounds with four local batches for each round. The dataset for LeNet-5 is MNIST and prepared in the same way as basic tests.

Among 30 clients, up to 14 malicious clients participate in the following scenarios. Thus, these experiments are more similar to the real-world scenarios since they consist of more clients with a well-known CNN network.

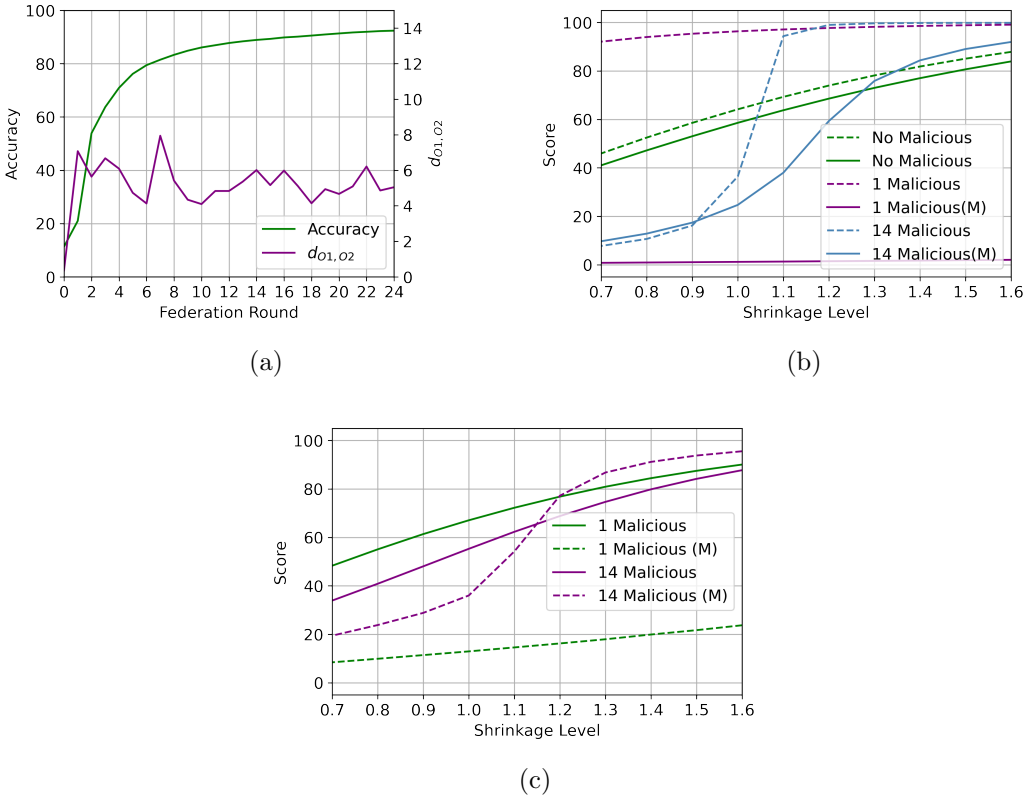


Figure 5.8: Comparison of clusters' centroids. $|C| = 30$, LeNet5, MNIST. (a) $d_{O1,O2}$ and accuracy, (b) 100% poison rate, gradient factor attack, (c) label flipping attack.

Figure 5.8a shows accuracy and $d_{O1,O2}$ values where all clients are benign and learn at full rate. Similarly, the accuracy increases dramatically in earlier rounds and then slowly climbs to 93 percent. As can be analyzed in Figure 5.8a, model updates of clients have lower similarity in earlier rounds; hence $d_{O1,O2}$ is higher. Nevertheless, $d_{O1,O2}$ value fluctuates between 4 and 8. Moreover, the LeNet5 version of tests has a lower $d_{O1,O2}$ value compared to other tests in the basic tests.

5.2.1. Gradient Factor Attack Detection

The impact of the gradient factor attack can be seen in Figure 5.8b. The gradient factor attack is used with the 100% poisoning rate. Additionally, the malicious clients started to attack after 25 rounds. Similar to experiments of the basic tests, Figure 5.8b includes three different tests with different numbers of the malicious client in

the system. There are two scores of cluster centroids for each test, and the dashed line represents the cluster with the higher participant. When there is no malicious client in the system, $d_{O1,O2}$ value becomes 5.04 as stated in Figure 5.8a. If there is one malicious client, the score of two clusters diverges, and $d_{O1,O2}$ is 95.32. When the number of malicious clients increases to fourteen, the clusters have diverged, and $d_{O1,O2}$ becomes 17.09. Thus, the detection mechanism can separate malicious clients even if the number of malicious clients becomes fourteen, similar to the basic tests.

5.2.2. Data Poisoning Attack Detection

The performance of the detection mechanism against the label flipping attack is presented in Figure 5.8c. The system is trained up to 90% accuracy before the attack.

In the LeNet5 tests, the label flipping attack generates $d_{O1,O2}$ values similar to the basic tests with low number of clients. The test with one malicious client generates two clusters. The detection system can easily catch as $d_{O1,O2}$ is 56.69. However, the label flipping attack with fourteen malicious clients leads to unanticipated clusters' centroids. The score of the malicious cluster exceeds the benign cluster's score after $\theta = 1.2$. The malicious client model's similarity could explain this impact due to the adjusted data label. The benign clients have no connection to their training data; nevertheless, the malicious clients do. All malicious devices can change their data labels to a specific value, such as zero in MNIST. Hence, each malicious device's dataset becomes similar. Thus, the cluster that contains malicious clients can increase its score with this similarity between malicious clients‘ models.

5.2.3. Layer Poisoning Attack Detection

In the following tests, the model layer poisoning is used for specific layers to poison instead of all models to hide their existence in general and harm all models via layers. The layer poisoning attack is based on the gradient factor attack applied to specific layers instead of all. Hence, this attack aims to collapse the global model

without getting noticed by the detection system since the proposed detection system controls all layers; then generates a general score representing them.

Recall that, LeNet 5 consists of four types of layers: convolution, dense, flatten, and pooling layers. Since the pooling layers have no trainable weight parameter, the layer poisoning is focused on three other layers. Figure 5.10 shows the layer poisoning attack on various layers with the accuracy and $d_{O1,O2}$.

In every test, there're 30 clients and fourteen of them are malicious while the detection mechanism is active. The γ value is fixed at eight percent, and the $d_{O1,O2}$ represents all layers. So, the cluster with lower participants will be deleted if the $d_{O1,O2}$ is higher than the γ.

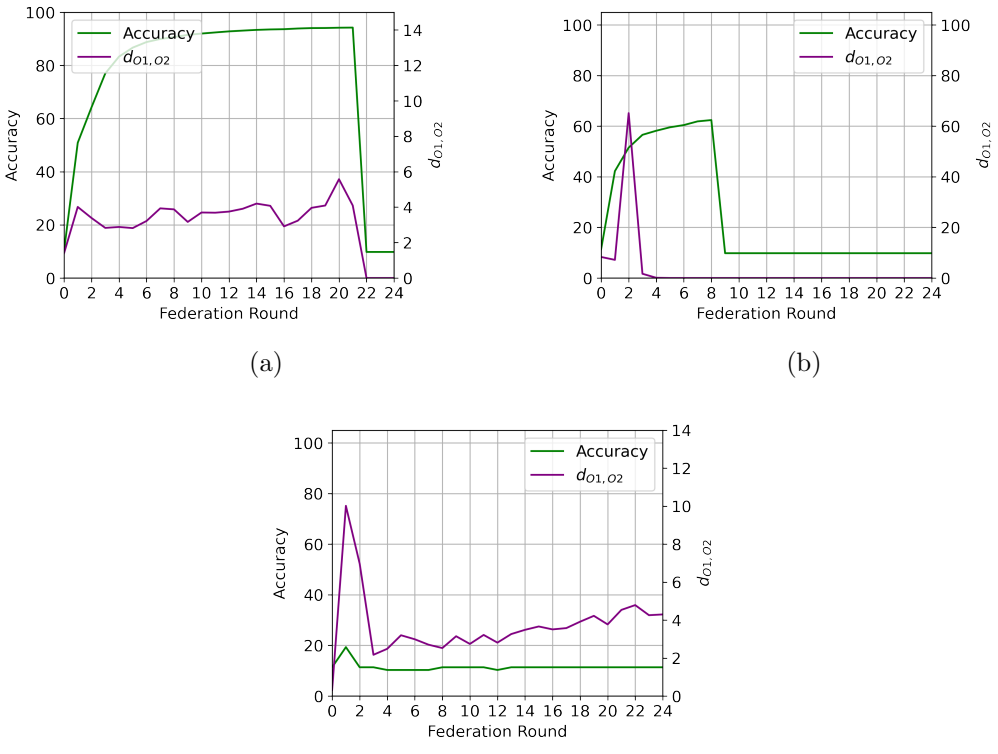


Figure 5.9: $d_{O1,O2}$ and accuracy without defense, layer poisoning attack. $|C| = 30$, $|M| = 14$, LeNet5, MNIST. (a) Convolution layers, (b) dense layers, (c) flatten layer.

Figure 5.9 shows the layer poisoning attack on a system with a disabled defense mechanism. As shown in Figure 5.9b and c, attacking flatten and dense layers directly collapses the global model. The accuracy increased while attacking CNN layers in Figure 5.9a. Nevertheless, the model still collapses after round 22. Thus, each layer is vulnerable to attacks.

In the layer poisoning of convolution layers, the first two convolution layers are selected to poison a small number of weight parameters compared to the last convolution layer. Figure 5.10a shows that attacking those layers does not impact accuracy and $d_{O1,O2}$ at earlier rounds. The accuracy and $d_{O1,O2}$ have similar results as the values from the test that has no malicious clients. Thus, the detection mechanism cannot notice and act to the layer poising attack due to low $d_{O1,O2}$. However, the model starts to collapse after some point (round 21), and clients are not able to train models after this point. Thus, we can say that the attackers successfully poison the global model, even though the defense mechanism is active.

Poisoning the flatten layer with layer poisoning results in a low $d_{O1,O2}$, as shown in Figure 5.10b. In the first round, $d_{O1,O2}$ exceeds the γ value; nevertheless, a single round is insufficient to prevent the layer-flipping attack. Other rounds also should have $d_{O1,O2}$ greater than γ. Thus, the detection mechanism cannot notice and act to the layer poising attack due to low $d_{O1,O2}$. Additionally, the accuracy value cannot increase. Thus, the layer poisoning on flatten layer is also successfully hidden from detection.

On the other hand, the layer poisoning on the dense layers leads to a high $d_{O1,O2}$ greater than γ, as plotted in Figure 5.10c. Thus, this attack can be avoided since the dense layers have over 10000 weight parameters, nearly 50% of the total weight parameters' count.

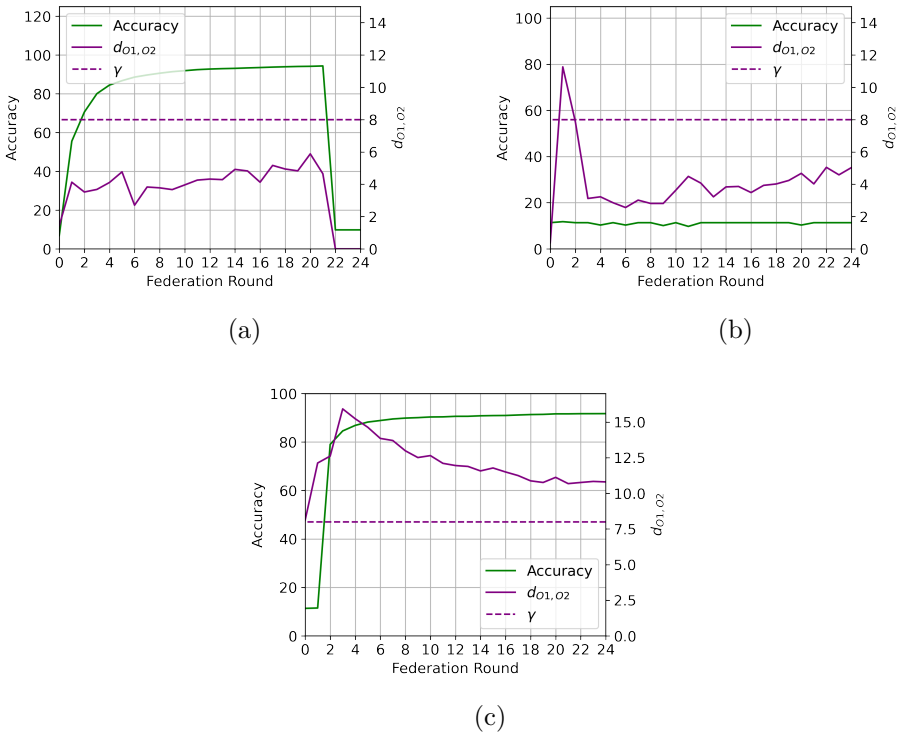


Figure 5.10: $d_{O1,O2}$ and accuracy with global defense, $\gamma = 8$. Layer poisoning attack, $|C| = 30$, $|M| = 14$, LeNet5, MNIST. (a) Convolution layers, (b) flatten layer, (c) dense layers.

Figure 5.10 shows that our defense mechanism with the global model checking cannot prevent the layer poisoning attack. The attacks targeted to specific layers, such as convolution and flatten layer can succeed in collapsing the global model. Layer poisoning affects the layers dramatically and can be seen within the layer-wise analysis, as shown in Figure 5.11. In the following tests, one benign and one malicious client is selected randomly in the server, and their values are compared. Y-axis shows the V_c value of specific layers taken from a benign and malicious client, the weight point within the threshold, and the x-axis shows the shrinkage level θ. The green line shows the values of a benign client, and the purple line shows the values of a malicious client. In convolution and dense layer, the V_c values are entirely different for the malicious and benign client. This can be easily monitored in layer-wise analysis, as shown in Figure 5.11a and c. On the other hand, the difference between V_c values of flatten layer is low compared to the convolution and dense layer, as seen in Figure 5.11b.

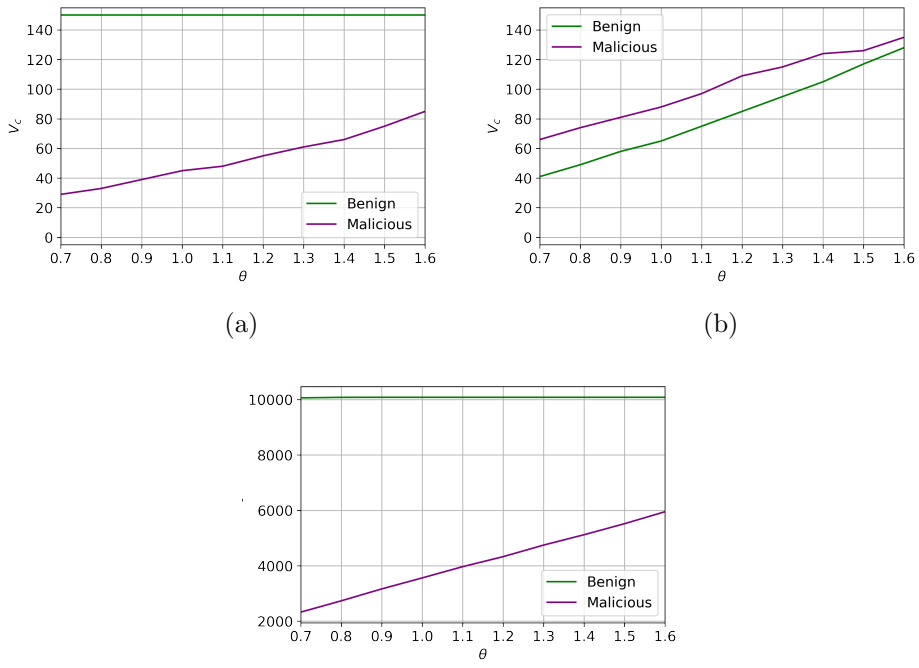


Figure 5.11: V_c over θ, layer poisoning attack. $|C| = 30$, $|M| = 14$, LeNet5, MNIST. (a) Convolution layers, (b) flatten layer, (c) dense layers.

In the layer-wise detection mode, the score is only computed for a specific layer. Then, k-means cluster each layer solely instead of all layers from the models. Moreover, $d_{O1,O2}$ values represent the layers' score. For instance, each flatten layer of a client model is clustered with other clients' flatten layers. This mechanism blocks malicious clients from sneaking into specific layers. In Figure 5.12, $d_{O1,O2}$ is not normalized within 0 and 100 and γ scaled for maximum $d_{O1,O2}$ to make $\gamma = 8$. As shown in Figure 5.12, the layer poisoning attack can be avoidable for convolution, dense, and flatten layers.

5.3. Adaptive Gamma

There are two ways to define γ. The first one is assigning a constant value to the γ. The client count and attack types should be analyzed to do that. In Figure 5.13, 5.14 and 5.15, the $\gamma = 7$ is used to measure the performance of the federated learning. This γ value is found during previous tests.

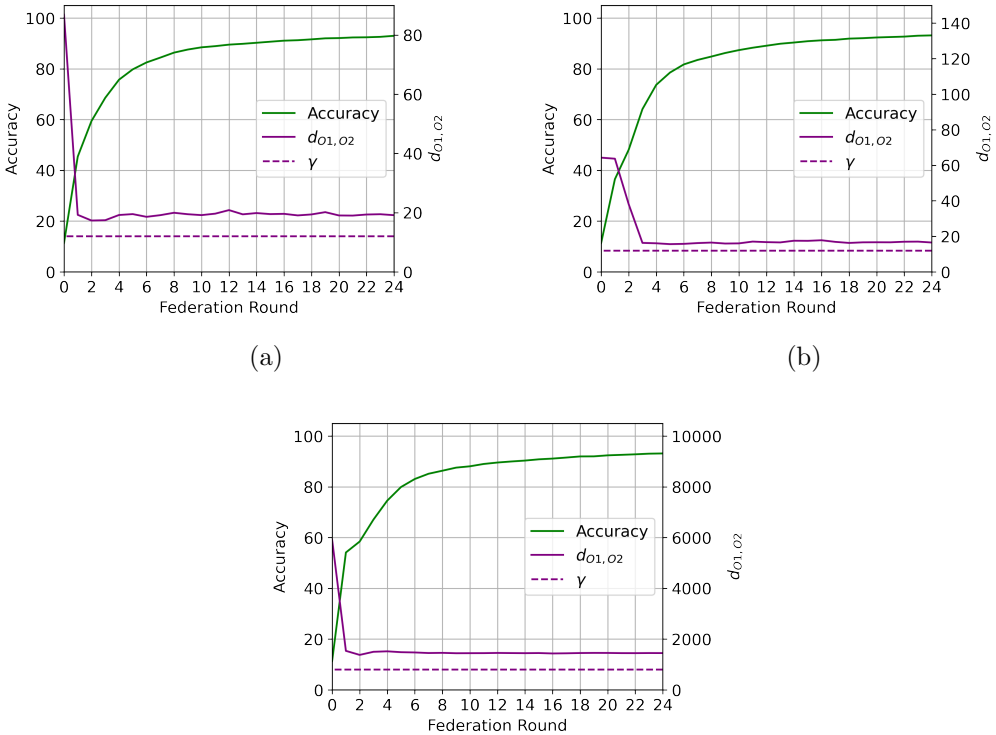
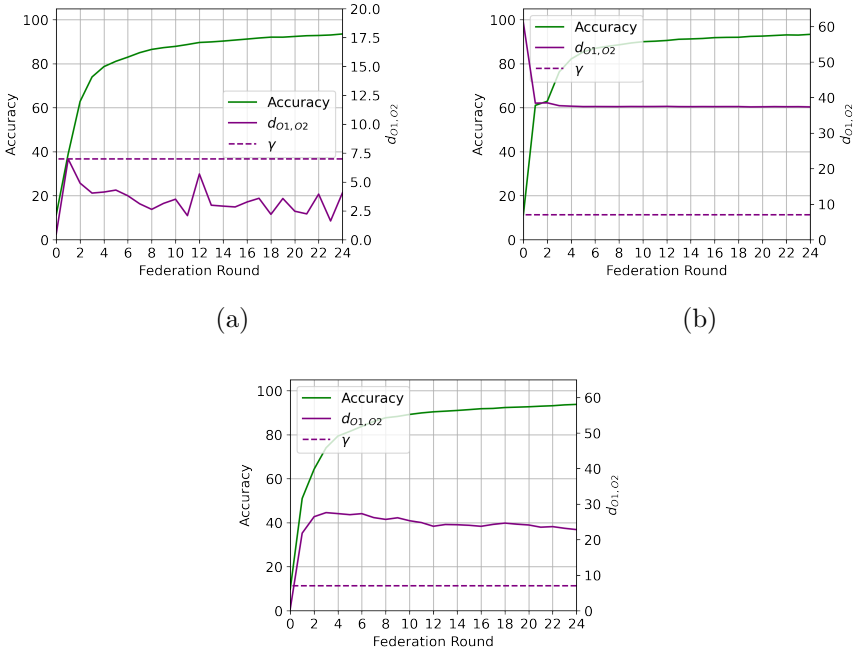


Figure 5.12: $d_{O1,O2}$ and accuracy with layer defense, layer poisoning attack. $|C| = 30$, $|M| = 14$, LeNet5, MNIST. (a) Convolution layers, (b) flatten layer, (c) dense layers.

While there is no attack, the gamma value is sufficient for benign devices not to be marked as malicious, as can be seen in Figure 5.13a, 5.14a and 5.15a. However, in Figure 5.15a, the $d_{O1,O2}$ increases to 8.56 in the second round, and some of the clients marked as malicious. However, the accuracy value is not affected by it. Additionally, the gamma value is also sufficient for the separation of the gradient factor attack. In earlier rounds, $d_{O1,O2}$ has a big value, then decreases.

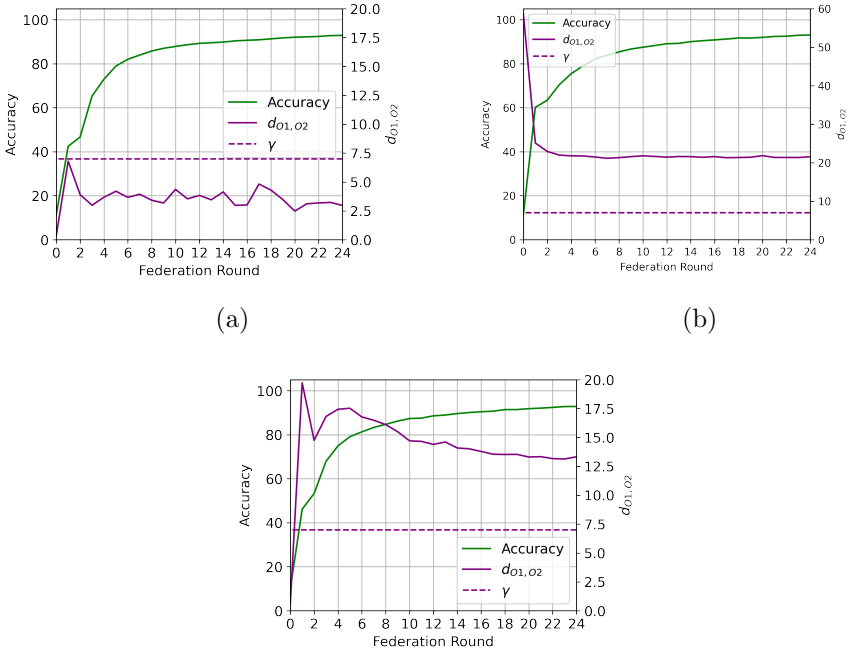
Furthermore, in the layer flipping attack, the first round of each test resulted in $d_{O1,O2}$ value of under one as can be seen in Figure 5.13c, 5.14c and 5.15c. Thus, the attack become successful and F1 score of these rounds become 0.75 for $|C| = 10$, 0.71 for $|C| = 20$ and 0.7 for $|C| = 30$.



(a) (b)

(c)

Figure 5.13: Fixed γ, $d_{O1,O2}$ and accuracy. $|C| = 10$, $|M| = 4$, $\gamma = 7$, LeNet5, MNIST. (a) No attack, (b) gradient factor attack, (c) label flipping attack.



(a) (b)

(c)

Figure 5.14: Fixed γ, $d_{O1,O2}$ and accuracy. $|C| = 20$, $|M| = 9$, $\gamma = 7$, LeNet5, MNIST. (a) No attack, (b) gradient factor attack, (c) label flipping attack.

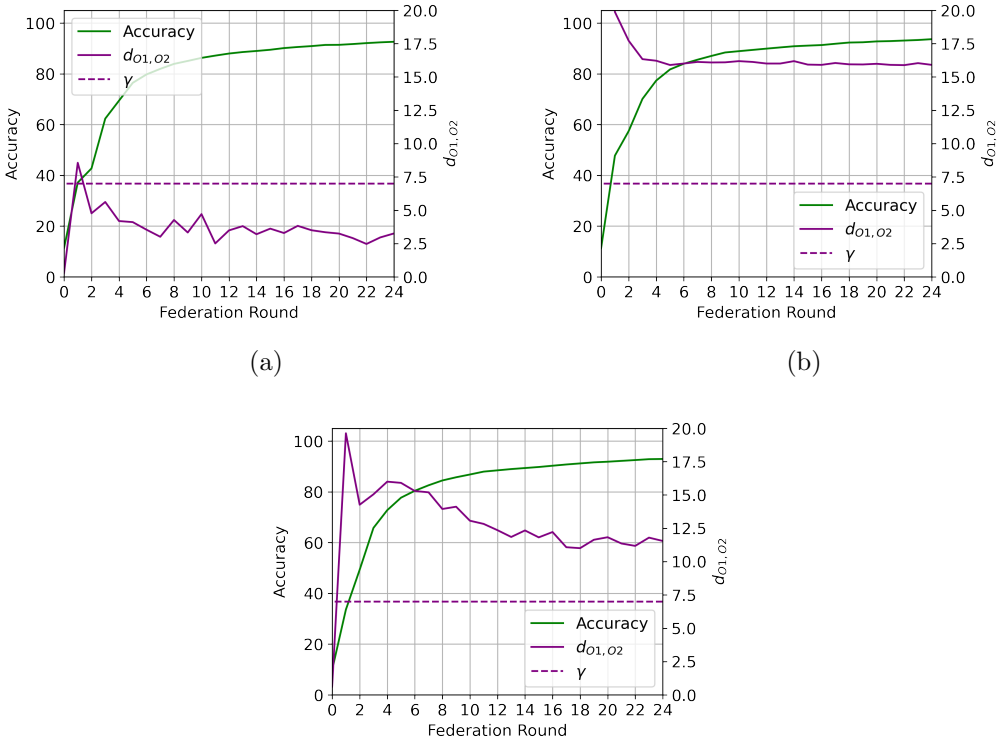


Figure 5.15: Fixed γ, $d_{O1,O2}$ and accuracy. $|C| = 30$, $|M| = 14$, $\gamma = 7$, LeNet5, MNIST. (a) No attack, (b) gradient factor attack, (c) label flipping attack.

As stated in Chapter 3, there are two initial values to compute adaptive gamma since the algorithm is bounded by the previous iterations of gamma calculation. Thus, the initial values are defined as $\gamma = 3$, and the $lastAcc = 0$ since there are no previous accuracy values. The tests are made with 30 clients, and results are shown in Figure 5.16. The dashed line represents adaptive γ.

Without an attack to the network, γ becomes stable and does not diverge from the initial value. After the fourth round, γ has never changed since the accuracy is not improved dramatically. However, in the first round, the gamma is insufficient to separate the benign clients, and the F1 Score becomes 0.72. Other than that, there is no problem with no attack test.

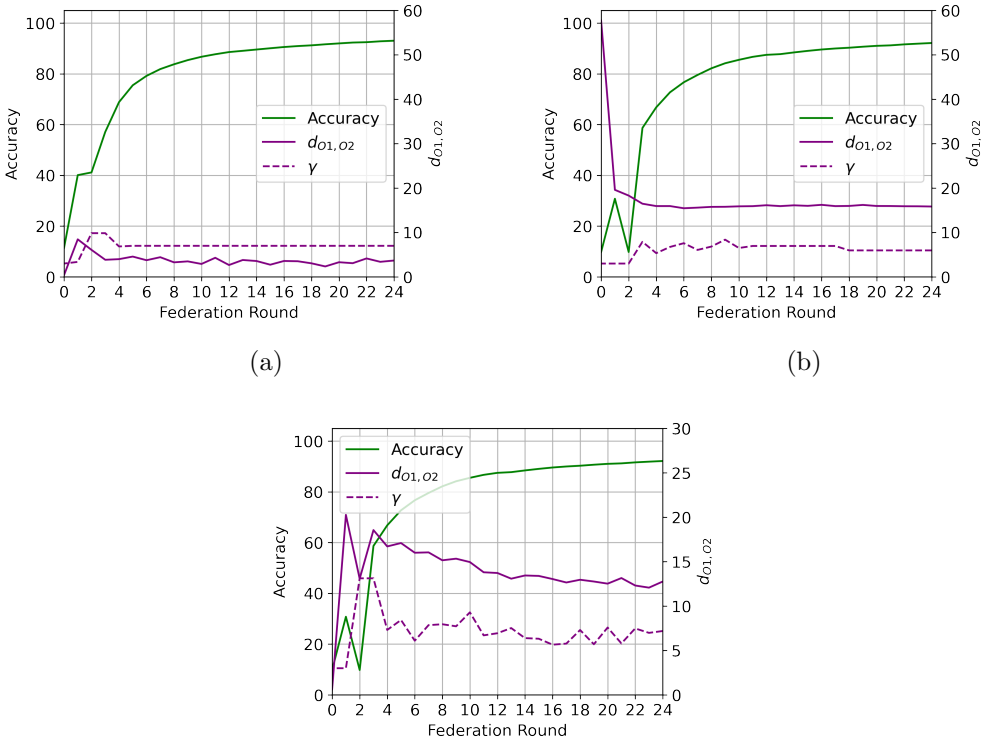


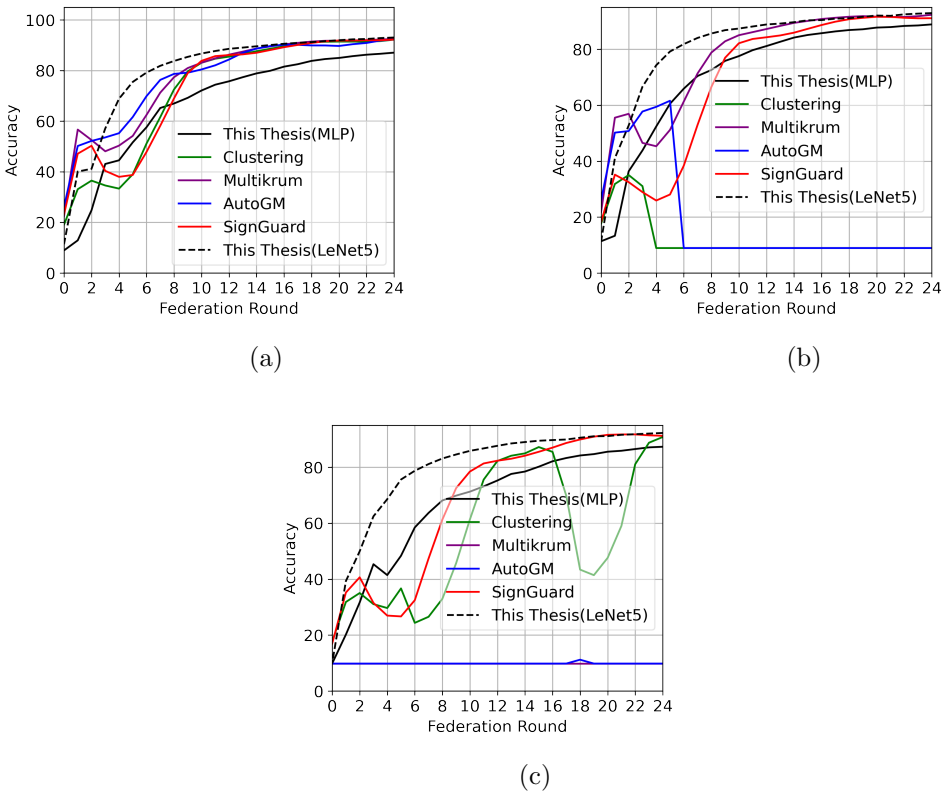
Figure 5.16: Adaptive γ, $d_{O1,O2}$ and accuracy. $|C| = 30$, $|M| = 14$, LeNet5, MNIST. (a) No attack, (b) gradient factor attack, (c) label flipping attack.

In the gradient factor attack, the $d_{O1,O2}$ starts from high values as in previous tests. On the other hand, the adaptive γ algorithm successfully separates benign and malicious. Besides, F1 values for all rounds are one.

In the label flipping attack, $d_{O1,O2}$ starts from low values again and then increases as expected. Thus, the detection system does not notice the malicious cluster with the adaptive gamma value in round one and three, and F1 values for these rounds become 0.7. As seen from Figure 5.16c, the detection mechanism clusters the benign and malicious clients without a mistake in following rounds, and the F1 value becomes 1. γ starts from high values since the global model changes dramatically. After the fourth round, γ is stabilized between five and seven.

5.4. Comparison with Existing Work

In this section, this thesis' approach is compared with the approaches from the literature. To do that, [29] is selected as the federated learning framework, which contains several detection mechanism such as Multikrum [30], Clustering [31], AutoGM [32], and SignGuard [33]. Approaches in [29] use the Multilayer Perceptron with three dense and one dropout layers as their DNN in MNIST compared to Multilayer Perceptron with three dense and one flatten layers in this thesis. The tests are made with 0.1 learning rate and 0.9 momentum. In each round, clients compute 16 batches of data with a batch size of 64 similar to the previous tests with four local batches and 25 rounds. This thesis uses adaptive gamma as its γ selection. Finally, the experiments use the gradient factor and label flipping attacks as attack types. Attacks are directly started in the first round. The results are shown in Figure 5.17.



(a) (b)

(c)

Figure 5.17: Robustness comparison of defense mechanisms. $|C| = 30$, $|M| = 14$, MNIST. (a) No attack, (b) gradient factor attack, (c) label flipping attack.

If there are no malicious devices in the network, the accuracy of each model increases as expected. After 24 rounds, the accuracy values become higher than 90 as can be seen in Figure 5.17a. However, this thesis approach with the Tensorflow is able to reach 87 in 25 rounds. Higher accuracy (higher than 90) is attended at 35 rounds. In LeNet5, the accuracy is able to increase up to 94 in 25 rounds. To compare its accuracy with the MLP, it is plotted as dashed line (LeNet5) in Figure 5.17. In the gradient factor attack, Clustering and AutoGM are not able to cope with the attack; thus, the accuracy converges to 9.8% as seen in Figure 5.17b. On the other hand, Multikrum and SignGuard can notice the attack and prevent them. However, their defence mechanism has lower robustness in lower accuracy values. Thus, their accuracy slowly converges to the top compared to the no-attack case.

In the label flipping attack, only Clustering, SignGuard and this thesis' method are able to notice the attack and prevent them. Nevertheless, Clustering and SignGuard have lower robustness in early rounds. Our work is able to detect the attack in earlier rounds, where it react attack with better robustness in the earlier rounds. As can be seen, AutoGM is failed in both attack types; however, its defence mechanism is robust if there are fewer malicious client in the network.

6. CONCLUSION

Multidimensional data points used in clustering and the pair-based comparison of their centroids developed in this work have proved efficient in various experiments. The framework that has been developed for experimental purposes can be used in real life with minor modifications. The advantage of the proposed method is that it can separate every layer or weight parameter and score them individually since it uses every weight parameter to utilize the detection system. Thus, a weight parameter from the convolution or dense layers can be analyzed individually rather than globally. This feature can make hidden attacks more observable.

We can assume that 13-15 of threshold γ is valid when ten clients are in the network with the simple CNN model. In the LeNet5 model, more clients participate in the learning, and γ could be lower such as 8. However, if we increase the local batch size, this threshold may not meet the expectations. Then the label-flipping attack may poison the global model. Adaptive gamma calculation should be used to make γ compatible for every attack types. If the malicious client appears in the system, it polarizes the malicious and benign clusters even with a low poisoning rate. Thus, lowering γ leads to a more robust defense mechanism. The layer poisoning attack enables malicious clients to deceive the defense mechanism as it checks the models instead of layers. The issue is that every layer reacts differently, so a constant γ value may not be optimal for every case and training round. Then, the layer control mechanism should be empowered in every round.

The follow-up studies may need to focus on various machine learning quantization techniques, such as pruning or quantization-aware training, with the real-world methods used in the industry. Other post-training quantization techniques may become more applicable in the future; thus, their impact could be investigated. Hence, adaptive γ calculation mechanism should also be optimized for the layer poisoning.

REFERENCES

1. McMahan, B., E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data", *Artificial Intelligence and Statistics*, edited by Singh, A. and J. Zhu, Fort Lauderdale, USA, Vol. 54, pp. 1273–1282, 2017.

2. Rey, V., P. M. S. Sánchez, A. H. Celdrán and G. Bovet, "Federated Learning for Malware Detection in IoT Devices", *Computer Networks*, Vol. 204, p. 108693, 2022.

3. Meidan, Y., M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher and Y. Elovici, "N-Baiot—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders", *IEEE Pervasive Computing*, Vol. 17, No. 3, pp. 12–22, 2018.

4. Nguyen, T. D., P. Rieger, R. De Viti, H. Chen, B. B. Brandenburg, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, S. Zeitouni, F. Koushanfar, A. R. Sadeghi and T. Schneider, "FLAME: Taming Backdoors in Federated Learning", *31st Security Symposium*, Boston, USA, pp. 1415–1432, 2022.

5. Wang, Y., D. Zhai, Y. Zhan and Y. Xia, "RFLBAT: A Robust Federated Learning Algorithm against Backdoor Attack", arXiv:2201.03772, 2022.

6. Muñoz-González, L., K. T. Co and E. C. Lupu, "Byzantine-Robust Federated Machine Learning Through Adaptive Model Averaging", arXiv:1909.05125, 2019.

7. Zhang, T., C. He, T. Ma, L. Gao, M. Ma and S. Avestimehr, "Federated Learning for Internet of Things", *19th Conference on Embedded Networked Sensor Systems*, Coimbra, Portugal, pp. 413–419, 2021.

8. Sánchez, P. M. S., A. H. Celdrán, T. Schenk, A. L. B. Iten, G. Bovet, G. M. Pérez and B. Stiller, "Studying the Robustness of Anti-adversarial Federated Learning

Models Detecting Cyberattacks in IoT Spectrum Sensors", arXiv:2202.00137, 2022.

9. Nguyen, T. D., S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan and A. R. Sadeghi, "DÏoT: A Federated Self-Learning Anomaly Detection System for IoT", *IEEE 39th International Conference on Distributed Computing Systems*, Dallas, USA, pp. 756–767, 2019.

10. Li, S., Y. Cheng, Y. Liu, W. Wang and T. Chen, "Abnormal Client Behavior Detection in Federated Learning", arXiv:1910.09933, 2019.

11. Zhao, Y., J. Chen, J. Zhang, D. Wu, J. Teng and S. Yu, "PDGAN: A Novel Poisoning Defense Method in Federated Learning Using Generative Adversarial Network", *Algorithms and Architectures for Parallel Processing: 19th International Conference*, edited by Wen, S., A. Zomaya and L. T. Yang, Melbourne, Australia, Vol. 11944, pp. 595–609, 2020.

12. Cao, X., J. Jia and N. Z. Gong, "Provably Secure Federated Learning Against Malicious Clients", *Conference on Artificial Intelligence*, Virtual Conference, Vol. 35, pp. 6885–6893, 2021.

13. Liu, W., H. Lin, X. Wang, J. Hu, G. Kaddoum, M. J. Piran and A. Alamri, "D2MIF: A Malicious Model Detection Mechanism for Federated Learning Empowered Artificial Intelligence of Things", *IEEE Internet of Things Journal*, Vol. 10, pp. 2141–2151, 2021.

14. Li, D., W. E. Wong, W. Wang, Y. Yao and M. Chau, "Detection and Mitigation of Label-Flipping Attacks in Federated Learning Systems with KPCA and K-means", *8th International Conference on Dependable Systems and Their Applications*, Yinchuan, China, pp. 551–559, 2021.

15. Nguyen, T. D., P. Rieger, M. Miettinen and A. R. Sadeghi, "Poisoning Attacks on Federated Learning-Based IoT Intrusion Detection System", *Workshop on Decen-*

tralized IoT Systems and Security, San Diego, USA, pp. 1–7, 2020.

16. Shen, S., S. Tople and P. Saxena, "Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems", *32nd Annual Conference on Computer Security Applications*, Los Angeles, USA, pp. 508–519, 2016.

17. Zhang, Z., X. Cao, J. Jia and N. Z. Gong, "FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients", *28th Conference on Knowledge Discovery and Data Mining*, Washington, USA, pp. 2545—2555, 2022.

18. Li, S., Y. Cheng, W. Wang, Y. Liu and T. Chen, "Learning to Detect Malicious Clients for Robust Federated Learning", arXiv:2002.00211, 2020.

19. Pasquini, D., D. Francati and G. Ateniese, "Eluding Secure Aggregation in Federated Learning via Model Inconsistency", *Conference on Computer and Communications Security*, Los Angeles, USA, pp. 2429–2443, 2022.

20. Bagdasaryan, E., A. Veit, Y. Hua, D. Estrin and V. Shmatikov, "How to Backdoor Federated Learning", *International Conference on Artificial Intelligence and Statistics*, edited by Chiappa, S. and R. Calandra, Virtual Conference, Vol. 108, pp. 2938–2948, 2020.

21. Reisizadeh, A., A. Mokhtari, H. Hassani, A. Jadbabaie and R. Pedarsani, "Fedpaq: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization", *International Conference on Artificial Intelligence and Statistics*, edited by Chiappa, S. and R. Calandra, Virtual Conference, Vol. 108, pp. 2021–2031, 2020.

22. Haddadpour, F., M. M. Kamani, A. Mokhtari and M. Mahdavi, "Federated Learning with Compression: Unified Analysis and Sharp Guarantees", *International Conference on Artificial Intelligence and Statistics*, edited by Banerjee, A. and K.

Fukumizu, Virtual Conference, Vol. 130, pp. 2350–2358, 2021.

23. Liu, Y., N. Kumar, Z. Xiong, W. Y. B. Lim, J. Kang and D. Niyato, "Communication-efficient Federated Learning for Anomaly Detection in Industrial Internet of Things", *IEEE Global Communications Conference*, Taipei, Taiwan, pp. 1–6, 2020.

24. Tensorflow, "Tensorflow Lite", `https://www.tensorflow.org/lite`, accessed on June 20, 2023.

25. Ma, C., J. Li, L. Shi, M. Ding, T. Wang, Z. Han and H. V. Poor, "When Federated Learning Meets Blockchain: A New Distributed Learning Paradigm", *IEEE Computational Intelligence Magazine*, Vol. 17, No. 3, pp. 26–33, 2022.

26. Deng, L., "The Mnist Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]", *IEEE Signal Processing Magazine*, Vol. 29, No. 6, pp. 141–142, 2012.

27. LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

28. Xiao, H., K. Rasul and R. Vollgraf, "Fashion-Mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms", arXiv:1708.07747, 2017.

29. Li, S., L. Ju, T. Zhang, E. Ngai and T. Voigt, "Blades: A Simulator for Attacks and Defenses in Federated Learning", arXiv:2206.05359, 2022.

30. Blanchard, P., E. M. El Mhamdi, R. Guerraoui and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent", *Advances in Neural Information Processing Systems*, edited by Bengio, S., H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, Long Beach, USA, Vol. 30, pp. 118-128, 2017.

31. Sattler, F., K.-R. Müller, T. Wiegand and W. Samek, "On the Byzantine Robustness of Clustered Federated Learning", *IEEE International Conference on Acoustics, Speech and Signal Processing*, Long Beach, USA, pp. 8861–8865, 2020.

32. Li, S., E. Ngai and T. Voigt, "Byzantine-Robust Aggregation in Federated Learning Empowered Industrial IoT", *IEEE Transactions on Industrial Informatics*, Vol. 19, No. 2, pp. 1165–1175, 2021.

33. Xu, J., S.-L. Huang, L. Song and T. Lan, "Signguard: Byzantine-Robust Federated Learning through Collaborative Malicious Gradient Filtering", arXiv:2109.05872, 2021.

APPENDIX A: USER GUIDE OF THE FRAMEWORK

The framework is designed as a distributed system, and each device is started as separate bash scripts. Also, each device has three main threads in its main flows. These are “listen”, “input”, and “handle_message”. The listen is used for listening to a specific port for upcoming messages assigned to the device. If there is a message for a device, this thread pushes it to the msgQueue array to be handled. The handle_message is used for handling these messages in the queue. The input is used for the user interface via the terminal so that a user can control the device.

The attack control and federated learning servers should be started firstly since the clients will be connected to these servers. Then, the clients should be started with the arguments of “Client Port, Server Port, Client ID, Compute Frequency”. That means a client starts with a specific port and client ID with a compute frequency variable. After starting a client, it directly transmits “addNetwork” message to both the federated learning server and attack control server to be enrolled in “clientPorts” array in the servers. Thus, the servers identify the clients with their configurations. Compute Frequency variable is used in the federated learning server, which recognizes to mock client’s hardware configuration. For instance, if this variable is Y for a slow learner, and the server orders from the default clients with X local training, the slow learner computes $\lceil X/Y \rceil$ times locally. The problematic client is imitated with a TensorFlow GPU package that cannot train the deep learning model in a CPU normally, then throws “Optimization loop failed” and trains differently from the rest of the clients. So, the problematic clients try to use GPU for the training without having a GPU. The sample story handling can be seen from Figure A.7. As can be seen, client one is poisoning its model with the gradient factor attack, while client two trains its data regularly. Malicious clients can also be detected with the name of the terminal, such as client one (malicious) in Figure A.7. Suppose the user or attack control server activates the malicious mode. In that case, the name of the terminal is automatically changed so that the user can realize the client mode quickly via the name of the

terminal.

After each client and server is started, the user can adjust the malicious rate from the terminal of the attack control server and adjust the attack from there. Then, the user can enter a story from the terminal of the federated learning server. The sample story can be seen in the Framework chapter or end of Appendix A.

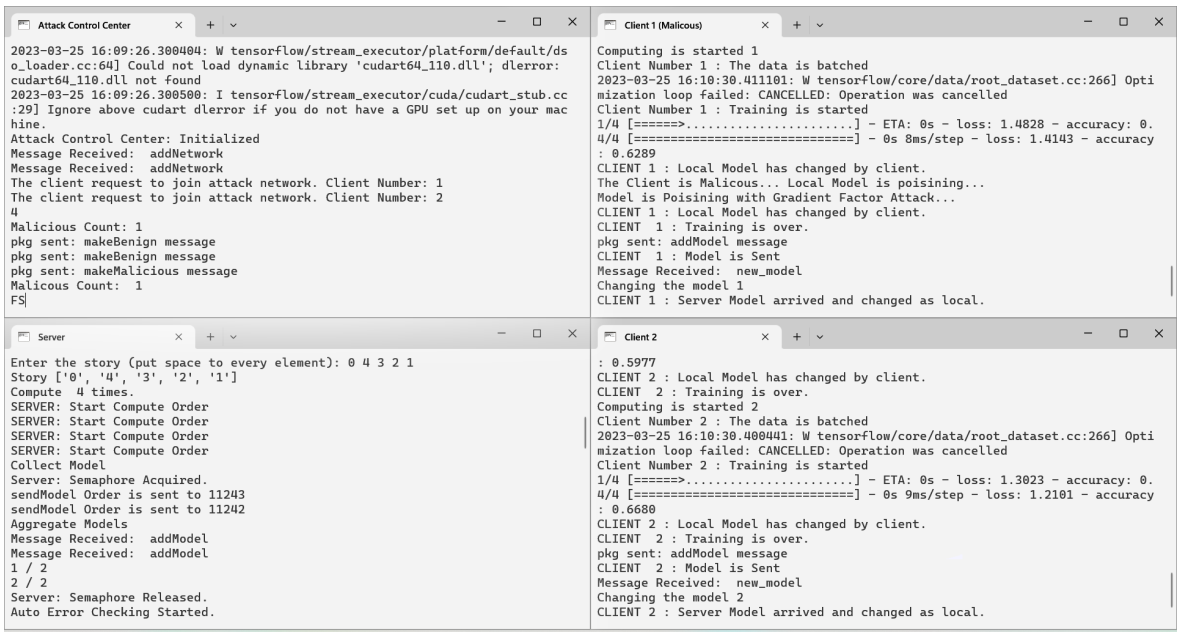


Figure A.1: Bash scripts of two clients, an attack control server and a federated learning server.

As stated before, the attack control server can change the malicious-benign client ratio for the experiments. If the attack control server selects a client as malicious, the client changes mode from “Benign” to “Malicious”, which triggers poisoning schemes after local training; thus, the malicious clients work synchronized and trained without additional adjustments. Being a client with quantization is also controlled via the terminal; however, being a problematic client needed to be adjusted before starting the bash script of the client. Regular and problematic clients use different virtual environments. At the beggining of bash scripts, specific virtual environment is assigned for each script, so that it can simulate the regular and problematic clients. As stated before, the problematic clients are using Tensorflow GPU framework, while regular

clients are using Tensorflow CPU framework.

A user should follow these steps to experiment with this thesis without errors. First, the user should generate batch scripts that generate client objects. For instance, with "python Client.py 11242 3864 1" in a batch script, the user creates a Client object with a port of 11242 and a client number of 1. 3864 is the server's port for communication. The user should also use a package manager to simulate the problematic client such as Anaconda. If the environment of the problematic client is used, the client will be problematic. Lastly, the client can be a slow learner with a "python Client.py 11244 3864 5 2". This script creates a client with a client number of 5 and a port of 11244. Additionally, it configures the Compute Frequency as two. Thus, this client does two local batches while regular clients do four local batches, for instance.

Secondly, the user generates objects of Server and Attack Control Server by triggering their batch scripts. After that, the user should create the client objects with their batch scripts. When all the clients and server start successfully, they are automatically connected with their port numbers. Clients send a packet of "addNetwork" to the server with a parameter of Compute Frequency and starting round. Thus, the server knows the configuration of each client respectively. The user can use terminals to interact with devices. As shown in Figure A.7, each device has its terminal.

The user should start injecting attacks via the terminal with the attack control server. First, he/she should select the number of malicious devices, then the attack type and parameters. After that, the malicious clients will be ready to attack the global system. Then, the user can interact with any client via its terminal to adjust the client configuration, such as changing the attack type or opening quantization. However, the user cannot specify a round to change the device configuration. It should be done between rounds via the client terminal.

Finally, the user should enter a story to the server terminal such as “0 4 3 2 1”, which means train four times locally (0 4), then collect the models from clients (3), then automatically check maliciousness and aggregate the remainder models (2), then send the aggregated model to every client (1). With these steps, the user can experiment the thesis simulator with full flexibility.