Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

# Mitigating Security Concerns for Federated Learning

*Author:*
Anne-Sophie Hannes

*Supervisor:*
Professor Kin Leung

Submitted in partial fulfillment of the requirements for the MSc degree in Computing of Imperial College London

August 2022

**Abstract**

Machine learning applications are crucial in many industries, e.g., healthcare, automobile insurance, communication services and national defense. Some prevalent issues, like data privacy, latency time and resource constrains, have led to the rise of Federated Learning (FL), which is a form of distributed machine learning where a global model is often trained on local edge nodes that possess the data sets. The main advantage of FL is to avoid sharing of private data among nodes and a prohibitive amount of communication resources to transfer data from different sources.

With the increasing importance of machine learning and more specifically FL, it is crucial that the machine learning model is protected from potential malicious actors (nodes), who intend to disrupt and harm the learning process, e.g., by reducing the model accuracy. In a FL environment, this problem is predominant because the nondisclosure of raw data with the system does make the identification of malicious actors challenging, and the distributed nature of the learning process can make the system more vulnerable to attacks.

The first goal of this project is to understand the behaviour of a FL environment when one or more malicious nodes intrude the system. Thereby, a FL testbed is developed and used for the analysis of multiple malicious scenarios. The testbed is implemented in a way that the number of malicious nodes, the amount of malicious data per malicious node as well as the time frame in which the node is malicious can be varied to facilitate investigation of their impacts on FL.

This has enabled a deeper understanding of how malicious nodes affect a Federated Learning system and has led to the development of a statistic parameter, namely the relative average Mean Squared Error (raMSE), for identifying those malicious nodes. The effectiveness of the raMSE for identifying malicious nodes is validated for multiple models and data sets.

Finally, the use of the raMSE has enabled the development of an automated malicious detection system. The newly proposed malicious detection system is able to classify malicious nodes and filter them from the global FL in each update round with a success rate of $\geq 93\%$. Furthermore, the global model's loss in a malicious environment with activated automated malicious detection system achieves similar values than the global model's loss of a healthy FL system. Thus, the automated malicious detection system is able to protect FL of the global model from potential malicious intentions.

# Acknowledgments

I would like to thank my supervisor Professor Kin Leung for his constant support, guidance and encouragement. In particular, I am grateful for all the time he has taken, the great effort he has put into every meeting, the encouraging words and the fantastic guidance in the right direction on this project.

Moreover, I would like to thank my second marker Professor William Knottenbelt for his great support and guidance.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation of this work

Federated Learning (FL), as a subsection of machine learning, is becoming increasingly important, but also brings significant security concerns along with it. In the following section, motivation and key drivers of this project are discussed.

### 1.1.1 Machine learning

In our modern world, machine learning applications have become increasingly essential. The global market is estimated at 15.44 billion USD in 2021 and expected to grow by a compound annual growth rate of 38.8% during the forecast period 2022 to 2029. Machine learning is important in most industries, as shown in figure 1.1 [1].

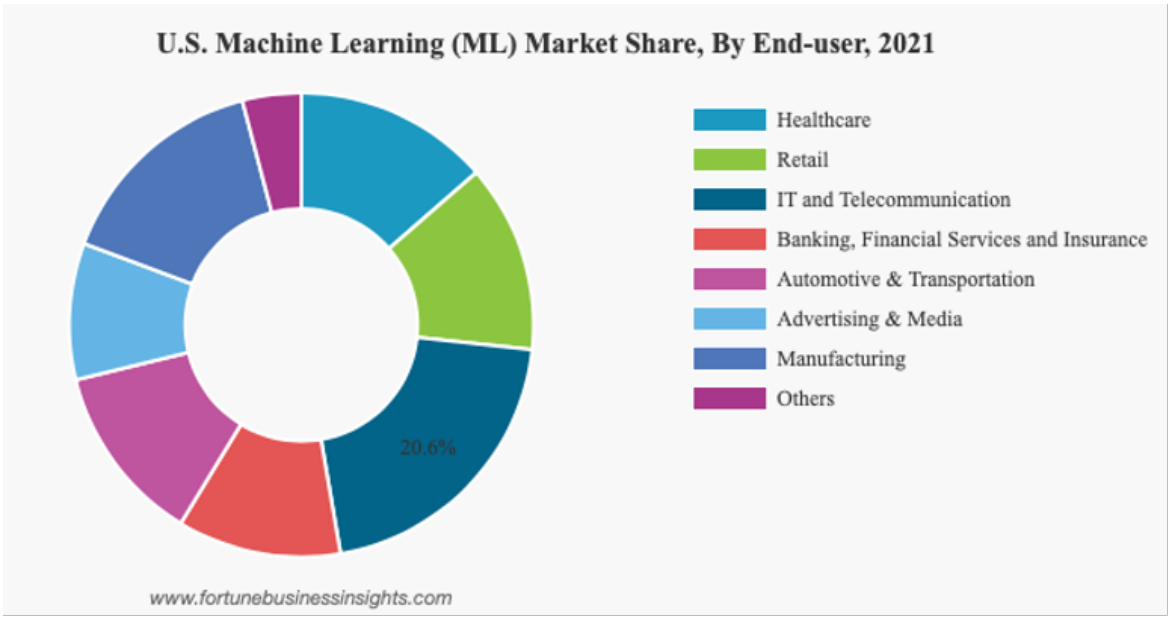



**Figure 1.1:** U.S. machine learning market share by end user in 2021 [1].

However, each of these industries face challenges within machine learning applications. Apart from improving the model accuracy, three very pivotal issues are predominant:

1. Within a traditional central machine learning model, devices which record data need to send their raw data to the central server. The raw data contains a lot of private information [2, 3]. Therefore, those models are not always able to preserve the privacy of their clients.

2. As the amount of data increases, it is no longer scalable to have the main server process all data. In addition, possible bandwidth limitations play an an influential role in the rising importance of FL [4, 5].

3. For many machine learning applications (for example, autonomous driving) low latency time for the prediction is crucial and cannot be achieved in a central server environment [6].

### 1.1.2 Federated Learning

Federated Learning (FL) is a proposed solution for the aforementioned problems, which enables clients to share a prediction model without sharing their raw data [7]. Each client trains the model on its own system with its local data set. The clients send their locally trained machine learning model parameters at certain intervals to the aggregator, which aggregates all parameters according to a certain aggregation rule, calculates new global model parameters and returns them to all end nodes [7, 8]. This has the advantage that all individual nodes can use a machine learning model and learn from each other (which promotes the performance of the model), but the privacy of the end node is better preserved because only the model parameter and not the data sets are shared with the aggregator [7]. Additionally, it means that nodes do not need a continuous connection to the internet, and bandwidth constraints as well as latency issues do not hinder predictions[7, 8].

FL can be useful in many industry applications, e.g., within autonomous driving, national defence systems or smart and fitness watches [9, 10]. However even though FL is a big research topic, it has not yet been adopted industry-wide due to its potential security issues [11].

### 1.1.3 Security

Security plays a crucial role in all digital activities. With the rise of “big data”, the security aspects of machine learning get more important [12]. Within a machine learning context, there exist a variety of security concerns and attack angles [13, 14].

A major risk with data-driven machine learning technologies is that a malicious actor may attempt to create training data that reduces the performance of the model [13, 14]. This attack vector is called a poisoning attack and can be very efficient even if only a small amount of data is compromised [15].

The risk of a successful poisoning attack is even greater with FL. Firstly, FL often relies on all nodes to contribute to the model optimization and it has a large attack surface for attackers to find vulnerabilities [4].

Secondly, FL introduces new challenges to detect and prevent those attacks. In comparison to a central machine learning model, a FL model does not have access to raw data. Therefore, the model cannot check the validity of the data and must trust the input parameters from the end nodes [16].

This issue becomes even more meaningful, as it is not only possible that an end node itself is malicious, but that a malicious actor tampers with an insecure connection and changes the model parameters [4]. This means that it is not sufficient only to make sure that the end nodes can be trusted, for example by performing a validation test before using them, but is also necessary to check their validity at each iteration.

## 1.2 Project objectives and aims

The aim of this project is to understand how malicious nodes in different scenarios influence the model performance and derive therefrom how a potential automated detection tool might be able to differentiate a malicious node from a non-malicious node. This automated detection tool can be classified as an anomaly detection tool [4].

This aim can be categorised in three objectives:

The first project objective is to understand the behaviour of the overall system when one or more malicious actors infiltrate the system. To achieve this, the project aims to develop a testbed that imitates malicious behaviour and is able to analyse different scenarios of maliciousness. An analysis along three main parameters is conducted:

1. Percentage of malicious nodes
2. Percentage of malicious data for each node
3. Time frame when the node is malicious

The second objective is to find a parameter to identify the malicious nodes in each of those scenarios. Therefore a relative statistical malicious detection parameter is designed that can be applied to identify the malicious node in various scenarios.

The third objective is to develop a system which can be placed between the edge

nodes and the aggregator that automatically filters out malicious nodes, thus protecting the global aggregation round and improving the accuracy of the overall model.

## 1.3 Project contributions

In line with these objectives, this project makes the following contributions under three assumptions. Firstly, Independent Identically Distributed (IID) data is used. Secondly, it is assumed that there are fewer malicious nodes than healthy nodes in the FL environment. Thirdly, the project defines malicious data as a change in the labels of the data, which can be classified as data poisoning, a subclass of model poisoning [4].

With these assumptions, the following contributions are achieved:

1. This project is able to provide a system which imitates a malicious FL Environment. The system is flexibly able to adjust the different parameters, such as the number of nodes, update rounds, models and data sets.

2. The project provides a testbed which observes how different scenarios of malicious behaviour influence the FL system and enables the export of graphs and statistical information. This testbed can be reused and applied to other scenarios.

3. This testbed enables the understanding of how malicious nodes in different cases of maliciousness affect the FL system. The analysis is conducted along the three parameters described above: different percentages of malicious nodes, malicious data and different time frames of when the node is turning malicious and potentially healthy again.

4. Based on the insights achieved through the first objective, a parameter, the relative average Mean Squared Error (raMSE), is identified, which is able to differentiate malicious nodes from healthy nodes in multiple scenarios described above.

5. The raMSE is applied in different ways and in several scenarios. This allows the presentation of the results on how the parameter performs in different (malicious) FL environments.

6. Lastly, the knowledge acquired from the analysis is used to create an automated malicious node detection system. The system utilises the raMSE and enables to detect a malicious node. If a malicious node is detected, the node will be excluded from the current update round. This automated malicious node detection system can be included in any FL environment to filter out potential malicious nodes.

To strengthen the integrity of the results, the model, data set, update rounds and number of nodes in the Federate Learning environment are varied to mimic different environments.

## 1.4 Report outline

Chapter 2 gives an overview of the existing literature in the area of security in FL. It classifies different security concerns and gives an overview of existing countermeasures. Chapter 3 presents the experimental setup for the malicious analysis. Chapter 4 discusses the malicious parameter which should be analysed and the malicious detection parameters proposed as a solution for how to identify a malicious node. Chapter 5 presents the results of the malicious analysis before the results are evaluated and classified into current literature in chapter 6. The report concludes with a summary and suggestions for future work in chapter 7.

# Chapter 2

# Security in Federated Learning

## 2.1 Security concerns

Subsection 2.1.1 gives an overview of various security issues in FL. Subsequently, section 2.1.2 gives a more detailed introduction to poisoning attacks, as the focus of this project is on poisoning attacks, more specifically on data poisoning, a subsection of poisoning attacks [4].

### 2.1.1 Overview attack vectors

Different FL security attacks can be classified. Figure 2.1 shows a variety of attack vectors.

The most likely and most dangerous attack vectors are poisoning attacks, inference attacks, backdoor attacks, malicious server attacks and generative adversarial network-based attacks (GANs), which are a combination of inference attacks and poisoning attacks [11, 17].

Inference attacks aim to circumvent the privacy measurements of Federate Learning systems in order to infer raw data from the given model parameters. Malicious server attacks use the aggregator directly to manipulate the global model [11]. As the attack surface is relatively small for a malicious server attack and inference attacks focus more on the privacy concerns of FL, poisoning and backdoor attacks will be prioritised in this project.

While poisoning attacks try to use incorrect model parameter updates to reduce the model accuracy, backdoor attacks aim to inject a malicious task into an existing model while not impacting the model accuracy directly [11]. Poisoning attacks are also called byzantine attacks [18].

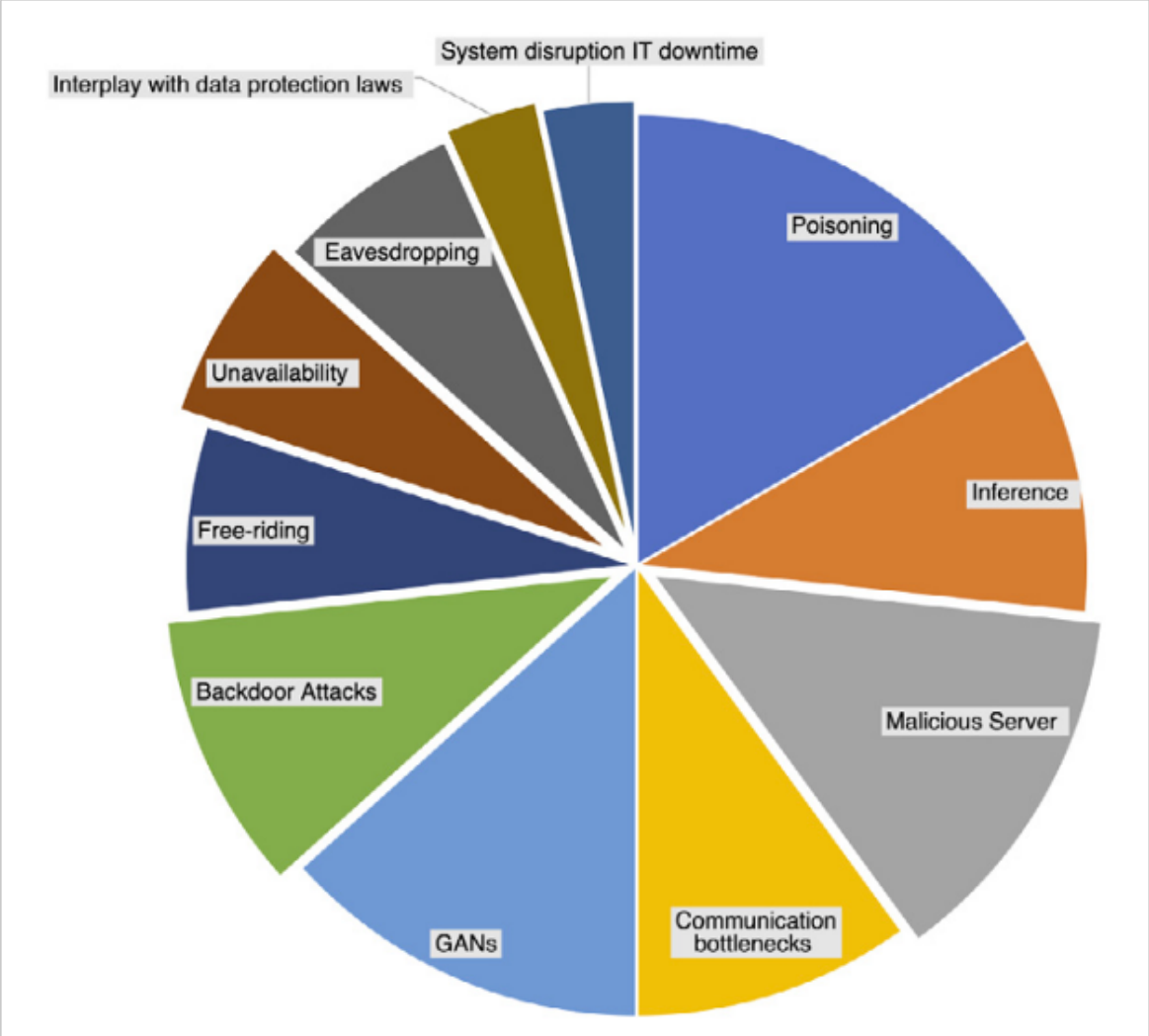



**Figure 2.1:** Attack vectors of Federate Learning security attacks. [11]

All attacks can be further differentiated between targeted and untargeted attacks. A targeted attack has a specific goal, while an untargeted attack tries to reduce the accuracy of the model without pursuing another specific goal [4, 19, 20]. Poisoning attacks can be targeted or untargeted, whereas backdoor attacks are by definition targeted attacks [20].

One example for a backdoor attack, is an image recognition algorithm where the model has been poisoned by malicious actors so that it identifies any image with a small white square at the bottom as a particular type [16]. Backdoor attacks are less straightforward than poisoning attacks, but can be highly effective [11]. Bagdasaryan et al. state that even a single shot attack from a single attacker in a single round of training leads to a 100% achievement on a targeted backdoor attack [17].

### 2.1.2 Poisoning attacks

In a poisoning attack, a distinction can be made between data poisoning and model poisoning. Data poisoning involves compromising the training data in the nodes to achieve malicious model parameters. This can, for example, involve label flipping or inserting of random or invalid label values. Model poisoning involves tampering with the model updates directly [4, 11, 17].

In data poisoning, a targeted attack can be the flipping of labels of certain data points, to decrease the classification success for a specific class. An untargeted attack can be to flip the labels of all classes and generally decrease model accuracy [4, 19].

In model poisoning, on the other hand, a targeted attack can be using a pre-built model to replace the machine learning model with a compromised model. An untargeted attack can be changing the model updates (randomly) to poisoned ones [4, 20].

To attack a distributed machine learning model, a malicious actor can use one edge node, or, for a more advanced attack, use multiple edge nodes. An attack in which several malicious nodes are coordinated and which, therefore, has a greater impact on the overall model, is called Sybil attack and is used often [16]. All this leads to the urgent question of how these attacks can be prevented.

## 2.2 Countermeasures

### 2.2.1 Classification of countermeasures

There are a few potential countermeasures already elaborated in past research. Tan et al. classify those countermeasures in three areas: robust aggregation, anomaly detection and hyper approach.

Firstly, robust aggregation describes efforts to average out the inputs from the nodes to reduce the influence of single nodes. This is achieved by using one of the byzantine-robust aggregation rules, for example, using the trimmed median instead of taking averages. This prevents the model from being dominated by single updates from individual nodes [19, 4].

Anomaly detection attempts to identify malicious nodes by detecting outliers. This is done by calculating the similarities between different updates from different nodes and mapping them onto a latent space. Model updates from identified outliers can be disregarded [4].

Lastly, the hyper approach combines robust aggregation and anomaly detection and reweights input parameters from potential malicious nodes instead of discarding

those parameters [4]. This is especially valuable, as it enables the retention of all model updates, but deprioritizes the model updates which are more likely to be malicious.

### 2.2.2 Malicious prevention algorithms

Research has been conducted to develop and test algorithms which prevent malicious actors to harm FL of the global model. Most of the previous research uses robust aggregation models and, for example, apply multi-Krum aggregation which calculates the Euclidean distance of the $n–f–2$ nearest neighbors and $f$ updates with the highest distances are removed [21, 16]. Another approach is the GeoMed, which is an aggregation method which calculates the geometric median of the means of the local model updates [22]. Also, the use of the trimmed mean as an aggregation algorithm to robust against malicious attacks is proposed [23].

In the area of anomaly detection, Li et al. uses a spectral anomaly detection framework to identify malicious nodes by putting data instances on a low-dimensional latent space and comparing the errors [18].

Tan et al. uses (among others) a Visual Basic Application (VBA) procedure as an hyper approach, which uses the training data set to determine whether a model parameter update is poisoned or not. For this purpose, the model determines a temporary new global model for each individual update and compares the accuracy with the previous accuracy. If the update does not decrease the model accuracy more than a given threshold, the model will be updated. Additionally, Tan et al. propose a Deep Reinforcement Learning (DRL)-based end node selection strategy to reduce the cost of model updates [4]. However, those mechanisms do not work for backdoor attacks, where the attacker does not aim to reduce the accuracy of the model.

Furthermore, Muñoz-González et al. propose Adaptive Federated Averaging (AFA), which considers the amount of data provided and the probability of the malicious node being malicious to aggregate the model parameters of each node in an iteration round [24].

Fung et al. developed a defense mechanism to identify malicious behavior within a Sybil attack and identifies malicious nodes by comparing the model updates from all nodes using the assumption that the parameters of malicious nodes are more similar and closer to each other than the parameters of honest nodes. However, the proposed solution does only work within a Sybil attack context [16].

A similar approach is taken with the Fed-Fi algorithm proposed by Zhou et al. [20]. The algorithm analyses similarities based on the hamming distance to protect a FL system from Sybil attacks [20].

As FL is a new technology, the ability to reliably detect malicious behaviour in every dimension and use case has been limited up to now. Currently a lot of research is conducted in the area of ruggedizing a FL system against byzantine attacks. However, the field is not yet fully explored and a perfect algorithm to detect malicious nodes has not yet been delivered. This project aims to contribute to reducing the amount of open security questions for FL by firstly providing analysis insights on the effects of malicious behaviour on FL. Secondly, by providing a new method of detecting malicious nodes and thirdly by extending this method to an automated malicious detection system, which can be applied to any FL environment.

# Chapter 3

# Experimental setup & tools

To achieve the aims and objectives of this project, an experimental setup is needed and provided. Three main technical contributions are made: Firstly, a FL testbed is developed which includes a tool that imitates malicious behaviour. Secondly, an analysis tool is built to a) capture the impact of malicious nodes and b) capture the performance of the statistical parameter to identify malicious nodes. Thirdly, an automated malicious detection system is included, which can be activated to identify malicious nodes and filter them out of the system in each update round based on flexibly designed input parameters.

This chapter gives an overview of the technical achievements as well as the chosen inputs, such as data sets and models, used in the experiments and analyses.

## 3.1 Data sets

Generally, the project focuses on FL systems and data sets in the image classification space. The baseline data set is the Modified National Institute of Standards and Technology (MNIST) data set, which is a data set of handwritten numbers between 0 and 9 as seen in figure 3.1 [25].

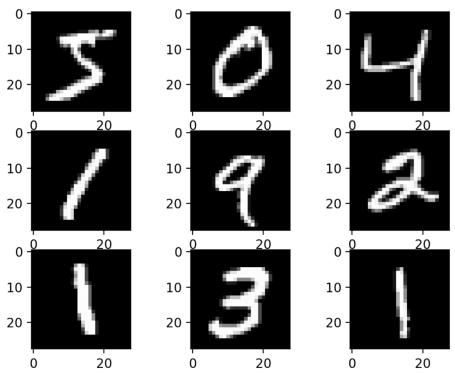


**Figure 3.1:** MNIST data set [26].

To strengthen the integrity of the results, the analyses are carried out with a second data set, the cifar-10 data set [27]. As can be seen in Figure 3.2, this is a data set that contains ten classes of images, e.g., animals and cars.

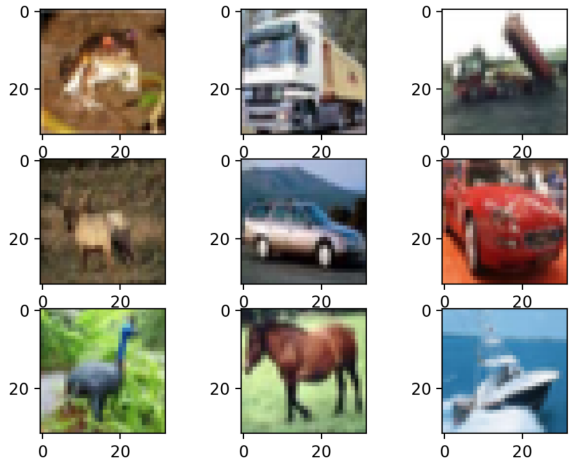


**Figure 3.2:** Cifar-10 data set [28].

## 3.2 Machine learning models

At the beginning, a Support Vector Machine (SVM) is used and applied to the MNIST data set [29]. To simplify the classification, the labels are classified as odd or even which leads to a binary classification problem.

Also, a more advanced model, a Convolutional Neural Network (CNN), is used [30]. The CNN is applied to the MNIST data set with a 0 to 9 label classification and it is applied to the cifar-10 data set.

In both models the standard settings are used, because the focus of the project is to compare the environment outcomes based on malicious actors, but not necessarily to optimise the FL model itself. A learning rate of 0.01 and a batch size of 100 with a mini batch sampler are used.

## 3.3 Federated Learning environment

A gradient descent FL testbed is developed using the Python programming language, reusing parts of the FL Environment developed by Wang et al. [31, 22].

The testbed consists of one server and multiple edge nodes. During initialisation, the server reads all information set as parameters by the user and, based on that, it

creates the FL system. Parameters which can be chosen by the user include model, data set, number of nodes and number of update rounds. After that, it connects to the nodes through a TCP connection and sends all relevant settings and information to the nodes, including the model and data set.

Each of the nodes has its own part of the data set, which is independent from the other nodes as shown in graph 3.3.

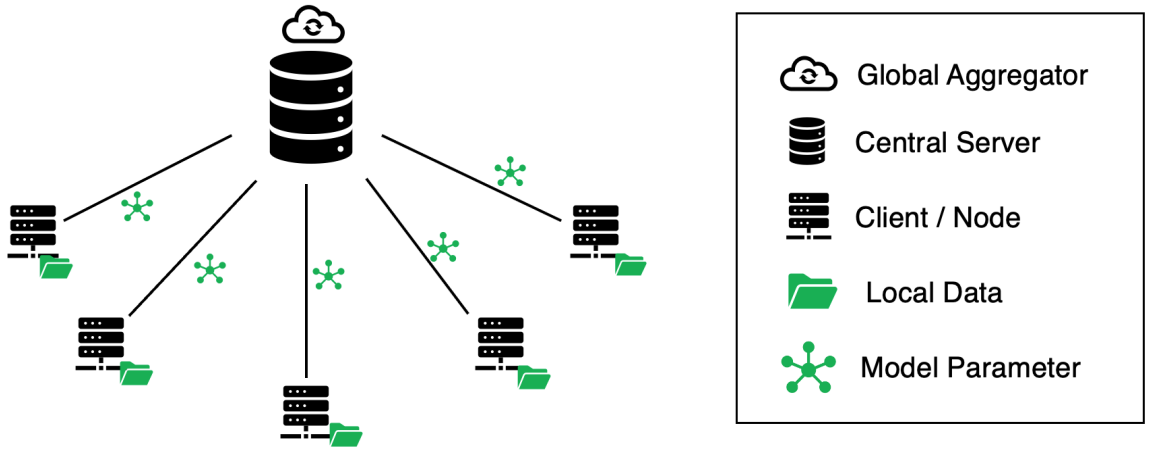



**Figure 3.3:** FL setup with multiple nodes and one central server.

The system will run for a given amount of rounds. In each round, each node updates its local machine learning parameters after a training round on a mini batch sample of its data set. Then, the nodes send their local model parameters to the central server, which waits for updates from all nodes before aggregating them with the mean-method, one of the potential methods to aggregate in FL [19].

The code parts used from S. Wang et al. are the functions related to remote procedure calls, the data extraction and processing of the data sets and the code for the (distributed) machine learning models [31]. The system from S. Wang et al. already includes the two data sets and models described above [31].

In this project, 500 update rounds are used as default, as the loss function shows conversion after those rounds. The project mainly works with 5 nodes to test the scenarios and systems, but the system is also tested with 10 and 20 nodes.

## 3.4 Tool to create malicious nodes

To understand how malicious nodes affect the FL, this project develops a tool into the FL system which artificially creates malicious nodes.

To gain flexibility, the tool allows the specification of the malicious nodes percentage, the percentage of malicious data and the time frame in which a node is malicious,

as well as the type of maliciousness. All those parameters are further elaborated in the section 4.1.

Depending on the given input from the user, the tool assigns the node in the setup phase a status (healthy or malicious) and the node behaves accordingly as seen in figure 3.4.

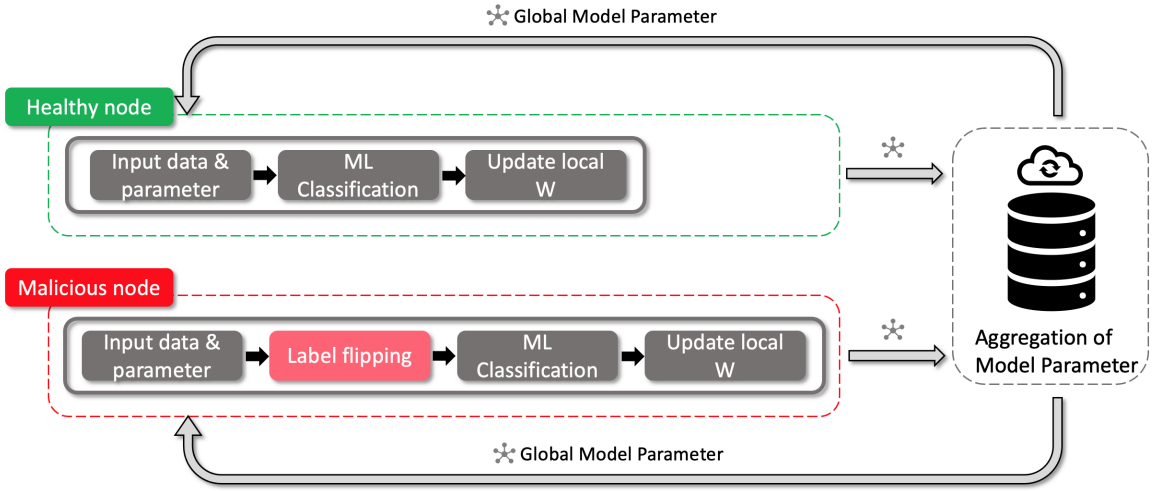



**Figure 3.4:** System overview of malicious and healthy nodes and the server aggregator.

If the node is classified malicious, the system will change the data set according to the type of maliciousness. If the node is only temporarily malicious, this will be captured and the behaviour of the node adjusted depending on the round.

To sum up, the focus is to design the system to be scalable and adjustable, so that the testbed can be used for different scenarios, settings and analyses.

## 3.5 Analysis tools

To evaluate the impact of malicious nodes on the FL, two types of analysis tools are build. The aim of the first part is to analyse how different scenarios of maliciousness impact the overall system. The second part aims to analyse how well the statistical parameter performs to identify malicious nodes.

### 3.5.1 Capturing impact of malicious nodes

The analysis of capturing the impact of malicious nodes to the overall system is divided into two parts. First each individual scenario (defined by the number of nodes, amount of malicious data and the time of maliciousness) is analysed and then different scenarios are compared to each other. For each scenario, the loss and accuracy as well as model parameters are captured and displayed in graphs.

Afterwards, an overall analysis is designed in order to compare the different scenarios and to show the differences in loss, accuracy or model parameters depending on the percentage of malicious nodes and malicious data. This is supposed to help understanding the bigger picture and how each different combination affects the global FL.

### 3.5.2 Capturing performance of detection parameter

To evaluate the performance of different statistical malicious detection parameters, the system calculates the statistical malicious detection parameter and outputs the results in diagrams. In addition, timings are recorded to document and compare the speed and efficiency of the different approaches (comparison of the limited model parameters approach with the full parameters approach is found in subsection 4.2.4).

## 3.6 Automated detection system

As part of the experimental setup, a detection tool is created which can be plugged into the system if needed. This tool is developed to automatically protect the FL from malicious nodes. The concept and idea is further elaborated in section 4.3.

The analysis tools described above can also evaluate the performance of the detection tool, e.g., how well the automated malicious detection system protects the FL from the malicious nodes.

# Chapter 4

# Method and analysis parameters

The described experimental setup is used to display and analyse the different areas of maliciousness, the statistical malicious detection parameter and the automated detection tool as described in this chapter.

This project considers different scenarios of a maliciousness in a FL environment to enhance the understanding of the influence of malicious nodes on FL, elaborated in section 4.1. Different combinations of those parameters are used to test how FL reacts to malicious nodes and thereby to design a statistical malicious detection parameter which is able to identify malicious nodes, as described in section 4.2. Lastly, this statistical malicious detection parameter is applied in an automated tool to protect the global FL (section 4.3).

## 4.1 Parameter of maliciousness

A number of different scenarios of maliciousness exist along which the analysis is varied. These are presented below.

### 4.1.1 Types of maliciousness

In this project, the focus is on “label changing” attacks, i.e., flipping the labels of the raw data. Label changing attacks are classified as data poisoning attacks as a subclass of model poisoning [4].

In the standard base case, the binary classification problem of the MNIST data set the labels are flipped from -1 to 1 and vice versa. In addition, the case where the labels are invalidated, i.e. 0, is analysed.

To simulate more complex image classification, the MNIST data set with number classification from 0-9 is used in combination with the CNN data set. To simulate maliciousness a random value is inserted. Another option is to use a value out of range, e.g. 10. This is not explored in this project, but could be a part of future work.

The cifar-10 data set classifies pictures into 10 different categories. Therefore, to simulate maliciousness, similarly to the CNN MNIST scenario a random value between 0 and 9 is inserted.

The main reason for using this type of malicious data is that it is a simple but effective way of generating maliciousness. In the course of the project, it turns out that this is a very "gentle" way of generating malicious data and that the differences in the model parameters and thus the classification of malicious and healthy nodes would be even greater in other cases.

Other ways to create malicious nodes are, for example, using different input data sets or even tampering with the model parameters directly, which would be classified as model poisoning by Tan et al. [4]. This is not part of the scope of this work, but should be a topic for future work and is discussed in section 7.3 in more detail.

### 4.1.2 Percentage of malicious nodes

It is possible to vary the number of malicious nodes. Within this project, the healthy case is compared with cases of 20%, 40%, 60%, 80% and 100% malicious nodes. It is noticeable that the cases with a higher percentage of malicious nodes compared to healthy nodes are edge cases which are only included to see the bigger picture and understand the behaviour of the data. However, in a realistic scenario, these may not be as relevant.

### 4.1.3 Amount of malicious data

The amount of malicious data within a malicious node is varied. Five different cases are compared: 20%, 40%, 60%, 80% and 100% malicious data.

Especially in a real world environment it is realistic that not always 100% of the data from a malicious node is malicious and the project aims to answer the question of whether a potential statistical malicious detection parameter could detect a malicious node with little malicious data. However, as Li et. al already elaborated, in an average aggregation FL (more specifically, mean aggregation mechanism in this project), a malicious node must have a big difference to healthy nodes to have a meaningful impact and overrule the healthy nodes [18]. Therefore, it can be stated, that high percentages of malicious data in one node are more likely to occur.

### 4.1.4 Time frame of maliciousness

Also, the time frame of maliciousness is varied. This is a realistic scenario, as it can be assumed that the malicious node is not malicious from the start but turns malicious after time or only for a short period of time. Therefore four cases are analysed: a scenario where a node is always malicious, a scenario where a node turns malicious after 1/3 of the rounds, another scenario where a node turns malicious after 1/3 of

the rounds and healthy after 2/3 of the rounds, and lastly a scenario where a node is malicious from start and turns healthy after 2/3 of the rounds.

## 4.2 Detection parameter

One goal of the project is to find a parameter which is able to detect one or more malicious nodes in a FL environment. To achieve this, the model parameters of the individual nodes are compared in each aggregation round. After analysing absolute model parameters, different statistical measurements are explored to express the differences between healthy and malicious model parameters.

### 4.2.1 Absolute average Mean Squared Error

The Mean Squared Error (MSE) of the nodes' model parameters is identified as a statistical malicious detection parameter to compare the model parameters of healthy and malicious nodes. As it is the squared error of the model parameters, it emphasises the outliers and gives a good approximation of the distribution of the nodes' model parameters as seen in Equation 4.1.

For each of the model parameters, the mean is calculated across all nodes and for each node the squared error to this mean is calculated. As machine learning models usually have a couple of hundred model parameters, it is not possible to compare all of them individually. Therefore, for each node the average MSE of all model parameters is taken.

It is important that the averaging is done after taking the MSE for each of the parameters. In this way, it is assured that the information of the error of the individual model parameters is captured, before aggregating it to one number to compare.

For each n in N:

$$Absolute\ average\ MSE = \frac{1}{M}\sum_{m=0}^{M}(\bar{w}_m - w_{mn})^2 \tag{4.1}$$

Where:
*N = number of nodes*
*n = node index*
*M = number of model parameters*
*m = model parameter index*

For each m in M:

$$\bar{w}_m = \frac{1}{N}\sum_{n=0}^{N} w_m \tag{4.2}$$

### 4.2.2 Relative average Mean Squared Error

The absolute average MSE identifies a malicious node, but fails to be applicable to any data set and model in the same way. This is especially important, as this analysis and later the automated tool should be transferable to multiple data sets and models. Therefore, a relative version of the average MSE is developed, namely the relative average Mean Squared Error (raMSE).

It becomes apparent that it is relevant to not lose the information about the size of the model parameters. Generally, the bigger the absolute value of the model parameters is in a machine learning model, the more important the model parameter [32]. By using the same formula as in the absolute average MSE and then dividing it by the median of all of these values as seen in Equation 4.3 the information about the distribution of the model parameters is not lost.

To relativise the absolute average MSE it is decided to divide it by the median of the absolute average MSE (Equation 4.3). The median is chosen to relativise the average MSE, as it is more robust to outliers than, for example, the mean. Outliers are in this case big absolute average MSE from potential malicious nodes.

For each n in N:

$$Relative\ median\ average\ MSE = \frac{\frac{1}{M}\sum_{m=0}^{M}(\bar{w}_m - w_{mn})^2}{median\ of\ [\frac{1}{M}\sum_{m=0}^{M}(\bar{w}_m - w_{mn})^2]_{n=[1,N]}} \tag{4.3}$$

Where:
*N = number of nodes*
*n = node index*
*M = number of model parameters*
*m = model parameter index*

For each m in M:

$$\bar{w}_m = \frac{1}{N}\sum_{n=0}^{N} w_m \tag{4.4}$$

### 4.2.3 Moving average of relative average Mean Squared Error

Model parameters are volatile and fluctuate a lot. Therefore, during the project a moving average of the raMSE is used as an indicator for maliciousness. This has the advantage that it is a more stable value. It comes with the cost that it also takes into account historical values and thereby a new behaviour is noticed later. The higher the value of moving averages, the later a potential behaviour change is noticed, but

the more stable the results are.

Within this project, a moving average of 25 and a moving average of 10 are used. While a human eye might not clearly differentiate which number is better to use, this is potentially a parameter which can be adjusted depending on model and data set and the decision for a moving average value could be automated by a machine learning tool.

### 4.2.4 Efficiency constraints and model parameter selection

It becomes evident that depending on the machine learning model there are a lot of different parameters and it might not be sufficient to compare all of them, as real world models do have time constraints and resource limitations [33]. This is also one limitation of many other systems in literature, as pointed out by Li et al. [18].

Therefore in this project it is tested whether the statistical malicious detection parameter would be able to detect a malicious node if it only has $1\%$ of the model parameters at hand. This reduces the calculation power needed and enables a more efficient malicious node detection.

This is inspired by the information that the size of the model parameters is important, as this implies that bigger model parameters potentially have a bigger influence on the classification variable.

In chapter 5 it is shown that it is not only possible with $1\%$ of the model parameters, but also, e.g., $10\%$ of model parameters.

Lastly, the system needs to define which model parameters should be considered. It is possible to do this each update round or to define the chosen model parameters during one update round and then use this selection for a couple of rounds. The second approach is more resource efficient, but might not be as accurate as defining the important model parameters each round. Both approaches are possible; however, in this research work the focus was set on redefining the most important model parameters each update round.

## 4.3 Automated detection

Lastly, after all the analysis described above, the insights are used to create a detection tool which is able to automatically detect a malicious node based on the moving average of the raMSE and exclude the node from the update round by building a detection layer around the central server as shown in graph 4.1.

The maintainer of a FL environment is able to specify parameters which determine how the automated malicious detection system operates. Currently the system oper-

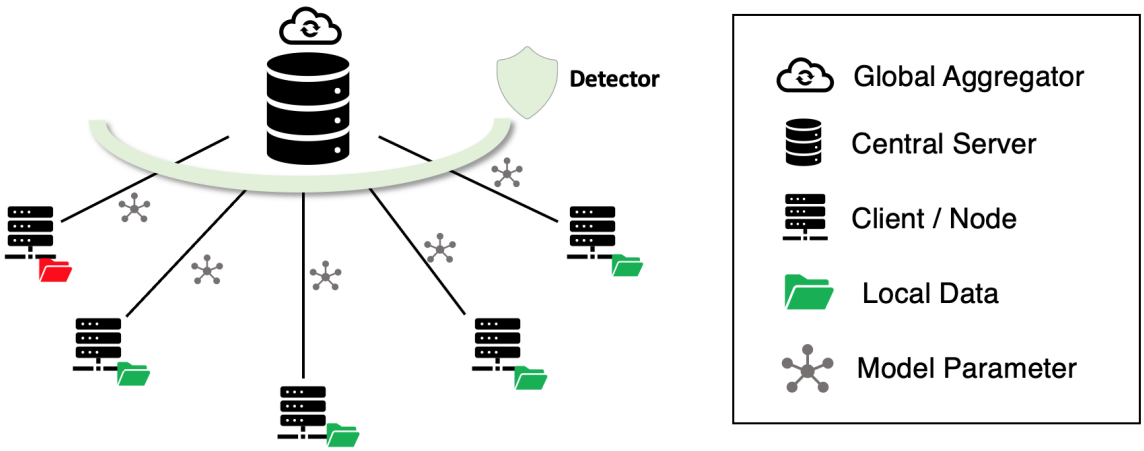



**Figure 4.1:** FL setup with multiple nodes, one central server and an integrated malicious detection tool.

ates with a static threshold strategy.

Therefore, the first parameter is the threshold of the moving average of the raMSE by which a node is classified as malicious. As a standard setting a threshold of 1.5 is used, as it has been shown in the analysis, which is presented in chapter 5, as a reliable borderline for most scenarios. The second parameter is the size of the moving average and during this project a moving average of 25 has been chosen as it is a compromise between quickly displaying a change of maliciousness of a node and a stable trajectory of the outputs. The third parameter is the number of model parameters to be considered. Currently, the project chooses to consider all model parameters (in contrast to only using 1% or 10%), as the automated malicious detection system is mainly tested on the SVM model, which is less resource intensive.

It is possible to determine the optimal values of those parameters with a machine learning model. This can be a path for future work and will be discussed in section 7.3.

# Chapter 5

# Analysis results

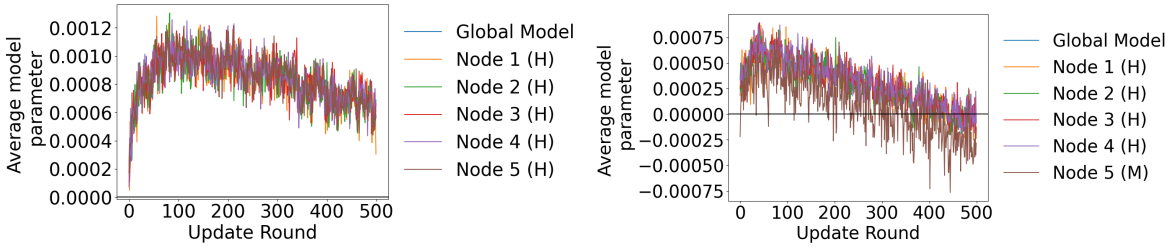
The methods and parameters described above are applied to the FL system described in chapter 3. This outputs a variety of information and diagrams from the analysis tools, which are elaborated in this section.

Unless otherwise stated, the base case is analysed, which means using the SVM model and the MNIST data set reduced to a binary classification, 500 update rounds, maliciousness defined as flipped labels, 5 nodes and continuous maliciousness if a node is classified as malicious (no transient maliciousness).

## 5.1 Impact of malicious nodes

Malicious nodes have an impact on FL, as the global aggregator calculates the mean across all model parameters as described in section 3.3. If a node is malicious and trains on flipped labeled data, which is the type of maliciousness considered here, the model parameters or model parameter distribution is different than for healthy nodes.

In graph 5.1 the average weight for each node and the global model is shown. The average weight is calculated by taking the average across all model parameters for each node and global model for each round. In graph 5.1a a healthy environment is displayed, while in graph 5.1b a malicious environment is shown, with 20% malicious nodes and 80% malicious data. It can be observed that in this specific case (for the MNIST data set and SVM model) the average weight from the malicious node is smaller than a healthy node. Therefore, the global model parameters are lowered.




**(a)** Healthy environment **(b)** Malicious Environment

**Figure 5.1:** Comparing the model parameters of the 5 nodes and the global model in an healthy environment in (a) to the model parameters of a malicious environment with 20% malicious nodes and 80 % malicious data in (b). H stands for healthy node, M stands for malicious node. The FL system uses the SVM model with the MNIST data set.

The influence of malicious nodes on the overall system becomes particular clear when analysing the loss of the global aggregated model in different scenarios. Comparing different numbers of nodes as well as different amounts of malicious data in graph 5.2 shows that the more malicious nodes and the more malicious data each of the malicious nodes has, the higher the loss value.

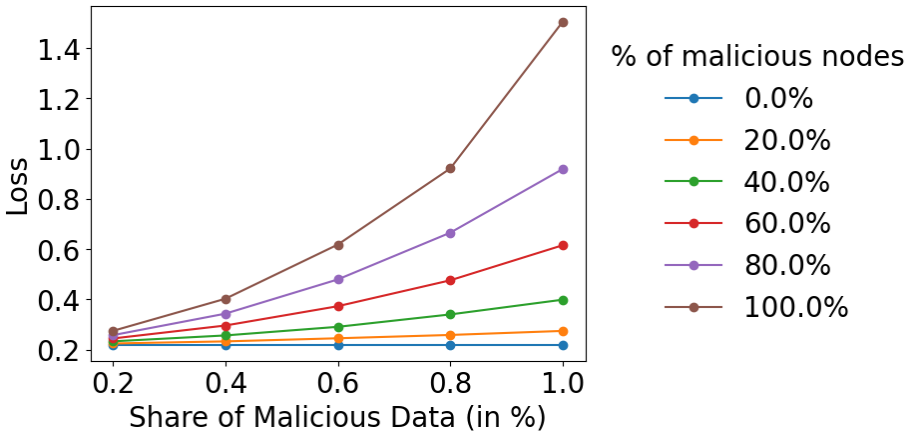



**Figure 5.2:** Loss value after 500 update rounds of the different malicious scenarios. The FL system uses the SVM model with the MNIST data set.

This is also true for transient malicious nodes. Graph 5.3 shows that the loss value increases similarly even when malicious nodes are only malicious between update round 150 - 300. This is reasonable, as during the rounds where the nodes are malicious the machine learning model learns less strongly. The effects of transient maliciousness are further discussed in subsection 5.2.2.

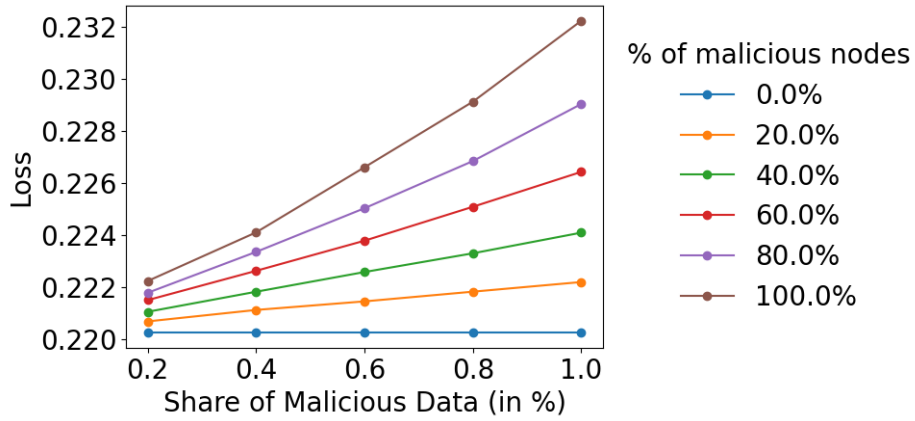



**Figure 5.3:** Loss value after 500 update rounds of the different malicious scenarios in a transient malicious case (Round 150-300 malicious, all other rounds healthy). The FL system uses the SVM model with the MNIST data set.

While the loss graphs show an unambiguous result, the accuracy is not so straightforward. When analysing all (non-transient) cases of maliciousness, it can be observed that for 0 - 40% malicious nodes the accuracy can be less clearly differentiated, as seen in graph 5.4 (lines for 0% and 20% malicious nodes are hidden behind the 40% line).

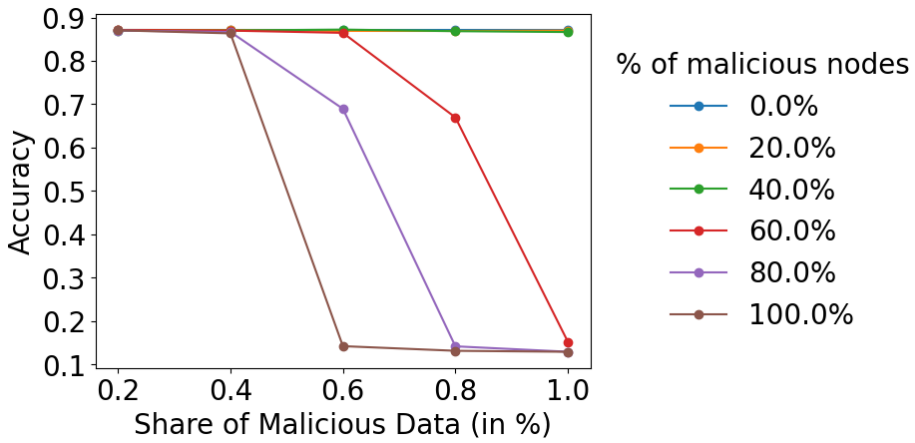



**Figure 5.4:** Accuracy after 500 update rounds of the different malicious scenarios. The FL system uses the SVM model with the MNIST data set.

When zooming in, it becomes clear that the accuracy is reduced in most cases as expected, but just very marginal (graph 5.5). One outlier is the scenario of 40 % malicious nodes and 60 % malicious data.

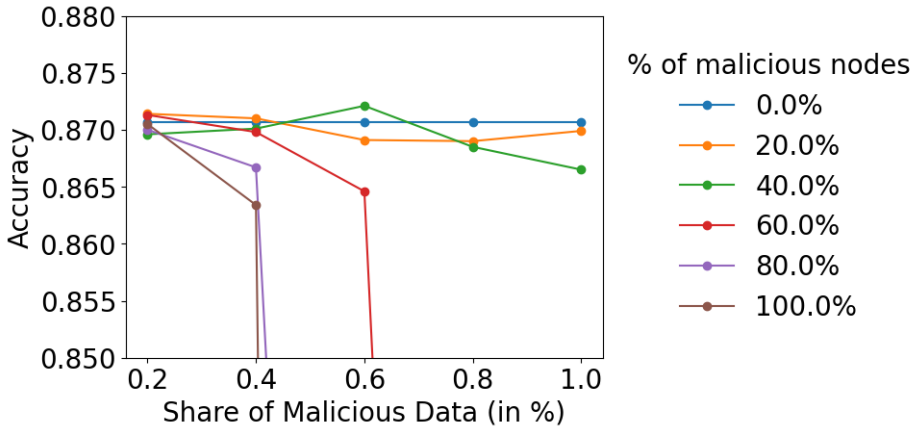



**Figure 5.5:** Zoomed in accuracy after 500 update rounds of the different malicious scenarios. The FL system uses the SVM model with the MNIST data set.

In the transient case the accuracy clearly drops for all cases as seen in graph 5.6, but the high malicious nodes cases reduce the accuracy less than the non-transient case, where the accuracy is almost equal to zero, as observed in graph 5.4. This is further analysed in chapter 6.

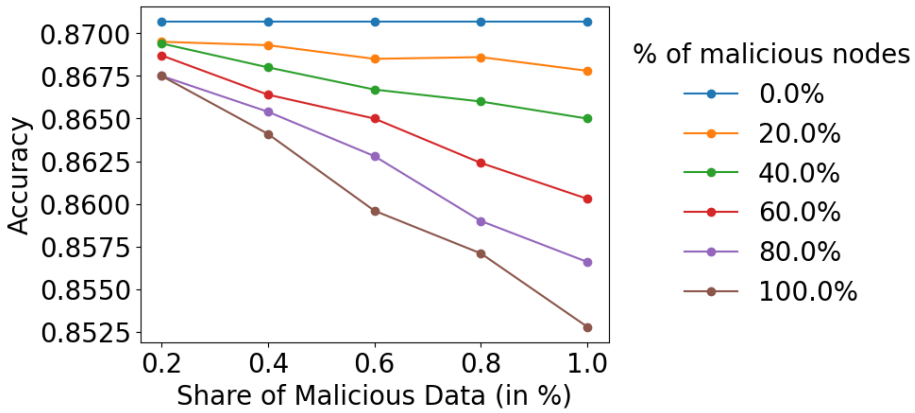
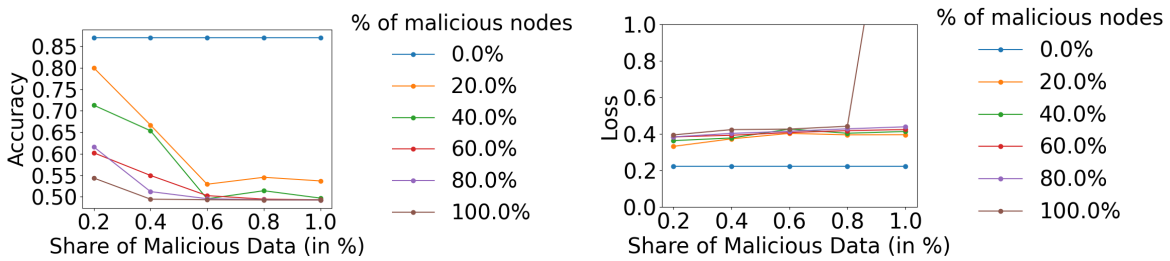



**Figure 5.6:** Accuracy after 500 update rounds of the different malicious scenarios in a transient malicious case (Round 150-300 malicious, all other rounds healthy). The FL system uses the SVM model with the MNIST data set.

It should be noted that the loss function describes the size of the errors of the model on the training data, while the accuracy can be described as the number of errors [33, 34]. The loss functions used in this project are the standard loss functions for each of the machine learning models, SVM and CNN.

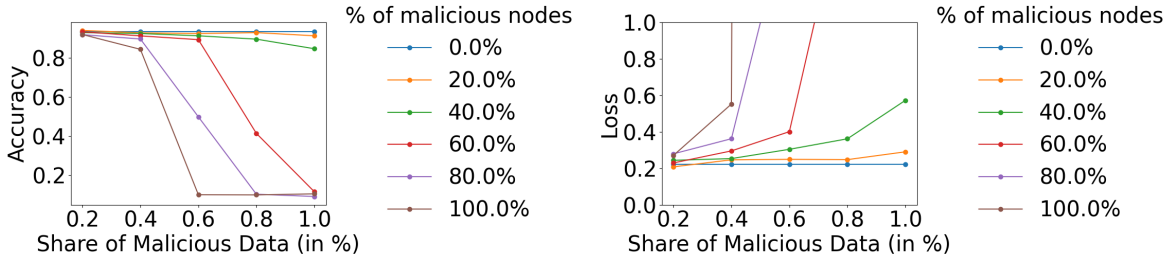
If not label flipping as a malicious data point is used, but using invalid data, the accuracy and loss are more affected by maliciousness. This can be observed in graph 5.7. The y axis for graph 5.7b is capped by 1, to enable better readability of the values.




**(a)** **(b)**

**Figure 5.7:** Accuracy sub-figure (a) and loss value sub-figure (b) after 500 update rounds of the different malicious scenarios with invalid labeling (= 0) as maliciousness. The FL system uses the SVM model with the MNIST data set.

The MNIST data set in a non-binary classification problem with the CNN model produces a similar loss and accuracy for different scenarios. In graph 5.8 the accuracy and loss for a non-transient case is shown. The y axis of the loss function in graph 5.8b is cut at 1, as the values above were so exorbitant big, that the graph was not readable anymore.




**(a)** **(b)**

**Figure 5.8:** Accuracy in sub-figure (a) and loss value in sub-figure (b) after 500 update rounds of the different malicious scenarios. The FL system uses the CNN model with the MNIST data set.

## 5.2 Detection parameter performance

To prevent a malicious node having influence on the global model, the second objective of this project is to identify a malicious node in the system. In this section the results are presented of how the statistical malicious detection parameter, the raMSE as described in subsection 4.2.2, performs in a malicious FL environment.

Unless otherwise stated, the same parameters as in section 5.1 apply. Furthermore, it can be assumed that the values shown are the moving average (25 values) of the raMSE.

### 5.2.1 Baseline maliciousness (varied malicious nodes and data)

To be able to identify the malicious nodes in the various cases, an understanding of the case with no malicious nodes is needed. In a healthy FL environment with no malicious nodesthe raMSE does not identify any malicious node as seen in graph 5.9. It can be observed that there are many fluctuations even though the moving average of 25 is used. Fluctuation in the raMSE is expected and not explainable, as the training of the model and therefore the model parameters are slightly different in each round. In subsection 4.2.3 the difference between different moving averages and the raMSE without a moving average is discussed.

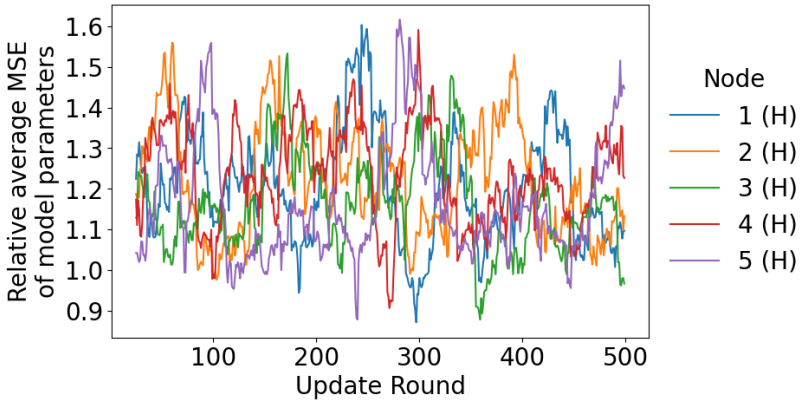
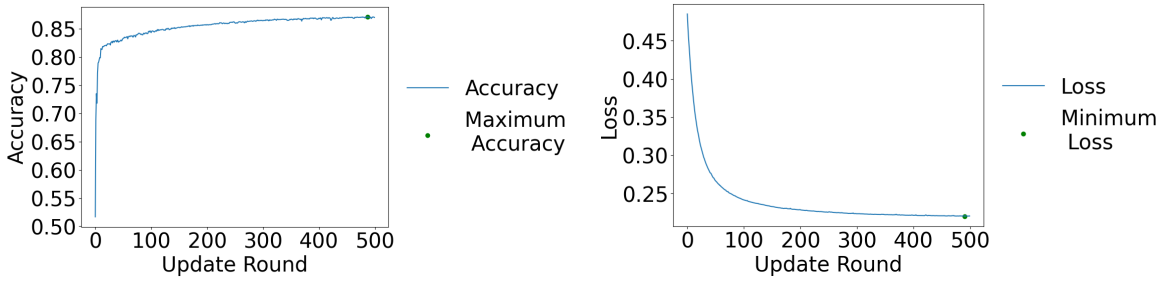



**Figure 5.9:** Relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes for 500 update rounds in an healthy FL environment. H stands for healthy. The FL system uses the SVM model with the MNIST data set.

500 update rounds are chosen as it becomes obvious that the models converge after this amount of update rounds, as seen in graph 5.10. Graph 5.10 shows the loss value and accuracy for the healthy scenario (no malicious nodes) in a FL environment.




**(a)** **(b)**

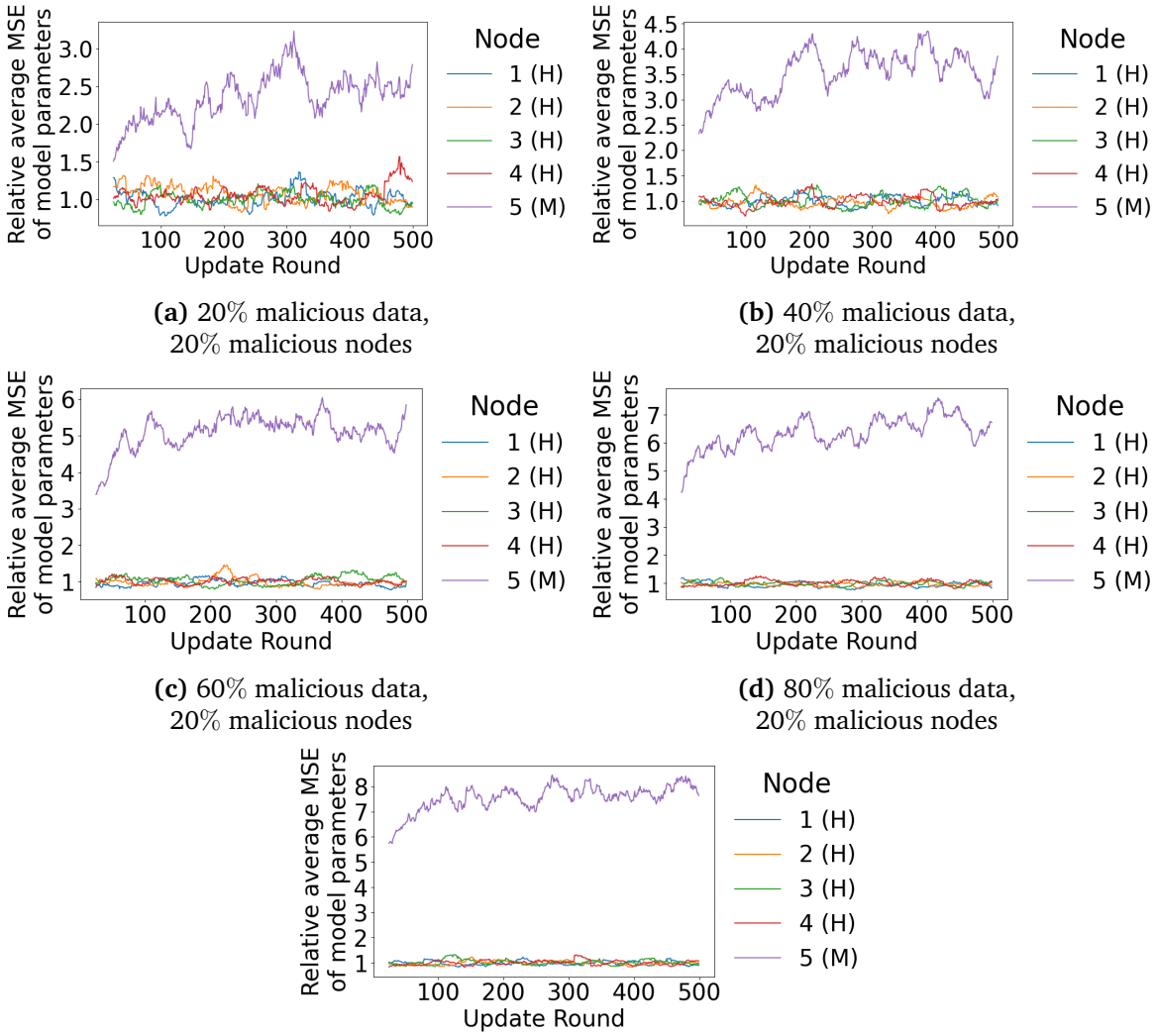
**Figure 5.10:** Accuracy in sub-figure (a) and loss value in sub-figure (b) for 500 update rounds in an healthy FL environment. The FL system uses the SVM model with MNIST data set.

To show the performance of the raMSE, the different malicious parameters introduced in section 4.1 are varied and different scenarios are presented. Two parameters are varied in this section: the number of malicious nodes as well as the amount of malicious data each malicious node has. The other parameters are staying constant in this subsection. If there is a malicious node, it stays malicious during all update rounds (no transient maliciousness) and maliciousness is defined as label flipping.

Starting with only 20% malicious nodes, five different data cases for this are analysed: 20%, 40%, 60%, 80% and 100% malicious data.

In graph 5.11 the results for these data cases are presented. Each sub-graph is one data case and shows the moving average of the raMSEs of the five nodes in the system and their classification (healthy or malicious). The higher the raMSE value, the more the node's model parameters deviate from the model parameters of the majority of the nodes.

It is observable that in all scenarios presented in graph 5.11 the raMSE is different for a malicious node (in comparison to a healthy node) and thus the malicious node can be identified.




**(a)** 20% malicious data, 20% malicious nodes

**(b)** 40% malicious data, 20% malicious nodes

**(c)** 60% malicious data, 20% malicious nodes

**(d)** 80% malicious data, 20% malicious nodes

**(e)** 100% malicious data, 20% malicious nodes

**Figure 5.11:** Sub-figures (a), (b), (c), (d) and (e) show different data cases of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in an environment with 20% malicious nodes. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

The further the amount of malicious data is increased, the bigger the difference of the raMSE and the easier the identification of a malicious node as observed in graph 5.11. This becomes even clearer in graph 5.12. Graph 5.12 shows for each node the mean value of the raMSE over all update rounds. The values for the healthy nodes are summarised and averaged to increase the readability of the graph. It can be observed, that the bigger the amount of malicious data, the further away is the raMSE value of the malicious node to the healthy nodes.

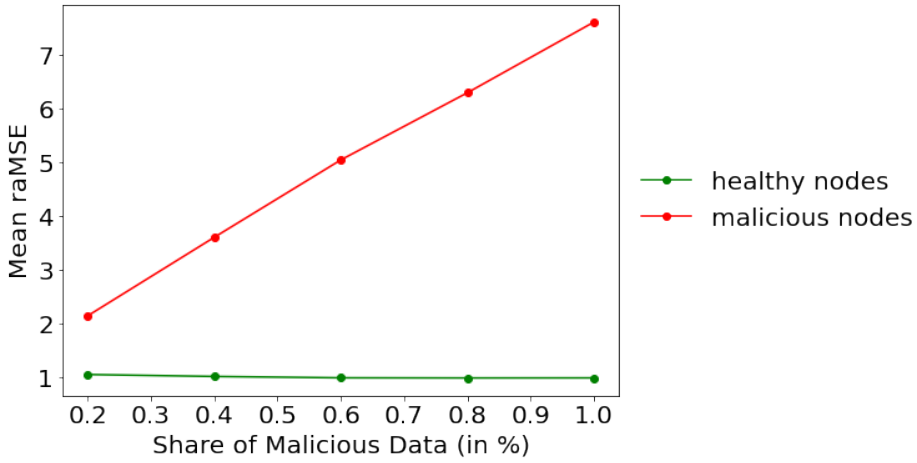
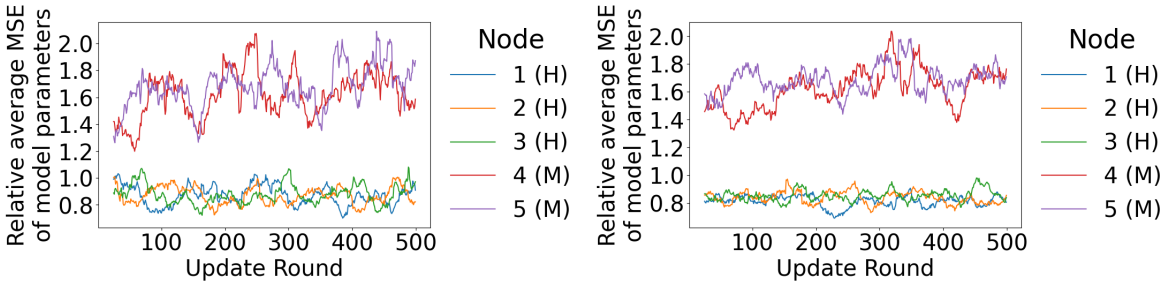



**Figure 5.12:** Mean value of the relative average Mean Squared Error (raMSE) for each node over 500 update rounds with 20% malicious nodes in different data cases. The FL system uses the SVM model and the MNIST data set.

Malicious nodes can also be identified in the scenario of 40% malicious nodes, which means in the case of 5 nodes, 2 malicious nodes and 3 healthy nodes. Again all different data scenarios (20% vs 40% vs 60% vs 80% vs 100%) are applied and an overview of all graphs for the 40% malicious nodes case can be found in appendix (graph B.1). Graph 5.13 shows two exemplary result for 40% malicious nodes and 60% and 80% malicious data.




**(a)** 40% malicious data, 40% malicious nodes

**(b)** 80% malicious data, 40% malicious nodes

**Figure 5.13:** Sub-figures (a) and (b) show two exemplary data cases of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in a FL system with 40% malicious nodes. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

Graph 5.14 shows the mean of the raMSE for both 20% malicious nodes (A) and 40% malicious nodes (B) for all data cases. It can be observed that the malicious nodes in the scenario of 20 % malicious nodes have a higher raMSE than the malicious nodes in the scenario of 40% malicious nodes. This means, that the identification of malicious nodes in the scenarios with 20% malicious nodes is clearer than the identification of malicious nodes in the scenarios with 40% malicious nodes. Furthermore, it is observable that in the scenarios with 40% malicious nodes the mean value of the

raMSE for the malicious nodes stays constant. In all cases a malicious node can be identified. The healthy node baseline of the two scenarios (20% malicious nodes vs 40% malicious nodes) differ, because in the scenario for 40% malicious nodes more malicious nodes affect the average MSE.

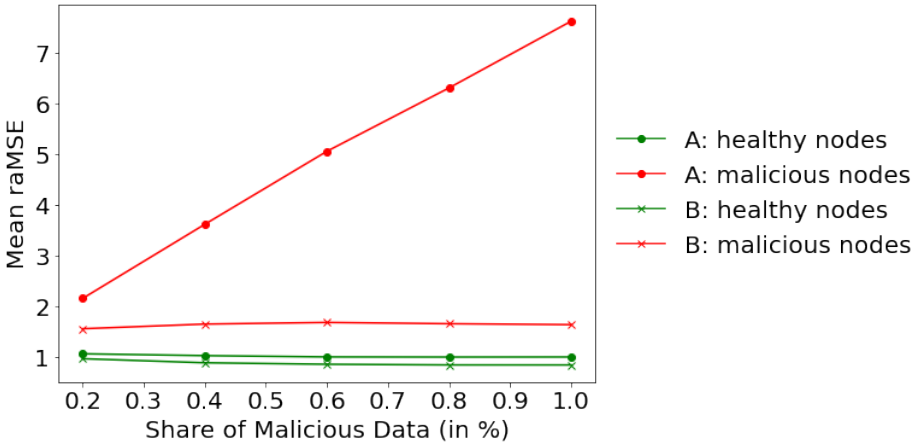
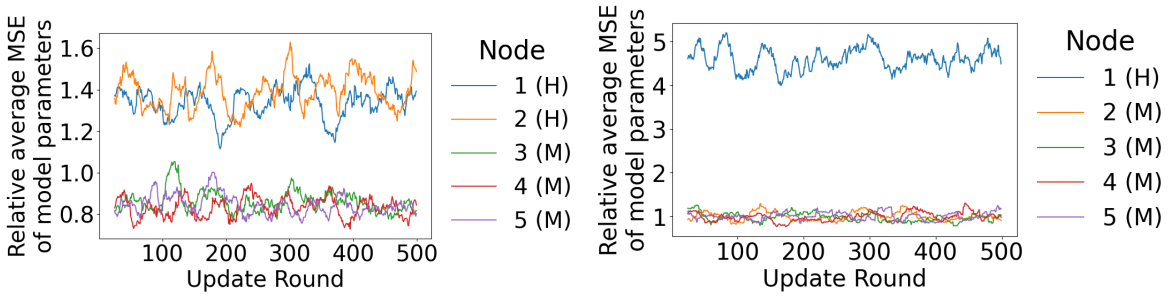



**Figure 5.14:** Mean value of the relative average Mean Squared Error (raMSE) for each node over 500 update rounds with 20% (A) and 40 % malicious nodes (B) in different data scenarios. The FL system uses the SVM model and the MNIST data set.

One limitation of the system is that it can only identify the odd ones out of a group of nodes. This means, with the raMSE, nodes can be classified into two subgroups, malicious and healthy nodes. However, this classification is depending on which of those groups is the bigger subgroup. The bigger subgroup is identified as healthy, whereas the smaller subgroup is identified as malicious. According to that, if there is a system with more malicious than healthy nodes it will classify all healthy nodes as malicious and vice versa. Therefore, during this project the assumption is made that there are more healthy than malicious nodes in the environment.

This limitation can be observed in graph 5.15. The groups are distinguishable for 60% and 80% maliciousness, but the parameter identifies the healthy nodes as malicious, because they are in the minority. For both of these graphs 80% of malicious data is chosen as a representative data point. The limitation of identifying malicious nodes in a primarily malicious environment can be addressed in future work.




**(a)** 60% malicious nodes, 80% malicious data

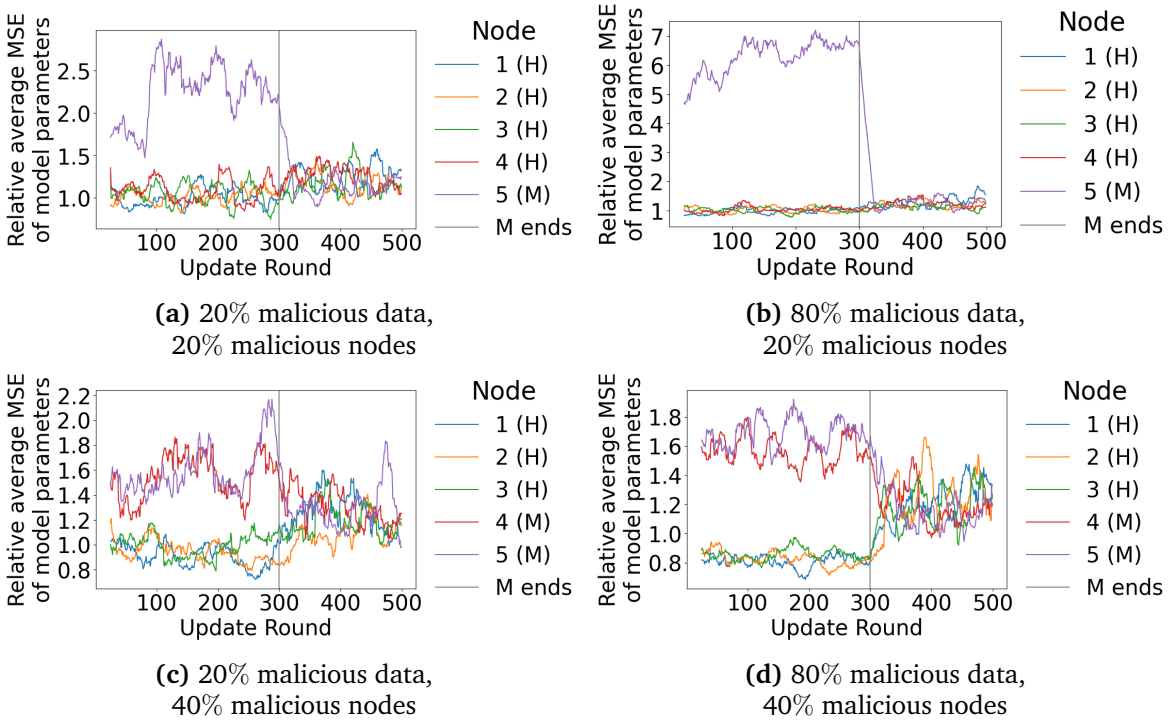
**(b)** 80% malicious nodes, 80% malicious data

**Figure 5.15:** Sub-figures (a) and (b) show different scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in a FL system. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

### 5.2.2 Transient maliciousness

In a real world environment it is most likely that a node turns malicious at some point, which doesn't necessarily have to be the starting point of the FL. Therefore, it is likely that nodes are only transiently malicious. Thus, multiple experiments to analyse transient malicious nodes are designed.

Firstly, a malicious node turns healthy again after $2/3$ of the rounds, which, in an environment of 500 update rounds, is after 300 rounds. Graph 5.16 shows for four exemplary cases the behavior of the raMSE. It becomes obvious, that in all cases the former malicious node is converging with the healthy nodes after round 300 and thereby not classified as malicious anymore.




**(a)** 20% malicious data, 20% malicious nodes

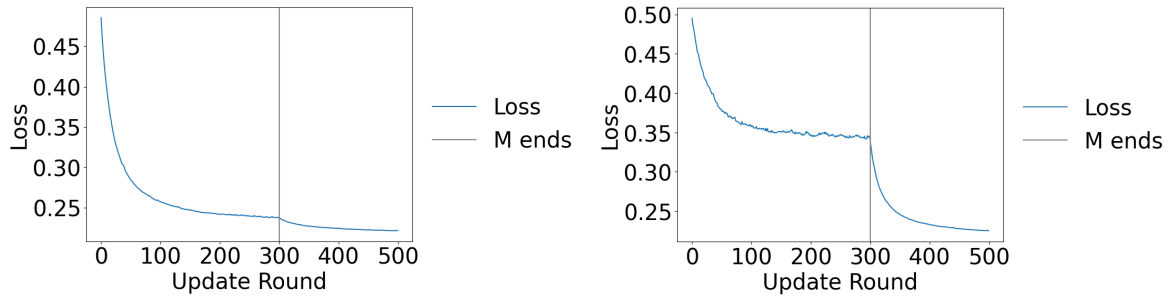
**(b)** 80% malicious data, 20% malicious nodes

**(c)** 20% malicious data, 40% malicious nodes

**(d)** 80% malicious data, 40% malicious nodes

**Figure 5.16:** Sub-figures (a), (b), (c) and (d) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in a FL system in a transient malicious case (Round 0-300 malicious, all other rounds healthy). H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

It is recognisable, that the transient maliciousness also affects the loss only during the time the node is actually malicious as seen in graph 5.17 and the loss converges as soon as the node turns healthy again. In graph 5.17 are two exemplary loss value graphs shown. The loss functions for other scenarios behave similarly.

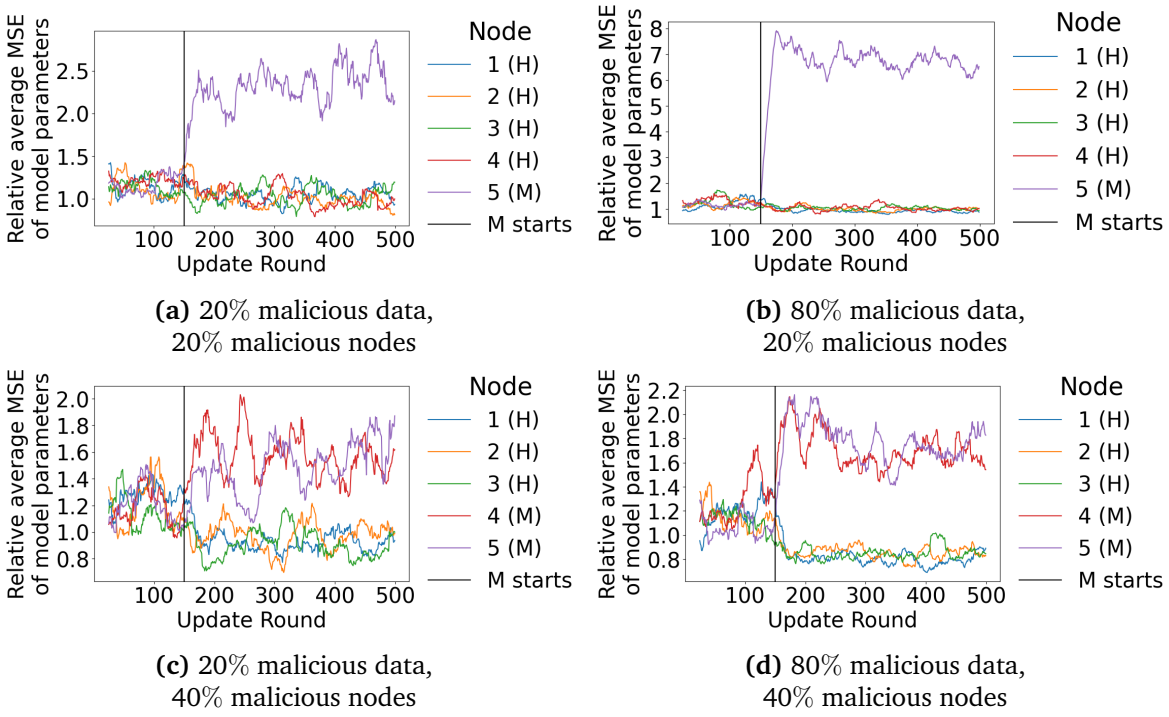



**(a)** 20% malicious nodes, 20% malicious data

**(b)** 20% malicious nodes, 80% malicious data

**Figure 5.17:** Sub-figures (a) and (b) show the loss value for 500 update rounds for different malicious scenarios in an transient malicious FL environment (Round 0-300 malicious, all other rounds healthy). M stands for malicious. The FL system uses the SVM model with MNIST data set.

The second case is that a malicious node is only starting to be malicious after $1/3$ of the rounds, meaning the node is malicious from round 150 to 500. This scenario applies for example when a malicious actor tampers with an existing healthy node and turns it malicious. The selected exemplary scenarios in graph 5.18 show, that the malicious nodes are identified correctly also within this scenario.




**(a)** 20% malicious data, 20% malicious nodes

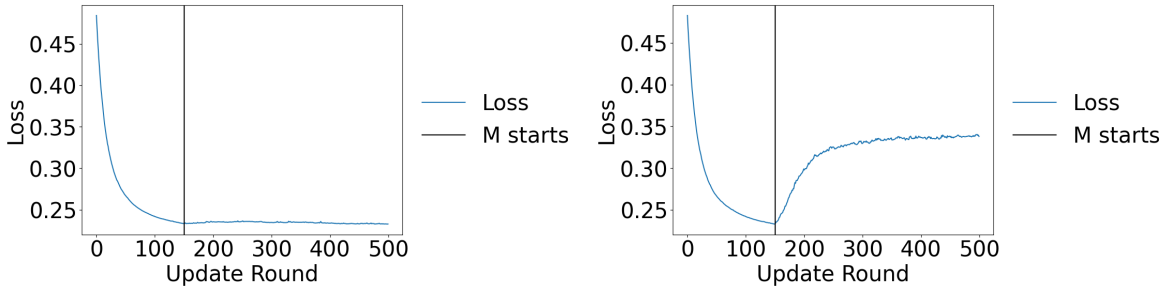
**(b)** 80% malicious data, 20% malicious nodes

**(c)** 20% malicious data, 40% malicious nodes

**(d)** 80% malicious data, 40% malicious nodes

**Figure 5.18:** Sub-figures (a), (b), (c) and (d) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in a FL system in a transient malicious case (Round 150-500 malicious, all other rounds healthy). H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

Also in this case the transient maliciousness affects the loss. This is shown in graph 5.19.

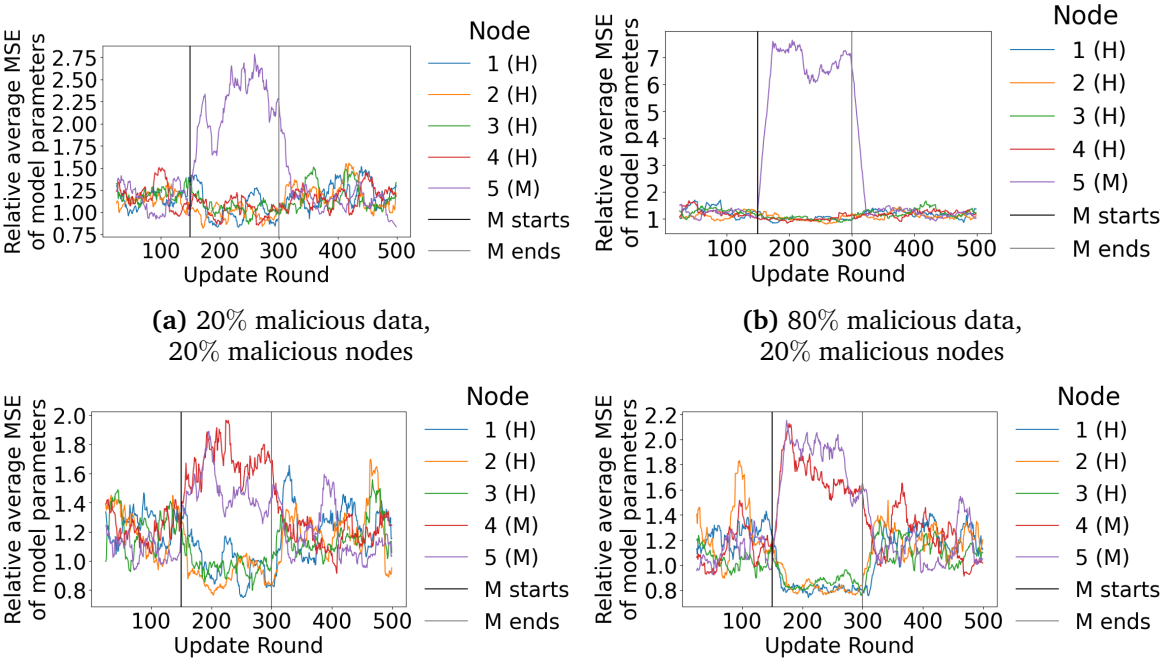



**(a)** 20% malicious nodes, 20% malicious data

**(b)** 20% malicious nodes, 80% malicious data

**Figure 5.19:** Sub-figures (a) and (b) show the loss value for 500 update rounds for different malicious scenarios in an transient malicious FL environment (Round 150-500 malicious, all other rounds healthy). M stands for malicious. The FL system uses the SVM model with MNIST data set.

Lastly, a malicious node is only starting to be malicious after $1/3$ of the rounds and turns healthy again after $2/3$ of the rounds, which means the node is malicious between round 150 and 300. In graph 5.20 again four different cases for this transient maliciousness are shown.




**(a)** 20% malicious data, 20% malicious nodes

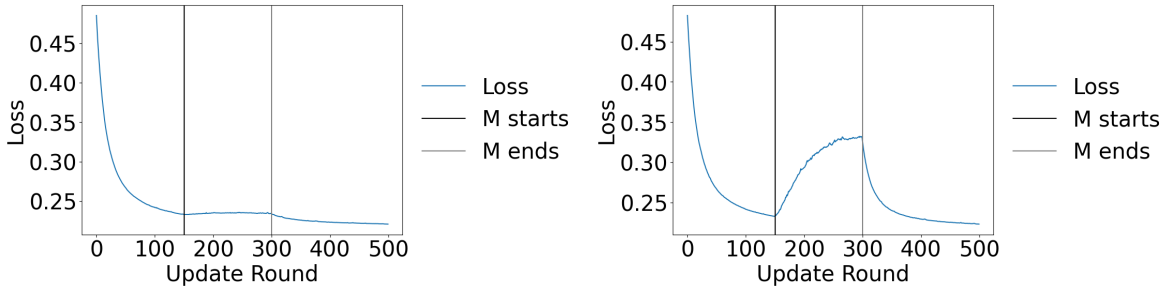
**(b)** 80% malicious data, 20% malicious nodes

**(c)** 20% malicious data, 40% malicious nodes

**(d)** 80% malicious data, 40% malicious nodes

**Figure 5.20:** Sub-figures (a), (b), (c) and (d) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in a FL system in a transient malicious case (Round 150-300 malicious, all other rounds healthy). H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

In this case, the loss functions behaviour differs only as long as the node is malicious, as seen in graph 5.21.




**(a)** 20% malicious nodes, 20% malicious data

**(b)** 20% malicious nodes, 80% malicious data

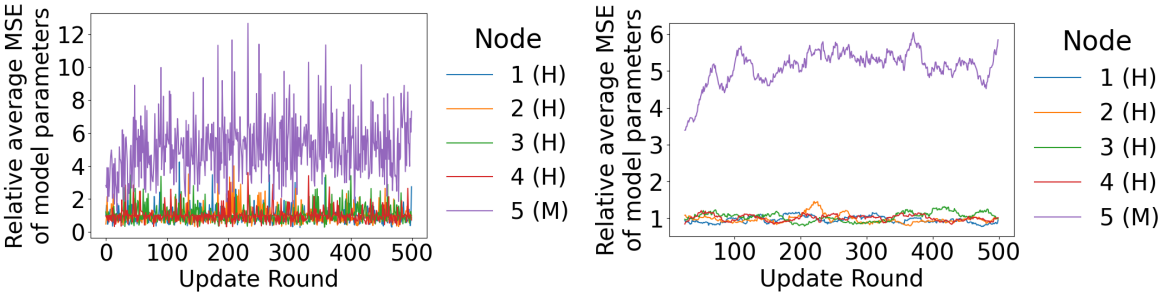
**Figure 5.21:** Sub-figures (a) and (b) show the loss value for 500 update rounds for different malicious scenarios in an transient malicious FL environment (Round 150-300 malicious, all other rounds healthy). M stands for malicious. The FL system uses the SVM model with MNIST data set.

It is observed above that transient maliciousness affects loss during the transient maliciousness. During rounds with a malicious node, the machine learning model learns less strongly as seen e.g. in graph 5.21, which is why the overall performance of the system after a fixed number of 500 rounds is also worse than for a healthy model as already observed in section 5.1

In all of these graphs the moving average of 25 values is used, which is why the transition phase of identifying the node as malicious or healthy is delayed. The advantages and disadvantages of using moving averages are discussed in the next subsection.

### 5.2.3 Moving average justification

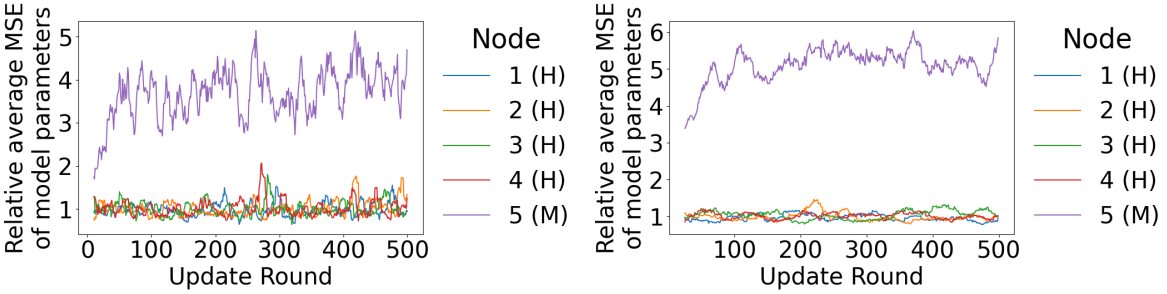
The reason to work with the moving average is that the model parameters are fluctuating. In Graph 5.22 it is compared how a scenario of 20% malicious nodes and 60% malicious data would appear without moving average and with a moving average of 25.




**(a)** Relative average MSE **(b)** Moving average (25) of relative average MSE

**Figure 5.22:** Comparing sub-figure (a) with the relative average Mean Squared Error (raMSE) (no moving average) to sub-figure (b) with the moving average of 25 values of raMSE in malicious scenario of 20% malicious nodes and 60% malicious data. H stands for healthy, M stands for malicious. The FL system uses the SVM model and the MNIST data set.

As explained in the previous chapter, the choice of the size of the moving average influences the stability of results as well as the speed at which changes are detected. This is especially relevant in the case of a transient malicious node. Graph 5.23 shows the difference of a moving average of 10 and 25 for 20% transient malicious nodes with 60% malicious data and a maliciousness time frame of round 150 to 300.



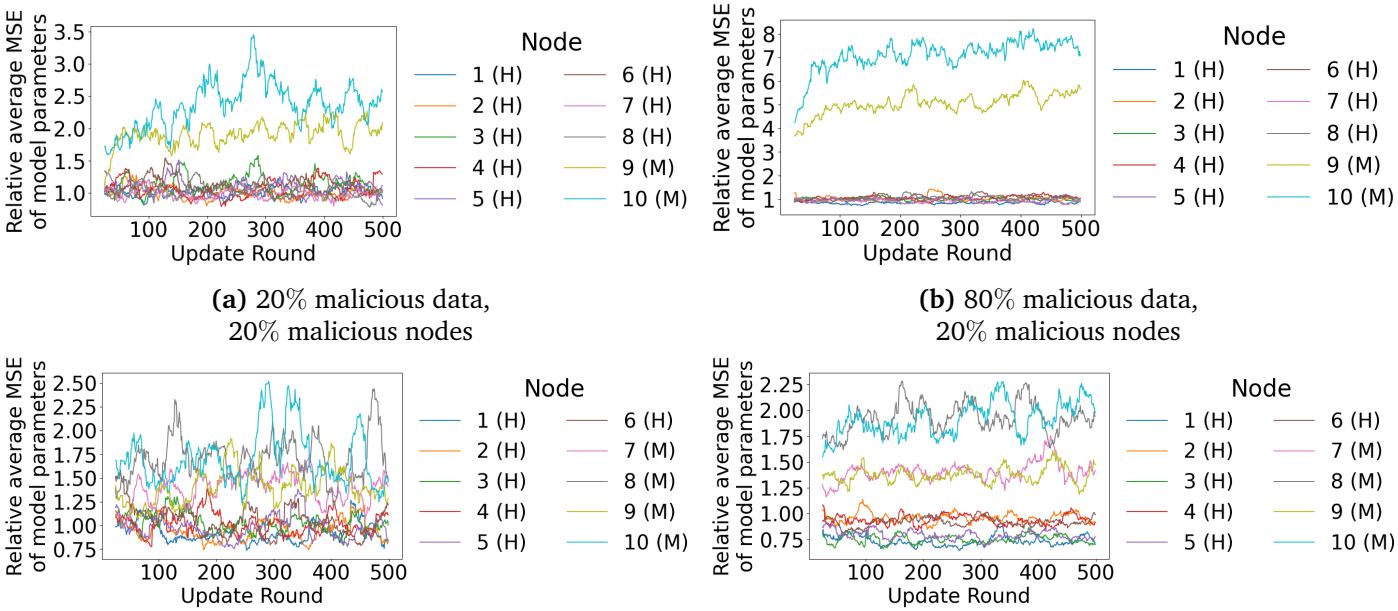

**(a)** Moving average (10) of relative average MSE **(b)** Moving average (25) of relative average MSE

**Figure 5.23:** Comparing sub-figure (a) with the moving average of 10 values of the relative average Mean Squared Error (raMSE) to sub-figure (b) with the moving average of 25 values of raMSE in malicious scenario of 20% malicious nodes and 60 % malicious data. H stands for healthy, M stands for malicious. The FL system uses the SVM model and the MNIST data set.

### 5.2.4 Larger Federated Learning systems

So far all analyses above are made with 5 nodes, but the following graphs confirm that it is possible to use this parameter in a bigger FL environment, e.g. with 10 or 20 nodes.

In graph 5.24, exemplary cases are shown for 10 nodes. All four cases show, that the raMSE works for a network of 10 nodes.




**(a)** 20% malicious data, 20% malicious nodes

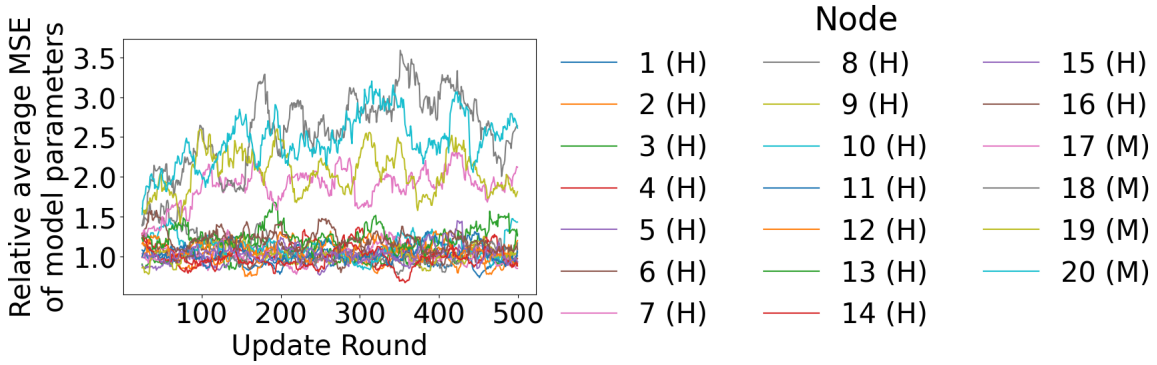
**(b)** 80% malicious data, 20% malicious nodes

**(c)** 20% malicious data, 40% malicious nodes

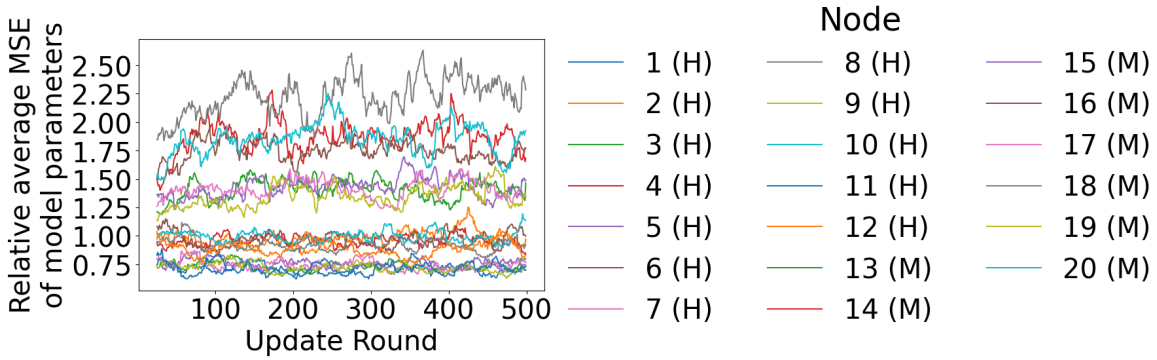
**(d)** 80% malicious data, 40% malicious nodes

**Figure 5.24:** Sub-figures (a), (b), (c) and (d) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 10 different nodes. H stands for healthy, M stands for malicious. The FL system uses the SVM model and the MNIST data set.

Likewise, in graph 5.25 the two exemplary cases of malicious node identification with the raMSE are shown for 20 nodes.




**(a)** 20% malicious data, 20% malicious nodes




**(b)** 80% malicious data, 40% malicious nodes

**Figure 5.25:** Sub-figures (a) and (b) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 20 different nodes. H stands for healthy, M stands for malicious. The FL system uses the SVM model and the MNIST data set.

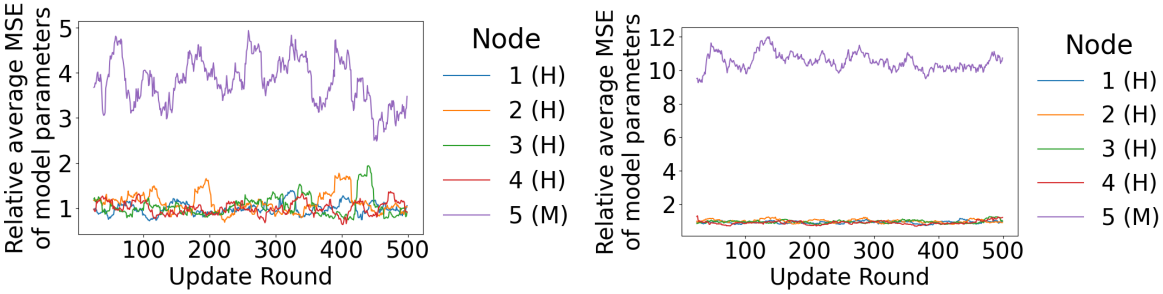
### 5.2.5 Reduction of the number of model parameters considered

As outlined before, computational efficiency plays an essential role in a real world environment [33]. FL systems can contain machine learning models with a large amount of model parameters (in this project: SVM 784 model parameters, CNN 500,000 model parameters). Calculating the raMSE for all these model parameters can be resource intensive in a real world environment. Therefore, this project aims to develop a version of the raMSE that uses fewer model parameters and while still able to detect malicious nodes. More specifically, a raMSE with 1% of the model parameters as well as a raMSE with 10% of the model parameters will be analysed.

In order to successfully accomplish this task, it is required to evaluate which model parameters are relevant for distinguishing a healthy from a malicious node. The analysis shows that the absolute size of the model parameters is decisive for the relevance. Therefore, only the most important 1% or 10% of the model parameters are considered.

Graph 5.26 with two example data distributions for 20% malicious nodes shows that

already 1% of the model parameters is sufficient enough to detect a malicious node.

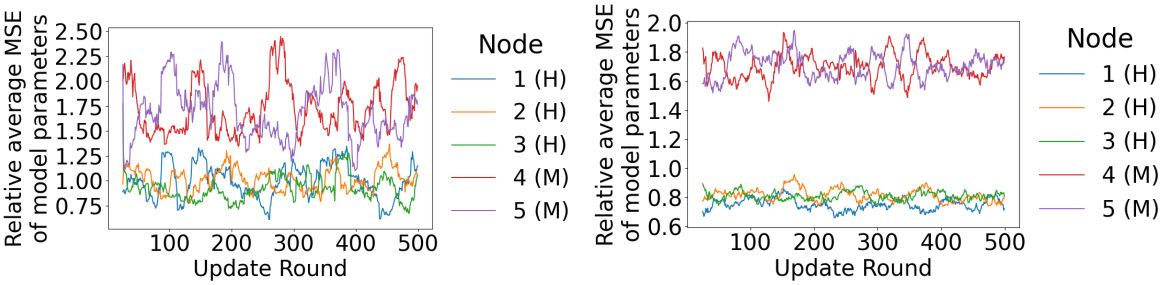



**(a)** 20% malicious data, 20% malicious nodes

**(b)** 80% malicious data, 20% malicious nodes

**Figure 5.26:** Sub-figures (a) and (b) show different data cases of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) with only 1% of the model parameters considered for 5 different nodes in an environment with 20% malicious nodes. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

Furthermore, this works for 40% nodes as shown for the same two sample data distributions in graph 5.27.




**(a)** 20% malicious data, 40% malicious nodes

**(b)** 80% malicious data, 40% malicious nodes

**Figure 5.27:** Sub-figures (a) and (b) show different data cases of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) with only 1% of the model parameters considered for 5 different nodes in an environment with 40% malicious nodes. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

Additionally, the time savings for using only a limited amount of model parameters are analysed, to identify how much time can be saved if only the most important model parameters are measured. Within the statistic detection parameter calculation there are 5 steps: 1. mean calculation, 2. identifying the biggest indices for the limited weight calculation, 3. MSE calculation, 4. raMSE calculation, 5. moving average of the raMSE calculation. The timings for graph 5.28 are from an average of 25 runs and show the calculation for 1 update round with the SVM model.

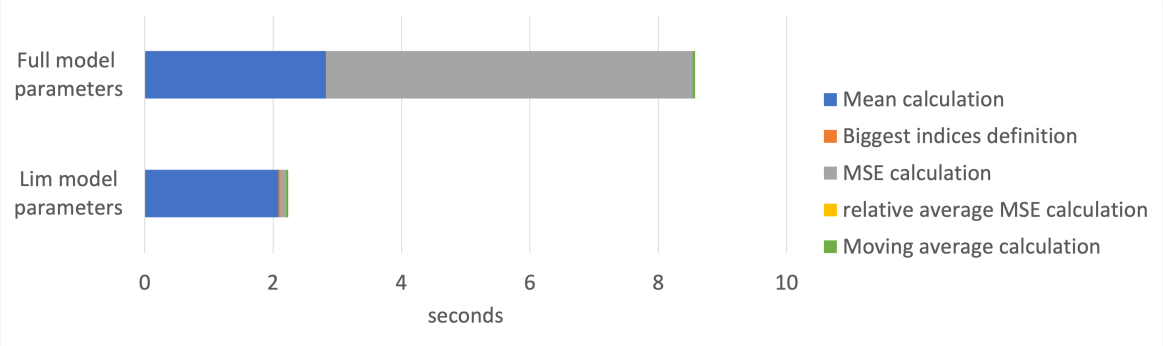



**Figure 5.28:** Timings of a limited model parameters of 1% relative average Mean Squared Error (raMSE) calculation and a full model parameters raMSE calculation in a FL system with SVM model and MNIST data set.

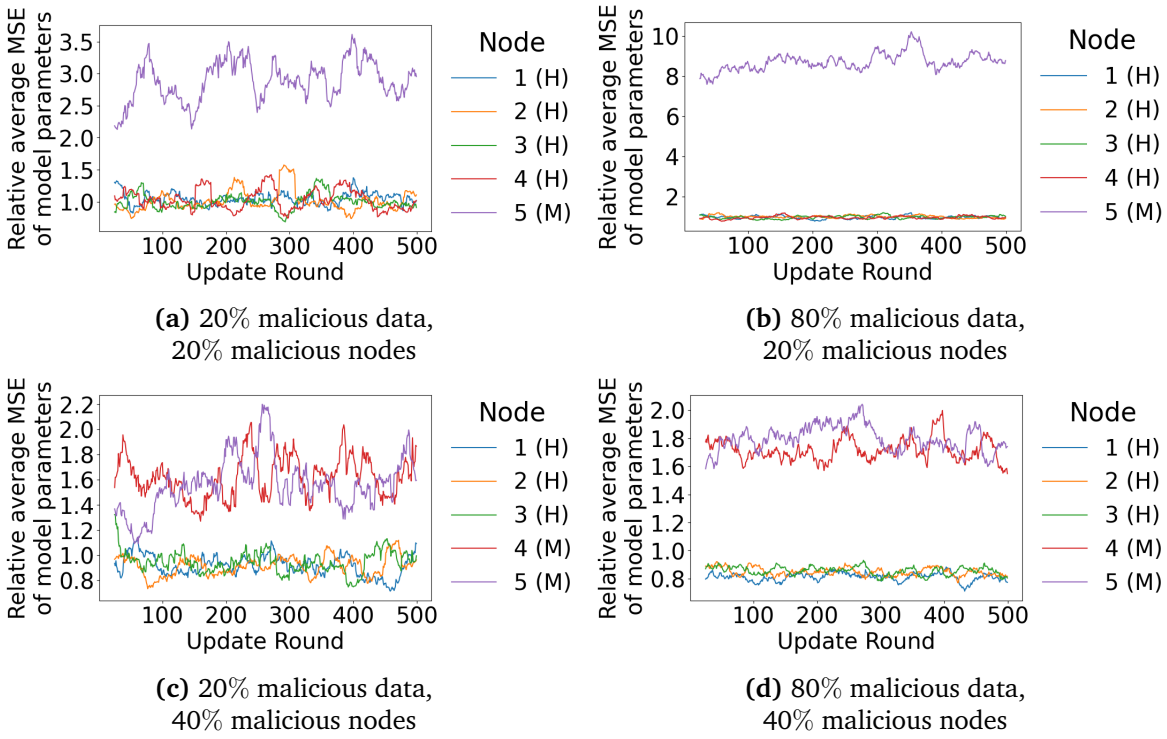
It is noticeable that the major time consumer for the "full model parameters" calculation is the MSE calculation. The time for this calculation is cut down from 5.7 seconds to 0.08 seconds, meaning a time saving of 98.57%. These are the timings for 500 update rounds with the SVM model and the MNIST data set. The SVM model in this scenario has 784 model parameters. The CNN model with the MNIST data set already has 500,000 model parameters. Assuming the timing is proportional to the number of model parameters (which is reasonable because the MSE calculation is looping through each model parameter) this leads to a potential saving of 60 minutes per update round. This calculation is shown in 5.29.

| Time in sec full model parameters | Time in sec limited model parameters | Time savings using limited model parameters in sec | SVM number of model parameters | Time savings in sec per model parameter |
|---|---|---|---|---|
| 5.7103 | 0.0818 | 5.6285 | 784 | 0.0072 |

| CNN Number of model parameter | Time savings using limited model parameters in sec | Time savings using limited model parameters in min |
|---|---|---|
| 500000 | 3589.6046 | 59.8267 |

**Figure 5.29:** Projection of time savings per round using limited model parameters of 1% (instead of all model parameters) in FL system with a CNN model and MNIST data set.

1% is just on potential percentage by which the number of model parameter to be considered could be cut down. It is a relevant value, as it is very low. If it works with only 1% of model parameters as well as with all model parameters shown above, it can be assumed that it will also work with anything in between. To give another example, graph 5.30 shows results for four exemplary cases with 10% model parameters. It is observable, that the differentiation between healthy and malicious nodes is a bit clearer than with 1% of model parameters.




**(a)** 20% malicious data, 20% malicious nodes

**(b)** 80% malicious data, 20% malicious nodes

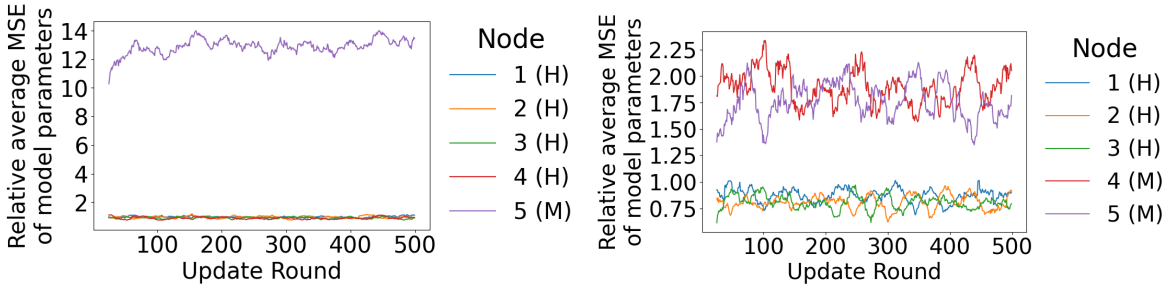
**(c)** 20% malicious data, 40% malicious nodes

**(d)** 80% malicious data, 40% malicious nodes

**Figure 5.30:** Sub-figures (a), (b), (c) and (d) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) with only 10% of the model parameters considered for 5 different nodes. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

### 5.2.6 Invalid malicious data

Another option to create malicious data for the non binary classification problem with the MNIST data set is to insert invalid data, as described in subsection 4.1.1. The results of two exemplary scenarios can be observed in graph 5.31.




**(a)** 20% malicious data, 20% malicious nodes

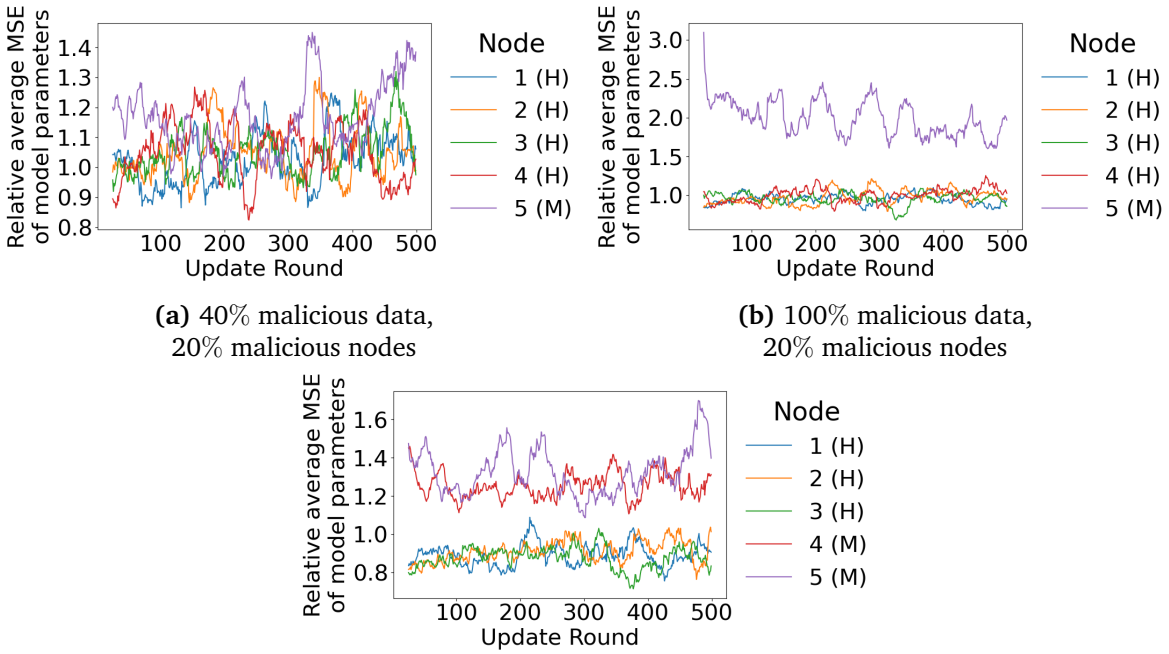
**(b)** 80% malicious data, 40% malicious nodes

**Figure 5.31:** Sub-figures (a) and (b) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) with invalid data (label = 0) as malicious data for 5 different nodes. H stands for healthy, M stands for malicious. The system uses the SVM model and the MNIST data set.

### 5.2.7 MNIST non-binary analysis

The results in the above sections show the results for the binary MNIST classification problem with the SVM model. To verify the results for other models, the MNIST data set is classified with the CNN model. In this case the pictures are labeled from 0 to 9.

In graph 5.32 it is observable that the raMSE is able to identify malicious nodes with 20% as well as 40% malicious nodes if there is a high percentage of malicious data. However, the raMSE is not able to identify a malicious node if only small amount of malicious data is available.



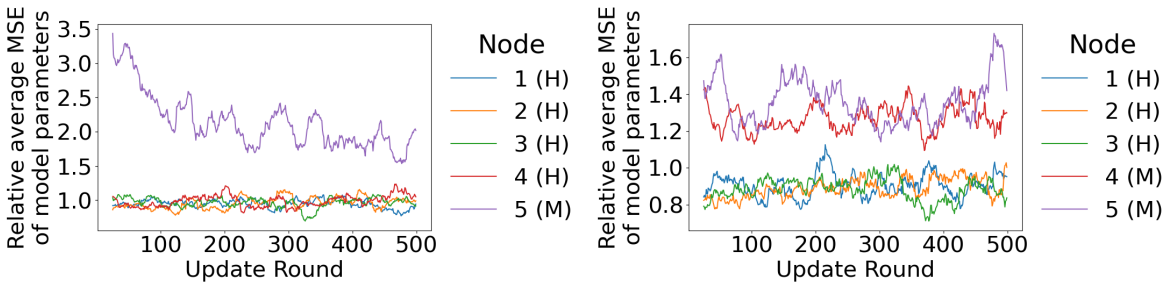

**(a)** 40% malicious data, 20% malicious nodes

**(b)** 100% malicious data, 20% malicious nodes

**(c)** 100% malicious data, 40% malicious nodes

**Figure 5.32:** Sub-figures (a), (b) and (c) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes. H stands for healthy, M stands for malicious. The system uses the CNN model and the MNIST data set.

Additionally, it is noticeable that in the scenarios with sufficient malicious data, the malicious node can also be identified with 1% of the model parameters as seen in graph 5.33.



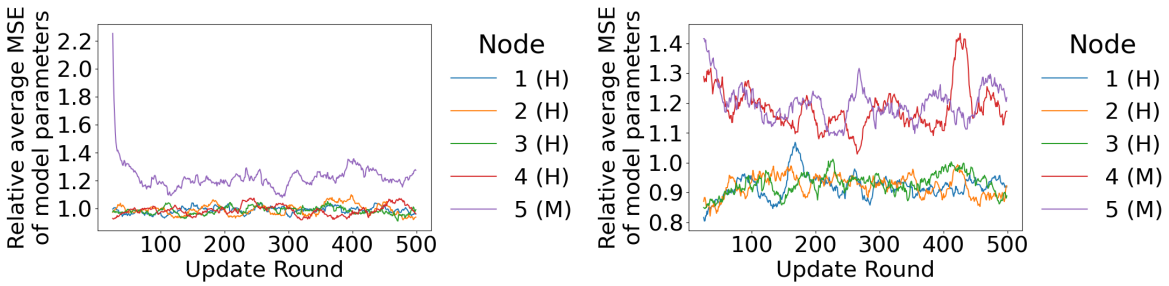

**(a)** 80% malicious data, 20% malicious nodes

**(b)** 80% malicious data, 40% malicious nodes

**Figure 5.33:** Sub-figures (a) and (b) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) with only 1% of the model parameters considered for 5 different nodes. H stands for healthy, M stands for malicious. The system uses the CNN model and the MNIST data set.

### 5.2.8 Cifar-10 analysis

To validate the usage of the raMSE, the parameter is applied to a second data set, the cifar-10 data set. The data set, as described in section 3.1 contains pictures which can be classified into 10 different categories. In graph 5.34 it is observable that the raMSE is capable to detect malicious node in scenarios with a high percentage of malicious data (as seen in graphs 5.34a and 5.34b).




**(a)** 100% malicious data, 20% malicious nodes

**(b)** 100% malicious data, 40% malicious nodes

**Figure 5.34:** Sub-figures (a) and (b) show different malicious scenarios of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes. H stands for healthy, M stands for malicious. The system uses the CNN model and the cifar-10 data set.

## 5.3 Automated detection performance

The automated malicious detection system aims to detect malicious nodes within a running FL environment. As described in section 4.3, the system allows the user to specify a few parameters: 1. threshold of when a node is considered malicious (according to the raMSE) 2. moving average value and 3. number of model parameters considered. In this experiments the threshold of 1.5 is chosen, moving average of 25 and all model parameters are considered.

The automated malicious detection system is tested with the SVM model and the MNIST data set. It is tested for all different scenarios of different percentage of malicious data and malicious nodes. Given the assumptions stated in section 1.3, it only makes sense to apply the system to an environment with fewer malicious than healthy nodes. Therefore the cases with 20% and 40% malicious nodes are analysed with all the different amounts of malicious data (20%, 40%, 60%, 80% and 100%). Furthermore, the results are confirmed in a transient malicious environment (maliciousness from update round 150 - 300).

As seen in the confusion matrix in figure 5.35 the automated malicious detection system is on average able to identify 93% of the malicious nodes and 96% of the healthy nodes correctly. The data are from multiple run throughs, with different amount of malicious nodes and different percentages of malicious data.

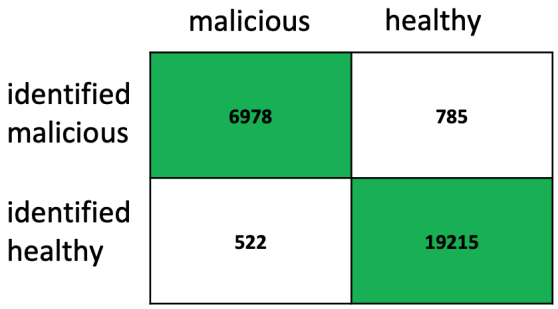



**Figure 5.35:** Confusion matrix for automated malicious detection system. FL system with SVM model and MNIST data set, various data cases, threshold of 1.5, moving average of 25 and all model parameters considered.

To confirm this result, figure 5.36 shows the confusion matrix for a transient malicious case, when nodes are only malicious between round 150 and 300. The figure shows, that also in a transient malicious case the automated malicious detection system is able to differentiate well.

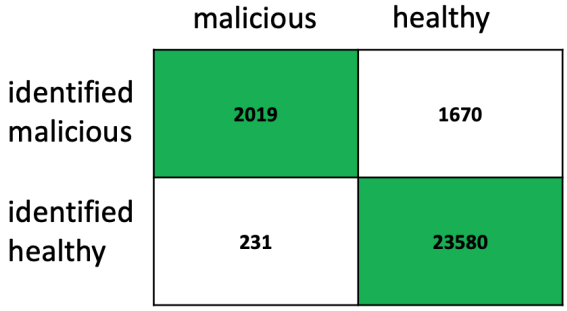



**Figure 5.36:** Confusion matrix for automated malicious detection system in an transient malicious environment (round 150-300 malicious, rest healthy). FL system with SVM model, MNIST data set, various data cases, threshold of 1.5, moving average of 25 and all model parameters considered.

How the automated malicious detection system is able to detect malicious nodes in most scenario can be seen in figure 5.37. In most scenarios (different percentages of malicious nodes and malicious data), the classification of malicious nodes as malicious works well. Only the scenario with 40% malicious nodes and 20% malicious data is less successful.

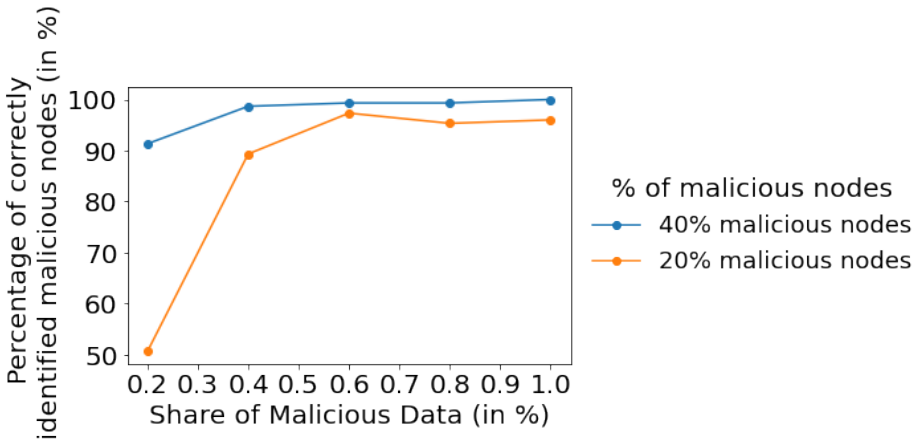



**Figure 5.37:** Percentage of nodes correctly classified malicious in an FL environment with a SVM model, MNIST data set environment with threshold of 1.5, moving average of 25 and all model parameters considered.

On the other hand, it is not only important that malicious nodes get classified as malicious. It is also relevant that the system does not classify healthy nodes wrongly. This could in the long term ruin the system performance more, than protect it. As seen in 5.38 in all of the scenarios, the classification of healthy nodes works well.

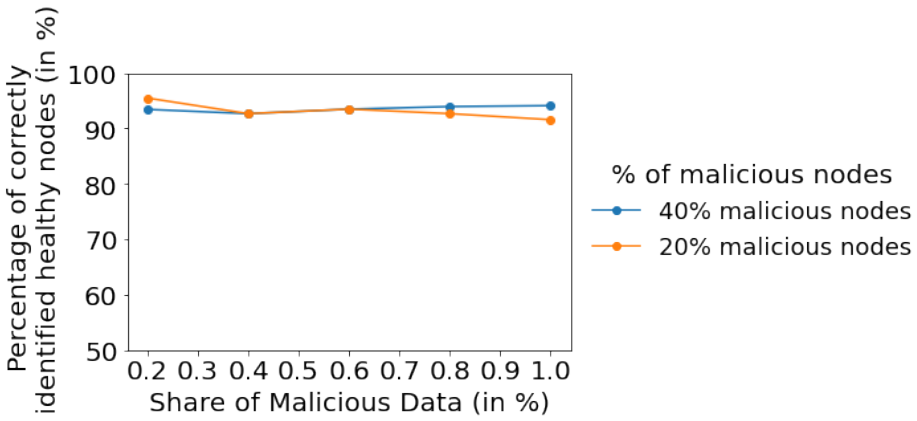



**Figure 5.38:** Percentage of nodes correctly classified healthy in an FL environment with a SVM model, MNIST data set environment with threshold of 1.5, moving average of 25 and all model parameters considered.

Graph 5.39 compares the loss value for the different data scenarios with automated malicious detection system activated and not activated. More specifically, the data distributions outlined above (20%, 40%, 60%, 80% and 100% malicious data) are presented with 20% malicious nodes and 40% malicious nodes. It can be observed, as already presented in section 5.1 that if no automated malicious detection system is activated the loss value increases when the amount of malicious data and the number of malicious nodes increases.

In the case when the detection system is not activated, the loss value stays stable for all scenarios. The line for 20% malicious nodes with activated detection system is hidden behind the line for 40% nodes and activated detection system. The loss value in all those cases is approximately the same value (0.22) as the healthy case.

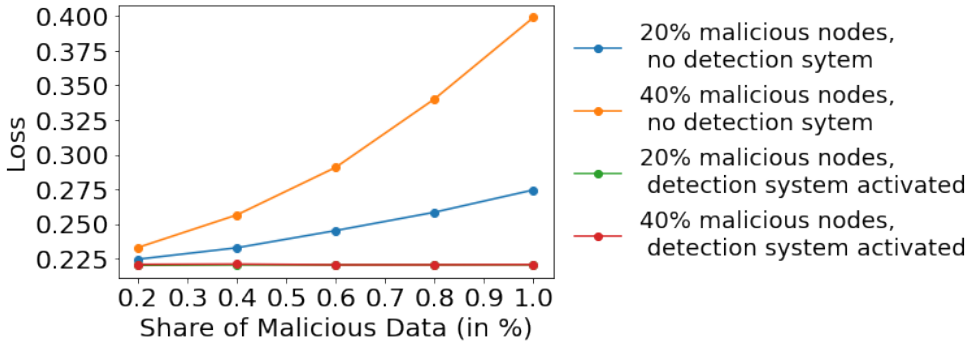



**Figure 5.39:** Loss value for different malicious scenarios with and without automated malicious detection system in an FL environment with a SVM model, MNIST data set environment with threshold of 1.5, moving average of 25 and all model parameters considered.

The same scenarios as in the graph for the loss value (Graph 5.39) are presented for the accuracy in graph 5.40. It can be observed that the accuracy stay stable in all cases when the automated malicious detection system is activated. All of the four lines overlap, as the accuracy of the SVM model and MNIST data set is not significantly reduced by the malicious attack.

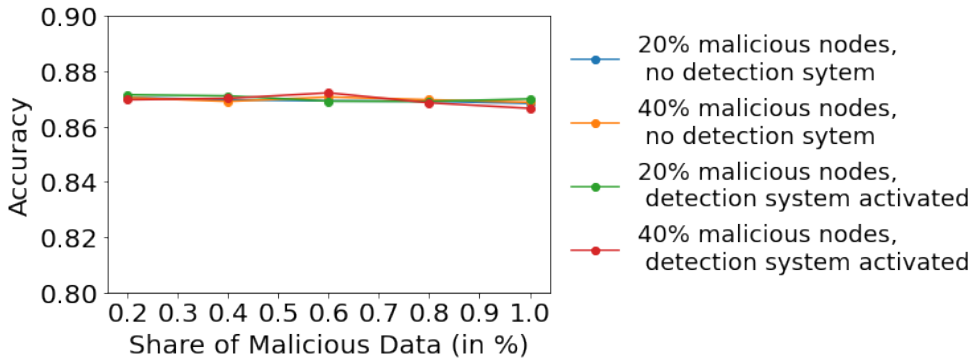



**Figure 5.40:** Accuracy for different malicious scenarios with and without automated malicious detection system in an FL environment with a SVM model, MNIST data set environment with threshold of 1.5, moving average of 25 and all model parameters considered.

# Chapter 6

# Evaluation and discussion

## 6.1 Impact of malicious nodes

This project contributes to the existing literature as it gives a detailed analysis of how different scenarios of malicious nodes influence FL. It is noticeable that there is an observable impact of malicious nodes on the overall system as shown in the graphs in section 5.1.

The strength of the influence depends on the environment setup. In section 5.1 each graph shows the loss and/or accuracy after 500 update rounds for all different cases (various percentage of malicious nodes and percentages of malicious data). However, this same analysis is conducted for multiple environments, varying the model, data set and the time frame of maliciousness. In detail four environment setups are presented: 1. SVM model with MNIST data set, label flipping and no transient maliciousness 2. SVM model with MNIST data set, inserting invalid data and no transient maliciousness 3. SVM model with MNIST data set, label flipping and transient maliciousness and 4. CNN model with MNIST data set label flipping (random value insertion) and no transient maliciousness. It becomes apparent that in all scenarios the maliciousness influences the loss as expected: the more malicious nodes and the more malicious data, the higher the loss value, but the evaluation of accuracy is less clear.

In the standard SVM model with the MNIST data set and a label flipping maliciousness, maliciousness has an affect on loss, but does not a strongly affect on accuracy (graph 5.4). A potential reason for this could be that the binary MNIST data set is only a simple problem and therefore malicious nodes might not have a big impact on the accuracy. This should be further explored.

If the type of maliciousness is set to an invalid data point (e.g. setting the labels to 0), the effect of the maliciousness on the model is stronger than for label flipping, suggesting that this type of maliciousness has a bigger effect on the overall system in the case of the MNIST data set with the SVM model. This could potentially be explained by the fact that label flipping still keeps the overall structure of the labels,

while an invalid label completely messes up the distribution of model parameter. These are just two examples of types of maliciousness. Extending this analysis for different types of maliciousness in a structured way could provide further explanations and will be discussed in future work.

Another point to make is that transient maliciousness has a less strong effect on the overall system than non-transient maliciousness. This proves a first intuition, since in a transient case the time when the system is affected by the maliciousness is smaller.

Lastly, it is observed, that in the non-binary classification problem of the MNIST data set (with the CNN model) the effect on FL is stronger. This is reasonable, as the non binary problem is a more complicated classification problem and therefore malicious nodes might have a bigger impact on the environment, loss and accuracy.

In summary, all of these findings add value to existing literature, as the intensity of different malicious cases has not been explored in such detail so far.

## 6.2 Malicious nodes identification

The statistical malicious detection parameter developed in the course of this project, the raMSE, is able to identify a malicious node in FL environments in most of the scenarios presented in the chapter 5.

It is especially striking to analyse different cases and scenarios. First of all, the raMSE works well on the SVM model for the binary classification problem of the MNIST data set as shown for example in graph 5.11. Furthermore, it is relevant to observe that the mean raMSE for 20% malicious nodes increases the higher the percentage of malicious data. However, the mean raMSE for 40% malicious nodes stays constant and is lower than the raMSE for 20% malicious nodes.

The raMSE is also able to uniquely identify a transient malicious node only during the time when the node is actually malicious as seen, for example, in graph 5.21. It becomes apparent that the maliciousness directly affects the loss value and can increase the loss of the overall system systematically as seen in graph 5.21.

Furthermore, analysing the graphs in subsection 4.2.4, shows that only a small percentage of model parameters is sufficient to identify a malicious user in the system. This is particular compelling, as it enables using this method in a resource constrained environment and saves up to 60 min for example while using the CNN model (as described in subsection 4.2.4). It is confirming to observe that taking only a small percentage of model parameters enables to identify the malicious nodes in different environments, with the CNN model as well as the SVM model.

As already discussed in section 6.1 it can be observed that invalid data has a stronger effect on the overall FL than label flipping. This can originate from the fact, that the model parameters of the malicious node is more different when using invalid data than when using label flipping. This leads to the point, that a malicious node can be better detected in this case as shown in the graphs in subsection 5.2.6.

For the non-binary classification problem of the MNIST data with the CNN data set, the parameter works very well for cases where a higher percentage of malicious data is present (see graph 5.32), but is not able to differentiate malicious from healthy nodes in cases with a lower percentage of malicious data. One potential explanation for this is that with 500,000 model parameter as in the CNN model the differences are less pronounced and need more malicious data to appear. One might assume that a small percentage of malicious data, say 20 %, is less likely in reality than, say, 80 % or 100 % of malicious data. If a malicious client wants to have an impact on the system, it must have a large difference in the model parameter updates it sends to overrule the healthy clients [18].

One potential challenge that becomes apparent throughout all these analyses is that the threshold varies depending on the maliciousness scenario and external environment parameters like model and data set. For the SVM model, most cases are above a threshold of 1.5, which is why this is used as the threshold during the test phase of the detection tool. However, this can not be a fixed value for an automated tool. Thus, the idea of an automatic adjustable threshold will be discussed further in the future work chapter.

In summary, the raMSE works well to identify malicious nodes under the given assumptions. In section 7.3 these assumptions as well as possible extensions are discussed further. In the context of the literature, this parameter adds value as it gives a new perspective on how to identify malicious nodes in a FL environment. Especially intriguing is the efficiency advantage by only using a small amount of model parameters. As pointed out in the literature, many proposed approaches to identify malicious nodes are computationally prohibitive in a real world environment [18]. This project was able to overcome this problem.

## 6.3 Automated detection

The detection tool results show that the system is able to filter out the malicious nodes in 93% of the cases. Furthermore it has improved the loss in average by 17,46% (for a FL system with SVM model and MNIST data set).

This implies that the raMSE is a good value to detect malicious nodes in a FL environment.

It should be especially noted, that the accuracy and the loss value for a malicious FL environment with activated automated malicious detection tool are on the same

level as an healthy FL environment. This holds true across all scenarios (different percentages of malicious nodes and malicious data as well as different time frames of maliciousness). This implies that the automated malicious detection system is successfully able to restore the healthy environment, when a malicious actor tries to infiltrate the system.

However, it has to be noted, that the system also classifies healthy nodes malicious from time to time as seen in figures 5.35 and 5.36 in section 5.3. Excluding them in the update round could potentially harm the systems performance. Future work should consider how to prevent the risk of wrongly classified nodes.

Also, the decision for the threshold is crucial for the success of the automated malicious detection system. E.g., a threshold of 2.5 was not able to identify malicious nodes in every scenario. This is why it is relevant that the threshold is determined automatically, as already mentioned and further elaborated on in the future work section.

It would be ideal to test the automated malicious detection system also with other models and data sets. Given the fact, that the automated malicious detection system shows very similar results to the graph analysis in the second part of this project, it is reasonable to take the assumption that the detection tool will work similarly well on those environments. However, it would be up to future work to test this hypothesis.

So far the automated malicious detection system operates on a per round basis to filter for malicious nodes, but keeps all nodes in consideration for the rounds thereafter. However, the system could also be used in other ways.

A potential different option is to fully kick a node out after it has been identified as malicious for a couple of rounds. This would enable the assurance of long term security of the system. However, it has the downside that a potential wrongly identified node has no chance to be part of the system again.

Another potential of the automated malicious detection system could be a reward system to incentivise nodes to behave normally, not maliciously. While it will not stop malicious actors, it might for example incentivise the nodes' authorities to check the integrity of their system. Furthermore, this could also enhance the problem of free riders in a system. This approach could then be compared to the approach proposed by Tan et al. [4].

In summary, this projects adds on existing literature by providing an anomaly detection system [4]. A next step is to compare this anomaly detection system to others, as for example the spectral anomaly detection framework [18].

Furthermore, as stated before this project only uses a mean aggregation algorithm to focus fully on the anomaly detection part. This gives the system the advantage, that

it can be applied to any aggregation method and thus be applied to any FL system, regardless of the design. In a next step, this project's results could be combined with a robust aggregation approach to create a new hyper approach. According to Tan et al. this enables the exploration of the full potential to protect FL [4].

## 6.4 Assumptions and limitations

The results are only valid under a few assumptions as stated in section 1.3. Firstly, the raMSE to identify malicious nodes can only be applied on IID data. However, this is often not the case in a real world FL environment. Therefore, addressing this limitation is a big part of future work proposals.

Another assumption is that the number of malicious nodes must be smaller than the number of healthy nodes. This assumption is acceptable because in a real environment it is unlikely that there are more malicious nodes than healthy nodes. However, to consolidate the results, it would be an interesting point to lift this limitation in future work.

# Chapter 7

# Conclusion

## 7.1 Ethical considerations

First of all, it is indisputable that machine learning models with their intensive computing power are not good for the environment. Therefore, during the project it is ensured that resource intensive processes are only run to confirm the results once and that it is tested on lighter machine learning models beforehand.

Secondly, the analysis on how malicious nodes can be identified could be used by malicious actors. However, it is the firm belief that the good of providing the analysis exceeds the bad influence it may have.

Furthermore, to not to harm any existing FL environment, all examples are tested in a isolated test environment.

## 7.2 Summary of achievements

The project has achieved its intended goal of implementing a testbed to gain new insights into how malicious nodes influence the behaviour of FL. The detailed analysis enabled to compare the influence of different malicious scenarios on the overall model accuracy and loss.

Furthermore, a technique to identify malicious nodes has been proposed for FL. Specifically, experimental results have revealed that a malicious node can be identified within different systems and scenarios and be differentiated from healthy nodes. Since the central aggregating node for FL receives all parameters of the global model from each participating node, the relative average Mean Squared Error – relative to the medians - of all model parameters for each node can be computed (raMSE). Large deviation of raMSE over successive update rounds for a given node from others nodes can reveal that node as malicious. With a new tool that artificially creates the malicious nodes on the testbed, the statistical effectiveness of the proposed detection parameter has been tested and confirmed for various environments with different degrees of malicious behaviours. Special considerations have been made

to make the malicious detection resource efficient, to overcome computational limitations [18].

Lastly, this project has developed an automated detection system that can identify and exclude a malicious node from the ongoing process of FL. With a success rate of $\geq$93% the system can detect malicious nodes in a FL process, in both non-transient and transient maliciousness scenarios. Furthermore, the automated malicious detection system is able to lower the global model's loss to the level of the loss values of a healthy FL system. Thus, the detection system can restore a healthy environment by filtering out malicious nodes. This represents a novel contribution to anomaly detection algorithms in the current literature [4].

Additionally, the project has developed a testbed with various built-in tools, which can be used by other researchers to investigate security issues and malicious behaviours for FL in the future.

## 7.3 Future work

As already mentioned above, future research issues for this work have been identified, which can be classified into two areas: 1. extend the applicability of the proposed malicious detection mechanism to other FL scenarios, and 2. enhance the automated malicious detection technique to remove malicious nodes from the learning process.

### 7.3.1 Extending analysis

Firstly, the proposed parameter and method to detect malicious behaviors are suitable mainly for FL with independent and identical distributed (IID) data. However, in many practical scenarios, the system often has non-IID data. Therefore, a next step would be to explore how the proposed parameter and technique could be adapted and enhanced so that it could also be applied to non-IID data.

Another limitation of the proposed method in this project is that it can only classify two groups of nodes, but it would not be efficient in identifying malicious nodes correctly if the environment has more malicious than healthy nodes. Therefore, it would be desirable to enhance the statistical detection method for environments where nodes are predominantly malicious.

One potential way of achieving this could be to measure the "health status" for each node. However, this may hard to achieve because it cannot guarantee that the individual nodes are healthy. Otherwise, combining the raMSE with other parameters to identify maliciousness could be explored.

Another research issue for future consideration is to explore other types of maliciousness. This project has focused on data poisoning, as a subclass of model poisoning

[3] and within the data poisoning area, this work focuses on untargeted attacks. The research can be further expanded to targeted label changing attacks. Furthermore, other model poisoning attacks, as well as backdoor attacks can also be analysed. The tools, which have developed during the project, can provide a useful testbed environment for such investigation.

Lastly, this work can be extended to other models and data sets to validate the results. Currently, the models and data sets used are related to image classification. The analysis can, for example, be extended to the sentimental analysis or language models.

### 7.3.2 Automated malicious detection tool

The automated detection tool has currently been tested only with the SVM model and the MNIST data set. The results from this test can be expanded with other machine-learning models and data sets.

The tool is currently based on statistical analysis. However, it is crucial that the detection threshold is properly set to the right value. Therefore, an interesting next step can be to use machine learning to identify such threshold for the optimal effectiveness for the given environment of model and data set.

This idea can be expanded even further by using machine learning to identify the right value for the moving average, which is applied to the raMSE to stabilise the results. This will help to maximise the success of a reliable detection of malicious node depending on the fluctuation of the model parameters.

To achieve the two steps above, the machine learning model can be trained using the testbed built during this project. The system knows which nodes are malicious and is therefore suited to train the model by a supervised-learning technique.

As mentioned before, one idea is to use the automated malicious detection system not only as a tool to protect the global model, but as a tool to incentivise nodes to be healthy or detect free riders in FL. This is also a research topic for further investigation.

# Bibliography


1. FortuneBusinessInsights. Machine Learning Market Size, Share, Growth & Trends [2029]. 2022. Available from: https://www.fortunebusinessinsights.com/machine-learning-market-102226 [Accessed on: 2022 May 31]
2. Goddard M. The EU General Data Protection Regulation (GDPR): European Regulation that has a Global Impact. International Journal of Market Research 2017 Nov; 59. Publisher: SAGE Publications:703–5. DOI: 10.2501/IJMR-2017-050. Available from: https://doi.org/10.2501/IJMR-2017-050 [Accessed on: 2022 May 30]
3. Yang Q, Liu Y, Cheng Y, Kang Y, Chen T, and Yu H. Federated Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning 2019 Dec; 13. Publisher: Morgan & Claypool Publishers:1–207. DOI: 10.2200/S00960ED2V01Y2019-10AIM043. Available from: https://www.morganclaypool.com/doi/10.2200/S00960ED2V01Y201910AIM043 [Accessed on: 2022 Aug 28]
4. Tan J, Liang YC, Luong NC, and Niyato D. Toward Smart Security Enhancement of Federated Learning Networks. IEEE Network 2021 Jan; 35. Conference Name: IEEE Network:340–7. DOI: 10.1109/MNET.011.2000379
5. Lim WYB, Luong NC, Hoang DT, Jiao Y, Liang YC, Yang Q, Niyato D, and Miao C. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. IEEE Communications Surveys Tutorials 2020; 22. Conference Name: IEEE Communications Surveys Tutorials:2031–63. DOI: 10.1109/COMST.2020.2986024
6. Ananthanarayanan G, Bahl P, Bodik P, Chintalapudi K, Philipose M, Ravindranath L, and Sinha S. Real-Time Video Analytics: The Killer App for Edge Computing. en. Computer 2017; 50:58–67. DOI: 10.1109/MC.2017.3641638. Available from: http://ieeexplore.ieee.org/document/8057318/ [Accessed on: 2022 Jun 1]
7. McMahan B and Ramage D. Federated Learning: Collaborative Machine Learning without Centralized Training Data. en. 2017. Available from: http://ai.googleblog.com/2017/04/federated-learning-collaborative.html [Accessed on: 2022 May 30]
8. McMahan HB, Moore E, Ramage D, Hampson S, and Arcas BAy. Communication-Efficient Learning of Deep Networks from Decentralized Data. Tech. rep. arXiv: 1602.05629. arXiv:1602.05629 [cs] type: article. arXiv, 2017 Feb. Available from: http://arxiv.org/abs/1602.05629 [Accessed on: 2022 May 30]

9. Schneble W and Thamilarasu G. Attack Detection Using Federated Learning in Medical Cyber-Physical Systems. en. *28th International conference on computer communications and networks (icccn)*. 2019 :1–8

10. Lu Y, Huang X, Zhang K, Maharjan S, and Zhang Y. Blockchain Empowered Asynchronous Federated Learning for Secure Data Sharing in Internet of Vehicles. IEEE Transactions on Vehicular Technology 2020 Apr; 69. Conference Name: IEEE Transactions on Vehicular Technology:4298–311. DOI: 10.1109/TVT.2020.2973651

11. Mothukuri V, Parizi RM, Pouriyeh S, Huang Y, Dehghantanha A, and Srivastava G. A survey on security and privacy of federated learning. en. Future Generation Computer Systems 2021 Feb; 115:619–40. DOI: 10.1016/j.future.2020.10.007. Available from: https://www.sciencedirect.com/science/article/pii/S0167739X20329848 [Accessed on: 2022 May 28]

12. Tankard C. Big data security. en. Network Security 2012 Jul; 2012:5–8. DOI: 10.1016/S1353-4858(12)70063-6. Available from: https://www.sciencedirect.com/science/article/pii/S1353485812700636 [Accessed on: 2022 May 30]

13. Barreno M, Nelson B, Joseph AD, and Tygar JD. The security of machine learning. en. Machine Learning 2010 Nov; 81:121–48. DOI: 10.1007/s10994-010-5188-5. Available from: http://link.springer.com/10.1007/s10994-010-5188-5 [Accessed on: 2022 May 30]

14. Joseph AD, Laskov P, Roli F, Tygar JD, and Nelson B. Machine Learning Methods for Computer Security (Dagstuhl Perspectives Workshop 12371). Dagstuhl Manifestos 2013; 3. Ed. by Joseph AD, Laskov P, Roli F, Tygar JD, and Nelson B. Place: Dagstuhl, Germany Publisher: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik:1–30. DOI: 10.4230/DagMan.3.1.1. Available from: http://drops.dagstuhl.de/opus/volltexte/2013/4356 [Accessed on: 2022 May 30]

15. Muñoz-González L, Biggio B, Demontis A, Paudice A, Wongrassamee V, Lupu EC, and Roli F. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. New York, NY, USA: Association for Computing Machinery, 2017 Nov :27–38. Available from: https://doi.org/10.1145/3128572.3140451 [Accessed on: 2022 May 30]

16. Fung C, Yoon CJM, and Beschastnikh I. The Limitations of Federated Learning in Sybil Settings. en. *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. 2020 :301–16

17. Bagdasaryan E, Veit A, Hua Y, Estrin D, and Shmatikov V. How To Backdoor Federated Learning. en. *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. ISSN: 2640-3498. PMLR, 2020 Jun :2938–48. Available from: https://proceedings.mlr.press/v108/bagdasaryan20a.html [Accessed on: 2022 May 24]

18. Li S, Cheng Y, Wang W, Liu Y, and Chen T. Learning to Detect Malicious Clients for Robust Federated Learning. Tech. rep. arXiv:2002.00211. arXiv:2002.00211 [cs, stat] type: article. arXiv, 2020 Feb. DOI: 10.48550/arXiv.2002.00211. Available from: http://arxiv.org/abs/2002.00211 [Accessed on: 2022 Aug 24]

19. Fang M, Cao X, Jia J, and Gong NZ. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. en. *29th USENIX Security Symposium (USENIX Security 20)*:1605–22

20. Zhou C, Sun Y, Wang D, and Gao Q. Fed-Fi: Federated Learning Malicious Model Detection Method Based on Feature Importance. en. Security and Communication Networks 2022 May; 2022. Publisher: Hindawi:e7268347. DOI: 10.1155/2022/7268347. Available from: https://www.hindawi.com/journals/scn/2022/7268347/ [Accessed on: 2022 Aug 25]

21. Blanchard P, El Mhamdi EM, Guerraoui R, and Stainer J. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. Available from: https://proceedings.neurips.cc/paper/2017/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html [Accessed on: 2022 Aug 24]

22. Chen Y, Su L, and Xu J. Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent. Tech. rep. arXiv: 1705.05491. arXiv, 2017 Oct. DOI: 10.48550/arXiv.1705.05491. Available from: http://arxiv.org/abs/1705.05491 [Accessed on: 2022 Aug 24]

23. Yin D, Chen Y, Ramchandran K, and Bartlett P. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. Tech. rep. arXiv: 1803.01498. arXiv:1803.01498 [cs, stat] type: article. arXiv, 2021 Feb. DOI: 10.48550/arXiv.1803.01498. Available from: http://arxiv.org/abs/1803.01498 [Accessed on: 2022 Aug 24]

24. Muñoz-González L, Co KT, and Lupu EC. Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging. Tech. rep. arXiv: 1909.05125. arXiv:1909.05125 [cs, stat] type: article. arXiv, 2019 Sep. DOI: 10.48550/arXiv.1909.05125. Available from: http://arxiv.org/abs/1909.05125 [Accessed on: 2022 Aug 24]

25. Lecun Y, Bottou L, Bengio Y, and Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE 1998 Nov; 86. Conference Name: Proceedings of the IEEE:2278–324. DOI: 10.1109/5.726791

26. Brownlee J. Plot-of-a-Subset-of-Images-from-the-MNIST-Dataset.png 1.280×960 Pixel. 2019. Available from: https://machinelearningmastery.com/wp-content/uploads/2019/02/Plot-of-a-Subset-of-Images-from-the-MNIST-Dataset.png [Accessed on: 2022 Aug 21]

27. Krizhevsky A. Learning Multiple Layers of Features from Tiny Images. en. 2009 :60

28. Brownlee J. Plot-of-a-Subset-of-Images-from-the-CIFAR-10-Dataset.png 1.280×960 Pixel. 2019. Available from: https://machinelearningmastery.com/wp-content/uploads/2019/01/Plot-of-a-Subset-of-Images-from-the-CIFAR-10-Dataset.png [Accessed on: 2022 Aug 21]

29. Cortes C and Vapnik V. Support-vector networks. en. Machine Learning 1995 Sep; 20:273–97. DOI: 10.1007/BF00994018. Available from: http://link.springer.com/10.1007/BF00994018 [Accessed on: 2022 Aug 28]

30. LeCun Y, Boser B, Denker J, Henderson D, Howard R, Hubbard W, and Jackel L. Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*. Vol. 2. Morgan-Kaufmann, 1989. Available from: https://proceedings.neurips.cc/paper/1989/hash/53c3bce66e43be4f209556518c2fcb54-Abstract.html [Accessed on: 2022 Aug 28]

31. Wang S, Tuor T, Salonidis T, Leung KK, Makaya C, He T, and Chan K. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. IEEE Journal on Selected Areas in Communications 2019 Jun; 37. Conference Name: IEEE Journal on Selected Areas in Communications:1205–21. DOI: 10.1109/JSAC.2019.2904348

32. MachineLearningPaperspace. Weights and Biases. 2021. Available from: https://machine-learning.paperspace.com/wiki/weights-and-biases [Accessed on: 2022 Aug 18]

33. Wang S, Tuor T, Salonidis T, Leung KK, Makaya C, He T, and Chan K. When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018 Apr :63–71. DOI: 10.1109/INFOCOM.2018.8486403

34. MachineLearningPaperspace. Accuracy and Loss. 2020. Available from: https://machine-learning.paperspace.com/wiki/accuracy-and-loss [Accessed on: 2022 Aug 19]

# Appendix A

# How to run the code

## A.1 Overall repository information

This repository is part of a MSc Computing Individual Project by Anne-Sophie Hannes. It build upon cornerstones of the "Adaptive Federated Learning in Resource Constrained Edge Computing Systems" code repository from Wang et al. and added functionality by changing it into an malicious FL environment analysis tool [31].

All code from *analyser.py*, *all-cases-analyser.py*, *detectionTool.py* is original. Almost all parts of the code in *serverAH.py*, *clientAH.py* and *config.py* are original, however small parts have been reused from the original repository.

This repository includes source code for the paper S. Wang et al. [31]. More specifically: all code in the folders *util-reused-code*, *models-reused-code* and *data-reader-reused-code*, as well as the function calls for those functions in the respective folders and the remote procedure calls within *serverAH.py* and *clientAH.py*.

## A.2 Code structure

'config.py' is the file where all configurations to run this repositories can be made. A more detailed instruction is provided in 'config.py'.

The folder 'analysis-results' contains the results, once a code ran through. It is stored depending on the setup in config file, but the default is storing it under the date.

Currently, the supported data sets are MNIST and CIFAR-10, and the supported models are SVM and CNN. The code can be extended to support other data sets and models too.

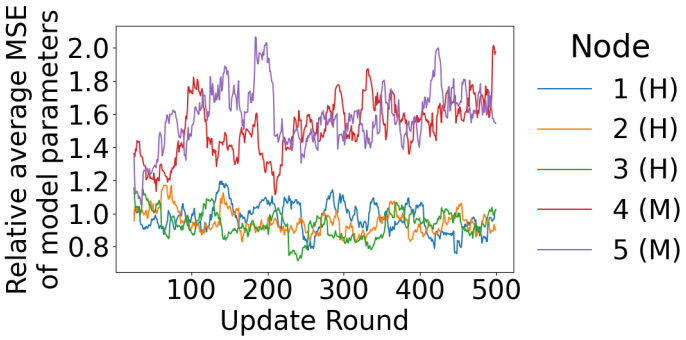
## A.3 Getting started

This repository requires Python 3 with Tensorflow version 1 ($(>= 1.13)$). If you have already a suitable environment install the dependencies by running: *pip3 install -r requirements.txt*.

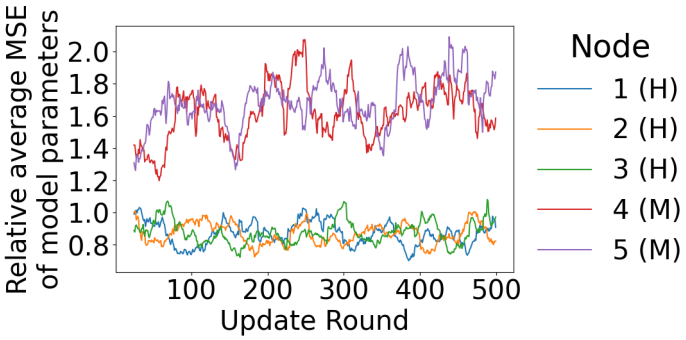
1. Otherwise, install the environment using Anaconda:
    - Download anaconda
    - Use the following command: ‘conda create –name env-project python=3.6.13 tensorflow=1.15.0 matplotlib=3.3.4 numpy=1.19.2‘
    - Activate enviornment: ‘Conda activate evn-project‘
    - To deactivate: ‘Conda deactivate‘
2. Download the ‘datasets‘ and put them into the dataset folder:
    - MNIST dataset, download from ¡http://yann.lecun.com/exdb/mnist/¿ and put the standalone files into ‘datasets/mnist‘.
    - For CIFAR-10 dataset, download the ‘CIFAR-10 binary version (suitable for C programs)‘from ¡https://www.cs.toronto.edu/ kriz/cifar.html¿, extract the standalone ‘*.bin‘ files and put them into ‘datasets/cifar-10-batches-bin‘.
3. Test the code:
    - Select all the settings in ‘config.py‘
    - Run ‘serverAH.py‘ and wait until you see ‘Waiting for incoming connections...‘ in the console output.
    - Run as many parallel instances of ‘clientAH.py‘ as selected in terminals
    - The terminals will show prints of what is happening in the background
    - The ‘analysis-results‘ folder will have all the results

# Appendix B

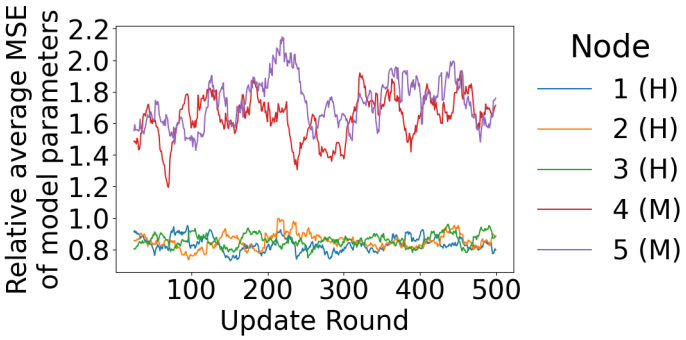
# Graphs



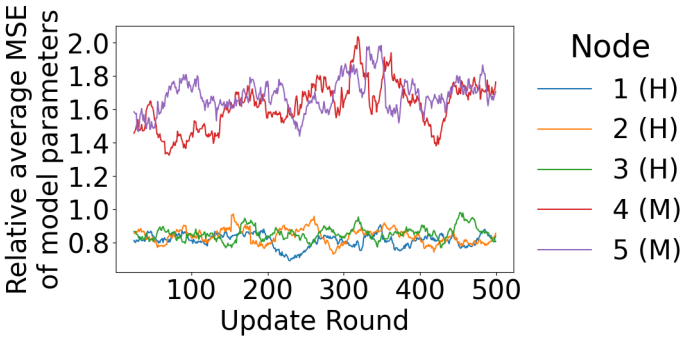

**(a)** 20% malicious data, 40% malicious nodes



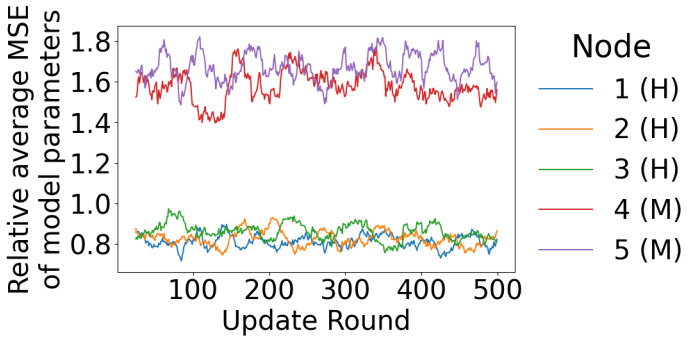

**(b)** 40% malicious data, 40% malicious nodes




**(c)** 60% malicious data, 40% malicious nodes




**(d)** 80% malicious data, 40% malicious nodes




**(e)** 100% malicious data, 40% malicious nodes

**Figure B.1:** Sub-figures (a), (b), (c), (d) and (e) show the data cases of the relative average Mean Squared Error (raMSE) (plotted as the moving average of 25 values) for 5 different nodes in a FL system with 40% malicious nodes. The system uses the SVM model and the MNIST data set