

Securing Federated Learning Model Aggregation Against Poisoning Attacks via  
Credit-Based Client Selection

by

Mohammadreza Khorramfar

A Dissertation Submitted in Partial Fulfillment of the  
Requirements for the Degree of

Master of Science

in the Department of Applied Computer Science

Securing Federated Learning Model Aggregation Against Poisoning Attacks via  
Credit-Based Client Selection

by

Mohammadreza Khorramfar

Supervisors :

---

Dr. Talal Halabi,  
(Department of Applied Computer Science)

---

Dr. Yaser Al Mtawa,  
(Department of Applied Computer Science)

## ABSTRACT

Federated Learning (FL) has emerged as a revolutionary paradigm in the field of machine learning, enabling multiple participants to collaboratively train models without compromising the privacy of their individual training data. However, the distributed and decentralized nature of FL also exposes it to a diverse array of poisoning attacks, wherein adversaries inject malicious updates to compromise the integrity and accuracy of the global model.

In this thesis, we embark on a critical exploration of defense strategies against poisoning attacks in FL. Our primary focus lies in proposing and evaluating a robust defense mechanism, aptly named Credit-Based Client Selection (CBCS). Leveraging a credit-based system, CBCS judiciously assigns credit scores to participating clients based on the accuracy and consistency of their historical model updates. By selectively incorporating reliable clients with higher credit scores into the model aggregation process, while subjecting low-credit clients to thorough scrutiny or exclusion, CBCS fortifies the defense against adversarial disruptions.

To further enhance our research comprehensiveness, we extend our evaluation to other scenarios that can be explored, such as normal conditions. We carefully assess the efficacy of these strategies across various FL settings.

Through an extensive series of experiments conducted on non-iid image classification datasets, we rigorously evaluate the performance of the CBCS defense mechanism. The results show that CBCS effectively identifies and excludes adversarial clients, maintaining model accuracy and data confidentiality in federated learning. The outcomes of our research underscore the profound impact of robust defense strategies on securing federated learning and their pivotal role in advancing collaborative and privacy-preserving machine learning applications. The proposed CBCS defense mechanism illuminates new avenues for enhancing the resilience and security of federated learning systems in the face of adversarial threats. As the world continues to embrace decentralized and privacy-focused learning approaches, our research contributes significantly to the safe and trustworthy deployment of federated learning across diverse domains.

# Contents

<b>Supervisory Committee</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Table of Contents</b>	<b>iv</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>ix</b>
<b>Acknowledgements</b>	<b>xi</b>
<b>Dedication</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Definition . . . . .	4
1.2 Proposed Approach . . . . .	5
1.3 Contributions . . . . .	5
1.4 Thesis Layout . . . . .	6
<b>2 Background Concepts</b>	<b>7</b>
2.1 Deep Neural Networks(DNNs) . . . . .	7
2.2 Federated Learning . . . . .	8
2.3 Privacy Concerns . . . . .	10
2.4 Aggregation Rules . . . . .	11
2.4.1 Federated Averaging . . . . .	12
2.4.2 Krum . . . . .	13
2.4.3 Median . . . . .	14
2.4.4 Trimmed-Mean . . . . .	15
2.4.5 Advantages of FedAvg . . . . .	15

2.4.6	Disadvantages of Robust Aggregation Algorithms . . . . .	17
<b>3</b>	<b>The Proposed Defence Model</b>	<b>18</b>
3.1	Adversarial Attacks in Federated Learning . . . . .	18
3.1.1	Data Poisoning Attacks . . . . .	19
3.1.2	Model Poisoning Attacks . . . . .	20
3.2	Defence Methods in Federated Learning . . . . .	23
3.2.1	Background Knowledge . . . . .	23
3.2.2	Proposed Defence Approach: Credit-Based Client Selection . .	25
<b>4</b>	<b>Datasets</b>	<b>34</b>
4.1	Mnist . . . . .	34
4.2	Fashion Mnist (FMnist) . . . . .	37
4.3	Cifar-10 . . . . .	39
4.4	Data Preprocessing . . . . .	42
4.4.1	Preprocessing Mnist . . . . .	43
4.4.2	Preprocessing FMnist . . . . .	43
4.4.3	Preprocessing Cifar-10 . . . . .	44
<b>5</b>	<b>Experiments, Results, and Discussion</b>	<b>46</b>
5.1	Federated Learning Parameters and Settings . . . . .	46
5.2	Model Artitucture . . . . .	47
5.3	Experiments . . . . .	49
5.4	Evaluation Metrics . . . . .	50
5.5	Evaluation of the Defense Mechanisms . . . . .	51
5.5.1	Experiments with FedAvg Algorithm without Credit-Based Client Selection . . . . .	52
5.5.2	Experiments with FedAvg Algorithm with Credit-Based Client Selection . . . . .	53
5.5.3	Experiments with FedAvg Algorithm under our Attack Model without Credit-Based Client Selection . . . . .	56
5.5.4	Experiments with FedAvg Algorithm and our Attack Model with Credit-Based Client Selection . . . . .	61
<b>6</b>	<b>Conclusion and Future Work</b>	<b>68</b>
<b>A</b>	<b>Selected Code Snapshots</b>	<b>70</b>

A.1	Attack Model . . . . .	70
A.2	Calculation of Credit-Based Client Selection . . . . .	71
A.2.1	Softmax Function . . . . .	72
	<b>Bibliography</b>	<b>74</b>

# List of Tables

Table 5.1 Federated Learning parameter values for all datasets . . . . .	47
Table 5.2 The CNN architecture for Mnist and FMnist . . . . .	48
Table 5.3 The CNN architecture for Cifar_10 . . . . .	48
Table 5.4 Mnist global model accuracy with FedAvg algorithm . . . . .	52
Table 5.5 FMnist global model accuracy with FedAvg algorithm . . . . .	53
Table 5.6 Cifar-10 global model accuracy with FedAvg algorithm . . . . .	53
Table 5.7 Cifar-10 global model accuracy with VGG-1 with FedAvg algorithm	54
Table 5.8 Mnist global model accuracy with FedAvg algorithm with credit- based client selection . . . . .	55
Table 5.9 FMnist global model accuracy with FedAvg algorithm with credit- based client selection . . . . .	55
Table 5.10 Cifar-10 global model accuracy with FedAvg algorithm with credit- based client selection . . . . .	56
Table 5.11 Testing accuracy of Mnist with our attack model ( $\omega = 0.3$ ) in %	58
Table 5.12 Testing accuracy of Mnist with our attack model ( $\omega = 10$ ) in %	58
Table 5.13 Testing accuracy of FMnist with our attack model ( $\omega = 0.3$ ) in %	58
Table 5.14 Testing accuracy of FMnist with our attack model ( $\omega = 10$ ) in %	59
Table 5.15 Testing accuracy of Cifar-10 with our attack model ( $\omega = 0.3$ ) in %	59
Table 5.16 Testing accuracy of Cifar-10 with our attack model ( $\omega = 10$ ) in %	60
Table 5.17 Mnist global model accuracy under attack ( $\omega = 10$ ) with credit- based client selection ( $\zeta = 0.1$ ) . . . . .	63
Table 5.18 Mnist global model accuracy under attack ( $\omega = 10$ ) with credit- based client selection ( $\zeta = 10$ ) . . . . .	63
Table 5.19 FMnist global model accuracy under attack ( $\omega = 10$ ) with credit- based client selection ( $\zeta = 0.1$ ) . . . . .	64
Table 5.20 FMnist global model accuracy under attack ( $\omega = 10$ ) with credit- based client selection ( $\zeta = 10$ ) . . . . .	64

Table 5.21 Cifar-10 global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 0.1$ ) . . . . .	65
Table 5.22 Cifar-10 global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 10$ ) . . . . .	65



# List of Figures

Figure 1.1	Next word predictions in Gboard. Based on the context “I love you”, the keyboard predicts “and”, “too”, and “so much” [25] .	2
Figure 1.2	Federated learning working process [50] . . . . .	2
Figure 1.3	Vulnerable Federated Learning system [50] . . . . .	3
Figure 2.1	Comparison between the classical centralized ML approach and centralized or P2P FL [22] . . . . .	10
Figure 2.2	Federated Learning process and environment [46] . . . . .	12
Figure 3.1	An overview of a single round of federated learning with an adversarial client [30] . . . . .	24
Figure 3.2	Client selection process in Federated Learning by involving credit scores . . . . .	26
Figure 3.3	Federated Learning with our defensive strategy . . . . .	27
Figure 4.1	Example of the Mnist database [4] . . . . .	35
Figure 4.2	Class names and example images in FMnist database [66] . . .	38
Figure 4.3	Visualization of the CIFAR 10 Train Dataset in the Deep Lake UI [2] . . . . .	40
Figure 4.4	Visualization of data preprocessing steps [21] . . . . .	42
Figure 5.1	Testing accuracy of Mnist, FMnist and Cifar-10 . . . . .	53
Figure 5.2	CNN and VGG-1 for Cifar-10 . . . . .	54
Figure 5.3	Testing accuracy of Mnist, FMnist and Cifar-10 with credit-based client selection . . . . .	55
Figure 5.4	Testing accuracy of Mnist with and without credit-based client selection . . . . .	56
Figure 5.5	Testing accuracy of FMnist with and without credit-based client selection . . . . .	57

Figure 5.6 Testing accuracy of Cifar-10 with and without credit-based client selection . . . . . 57

Figure 5.7 Testing accuracy of Mnist under attack ( $\omega = 0.3$  and  $\omega = 10$ ) . 59

Figure 5.8 Testing accuracy of FMnist under attack ( $\omega = 0.3$  and  $\omega = 10$ ) 60

Figure 5.9 Testing accuracy of Cifar-10 under attack ( $\omega = 0.3$  and  $\omega = 10$ ) 61

Figure 5.10 Testing accuracy of Mnist, Mnist with CBCS, Mnist under attack 61

Figure 5.11 Testing accuracy of FMnist, FMnist with CBCS, FMnist under attack . . . . . 62

Figure 5.12 Testing accuracy of Cifar-10, Cifar-10 with CBCS, Cifar-10 under attack . . . . . 62

Figure 5.13 Comparison of accuracy of Mnist under attack ( $\omega = 10$ ) using our credit-based client selection defence ( $\zeta = 0.1$  and  $\zeta = 10$ ) . 64

Figure 5.14 Comparison of accuracy of FMnist under attack ( $\omega = 10$ ) using our credit-based client selection defence ( $\zeta = 0.1$  and  $\zeta = 10$ ) . 65

Figure 5.15 Comparison of accuracy of Cifar-10 under attack ( $\omega = 10$ ) using our credit-based client selection defence ( $\zeta = 0.1$  and  $\zeta = 10$ ) . 66

Figure 5.16 Comparison of Mnist with FedAvg, Mnist with CBCS, Mnist with only the attack and Mnist with attack + CBCS ( $\omega = 10$  and  $\zeta = 0.1$ ) . . . . . 66

Figure 5.17 Comparison of FMnist with FedAvg, FMnist with CBCS, FMnist with only the attack and FMnist with attack + CBCS ( $\omega = 10$  and  $\zeta = 0.1$ ) . . . . . 67

Figure 5.18 Comparison of Cifar-10 with FedAvg, Cifar-10 with CBCS, Cifar-10 with only the attack and Cifar-10 with attack + CBCS ( $\omega = 10$  and  $\zeta = 0.1$ ) . . . . . 67

Figure A.1 Python code for the implementation of the attack model . . . . 70

Figure A.2 Python code for the implementation of the attack model (continued) . . . . . 71

Figure A.3 Python code for the implementation of our credit-based client selection . . . . . 71

Figure A.4 Python code for the implementation of our credit-based client selection (continued) . . . . . 72

Figure A.5 Softmax Function . . . . . 73

## ACKNOWLEDGEMENTS

Learning is an unceasing journey, enriched by the guidance and support of many remarkable individuals. With immense gratitude, I extend my heartfelt appreciation to those who have played pivotal roles in shaping my academic path at the University of Winnipeg and beyond.

First and foremost, I express my profound thanks to my co-supervisors Dr. Talal Halabi and Dr. Yaser Al Mtawa for bestowing upon me the invaluable opportunity to pursue my studies under their esteemed guidance. Throughout my research journey, their unwavering support and encouragement have been instrumental in helping me achieve my goals and their mentorship have instilled in me greater confidence in writing this paper.

I am indebted to the esteemed professors in the Department of Applied Computer Science at the University of Winnipeg. Their profound expertise and invaluable instructions during the courses I attended have been truly enlightening. The knowledge and insights I gained from them have been critical to the success of my research.

I extend my gratitude to the Dean of Graduate Studies, Dr. Mavis Reimer, Applied Computer Science Department Chair, Dr. Christopher Henry, and the dedicated staff of Graduate Studies, Ms. Dagmawit Habtemariam, Mr. Dylan Jones, Mr. Eric Benson and Mr. Kent Suss for their assistance and facilitation throughout my academic journey.

To Ms. Connie Arnhold from the ACS department, I extend my appreciation for her valuable contributions to my academic pursuits.

I am immensely grateful to my family, especially my father, mother and sister, whose unwavering encouragement and support have been a constant source of strength during challenging times. Their unwavering faith in me has been a driving force behind my achievements.

To all my friends, whose invaluable assistance and support have been immeasurable, I extend my heartfelt thanks.

Lastly, I express my deep appreciation to everyone who has played a part in assisting and supporting me on this journey, enabling me to reach my academic goals. Your contributions have been instrumental in making this endeavor a reality. Thank you all for being an integral part of my journey of learning and growth.

Mohammadreza Khorramfar

DEDICATION

*To those who lost their lives in the path of freedom.  
To my aunt, may she rest in peace.*

# Chapter 1

## Introduction

Federated Learning (FL) represents a revolutionary approach in the field of machine learning, empowering thousands, or even millions, of participants to collaboratively train deep learning models [32, 43]. At the core of FL lies the idea of a decentralized learning process, where a central server coordinates the joint training of a global model, while individual clients locally train their models using their private data. The power of FL lies in its ability to harness the collective knowledge of diverse clients without compromising data privacy. By aggregating contributions from all clients, the central server constructs the global model, which is a collective representation of the distributed knowledge from the participants.

This decentralized approach brings numerous advantages, especially in terms of computational efficiency. FL achieves significant computational savings on the server side by outsourcing and parallelizing the model training process. Furthermore, data privacy is strictly maintained, as only local model updates are transmitted between clients and the central server, ensuring that raw training data remains confidential and is not shared with the service provider or other clients.

The flexibility and versatility of FL have spurred its wide adoption across various domains and applications. Technology giants like Google [25, 69] have embraced FL as a powerful tool for next-word prediction on Android Gboard as depicted in Figure 1.1, while Apple harnesses its potential to develop highly accurate voice recognition models [23].

These real-world applications showcase the immense promise of FL in enabling collaborative intelligence while safeguarding data ownership and privacy. The working process of federated learning is given in Figure 1.2 [50].

However, despite its potential, FL also faces significant security challenges, par-

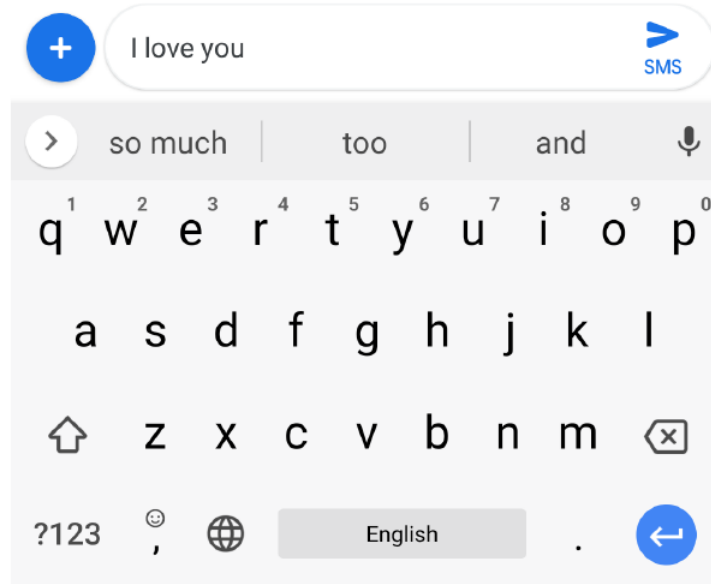


Figure 1.1: Next word predictions in Gboard. Based on the context “I love you”, the keyboard predicts “and”, “too”, and “so much” [25]

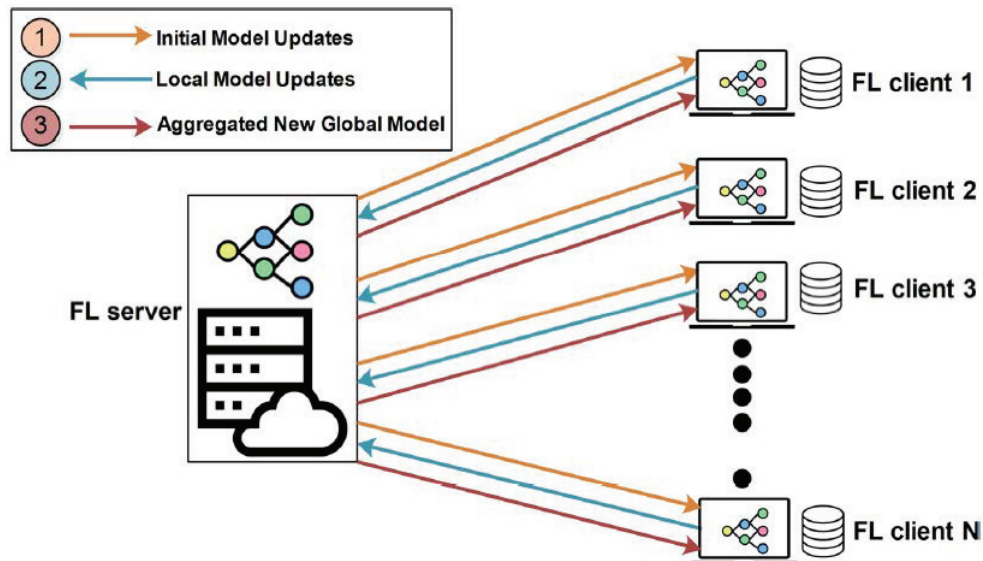


Figure 1.2: Federated learning working process [50]

ticularly in the form of model poisoning attacks as depicted in Figure 1.3 [50]. The distributive nature of FL exposes it to potential threats from malicious clients, which may manifest as either compromised legitimate participants or malicious clients inserted by attackers. These adversaries strategically manipulate model updates during

the training process, intending to compromise the integrity and accuracy of the global model. Consequently, the testing accuracy of the overall system may be compromised due to the influence of these malicious updates [17, 7, 16].

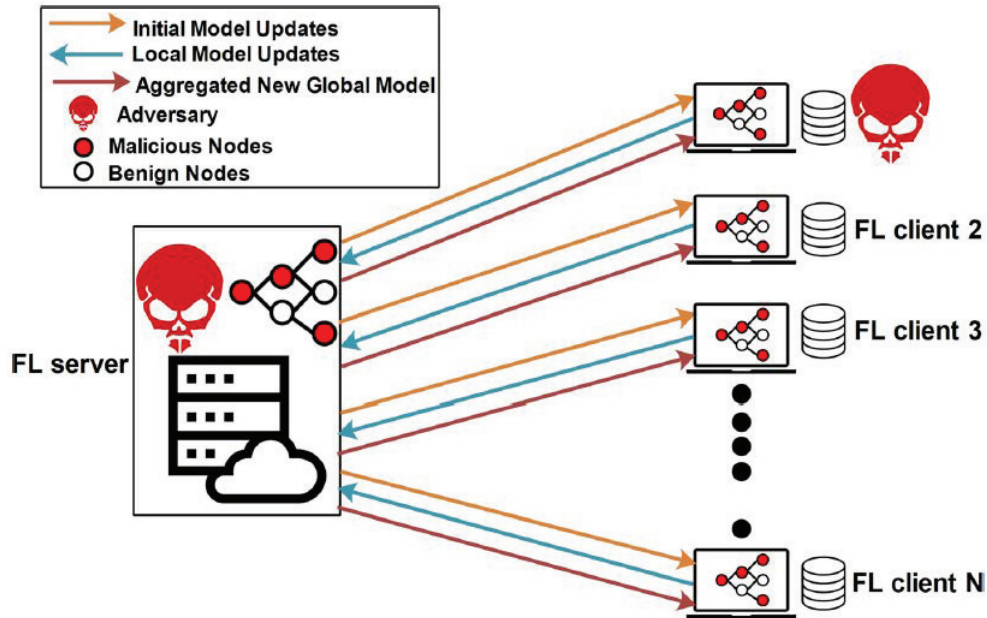


Figure 1.3: Vulnerable Federated Learning system [50]

One of the widely-used algorithms in FL, known as FedAvg [43], is especially used in most applications. FedAvg relies on computing the global model through a weighted average of model parameters contributed by each client. Alarming research findings indicate that if clients are compromised, the resulting global model can be arbitrarily manipulated by adversaries [7].

In response to these pressing security concerns, this research aims to explore a robust defense strategy that effectively safeguard FL systems against adversarial disruptions. By identifying and mitigating the impact of malicious model updates, we aspire to fortify the integrity and accuracy of the global model within the collaborative learning environment. Our proposed defense strategy, the "Credit-Based Client Selection (CBCS)," is designed to identify and select trustworthy clients for model aggregation. The CBCS mechanism leverages a credit-based system, where clients are assigned credit scores based on the accuracy and consistency of their historical model updates. Trustworthy clients with higher credit scores are given more significant contributions during model aggregation, while clients with lower credit scores undergo rigorous scrutiny or may even be excluded from the aggregation process

altogether. Through extensive experiments and evaluations on non-iid image classification datasets, we assess the efficacy of the CBCS defense strategy, along with other scenarios and mechanisms integrated into the aggregation rule(FedAvg) of FL.

Securing FL against model poisoning attacks is of utmost importance to enable privacy-preserving, collaborative machine learning. The research presented here highlights the critical significance of developing robust defense strategies that effectively counteract adversarial disruptions, paving the way for the continued growth and advancement of federated learning in diverse domains.

## 1.1 Problem Definition

Numerous prior studies have proposed Byzantine robust aggregation algorithms with the primary goal of mitigating the impact of model poisoning attacks [3, 7, 11, 16, 67, 70]. These defense mechanisms operate by detecting and eliminating outliers among model updates before incorporating them into the global model. However, a significant drawback of many of these aggregation algorithms is their reliance on the assumption of independent and identically distributed (iid) training data, which does not hold in real-world scenarios where non-iid datasets are prevalent [18, 55]. Recent research has shown that these existing defenses are inadequate in preventing poisoning attacks in Federated Learning (FL), particularly when faced with non-iid datasets. Furthermore, previous defense approaches assume a fixed number of malicious clients participating in the process, leading to the removal of a specific number of model updates in every training round. Additionally, many implementations rely on random client selection as the client selection strategy, potentially leading to the exclusion of a substantial number of benign updates. Consequently, the aggregation algorithms such as Krum or Median, may inadvertently cause a decrease in the global model’s accuracy, even in the absence of attackers.

In this thesis, one of our contributions lies in the development of a generic model poisoning attack against FL. Our attack methodology employs a simple yet effective poisoning technique, generating malicious updates for integration into the aggregation process. We frame this attack as model poisoning attack, drawing parallels with prior research efforts [18, 55, 6]. The results of our experiments indicate that this attack method, used as the first step in our defense design, can significantly reduce the accuracy of models employing random client selections, leading to a decrease in the global model’s testing accuracy by approximately 50%. As previously highlighted,



we encountered an additional challenge linked to the inherent nature of the federated learning algorithm proposed by Google, namely FedAvg. In this algorithm, clients are selected randomly during each round, leading to a query about the algorithm’s credibility. In response, we chose to adopt this algorithm as our base and fortified its performance through the integration of our credit-based client selection mechanism. This strategic augmentation significantly enhanced the outcomes of the algorithm.

## 1.2 Proposed Approach

In addition to exploring the attack aspect, we propose a new defense framework known as Credit-Based Client Selection (CBCS) to effectively counter untargeted model poisoning attacks. Our defense mechanism is designed to identify and select trustworthy clients while excluding potentially malicious or compromised clients from the model aggregation process. The core principle behind CBCS is to assign credit scores to individual clients based on their historical behavior and model update contributions. Clients with higher credit scores are considered more reliable and are given greater influence during the model aggregation, while those with lower credit scores undergo more stringent scrutiny or may be excluded altogether.

Our extensive evaluation demonstrates that our defense achieves only a slight reduction in testing accuracy (less than 3% ) compared to FedAvg under non-adversarial settings, while offering significantly improved robustness. To thoroughly assess the defense’s effectiveness, we conducted evaluations using three datasets, considering various scenarios involving normal poisoning attacks. By comparing the dataset’s performance before and after implementing our defense strategies, the results confirm the efficacy of CBCS in fortifying Federated Learning against model poisoning attacks, thereby enhancing the security and accuracy of collaborative machine learning systems.

## 1.3 Contributions

The main contributions of this work are outlined as follows:

- We evaluate the performance of the FedAvg aggregation rules on three datasets (Mnist, FMnist, and CIFAR-10) under both benign and adversarial threat sce-

narios to emphasize the impact of poisoning attacks in federated learning systems.

- We propose a new defense approach, namely Credit-Based Client Selection (CBCS), to effectively counter untargeted model poisoning attacks in federated learning.
- We conducted a comprehensive evaluation of our defense strategy under model poisoning attacks and demonstrated the effectiveness of our credit-based client selection mechanism using three datasets, namely Mnist, FMnist, and CIFAR-10, under normal and benign conditions.

## 1.4 Thesis Layout

The rest of this thesis organized as follows:

**Chapter 2** provides an overview of important concepts in federated learning.

**Chapter 3** Provides a theoretical framework for data and model poisoning attacks, and most importantly provides extensive details about our proposed defence strategy.

**Chapter 4** explains the three datasets used as a case study in this thesis.

**Chapter 5** gives experiments conducted on the datasets with various scenarios followed by a discussion of the results.

**Chapter 6** concludes the thesis and provides future research directions.

## Chapter 2

# Background Concepts

Federated Learning (FL) is a groundbreaking approach to train deep neural networks (DNNs) across multiple decentralized devices or servers while keeping the data locally on these devices. The core idea behind FL is to enable collaborative learning without the need to share raw data with a central server, thereby preserving user privacy and data security.

In the context of DNNs, the FL process begins with a central server, often referred to as the aggregator, initiating the model training. This central server maintains a global model that serves as the starting point for all participants. These participants can be individual mobile devices, edge servers, or other remote nodes.

### 2.1 Deep Neural Networks(DNNs)

A deep neural network (DNN) is a powerful architecture used in modern machine learning to process complex data and make predictions across various tasks [52]. It is comprised of multiple layers, each responsible for extracting increasingly abstract and meaningful features from the input data. These layers consist of learnable parameters, namely weights and biases, which are optimized during the training process.

In a DNN classifier, the ultimate objective is to map inputs, denoted as  $x \in X \subseteq R^d$ , to their corresponding class labels, represented as  $y \subseteq R^k$ , where  $K$  is the number of distinct classes. This mapping is realized by a function  $f : X \rightarrow \mathbb{Y}$ , which assigns likelihood scores to input data for each class. The output layer of the DNN employs the softmax activation function, which transforms the likelihood scores into a vector of real values within the range  $[0, 1]$ , where the sum of the values equals

1. Consequently, the output can be interpreted as the probabilities of the input belonging to each prediction class, facilitating the classification decision.

To train the DNN classifier effectively, a differentiable loss function  $\ell$  is typically employed to quantify the discrepancy between the model’s predictions and the actual ground-truth labels. The training process aims to minimize this loss function over the model parameter  $w$  with respect to the training dataset  $D$ . Mathematically, the empirical loss on the model parameter  $w$  is defined as  $l(\omega) = \frac{1}{N} \sum_{(x_i, y_i) \in D} L(f(x_i; \omega), y_i)$ , where  $N$  is the number of training samples. The objective is to find the optimal model parameter  $\omega$  that minimizes the empirical loss, thereby maximizing the accuracy of the predictions.

To achieve this optimization, the mini-batch stochastic gradient descent (SGD) algorithm is widely used. During each iteration of the algorithm, a mini-batch of training samples is randomly selected, and the backpropagation algorithm is applied to compute the gradient of the loss function with respect to the model’s weights. This gradient information guides the update of the model weights, nudging them in the direction of the local minimum. By iteratively repeating this process over the entire training dataset, the DNN learns to better generalize from the training data to make accurate predictions on new, unseen data.

The versatility and adaptability of DNNs have enabled them to achieve impressive results in various machine learning tasks, such as image and speech recognition, natural language processing, and recommendation systems. Their ability to automatically learn hierarchical representations from data has been instrumental in pushing the boundaries of artificial intelligence, making DNNs a foundational element in contemporary deep learning research.

## 2.2 Federated Learning

Federated Learning (FL) [32, 43] represents a groundbreaking paradigm in the field of machine learning, offering a distributed approach where multiple data owners collaboratively train a global machine learning model without the need to share their private data [20]. This innovative setting addresses the significant privacy concerns associated with traditional centralized machine learning approaches, where sensitive data is centralized, raising potential security risks.

In the FL framework, we consider  $N$  clients, each possessing its local training dataset, denoted as  $D_i$ . Every client maintains a local machine learning model, and

the model parameters are determined by solving the optimization problem  $\arg \min_{\omega} l(\omega)$  on its specific dataset, using a designated loss function  $L$ . By doing so, each client tailors its model to the intricacies of its local data, ensuring a more personalized and context-aware model.

A central server plays a crucial role in the FL ecosystem, acting as a coordinator to facilitate the collective learning process. This server maintains the global model by aggregating the locally trained models from all participating clients. The global model, which represents the summation of knowledge from diverse data sources, offers a more comprehensive and robust model.

The FL process unfolds iteratively in rounds, allowing for continuous learning and adaptation. In each FL round, the server strategically selects a subset of  $n$  clients from the total  $N$  clients and shares the current global model weights  $\omega_g^t$  with this cohort. The selected clients then perform fine-tuning on  $\omega_g^t$  using their local datasets, iteratively updating their models over a fixed number of epochs  $E$  to obtain new local models  $\omega_k^t$ .

After the fine-tuning process on the selected clients, each of the  $n$  clients computes its model update  $\Delta_k^t = \omega_k^t - \omega_g^t$  and transmits  $\Delta_k^t$  back to the server. At this stage, the server employs an aggregation algorithm, denoted as *fagg*, which combines all client updates to obtain the aggregate update  $\vec{\nabla}_{t_g} = f_{\text{agg}}(\{\vec{\nabla}_{t_k} : k \in [n]\})$ . This aggregate update represents the collective intelligence from the selected clients and constitutes the refined global model.

The updated global model with the new weights  $\omega_g^{t+1} = \omega_g^t + \vec{\nabla}_{t_g}$  is broadcast to a fresh subset of selected clients in the subsequent round, and the process continues iteratively until convergence. The iterative nature of FL allows the global model to benefit from the cumulative knowledge contributed by all participating clients, leading to enhanced performance and generalization.

It is crucial to note that FL can be classified into two primary types: cross-silo and cross-device. In cross-silo FL [26], organizations, such as financial or medical institutions, act as clients, and a trusted third-party entity serves as the global server to coordinate the training process. In contrast, cross-device FL encompasses scenarios with numerous mobile or IoT devices, where only a fraction of clients participate in each training round due to resource constraints and varying availability.

In the context of this thesis, our focus is on studying the vulnerability of FL to poisoning attacks, specifically honing in on the cross-device setting. In such settings, clients are often highly unreliable due to factors like intermittent connectivity,

hardware limitations, or malicious behavior. Consequently, comprehending and mitigating the risks of poisoning attacks are of paramount importance for ensuring the robustness, privacy, and security of the FL framework. By exploring effective defense strategies against poisoning attacks, we aim to bolster the reliability and trustworthiness of FL models in these challenging cross-device scenarios.

## 2.3 Privacy Concerns

Federated Learning (FL) has emerged as a promising paradigm for collaborative machine learning without the need to share raw data. In FL, multiple participants, often referred to as clients, collectively train a global machine learning model while keeping their local data securely on their devices, as depicted in Figure 2.1 [22]. This decentralized approach ensures that sensitive data remains private and is not exposed to a central server or any other third-party entity. FL presents a powerful solution to address privacy concerns associated with traditional centralized machine learning approaches, where aggregating data from various sources may lead to potential data breaches and privacy violations.

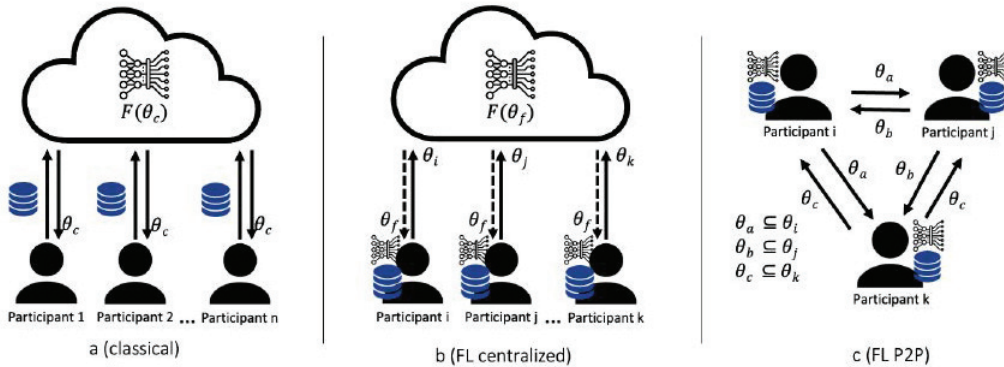


Figure 2.1: Comparison between the classical centralized ML approach and centralized or P2P FL [22]

However, despite its inherent privacy advantages, FL is not immune to privacy challenges and potential threats. Privacy risks can arise in scenarios where adversaries attempt to infer sensitive information about individual users based on the aggregated model updates. These privacy attacks, often referred to as membership inference attacks or model inversion attacks, can exploit the statistical information present in the model updates to determine if a particular client’s data was used during the

training process.

To mitigate such privacy risks, researchers and practitioners have been exploring various privacy-preserving techniques [8] and robust aggregation algorithms in FL. Differential Privacy is one such technique that aims to inject noise into the model updates to prevent the inference of individual data points. By adding carefully calibrated noise to the updates, differential privacy offers a strong guarantee of privacy while still enabling effective global model training.

Additionally, federated learning frameworks can incorporate secure communication protocols, such as homomorphic encryption and secure multi-party computation (SMPC) [63], to further protect sensitive gradients during transmission. These cryptographic methods allow clients to collaborate without explicitly sharing their updates, ensuring that the aggregated model remains secure and privacy-preserving.

## 2.4 Aggregation Rules

In the domain of federated learning, an aggregation algorithm plays a critical role in combining the results obtained from training multiple smart models on individual clients' local data. This algorithm handles the fusion of local client updates and updates the global model accordingly [46]. The typical learning process in federated learning involves the following steps:

1. The central server establishes connections with the clients and distributes the initial global model.
2. Each client receives a copy of the initial model, performs training using its local data, and sends the updated models back to the central server.
3. The central server collects the locally trained models from all clients and aggregates them using the designated algorithm.
4. Based on the aggregation results, the central server updates the global model and sends the updated version back to the respective clients.
5. Lastly, the central server collects the locally trained models from all clients and aggregates them using the designated algorithm.

The above steps are iteratively repeated until the global model converges or until the server decides to terminate the process. The underlying architecture, entities, and steps of this federated learning process are depicted in Figure 2.2 [46].

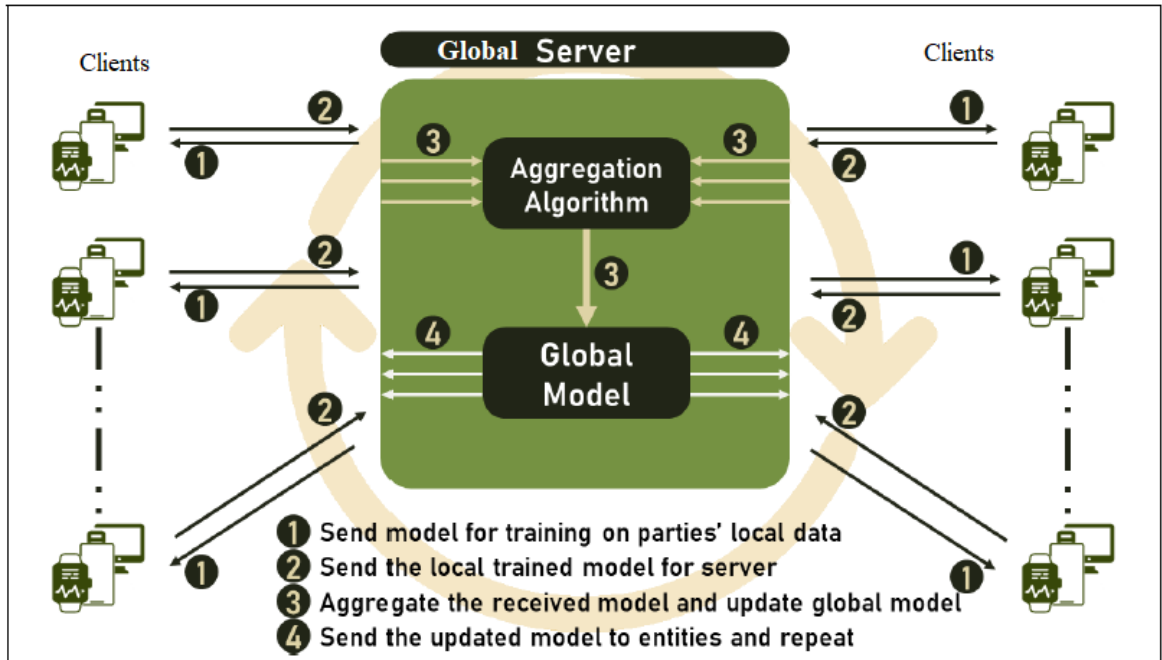


Figure 2.2: Federated Learning process and environment [46]

This iterative process of training, aggregating, and updating enables the central server to learn from the collective knowledge of all participating clients without accessing their raw data directly. It ensures privacy and data security, making FL an appealing approach for collaborative machine learning applications in distributed environments.

### 2.4.1 Federated Averaging

Federated Averaging (FedAvg) [40] is a widely used aggregation algorithm in federated learning, and it offers several key advantages over other algorithms that make it an attractive choice for many scenarios. FedAvg operates in a decentralized manner [60], allowing multiple participants (e.g., devices, edge nodes, or clients) to train their local models using their private data without sharing raw data centrally. Instead, participants only send their model gradients to a central server (e.g., a cloud or coordinator) for aggregation. At the end of this section, we will compare the advantages of FedAvg to other algorithms.



The equation for Federated Averaging (FedAvg) in federated learning involves the aggregation of model gradients from multiple participants. Let's break it down step by step: [45]

1. **Local Model Update:** In the local training phase, each participant (e.g., device or client) performs training on its private data using the current global model. After training, it computes the gradient of the model's loss function with respect to its local parameters. Let's denote the local model's parameters for participant  $i$  as  $\theta_i$ , and the corresponding gradient as  $g_i$ .
2. **Aggregation Phase:** After local training, participants send their gradients to a central server (e.g., a cloud or coordinator) for aggregation. During this phase, the gradients are combined to form the updated global model.
3. **Federated Averaging:** The central server aggregates the gradients received from all participants using a simple averaging process. Given  $N$  participants, the aggregation equation for FedAvg can be represented as:

$$\theta_{\text{new}} = \frac{1}{N} \sum_{i=1}^N g_i \quad (2.1)$$

Here,  $\theta_{\text{new}}$  represents the updated global model parameters after aggregation,  $N$  is the total number of participants, and  $\sum_{i=1}^N g_i$  represents the sum of all gradients  $g_i$  received from the  $N$  participants. The resulting  $\theta_{\text{new}}$  becomes the new global model, which is then sent back to the participants for the next round of training. Federated Averaging ensures that each participant's contribution is equally weighted in updating the global model, making it a fair and privacy-preserving approach for collaborative learning without directly sharing raw data.

### 2.4.2 Krum

Krum, proposed by [7], is an aggregation algorithm utilized in federated learning to select one update among the  $n$  updates that has the lowest score based on the  $\ell_2$  distance. The score of the  $i$ th update is calculated using the following equation:

$$\text{Score}(\nabla_i) = \sum_{\nabla_j \in P} \|\nabla_i - \nabla_j\|_2^2 \quad (2.2)$$

where  $m$  is an estimation of the upper bound on the number of malicious clients, and  $P$  represents the set of the  $n - m - 2$  neighboring updates of  $\nabla_i$ . The underlying rationale behind this approach is to identify malicious updates, which would need to be far from benign updates to have a significant impact in poisoning the global model. Multi-krum [7] represents a notable variant of the Krum aggregation algorithm, distinguished by its significantly higher global model accuracy. The process of Multi-krum involves iteratively selecting updates from the remaining set multiple times, following the same approach as Krum. This iterative selection procedure yields a selection set  $S$ , comprising  $c$  updates, where  $c < n - 2m - 2$ , with  $n$  being the total number of clients and  $m$  being an estimation of the upper bound on the number of malicious clients.

### 2.4.3 Median

Median [70] represents another aggregation algorithm used in federated learning, which performs aggregation along each dimension of the updates. The Median algorithm aggregates the updates by calculating the median of the values along each dimension. Similar to the trimmed-mean aggregation rule, the Median approach also achieves an error rate when the objective function is strongly convex. The Median algorithm is used when dealing with highly non-linear objective functions or scenarios where the data distributions across clients are diverse and non-identically distributed. By taking the median along each dimension, the algorithm can mitigate the impact of outliers and provide a robust global model. Given a set of updates  $\{\nabla_1, \nabla_2, \dots, \nabla_n\}$ , where  $\nabla_i$  represents the update from the  $i$ th client, the Median aggregation along each dimension can be expressed as follows:

$$\text{Median}(d) = \text{median}(\{\nabla_{1d}, \nabla_{2d}, \dots, \nabla_{nd}\}) \quad (2.3)$$

Where  $\text{Median}(d)$  represents the aggregated value along the  $d$ th dimension of the updates and  $\nabla_{id}$  represents the value of the  $d$ th dimension of the update from the  $i$ th client. The Median algorithm computes the median value of each dimension across all client updates, resulting in a new update that represents the aggregated information along each dimension.

#### 2.4.4 Trimmed-Mean

Trimmed-mean [70] is a coordinate-wise aggregation algorithm used in federated learning. For each dimension  $k$ , the trimmed-mean method sorts the values in the  $k$ th dimension of the  $n$  vectors and removes the  $m$  smallest and largest values. After this trimming process, dimensional-wise averaging is performed on the remaining  $n - 2m$  vectors. The intuition behind this approach is to reduce the impact of extreme values or outliers in the aggregation process, thereby enhancing the robustness of the global model.

Dong et al [70] provided theoretical evidence that the trimmed-mean aggregation achieves order-optimal statistical error rates, particularly when dealing with convex loss functions. This property makes trimmed-mean a choice for aggregation in scenarios where the objective function exhibits strong convexity, and it has shown good performance in practice as well. For a given dimension  $k$  and a set of  $n$  vectors  $\{\nabla_{1k}, \nabla_{2k}, \dots, \nabla_{nk}\}$ , the trimmed-mean aggregation is computed as follows:

$$\text{Trimmed-mean}(d_k) = \frac{1}{n - 2m} \sum_{i=m+1}^{n-m} \nabla_{ik} \quad (2.4)$$

where  $n$  is the total number of vectors (updates) in the set,  $m$  is the number of vectors removed from the smallest and largest values in the dimension  $k$  before aggregation, and  $\vec{\nabla}_{ik}$  represents the value of the  $k$ th dimension of the  $i$ th vector in the set.

#### 2.4.5 Advantages of FedAvg

In this chapter, we present an in-depth analysis of aggregation rules used in Federated Learning (FL) experiments, with a primary focus on the widely used Federated Averaging (FedAvg) algorithm. We explore the merits of FedAvg in comparison to other robust aggregation rules, such as Krum, Median, and Trimmed Mean. Our investigation aims to identify the strengths and weaknesses of these aggregation methods in the context of FL, and justify the choice of FedAvg as the algorithm used in our experiments. In federated learning, Federated Averaging (FedAvg) has many advantages [19, 58, 14] over other (Krum, Trimmed mean and median) aggregation methods:

1. **Robustness to Outliers:** FedAvg is more robust to outliers compared to Krum and Median aggregation. In Krum and Median or trimmed mean, a single malicious or faulty participant can significantly impact the final model,

whereas FedAvg mitigates this by averaging gradients from multiple participants, reducing the impact of outliers.

2. **Communication Efficiency:** FedAvg typically requires less communication compared to Krum and Median. In Krum, participants need to exchange information about their gradients, and in Median, they need to transmit their entire model. In contrast, FedAvg only requires exchanging gradients, making it more communication-efficient.
3. **Scalability:** FedAvg is more scalable as the number of participants increases. Krum’s communication overhead grows quadratically with the number of participants, while Median’s communication cost increases linearly. FedAvg’s linear communication cost makes it more suitable for larger federated learning setups.
4. **Ease of Implementation:** FedAvg is relatively straightforward to implement, making it a popular choice in federated learning research and applications. Krum and Median require additional steps to handle outliers and secure communication, which can be more complex to implement.
5. **Privacy-Preserving:** FedAvg provides inherent privacy preservation since participants only share their gradients, not their raw data. This reduces the risk of exposing sensitive information during the aggregation process. On the other hand, Krum and Median may require more data sharing or model parameters, potentially raising privacy concerns.
6. **Lower Computational Overhead:** FedAvg often has lower computational overhead at the participant’s end compared to Krum and Median. In Krum, participants need to compute distances between gradients, and in Median, they need to sort gradients, both of which can be computationally expensive, especially in resource-constrained devices.
7. **Model Convergence:** FedAvg tends to achieve better model convergence in many scenarios. By averaging gradients from multiple participants, it can leverage a larger and more diverse dataset, leading to improved generalization and convergence properties.
8. **Simplicity in Handling Non-IID Data:** FedAvg is known to handle non-IID (non-identically distributed) data across participants better than Krum

and Median. Non-IID data arises when participants have different data distributions, and FedAvg’s averaging mechanism helps balance the impact of such differences.

### 2.4.6 Disadvantages of Robust Aggregation Algorithms

While secure aggregation methods like Krum, trimmed mean or Median can enhance privacy and security in federated learning, they do have many disadvantages [12, 68, 44]:

1. **Increased computation overhead:** Secure aggregation requires additional cryptographic operations, which can lead to higher computational costs and longer training times, especially when dealing with large-scale models and datasets.
2. **Communication overhead:** Secure aggregation methods often involve exchanging encrypted data and cryptographic keys between the participating devices or nodes, leading to increased communication overhead. This can be particularly challenging in low-bandwidth or high-latency environments.
3. **Vulnerability to Byzantine attacks:** While these methods aim to handle malicious participants (Byzantine nodes), they may not be completely robust against sophisticated adversaries who actively manipulate their model updates to mislead the aggregation process.
4. **Limited scalability:** As the number of participating devices or nodes increases, secure aggregation techniques may face scalability issues, impacting the overall performance and convergence of the federated learning process.
5. **Reduced model convergence speed:** secure aggregation methods introduce noise or additional computations during aggregation, which may slow down the convergence rate of the federated learning process compared to traditional aggregation techniques.
6. **Increased model size:** Cryptographic techniques can increase the size of the model updates, leading to larger communication overhead, especially for low-bandwidth or resource-constrained devices.

## Chapter 3

# The Proposed Defence Model

In this chapter, we explore attacks in federated learning, specifically data poisoning and model poisoning attacks. We critically examine the advantages and disadvantages of these attack strategies, emphasizing their potential impact on the global model's integrity and performance. While our primary objective in this paper is to evaluate defense mechanisms in FL, we briefly introduce a simple model poisoning attack to showcase its effect on the system's vulnerability.

Then, we present the Credit-Based Client Selection (CBCS) defense, which utilizes a credit-based system to identify reliable clients for model aggregation. We outline the steps taken to enhance the effectiveness of this defense mechanism. By conducting a rigorous examination of our defense strategy, we aim to contribute valuable insights into the robustness and efficacy of the defense mechanism in FL. Our research endeavor seeks to enhance the understanding of the security landscape in federated learning and foster the development of more resilient and privacy-preserving machine learning systems.

### 3.1 Adversarial Attacks in Federated Learning

Existing research has demonstrated that Federated Learning (FL) is susceptible to various poisoning attacks [18, 6, 55, 47, 28, 62]. These attacks can be broadly categorized into two types based on the adversary's goal: targeted and untargeted attacks. In targeted attacks, the adversary aims to reduce the model's accuracy on specific inputs. Conversely, untargeted attacks are designed to minimize the model's accuracy on any test data, thus undermining the overall performance of the FL system.

A specific subset of targeted attacks is known as backdoor attacks [17] wherein the attacker alters the model’s behavior only on selected inputs known as backdoor triggers while keeping the model’s accuracy intact on the primary FL learning task. Backdoor attacks can be particularly dangerous as they can introduce subtle biases into the model, leading to incorrect predictions on specific inputs without raising suspicion during normal use.

While targeted and backdoor attacks result in the model misclassifying specific input samples, untargeted attacks are intended to degrade the overall utility of both the global model and individual client models. Although existing Byzantine-robust aggregation algorithms offer asymptotic bounds on the error rates of the global model up to a constant factor, such guarantees do not necessarily ensure empirical robustness of the model [12]. For instance, even when robust aggregation algorithms (*RAA*) are employed, a successful untargeted attack can lead to a significant reduction of more than 20% in the global model’s accuracy on datasets like CIFAR-10, making the model unsuitable for practical use [44].

Given the severity of the threat posed by untargeted attacks, this thesis concentrates on exploring defenses against this category of attacks. Understanding and countering untargeted attacks are crucial since they pose a more severe challenge to the practical applicability of FL. Depending on the adversary’s capabilities, two types of FL poisoning attacks can be identified: data poisoning attacks and model poisoning attacks. Data poisoning attacks involve manipulating the data used for training, while model poisoning attacks alter the model’s updates or parameters directly. Addressing both these types of attacks is essential to ensure the security and privacy of FL systems in real-world scenarios.

### 3.1.1 Data Poisoning Attacks

Data poisoning attacks have been extensively studied in the centralized setting of machine learning [47, 28]. In data poisoning attacks, the adversary indirectly manipulates the model updates by injecting poisoning data into the local training datasets of compromised clients. Fang et al. [18] adapted the simple label flipping attack to the federated learning (FL) settings. In this attack, the adversary alters the labels of the local data on compromised clients, replacing them with false labels to introduce malicious behavior into the FL process. A data poisoning attack can be formulated

as an optimization problem as shown in the following equations:

$$\operatorname{argmax}_{D_{k \in [m]}^m \subset D} \|\nabla^b - \text{fagg}(\nabla_{\{k \in [m]\}}^m \cup \nabla_{\{k \in [m+1, n]\}})\|, \quad (3.1)$$

$$\nabla_{k \in [m]}^m = \tau(\omega_g, D_{k \in [m]}^m) - \omega_g, \quad \nabla^b = \text{Avg}(\nabla_{\{k \in [n]\}}) \quad (3.2)$$

In this context,  $\tau$  represents a training algorithm, specifically Stochastic Gradient Descent (SGD), which utilizes the malicious input data  $D_m \subset D$  to fine-tune the current model weight  $\omega_g$  and obtain the new weight. However, Fang et al. [18] demonstrated that using gradient-based optimization to compute  $D_m$  is time-consuming and not very effective.

Shejwalkar et al. [56], on the other hand, proposed an alternative approach for data poisoning attacks. They suggest using label-flipped data to replace the compromised client’s dataset and carefully optimizing the number of data samples based on the server’s aggregation algorithm. This approach aims to improve the efficiency and effectiveness of data poisoning attacks in the federated learning setting. In our study, we observe that data poisoning attacks can be transformed into model poisoning attacks. By training the local model on a poisoned dataset and considering the changes in model weights as malicious model updates, we can effectively carry out model poisoning attacks. Additionally, previous research [18, 6, 55] has indicated that model poisoning attacks tend to be more impactful in various federated learning settings compared to data poisoning attacks. Consequently, our focus in this thesis is on investigating untargeted model poisoning attacks, as they pose a more significant threat in the context of federated learning such as manipulating the model update’s weight, which is what we did in our simple model poisoning attack to demonstrate the effectiveness of our defence approach.

### 3.1.2 Model Poisoning Attacks

In the context of federated learning, the distributed nature of the process introduces a new vulnerability to poisoning attacks compared to the centralized setting. Model poisoning attacks are a type of attack in which the adversary directly manipulates the model updates sent from compromised clients to the server. This class of attacks is particularly concerning as they can have a significant impact on the accuracy of the global model due to the increased capabilities of the attacker. Various model



poisoning attacks have been proposed in the literature [5, 18, 55]. One such attack is the Little Is Enough (LIE) attack [5], which introduces a small perturbation noise to each dimension of the averaged benign updates. To carry out this attack, the adversary requires knowledge of the benign updates. Initially, the attacker computes the dimensional-wise average of the available benign updates  $\nabla^b$  and then adds calibrated noises based on the standard deviation  $\sigma$  of the benign updates. The resulting poisoned update is constructed as  $\nabla^m = \nabla^b + z\sigma$ , a scalar value determined by the number of malicious and total clients. Baruch et al. [5] demonstrated that LIE effectively evades the detection of many robust aggregation algorithms, enabling successful poisoning of the global model.

Fang [18] and Shejwalkar et al. [55] proposed a tailored attack formulation, considering the scenario where the attacker has knowledge of the aggregation algorithm used in the Federated Learning (FL) system. In this attack, the adversary aims to maximize the discrepancy between the final aggregated update and a reference update computed from available benign updates. The tailored attack starts by calculating the average of the available benign updates, which serves as the reference update denoted as  $\nabla^b$ . Next, the attacker determines a malicious perturbation direction, denoted as  $\nabla^p$ , by taking the sign of the reference update  $\nabla^b$ . This direction indicates the adversarial direction in which the updates will be perturbed. The malicious update  $\nabla^m$  is then constructed as a perturbed version of the reference update  $\nabla^b$  in the direction of  $\nabla^p$ , i.e.,  $\nabla^p \leftarrow \text{sign}(\nabla^b)$ . The optimization objective is to maximize the distance between the final aggregated update after aggregation and the reference update  $\nabla^b$ . Mathematically, the tailored attack can be formulated as follows [55]:

$$\underset{\gamma}{\text{argmax}} \|\nabla^b - \text{fagg}(\nabla_{k \in [m]}^m \cup \nabla_{k \in [m+1, n]})\| \quad (3.3)$$

$$\nabla_{k \in [m]}^m = \nabla^b + \gamma \nabla^p, \Delta^b = \text{Avg}(\nabla_{k \in [n]}) \quad (3.4)$$

In the tailored attack, the adversary aims to find an optimal scalar  $\gamma$  that evades the server's aggregation algorithm. This attack assumes that the aggregation algorithm, denoted as  $\text{fagg}$ , is known to the attacker, allowing them to check whether the tailored malicious update  $\nabla^m$  is selected by  $\text{fagg}$ . The objective of the optimization is to maximize the distance between the reference update  $\nabla^m$  and the final update after aggregation.

The tailored attack starts by initializing  $\gamma$  to a large value and gradually de-

creases it until the adversarial objective is achieved. The algorithm uses a line search approach for the optimization, as fagg is typically non-differentiable. The attack computes the reference update  $\nabla^b$  as the dimensional-wise average of available benign updates ( $\vec{\nabla}_{k \in [n]}$ ). Then, it sets a malicious perturbation direction  $\vec{\nabla}^p \leftarrow \text{sign}(\vec{\nabla}^b)$ , where  $\text{sign}\nabla^b$  denotes the element-wise sign function applied to  $\nabla^b$ .

The poisoned update  $\nabla^m$  is constructed as  $\vec{\nabla}_m = \vec{\nabla}_b + \gamma\vec{\nabla}_p$ , where  $\gamma$  is the scalar parameter being optimized. By tuning  $\gamma$ , the attacker can effectively manipulate the aggregated update to achieve their adversarial goal. The optimization process aims to find the optimal  $\gamma$  that maximizes the distance between  $\nabla^b$  and the aggregated update selected by fagg.

Shejwalkar et al. [55] enhance this attack by utilizing more advanced line search techniques and considering alternative choices for the perturbation vectors  $\nabla^p$ , such as a normalized version of  $\nabla^b$ . These improvements make the tailored attack more effective and difficult to detect, posing a significant threat to the security of federated learning systems. In scenarios where the underlying aggregation algorithm is hidden from the adversary, this makes it difficult to compute the result of the aggregation. Shejwalkar et al. [55] proposed two Agg-agnostic attacks known as Min-max and Min-sum. These attacks utilize the same construction of the malicious vector  $\nabla^m$  as the tailored attack described earlier. The main concept behind both attacks is to ensure that the malicious updates lie close to the clique of benign updates. The Min-max attack aims to maximize the scalar  $\gamma$  while ensuring that the maximum distance between the malicious update and any other updates is upper bounded by the maximum distance between any two benign updates. In mathematical terms, it ensures the following 3.5:

$$\max_{i \in [m+1, n]} \|\nabla^m - \nabla^i\|_2 \leq \max_{i, j \in [m+1, n]} \|\nabla^i - \nabla^j\|_2 \quad (3.5)$$

On the other hand, Min-sum attack ensures that the sum of distances between the malicious update and all the benign gradients is upper bounded by the sum of distances between any benign gradient and the other benign gradients. This is defined formally as:

$$\sum_{i \in [m+1, n]} \|\nabla^m - \nabla^i\|_2 \leq \sum_{i, j \in [m+1, n]} \|\nabla^i - \nabla^j\|_2 \quad (3.6)$$

Shejwalkar [55] presents empirical evidence demonstrating that Min-max and Min-sum attacks have a more significant impact on various defenses compared to the LIE

attack.

## 3.2 Defence Methods in Federated Learning

In this chapter, we discuss about defences against poisoning attacks in federated learning, then introduce our own defence strategy and process. Also, for comparison purposes, we briefly explain two other defence strategies that are used in federated learning.

### 3.2.1 Background Knowledge

#### Adversary’s Knowledge

**Attacker’s Ability:** In untargeted model poisoning attacks, the adversary aims to compromise the global model’s accuracy on any test input by crafting malicious updates on a subset of participating clients. We assume that the attacker has control over a fraction,  $P$ , of the total  $N$  clients in the Federated Learning (FL) system [17, 5, 6, 18]. These clients can be either normal clients that are compromised by the attacker or fake clients injected into the FL system. To maintain the effectiveness of used algorithms, we usually constrain the malicious fraction  $q = \frac{P}{N}$  to be less than 50%. The attacker can manipulate the updates sent from the compromised clients to the central server during each FL round, allowing them to inject malicious information into the model. In our case, we injected the malicious updates for the attacker.

**Attacker’s Knowledge:** The attacker possesses basic knowledge about the FL system [10, 18, 55], including access to the local training code, training dataset, and model updates of the compromised clients. Figure 3.1 [30] shows the steps of a single round of the attack. Additionally, in adversarial FL settings, researchers consider two other dimensions of the attacker’s knowledge: knowledge of the aggregation algorithm and knowledge of other benign model updates.

The knowledge of the aggregation algorithm is a relatively strong assumption for the adversary, as it enables them to tailor attacks to exploit vulnerabilities in the aggregation process. Similarly, having knowledge of other benign model updates can be advantageous for crafting more effective attacks.

In the literature, tailored attacks typically assume full knowledge of the aggregation algorithm [18, 55], while the LIE algorithm [5] assumes full knowledge of the benign updates. In our scenario, our focus is to demonstrate the capability of our

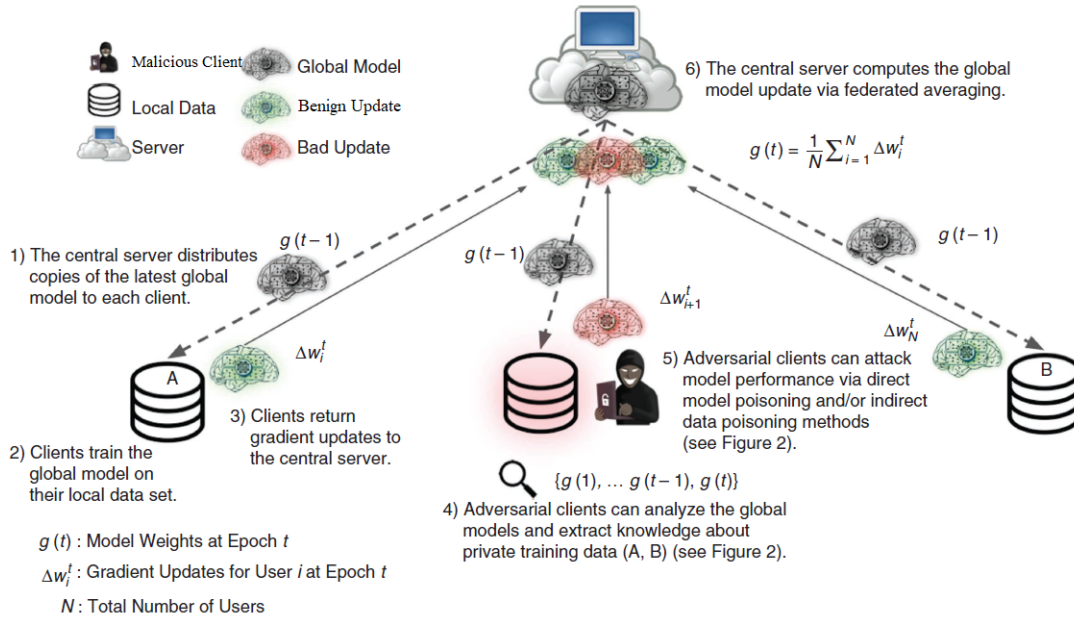


Figure 3.1: An overview of a single round of federated learning with an adversarial client [30]

defence strategy, but our defence can also be used in scenarios where there is no threat as it is also effective for cost saving and time consumption. Hence, our attack approach involves an attacker who has the capability to manipulate the weight of the client in order to decrease the performance of the model, which was helpful to demonstrate the effectiveness of the defence strategy.

### Defender's Knowledge

**Defender's Goal:** The defender's objective is two-fold. Firstly, the global model should exhibit robustness against various threat models, ensuring that it maintains high accuracy on the main task even in the presence of potential attackers. Secondly, the defense mechanism should not compromise the model's accuracy when there are no attacks, preserving *fidelity*. Unfortunately, many existing defense algorithms focus solely on robustness and overlook fidelity as we gathered from other research papers; they only consider using robust algorithms that require heavy computational resources and consume time, which is why we focused on using FedAvg algorithm [43] in our work. As we discussed before, FedAvg is still the algorithm that Google is using in its projects [59]. For example, research shows that using Krum [7] leads to a significant drop in testing accuracy (over 20%) on the Mnist dataset, rendering

the model practically useless. We delve into the common weaknesses of existing defenses. Hence, when devising defense mechanisms, it is crucial to compare the model’s accuracy in the absence of attacks with a baseline algorithm such as FedAvg, as shown in chapter 5. To address the robustness challenges, we design our defense under adversarial settings where the attacker has complete knowledge of the benign updates. Additionally, we propose an attack scenario where the defense algorithm is disclosed to the attacker.

**Defender’s Knowledge:** The defender in our paper lacks any client-side information, including access to local training data. Unlike previous approaches, we also assume that the server does not require knowledge of the upper bound of the malicious fraction  $q$ . The server can solely access the model updates received from participating clients in each round. To ensure both fidelity and robustness, we assume the server possesses a tiny clean labeled dataset. This only eliminates the threats of the data poisoning attacks. Although this dataset comes from a similar domain, it may not follow the same distribution as the training data, and it does not need to be independent and identically distributed (iid) and may even be skewed. Our experiments in chapter 5 demonstrate that a very small dataset (e.g., 100 samples for Mnist and FMnist) suffices to achieve our objectives. Such a small dataset can often be collected from the public domain and manually labeled, making it a feasible approach.

### 3.2.2 Proposed Defence Approach: Credit-Based Client Selection

Our defence strategy is inspired by the algorithm that was first proposed in the end of 2021 (recall that in 2019, Google open-sourced the Federated Learning framework, making it accessible to a broader community of researchers and developers). This move encouraged further research and innovation in the field, including client selection approaches.

Client Selection as depicted in Figure 3.2 [29], is an essential aspect of Federated Learning defense mechanisms. The idea of utilizing the historical behavior of clients to assess their trustworthiness has become more prominent in recent years as FL faces security challenges. Value-based client selection gained traction as a defense strategy in the mid-2021s. The idea of assigning value to clients based on their past contributions and behavior was proposed as an effective way to detect and mitigate poisoning attacks. As research in Federated Learning defense progressed, various metrics and

algorithms for computing values were proposed [65]. Researchers experimented with different weighting schemes and update mechanisms to enhance the accuracy of the model. Client selection became a crucial component of adversarial defense strategies in FL. By excluding clients with low values, the defense mechanism aimed to improve the robustness of the global model against poisoning attacks.

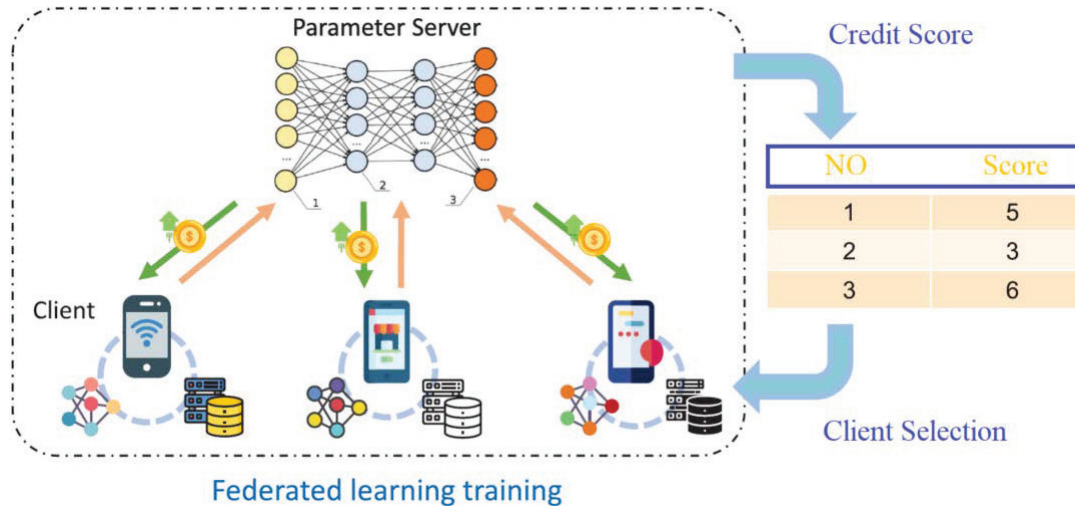


Figure 3.2: Client selection process in Federated Learning by involving credit scores

In our defense strategy, we explored existing client selection strategies and drew inspiration from a method that credits clients based on their performance. Initially, we decided to assign each client’s model accuracy as their corresponding score. However, we later decided to incorporate the loss function of each client’s model into the score calculation. We then implemented a sorting mechanism in the credit algorithm, where clients with low credit scores are excluded from the training round, and only clients with high accuracy and low loss metrics are allowed to participate.

Through extensive validation experiments involving two different model architectures and three datasets, our approach demonstrated stability over non-iid data and under imbalanced distribution. The proposed method aims to enhance the use of federated learning in training environments, particularly in addressing privacy concerns. The objective is to achieve a higher accuracy of the aggregated global model (final model) by disregarding poorly performing models or models that do not contribute significantly to the overall improvement.

It is worth mentioning that the excluded models could be provided by two categories of users: 1) malicious users who may attempt data poisoning attacks, and 2) cooperative users who might contribute data of poor quality. The proposed frame-

work aims to eliminate the contributions of these user sets, as illustrated in Figure 3.3. The process involves calculating a Credit score for each user, representing their contributions and performance.

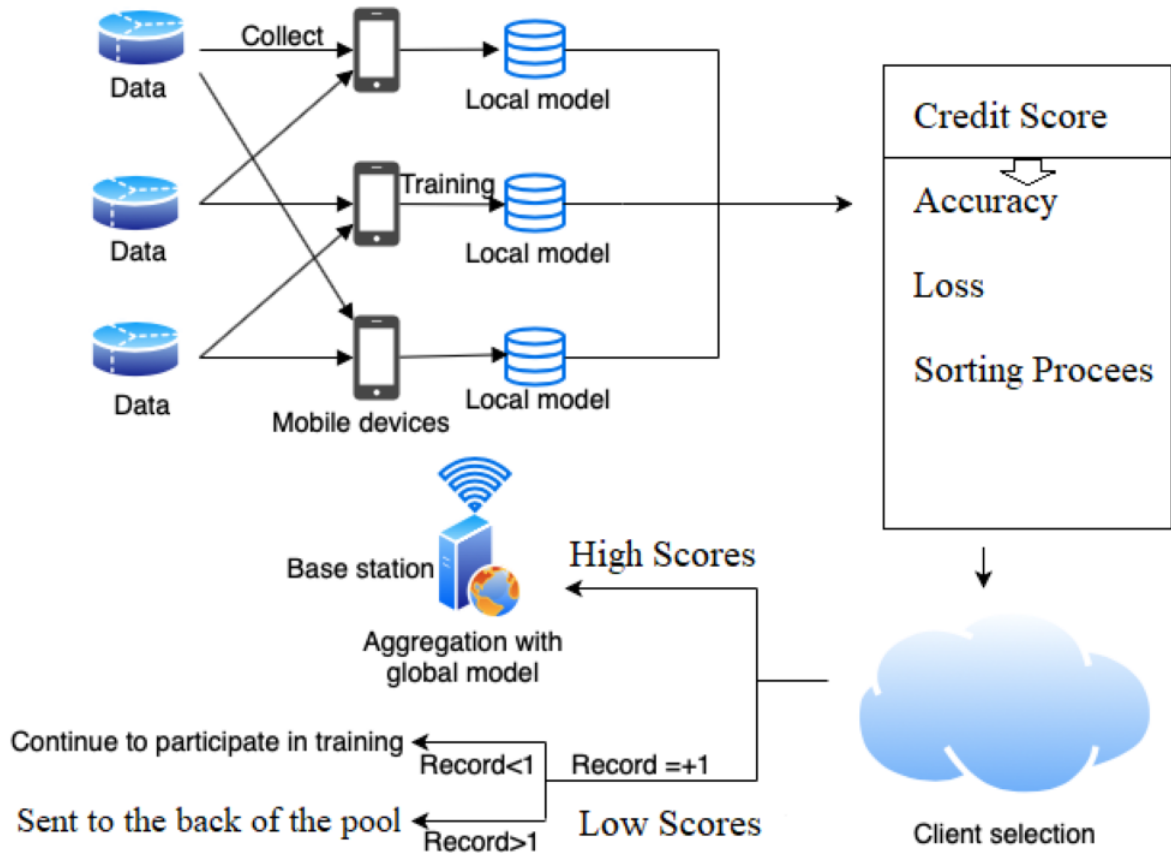


Figure 3.3: Federated Learning with our defensive strategy

In a federated learning scheme involving a base station and a set of  $C$  clients, the clients collaborate with the base station to execute federated learning. Federated learning ensures that the base station and users learn a shared learning model collaboratively while keeping all datasets locally on the clients' devices. Each client trains its local model using the data generated locally. The global model is aggregated at the base station and distributed as a shared learning model. This shared federated learning model is used to improve each client's local model without transmitting their data to a central server. The global federated learning model is generated at the base station using the local model parameters from each client, which are uploaded from the client to the base station, and then downloaded by each client's device.

In a setting with  $N$  clients, we denote  $e$  as the number of epochs and  $i$  as the identifier of a local client that participates in training. The data related to each

client is stored in the dictionary  $Local[i]$ , which includes information such as accuracy, training loss, and accuracy score during the training process. Specifically, the training results of a particular client in the  $e - th$  epoch are saved in  $Local[i][e]$ . On the other hand, the global model features are stored in the dictionary  $Global[]$ .

Algorithm 1 outlines the overall process of our proposed credit-based client selection. The selection of clients to participate in training in the current epoch is based on the credit score of the models updated by the selected clients in previous epochs. This credit score considers the accuracy and performance of each client’s model over time and helps determine their eligibility for participation in the current training round and will process a sorting algorithm in the rounds to determine the best updates and clients with the best performance in each round.

---

**Algorithm 1:** Credit-Based Client Selection Procedure

---

**Data:** Data of local model, dictionary  $Local[]$   
**Result:**  $L$ : list of client IDs

```

1 for  $i$  in  $N$  users do
2   if  $Local[client_i]$  is an empty dictionary; /* First epoch or first time
   this client being selected */
3     then
4        $L \leftarrow L \cup \{i\}$ 
5     else
6        $e \leftarrow$  number of epochs
7       loop over  $e$ 
8       Let  $n_i$  be the total of ( $Local[client_i][e][\textit{CreditScore}] > 0$ )
9       if  $n_i < n_{i-1}$  then
10         $z = n_i$ ;
11         $n_{i-1} = n_i$ ;
12         $n_i = z$ ;
13        ignore  $i$  from  $L$ ;
14        Sorted;

```

---

Now, we are going to describe how to calculate the credit score, but first we are going to give a brief mathematical description of how federated learning works and also explain the loss function that we used for calculating our credits for the clients. According to [71], the federated learning training process can be summarized into three steps: initialization, local model training, and global aggregation.

1. **Step 1: Initialization.** The federated learning process begins with the parameter server determining the architecture of the global model. The parameters



of the global model are then initialized either randomly or through pretraining on a public dataset, this approach is the normal process of federated learning randomly choosing the clients, that is why we decided to implement our own approach and choosing the clients based on their performance and not randomly. After that, the clients will train depending on the specific training task. Subsequently, the parameter server distributes the initial global model parameters  $\omega_0$  to the selected clients.

2. **Step 2: Local model training.** In the  $t$ -th communication round, each selected client updates its local model parameters  $\omega_t^i$  based on the received global model parameters  $\omega_t$  and using its local dataset. After local model training, the updated local model parameters  $\omega_{t+1}^i$  are sent back to the parameter server. The objective of client  $i$  in the  $t$ th round is to minimize the empirical loss  $F(\omega_t^i)$  based on its local dataset, according to Equation 3.7 and Equation 3.8, to make its local model well fit to the data as well. In equation 3.8,  $|D_i|$  denotes the number of samples in dataset  $D_i$ .

$$\omega_t^i = \arg \min_{\omega_t^i} F(\omega_t^i), \quad (3.7)$$

$$F(\omega_t^i) = \frac{1}{|D_i|} \sum_{j \in D_i} f_j(\omega_t^i), \quad (3.8)$$

The update process in each client can be achieved by performing stochastic gradient descent (SGD) with mini-batches sampled from its local dataset  $D_i$ , as per Equation 3.9. The local model parameters  $\omega_t^i$  in the  $t$ -th round are updated as per Equation 3.9, where  $\nabla F(\omega_t^i)$  represents the gradient of the loss function with respect to the local model parameters  $\omega_t^i$ , and  $\eta$  is the learning rate. The update is performed to minimize the empirical loss function  $F(\omega_t^i)$  based on the local dataset  $D_i$ .

$$\omega_t^i = \omega_t^i - \eta \cdot \nabla F(\omega_t^i), \quad (3.9)$$

3. **Step 3: Global Aggregation.** In each round, the parameter server aggregates the locally updated parameters from selected clients and replaces the global model with the average model. The updated global model parameters  $\omega_t^i$  are then sent back to the selected clients. The aim is to minimize the global loss

function, which can be expressed as follows:

$$F(\omega_t) = \frac{1}{|D|} \sum_{i=1}^N |D_i| \cdot F(\omega_t^i), \quad \text{where } i \in \{1, 2, \dots, N\}, \quad (3.10)$$

This iterative process continues for multiple rounds until the global model converges or a stopping criterion is met. The aim is to collaboratively learn a shared learning model across all clients without sharing their raw data, thereby achieving privacy preservation while improving the overall performance of the global model. These steps are repeated until the desired accuracy is achieved. Compared to traditional model training approaches, federated learning offers several advantages such as privacy preservation, Efficient resource utilization and lower inference latency.

Now that we have explained how federated learning works and what a loss function is, we are going to describe the credit score used in our defense method.

**Credit score:** It serves as an assessment of each local model’s suitability for aggregation into the global model. It plays a crucial role in identifying underperforming models and malicious users, thereby ensuring the accuracy of the global model. At the beginning of each epoch, every user’s credit score is calculated. The following equation shows the parameters we used to calculate the credit score.

$$CreditScore = F(W_{acc}, W_{loss}) = W_1(acc) - W_2(\mu \times loss), \quad (3.11)$$

Algorithm 2 outlines the entire process of calculating the credit score, which takes into account two key parameters: accuracy and loss, and more importantly, the sorting process that will take place. These parameters are used to determine the contribution of each local model, influencing their credit score.

The training process halts either when the learning curve converges or when the system exhausts the available local users, or when it reaches the appointed number of clients. The evaluation of each local model’s test accuracy is done using two metrics to determine their credit score (equation 3.11).

First, the test accuracy of local models in each epoch is compared to compute an average test accuracy while appending the loss of their desired performance. Models that perform below the average are given a lower credit, while those that outperform the average are favored.

Second, the test accuracy of each local model is compared to the rest of the clients

---

**Algorithm 2:** Computation of Credit-Score in the local model in epoch  $e$ 


---

**Data:** Data of local model, dictionary Local[]  
**Result:** Local[ $i$ ][ $e$ ][Credit - Score]

- 1 **for** all users  $i$  **do**
- 2   total += Local[ $i$ ][ $e$ ][accuracy]
- 3 average =  $\frac{\text{total}}{n}$
- 4 **for** all user  $i$  **do**
- 5   **if** Global[ $e-1$ ] has no value **then**
- 6     Local[ $client_i$ ][ $e$ ][Credit-Score] =  
       (Local[ $client_i$ ][ $e$ ][accuracy])  $\times w1 - (\mu \times \text{Global}[e][\text{Loss}]) \times w2$
- 7   **else**
- 8      $e \leftarrow$  number of epochs
- 9     loop over  $e$
- 10    Let  $n_i$  be the total of (Local[ $client_i$ ][ $e$ ]['CreditScore'] > 0)
- 11    **if** Local[ $client_i$ ][ $e$ ][Credit-Score] < Local[ $client_{i-1}$ ][ $e$ ][Credit-Score] **then**
- 12     Store = Local[ $client_{i-1}$ ][ $e$ ][Credit-Score];
- 13     Local[ $client_{i-1}$ ][ $e$ ][Credit-Score] = Local[ $client_i$ ][ $e$ ][Credit-Score];
- 14     Local[ $client_i$ ][ $e$ ][Credit-Score = Store;
- 15     ignore  $i$  from  $L$ ;
- 16     Sorted;

---

credit score; if a local model performs worse than other clients, it receives a negative contribution to its credit score. This metric helps in selecting the best models at each epoch, eliminating poor models with negative credit score before they are aggregated into the global model. However, they will have a chance to come back at the next epoch.

Third, each client's weight is compared to other nine clients to assess its improvement. A positive comparison indicates a positive contribution to the credit score, and this is done via the sorting algorithm and signifying an improvement over the previous global model.

By considering these two metrics, a positive credit score indicates a positive contribution from the local model. Conversely, a poorer performance compared to the other clients' weights or lack of improvement over the previous global model would result in a negative impact on the credit score. As local models are trained further after each epoch, their performance is likely to improve, leading to positive contributions to their credit score. The sorting algorithm used in our research is called "argsort." It is a function provided by the NumPy library in Python. The argsort

function performs an indirect sort on an array and returns the indices that would sort the array in ascending order.

In the context of the code, the `argsort` function is used to sort the clients based on their current credit scores, or most importantly, client weights in descending order. The line `sortedclients = np.argsort(clientweights)` performs the `argsort` operation and stores the indices of the clients in the sorted order with the highest credit score at the beginning of the `sortedclients` list.

By sorting the clients based on their credit scores, the code ensures that the clients with the highest credit or highest scores are considered first during the federated learning process, allowing their model updates to have a greater influence on the global model. This sorting strategy is an essential part of our credit-based client selection mechanism and helps prioritize reliable clients with better performance in the federated learning process.

Of course, this will raise the question about the clients that do not participate in the current round; if a client does not participate in the current round (i.e., it is not selected for model update in the current round), it will not have any effect on the global model for that round. The client’s model will not be updated with new data, and its model update will not be included in the computation of the global model.

As we mentioned, during the federated learning process, the clients are selected based on their credit scores (`clientweights`) and are sorted in descending order of their credit scores using the `argsort` function. The clients with the highest credit scores are selected first and used for model updates, while the clients with lower credit scores may not be selected in the current round. After calculating the credit scores, the `clientweights` list is populated with the credit scores, which act as weights for the clients during the federated averaging process. These weights are normalized to ensure they sum up to 1, effectively determining the contribution of each client to the global model.

As a result, clients that are not selected in a particular round will keep their models unchanged, and their previous model updates will not be considered in the aggregation of the global model for that round. However, these clients will still be considered for future rounds, and their credit scores may change based on their performance in subsequent rounds. The hyperparameter that we used  $\mu$ , and the term  $0.5 \times \text{loss}$  in the credit score calculation is an arbitrary coefficient used to balance the contribution of accuracy and loss in the overall credit score. Our goal was to combine both accuracy and loss metrics into a single credit score that reflects the

client's performance in the federated learning process. Here, "accuracy" represents the accuracy of the client's model on the test data, and loss represents the loss (in this case, the cross-entropy loss) of the client's model on the same test data. The loss metric measures how well the model's predictions match the true labels, while accuracy measures the overall correctness of the model's predictions.

By subtracting  $0.5 \times \text{loss}$  from accuracy, the credit score gives more weight to accuracy while still taking the loss into account. The coefficient 0.5 is chosen arbitrarily and can be adjusted based on the application or desired behavior. For example, a higher coefficient for loss might be used if the loss is considered to be more critical in certain scenarios.

The idea behind including both accuracy and loss in the credit score is to incentivize clients to not only improve their accuracy but also to reduce their model's loss during the federated learning process. This helps in selecting clients with better-performing models for model aggregation and potentially improving the overall performance of the global model.

# Chapter 4

## Datasets

In our study, we draw inspiration from existing research in Federated Learning (FL) [5, 18, 55, 56, 62] and we focus on three widely used image classification datasets: Mnist [38], FMnist [13], and CIFAR-10 [33].

### 4.1 Mnist

The Mnist dataset [38, 39] is a widely recognized and extensively used 10-class grayscale digit image classification dataset as shown in Figure 4.1 [4]. It comprises a total of 70,000 images, out of which 60,000 images are designated for training, and the remaining 10,000 images are reserved for testing purposes. Each image in the dataset is of dimensions  $28 \times 28$  pixels.

For the purpose of our research, we distribute the training data among 10 clients and 100 clients, with each client receiving an equal subset of the number of samples. This distribution ensures that each client has a representative and diverse set of training examples to develop robust and accurate local models. By dividing the data among multiple clients, we aim to explore federated learning scenarios where each client trains its model locally without sharing their raw data, enabling privacy-preserving collaborative model training.

The Mnist dataset's popularity arises from its simplicity, making it an excellent benchmark for evaluating various image classification algorithms, including traditional machine learning methods and deep learning techniques. Its manageable size and balanced class distribution contribute to its suitability for exploring federated learning approaches and providing valuable insights into model performance across

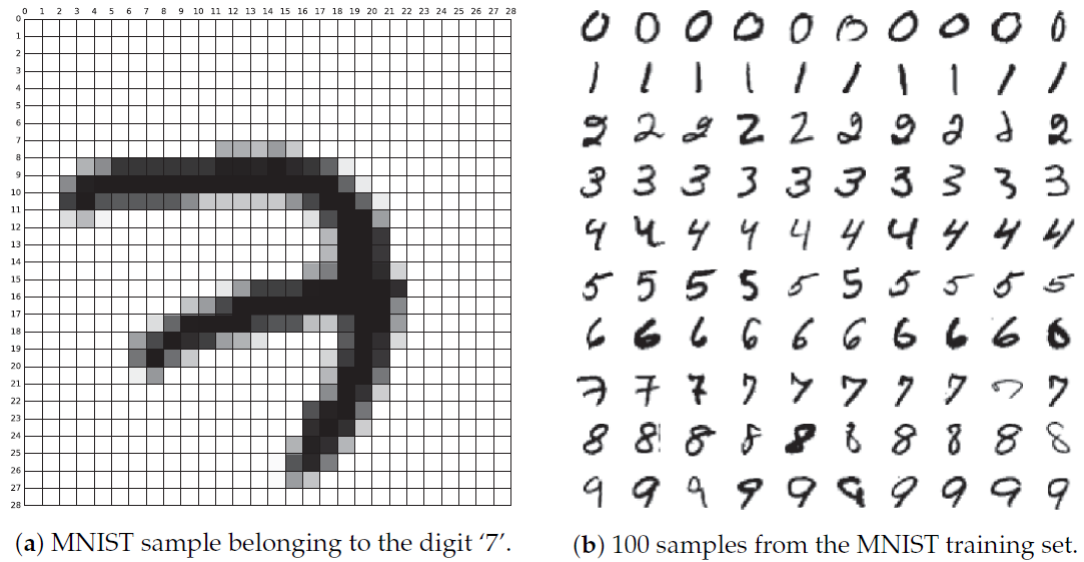


Figure 4.1: Example of the Mnist database [4]

decentralized environments. Below is a full description and history of the Mnist database:

1. Name: Mnist (Modified National Institute of Standards and Technology) Dataset
2. Source: The Mnist dataset is derived from the original NIST Special Database 19, which was collected by the National Institute of Standards and Technology (NIST) for the purpose of developing and evaluating optical character recognition (OCR) algorithms. The Mnist dataset, as used in this study, is a processed and modified version of the NIST dataset to fit the requirements of machine learning applications.
3. Description: The Mnist dataset is a well-known and widely used benchmark dataset in the machine learning community. It consists of a collection of grayscale images of handwritten digits (0 to 9) [4], with each image represented as a  $28 \times 28$  pixel matrix. The images have been normalized and centered, making the dataset suitable for machine learning tasks, particularly image classification.
4. Size: The Mnist dataset comprises a total of 70,000 samples, which are further divided into 60,000 training samples and 10,000 test samples. The division of data into training and test sets allows for the evaluation and comparison of various machine learning models in a controlled manner.

5. **Image Format:** Each image in the Mnist dataset is represented as a grayscale image, where each pixel's intensity value ranges from 0 (black) to 255 (white). The grayscale representation simplifies the input for machine learning algorithms by reducing the complexity of color-based information.
6. **Label Format:** Each image in the Mnist dataset is associated with a corresponding label, indicating the digit (0 to 9) it represents. The digit labels are used as the ground truth for the training and evaluating of classification models.
7. **Data Distribution:** The Mnist dataset is carefully constructed to ensure a balanced distribution, where each digit class is equally represented, resulting in approximately 7,000 samples per digit class. This balanced distribution ensures that machine learning models do not exhibit biases towards specific digits during training and testing.
8. **Purpose:** The primary purpose of the Mnist dataset is to serve as a benchmark for evaluating the performance of various machine learning algorithms, particularly in the domain of image classification. It has been widely used to assess the effectiveness of different approaches, including traditional machine learning methods and deep learning models.
9. **Difficulty Level:** While the Mnist dataset is considered relatively simple compared to more complex real-world datasets, it remains a valuable resource for introductory machine learning tasks and serves as a stepping stone for researchers and learners to grasp fundamental concepts in image recognition and classification.
10. **Use Case:** The Mnist dataset is commonly employed in training and testing classifiers to recognize handwritten digits. It finds applications in optical character recognition (OCR) systems, digit classification tasks, and various other image recognition scenarios.
11. **Relevance:** Despite being introduced in the late 1990s, the Mnist dataset remains relevant as a standard benchmark in the machine learning community. Its popularity stems from its simplicity, which facilitates quick experimentation and comparison of different algorithms, making it a valuable resource for educational purposes and initial model testing.



The Mnist dataset’s longevity and widespread usage can be attributed to its ability to provide valuable insights into the performance and limitations of various machine learning models, making it an essential component of machine learning research and experimentation.

## 4.2 Fashion Mnist (FMnist)

FEMnist [13, 53], short for Federated Extended Mnist, is an augmented version of the Mnist dataset, encompassing a more extensive and diverse collection of handwritten character digits. The dataset is notably larger, comprising a total of 814,255 grayscale images. These images are derived from handwritten examples provided by 3400 individuals, and each example is converted into a standardized  $28 \times 28$  pixel image format. Also there is a second version of FEMnist called Fashion Mnist(FMnist) dataset as shown in Figure 4.2 [66], which is a popular image classification dataset used as a benchmark for machine learning and computer vision tasks. It serves as a direct replacement for the classic Mnist dataset but offers more diverse images. The dataset comprises 10 distinct classes, each representing a different fashion item: a. T-shirt/top, b. Trouser, c. Pullover, d. Dress, e. Coat, f. Sandal, g. Shirt, h. Sneaker, i. Bag and j. Ankle boot. Fashion Mnist is commonly used to evaluate the performance of various machine learning and computer vision algorithms, such as deep learning models and image classifiers.

What sets FMnist apart from Mnist is the inclusion of 62 distinct classes in the dataset. These classes encompass both upper and lower case letters, along with the traditional digits. Specifically, there are 52 classes representing various alphabetic characters and an additional 10 classes corresponding to numerical digits.

The expansion of classes in FMnist not only increases the dataset’s complexity but also adds greater variability and challenges for model training and classification tasks. This augmentation makes FMnist an excellent choice for evaluating the performance and robustness of federated learning models across a more diverse set of image recognition tasks. As with Mnist, FMnist facilitates the exploration of privacy-preserving collaborative model training approaches, where individual client data remains locally secure, yet contributes collectively to the development of a more accurate and versatile global model. Hence, it is perfect for federated learning settings.

Below, we provide a detailed overview and history of FMnist, covering 11 aspects of the dataset:











Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 4.2: Class names and example images in FMnist database [66]

1. Dataset Origin: FMnist is generated by collecting handwritten character digits from a diverse group of 3400 individuals.
2. Image Format: Each example in FMnist is represented as a grayscale image of dimensions  $28 \times 28$  pixels.
3. Dataset Size: FMnist is substantially larger than the original Mnist dataset, comprising a total of 814,255 images.
4. Number of Classes: Unlike Mnist, which contains 10 classes representing numerical digits (0 to 9), FMnist expands the class set to include a broader range of characters.
5. Class Distribution: FMnist encompasses 62 distinct classes, organized into different categories: a. 52 classes represent uppercase and lowercase letters of the

alphabet. b. 10 classes represent numerical digits (0 to 9).

6. **Class Imbalance:** Due to the inclusion of multiple alphabetic characters, FMnist exhibits class imbalance, with some classes having more samples than others.
7. **Handwriting Variation:** As FMnist is derived from real handwritten examples, it captures natural variations in writing styles, shapes, and sizes.
8. **Privacy Preservation:** FMnist is designed for federated learning scenarios, where data privacy is crucial. Each individual's data is kept locally on their respective devices, ensuring privacy while contributing collectively to model training.
9. **Federated Learning Setting:** The dataset is intended for evaluating and developing federated learning algorithms, where local models on individual devices are updated and aggregated to create a more accurate global model.
10. **Multiclass Classification:** FMnist presents a challenging multiclass classification task, with models required to identify both letters and numerical digits.
11. **Real-World Application:** FMnist is well-suited for testing federated learning in real-world applications, where users' devices collectively collaborate in model training without sharing raw data.

Overall, FMnist provides a valuable resource for researchers and practitioners in the field of federated learning. Its larger size, diversified classes, and privacy-aware nature make it a suitable benchmark dataset for evaluating the performance and robustness of federated learning algorithms in handling real-world scenarios and protecting data privacy.

### 4.3 Cifar-10

CIFAR-10 [33] is a well-known color image classification dataset, widely used in the field of machine learning and deep learning. The dataset consists of images, which are divided into a training set and a test set as shown in Figure 4.3 [2].

The CIFAR10 dataset, created by the Canadian Institute For Advanced Research, comprises ten distinct classes, each containing 60,000 color images of  $32 \times 32$  resolution. The dataset is further divided into 50,000 training images and 10,000 testing images. Within the testing dataset, in most cases (not our case) there are precisely

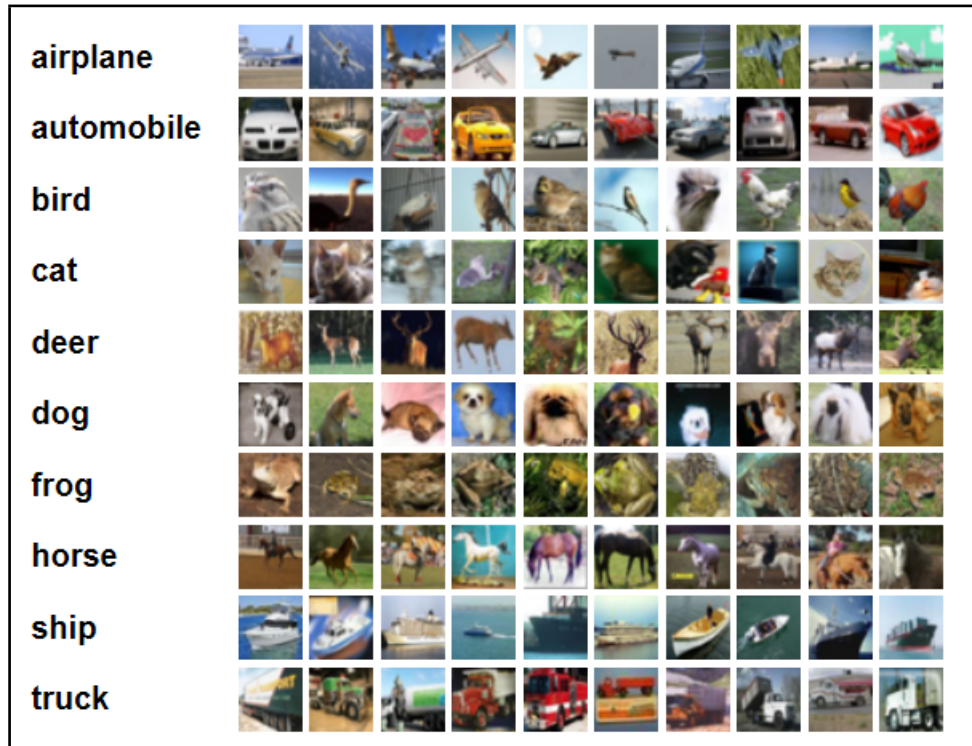


Figure 4.3: Visualization of the CIFAR 10 Train Dataset in the Deep Lake UI [2]

1,000 images randomly selected from each class, ensuring a balanced representation of all categories during evaluation [34, 35]. On the other hand, the training datasets may exhibit slight variations in the number of images per class, but on average, they contain 5,000 images per class. This variability arises due to the random arrangement of images, which adds diversity to the training process and fosters a more robust and generalizable model. Below is a detailed overview of the dataset:

1. **Training Set:** The training set comprises 50,000 color images, carefully labeled with corresponding class labels. These images are used to train machine learning models and neural networks to recognize various object categories.
2. **Test Set:** The test set consists of 10,000 color images, used for evaluating the performance of the trained models. It serves as an unseen dataset during training, enabling researchers to assess the generalization capabilities of their models.
3. **Image Format:** Each image in the CIFAR-10 dataset is represented in RGB (Red, Green, Blue) color space, allowing for rich color information to be captured. This makes it suitable for tasks that involve color-based classification.

4. **Image Size:** The images are standardized to a resolution of  $32 \times 32$  pixels. This relatively small size presents a challenge for models, as they need to infer object categories from limited spatial information.
5. **Class Labels:** CIFAR-10 consists of ten distinct classes, each representing a specific object category. The classes include: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck.
6. **Class Distribution:** The dataset maintains a balanced distribution of samples across its ten classes. Each class contains an equal number of images, totaling 6,000 images per class.
7. **Real-World Data:** CIFAR-10 is composed of real-world images, depicting various everyday objects encountered in daily life. The diversity of object categories enhances the dataset's ability to address real-world image recognition challenges.
8. **Benchmarking:** Due to its challenging nature and wide range of object categories, CIFAR-10 serves as a prominent benchmark dataset for evaluating the performance of image classification algorithms. Researchers often use it to compare and assess the effectiveness of different models and techniques.
9. **Data Augmentation:** To augment the dataset and increase the robustness of models, data augmentation techniques are commonly applied during training. These techniques include random rotations, translations, flips, and color transformations, enabling models to generalize better to unseen data.
10. **Research Impact:** CIFAR-10 has significantly contributed to the advancement of computer vision research, particularly in the development of deep learning algorithms. It has been used to train state-of-the-art convolutional neural networks (CNNs) and explore various techniques, such as transfer learning and network architectures.

In conclusion, CIFAR-10 is a valuable and widely recognized dataset that offers researchers an opportunity to study image classification tasks in real-world scenarios. Its diverse object categories, realistic images, and challenging characteristics make it an essential resource for advancing the field of computer vision and machine learning.

## 4.4 Data Preprocessing

Preprocessing refers to the series of steps and techniques applied to raw data before it is used in a machine learning or data analysis task. The main purpose of preprocessing is to clean, transform, and format the data in a way that makes it suitable for further analysis and model training [15, 21]. Preprocessing is a critical step in the data science workflow, as the quality and structure of the data directly impact the performance and accuracy of machine learning models. Some common preprocessing steps are shown in Figure 4.4 [21] and include:



Figure 4.4: Visualization of data preprocessing steps [21]

- **Data Cleaning:** Removing or handling missing data, outliers, or noisy data points that could negatively impact model performance.
- **Data Normalization or Scaling:** Scaling the data to a similar range or normalizing it to have a mean of zero and a standard deviation of one. This step ensures that features with different scales do not dominate the model training process.
- **Feature Selection:** Selecting relevant features or attributes from the data to reduce dimensionality and focus on the most important information.
- **Feature Engineering:** Creating new features or transforming existing ones to extract meaningful information and improve model performance.
- **Encoding Categorical Variables:** Converting categorical variables into numerical representations that machine learning algorithms can understand.

- Handling Imbalanced Data: Addressing class imbalances in classification tasks to prevent bias towards the majority class.
  - Splitting the Data: Dividing the data into training, validation, and testing sets to evaluate the model's performance accurately.
- Handling Text and Image Data: For natural language processing (NLP) tasks, tokenizing and vectorizing text data.

For image processing tasks, the steps also include resizing and normalizing image data. The specific preprocessing steps and techniques used depend on the nature of the data and the machine learning algorithm being applied. Proper preprocessing ensures that the data is in the best possible form for accurate model training and reliable insights.

#### 4.4.1 Preprocessing Mnist

For preprocessing the Mnist dataset [61, 64], we first loaded the Dataset, The Mnist dataset is loaded using TensorFlow's *tf.keras.datasets.Mnist.load\_data()* function. This function directly provides the train and test images along with their corresponding labels. Second is data Splitting: the loaded training images and labels are combined into a list of tuples called *federated\_train\_data*, where each tuple contains an image and its corresponding label. The entire dataset is then split into 100 shards using the *np.array\_split()* function. Each shard represents a subset of the dataset, which simulates the distribution of data among multiple clients in federated learning. Third, the model architecture is defined using *tf.keras.Sequential*. It consists of three layers: a Flatten layer, a hidden Dense layer with *ReLU* activation, and an output Dense layer with Softmax activation. This architecture is designed to handle the 28x28 pixel images of Mnist, then we start the training on Shards; The *train\_on\_shard()* function is defined to train the model on a shard of the dataset. The function takes the shard data (images and labels) and the model as inputs. The images and labels are converted to numpy arrays using *np.array()* for efficient processing, and then the model is trained on this shard for one epoch using *model.fit()*.

#### 4.4.2 Preprocessing FMnist

The preprocessing of the Fashion Mnist dataset [41] is done using the TensorFlow and NumPy libraries, same as the Mnist dataset as they both has the similar structure.

Here’s a step-by-step description of the preprocessing [36]. Loading the Dataset: The Fashion Mnist dataset is loaded using TensorFlow’s built-in function *tf.keras.datasets.fashion\_mnist.load\_data()*. This dataset contains images of clothing items like shirts, trousers, dresses, etc. Data Split: The loaded dataset is divided into two sets: the federated training data and the test data. The federated training data is further split into 100 shards using *np.array\_split()* function. The model architecture is defined using TensorFlow’s Sequential API, the same as Mnist dataset. The model consists of a Flatten layer to convert the  $28 \times 28$  image data into a 1D array, followed by a Dense layer with 128 units and a *ReLU* activation function. The final layer is another Dense layer with 10 units (equal to the number of classes in Fashion Mnist) and a softmax activation function to predict the class probabilities. Training the Model on a Shard: The function *train\_on\_shard()* is defined to train the model on a single shard of the federated training data. The function takes a shard of images and their corresponding labels, converts them to NumPy arrays, and fits the model on the data for one epoch.

### 4.4.3 Preprocessing Cifar-10

The preprocessing of Cifar-10 is slightly different [51]. In our research, the CIFAR-10 dataset is loaded and preprocessed before being used for federated learning. The preprocessing steps include loading the dataset, splitting the data into shards, and defining the model architecture, so the steps are same as the Mnist and FMnist databases. First is loading the CIFAR-10 dataset. The implementation uses the *tf.keras.datasets.cifar10.load\_data()* function to load the CIFAR-10 dataset. Then the dataset is split into training and testing sets, containing images and corresponding labels. Sharding the data into 100 shards: The *federated\_train\_data* variable is created as a list of tuples, where each tuple contains an image ( $32 \times 32 \times 3$  array) and its corresponding label (single integer). The data is then split into 100 shards using the *np.array\_split()* function. This step is specific to federated learning, where data is divided into non-overlapping subsets (*shards*) to simulate distributed clients. Lastly is defining the model Architecture: The model architecture is defined using *tf.keras.Sequential*, which is a linear stack of layers. The model starts with three Convolutional layers (*Conv2D*) with *ReLU* activation, followed by two Max Pooling layers (*MaxPooling2D*). Then, a Flatten layer is added to convert the 3D feature maps into a 1D vector. After that, a Dense layer with *ReLU* activation is used as



a hidden layer, followed by the output Dense layer with 10 neurons using Softmax activation for 10 classes. Compilation of the Model: The model is compiled using the *compile()* method of the Keras model. The optimizer used is *'adam'*, a popular optimization algorithm based on adaptive learning rates. The loss function is set to *'sparse\_categorical\_crossentropy'* since the dataset contains integer labels. The metrics are defined as *'accuracy'* to monitor the model's performance during training. These preprocessing steps prepare the CIFAR-10 dataset for the federated learning process [51]. The federated learning process includes training models on shards of the dataset, aggregating models using federated averaging, and updating client weights based on credit scores that we described in chapter 3.

## Chapter 5

# Experiments, Results, and Discussion

In this chapter, we present the setup for our implementation, the frameworks utilized in our experiments, and the parameters explored during our study. We demonstrate the results of employing the credit-based client selection approach both with and without attacks. We conduct experiments under normal settings and investigate the impact of varying parameters for the attack weight and loss function. Through this analysis, we showcase how these mechanisms and parameter choices influence the performance of the model.

### 5.1 Federated Learning Parameters and Settings

In our experiments, we set the number of clients  $N$  to 100 for the Mnist and CIFAR-10 datasets, and for the larger FMnist dataset, we consider  $N$  as 500 to simulate the cross-device federated learning scenario. In each federated learning round, we select  $n$  clients to participate based on our state-of-the-art credit-based client selection mechanism, where  $n$  is set to 30 for Mnist and CIFAR-10, and 50 for FMnist. But for demonstration purposes, we are going to simulate 10 clients for each dataset so we can demonstrate the results in the tables. To represent malicious clients, we assume a default malicious fraction which is the malicious weight  $\omega$  equal to 0.1, but we also investigate the impact of varying  $\omega$  values. Most importantly, to represent the impact of our defence, we incorporated  $\zeta$  as a parameter for our loss function in our equation, the initial value is 0.5, but we test our results with different values.

For training the Mnist and CIFAR-10 datasets, we employ the SGD optimizer with a learning rate initialized to 0.01. The training process is carried out for a total of  $R = 100$  FL epochs, and the learning rate is reduced after every 30 epochs. As for FMnist, we conduct  $R = 100$  FL epochs with a learning rate of 0.02. An epoch is one complete pass through the entire training dataset during the training process. In other words, it is the number of times the model sees the entire dataset. For example, if we have 100 training samples and you train your model for 10 epochs, it means the model will go through the entire dataset 10 times during training. The local batch size  $B$  is set to 10 for Mnist and FMnist, while for CIFAR-10, it is set to 32. The batch size is the number of samples that are processed by the model at each iteration during training. Instead of updating the model’s weights after processing each individual sample, the model computes the gradients based on a small batch of samples and updates the weights after each batch. The batch size is typically a hyperparameter that can be adjusted to control the trade-off between computation efficiency and convergence speed. Larger batch sizes can be more computationally efficient, but smaller batch sizes may offer better generalization and convergence. And lastly, the number of local epochs is defined as 10 for Mnist, CIFAR-10, and also 10 for FMnist. Table 5.1 presents a summary of the federated learning parameters and their default values used in our experiments unless stated otherwise.

Table 5.1: Federated Learning parameter values for all datasets

Parameter Name	Mnist	CIFAR-10	FMnist
$N$ (Total # of Clients )	100	100	500
$n$ (Total # of Participants)	30	30	50
$R$ (# of FL Rounds)	100(10)	100(10)	100(10)
$B$ (Batch Size)	32	32	32
$\omega$ (Malicious Weight)	0.2	0.2	0.2
$\zeta$ (loss)	0.5	0.5	0.5
$r$ (Local Epochs)	10	10	10

## 5.2 Model Architecture

We employ diverse model architectures tailored to each dataset. Specifically, for Mnist and FMnist, we utilize a convolutional neural network (CNN) with the architecture

presented in Table 5.2 [37]. As for CIFAR-10, we adopt the VGG-11 [57] network which is presented in Table 5.3. It is important to note that all clients use the same model architecture as the global model for consistency and fair comparison.

Table 5.2: The CNN architecture for Mnist and FMnist

<b>Layer</b>	<b>Size</b>
Input	$1 \times 28 \times 28$
<i>Conv_1 + Relu</i>	$5 \times 5 \times 10$
Max Pooling	$2 \times 2$
<i>Conv_2 + Relu</i>	$5 \times 5 \times 20$
Max Pooling	$2 \times 2$
<i>FC<sub>1</sub> + Relu</i>	50
FC_2	10

Table 5.3: The CNN architecture for Cifar\_10

<b>Layer</b>	<b>Size</b>
Input	$3 \times 32 \times 32$
Conv_1 + Relu	$30 \times 30 \times 32$
Max Pooling	$15 \times 15 \times 32$
Conv_2 + Relu	$13 \times 13 \times 64$
Max Pooling	$6 \times 6 \times 64$
FC_1 + Relu	64
FC_2	10

The CNN layers are described as follows:

- The input size of the model refers to the dimensions of the input data that the model expects to receive. For example, for the setting dataset, the input size is (32, 32, 3). The first two numbers, 32x32, represent the spatial dimensions of the input image. Each image is 32 pixels wide and 32 pixels high. The last number, 3, represents the number of color channels in the input image. It indicates that the images are in RGB format, with three color channels (Red, Green, and Blue). So, the model expects input images with a resolution of 32x32 pixels and three color channels, which is a common size for images in the CIFAR-10 dataset [54, 27, 48, 1, 42, 49, 31, 24].

- The Convolutional Layer 1 + *ReLU* size refers to the output size of the first convolutional layer in the model after applying the Rectified Linear Unit (ReLU) activation function. The ReLU activation function introduces non-linearity, allowing the model to learn complex patterns and representations from the input data.
- The Max Pooling Layer 1 reduces the spatial dimensions of the input tensor. In *Cifar\_10* architecture, the Max Pooling Layer 1 is applied after Convolutional Layer 1. The Max Pooling operation helps in reducing the number of parameters and computational complexity in the network while preserving the most important features.
- The Convolutional Layer 2 + ReLU size refers to the spatial dimensions of the output tensor after applying Convolutional Layer 2 followed by the ReLU activation function. In *Cifar\_10*, the second Convolutional Layer is defined with 64 filters, each having a kernel size of (3, 3), and the ReLU activation function is applied after the convolution operation.
- The Fully Connected Layer 1 + ReLU refers to the number of neurons in the layer after applying the first fully connected (dense) layer followed by the ReLU activation function. In our setting, the first fully connected layer is defined with 64 neurons, and the ReLU activation function is applied after this dense layer.
- In our experiment, the "Fully Connected Layer 2" refers to the last layer of the neural network, also known as the output layer. This layer is a fully connected layer with 10 neurons, corresponding to the 10 classes in the settings. The output of this layer is used to make predictions for the input images. Each neuron in this layer represents a different class, and the final predicted class is the one with the highest activation value among these neurons.

### 5.3 Experiments

All experiments were conducted on an ASUS TUF Laptop equipped with a 3.30 GHz Intel Core i7 CPU, 1TB GB hard disk, and 8GB of memory. The experiments were performed on a Windows 10 Professional operating system with a 64-bit architecture. The code used for the experiments was implemented in Python 3.7, and the libraries used were TensorFlow 2.3.4 and TensorFlow federated 0.17.0; also, we used Keras and

numpy libraries with TensorFlow datasets. for experimental purposes, be advised that the pip must be updated in order to have the required libraries and packages such as grpcio, h5py, tensorboard and wheel.

## 5.4 Evaluation Metrics

To assess the performance of our proposed credit-based client selection method in federated learning, we employ two key evaluation metrics, the *Accuracy* and *Loss* functions. These metrics help us gauge the effectiveness and robustness of our approach. The accuracy metric as per equation (1) measures the overall performance of the global model on the test dataset. We evaluate the global model’s accuracy before and after applying our client selection method. The goal is to observe how the accuracy is affected when only a subset of clients, specifically 10 out of 100 clients, are selected to participate in each round of federated learning under different conditions. We perform 100 rounds of federated learning to obtain reliable results.

In addition to accuracy, we also consider the loss function in equation (2) as an evaluation metric. The loss function measures the discrepancy between the predicted outputs of the model and the actual labels of the data points in the training dataset. Similar to accuracy, we evaluate the global model’s loss function before and after applying the client selection method to assess the impact on model performance under various conditions. By analyzing the accuracy and loss function of the global model under different rounds and using our client selection method, we aim to demonstrate the effectiveness and efficiency of our approach in improving the performance of federated learning while reducing computational and communication overhead.

Overall, our evaluation process is designed to provide insights into the performance of our client selection method, considering accuracy, loss function, and the impact of selecting a subset of clients on the global model’s performance over 100 rounds of federated learning. The calculation of Precision and Recall is also given in Equations (3) and (4), respectively, as one of the common metrics for evaluating the performance of the models. Let True Positive (TP) be the number of instances that are correctly classified as positive by the model. In other words, these are the instances that belong to the positive class, and the model correctly predicted them as positive. True Negative (TN) be the number of instances that are correctly classified as negative by the model. These are the instances that belong to the negative class, and the model correctly predicted them as negative. Let False Positive (FP) be the number

of instances that are incorrectly classified as positive by the model. These are the instances that belong to the negative class, but the model incorrectly predicted them as positive. Lastly, let False Negative (FN) be the number of instances that are incorrectly classified as negative by the model. These are the instances that belong to the positive class, but the model incorrectly predicted them as negative.

$$\text{Accuracy}_c = \frac{\text{True\_Positive} + \text{True\_Negative}}{\text{True\_Positive} + \text{True\_Negative} + \text{False\_positive} + \text{False\_Negative}} \quad (1)$$

$$\text{Loss}_c = L = -\frac{1}{N} \sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (2)$$

$$\text{Precision}_c = \frac{\text{True\_Positive}}{\text{True\_Positive} + \text{False\_Positive}} \quad (3)$$

$$\text{Recall}_c = \frac{\text{True\_Positive}}{\text{True\_Positive} + \text{False\_Negative}} \quad (4)$$

## 5.5 Evaluation of the Defense Mechanisms

Firstly, we evaluated the performance of our model under normal conditions with the aggregation algorithm Federated averaging (FedAvg), without the credit-based client selection algorithm to determine the benchmark for our mechanism. Then, we evaluated our attack with different malicious weights and different numbers of poisoning rounds to demonstrate the impact of the attack without the defence mechanism. In the context of our attacks, the objective of the attacker is to reduce the global model’s accuracy on any input data. Thus, in this study, we quantify the impact of the attack by evaluating the difference in the global model’s testing accuracy before and after the attack. Then, we evaluated our model without the presence of the attacker, using our credit-based client selection approach to showcase the fidelity of our mechanism. Lastly, we consider the worst-case adversarial conditions for our defense mechanism to emphasize the robustness of our method, using different poisoning rounds and various malicious weights when applying our credit-based client selection method.

Note that we used the qualifying method for considering the clients; in each round, there are 10 clients out of 100 that participate in the current rounds, but our algorithm chooses different clients in each round. For demonstration purposes, we assigned

numbers 1 to 10 to the clients in each round, i.e., client number 1 in round number 2 is the client who had the first results in round number 2, so client number 1 in our table does not mean client with index 1 in the dataset, it means first client that was displayed in our output.

### 5.5.1 Experiments with FedAvg Algorithm without Credit-Based Client Selection

In this section, we present the benchmark results obtained from our experiments. The Federated Averaging algorithm (FedAvg) yielded an impressive accuracy of 90.1% on the Mnist dataset as seen in table 5.4, while for the FMnist dataset, we achieved an accuracy of 84.5% as shown in table 5.5. For the CIFAR-10 dataset in table 5.6, we evaluated two architectures, and the VGG-11 model emerged as the most successful one as seen in table 5.7, achieving a high accuracy of 69.8%. These benchmark results serve as the foundation for comparing the performance of our proposed approaches under different scenarios. In Figure 5.2 and Figure 5.1 we show the comparison of three datasets with their accuracy and also the comparison of the accuracy of Cifar-10 with two architectures.

Table 5.4: Mnist global model accuracy with FedAvg algorithm

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	80.6	80.5	80.4	79.6	80.9	81.6	80.7	79.3	80.4	80
Round 11-20	81.9	82.9	82.7	83.6	83.1	83.9	84	83.4	83	83.5
Round 21-30	83.5	84.9	84	84.1	84.3	84.7	83.6	83.9	83.5	83.6
Round 31-40	84.3	84.1	84.5	83.9	84.7	84.5	84.6	84.7	84.1	85
Round 41-50	84.9	85.1	85.3	85.7	85.9	85.6	85.9	85.1	85.7	85.6
Round 51-60	86.1	86.7	86.2	86.3	86.1	86.4	86.9	86.8	87	86.9
Round 61-70	86.9	87.1	87.6	87.9	87	87.3	87.4	87.5	87.6	87.6
Round 71-80	87.5	88.1	88.3	88.6	87.6	88.2	88.4	88.1	88.6	88.9
Round 81-90	89.3	89.1	89.5	89.4	89.7	89.1	89.6	89.3	89.1	89.4
Round 91-100	89.9	90.5	90.8	90.7	90.6	90.4	90.7	90.6	90.9	90.8



Table 5.5: FMnist global model accuracy with FedAvg algorithm

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	79.9	79.8	79.7	79.9	79.5	79.6	79.5	79.6	79.8	79.1
Round 11-20	79.9	79.7	80.2	80.5	79.9	80.9	80.7	80.8	80.6	80.1
Round 21-30	80.2	80.7	80.9	80.6	80.7	80.9	80.3	80.7	80.8	80.9
Round 31-40	81.1	80.9	81.3	81.4	81.8	81.7	81.6	81.6	81.7	81.6
Round 41-50	81.9	82	81.8	81.6	81.7	82.3	82.1	82	82.1	82.2
Round 51-60	82.3	82.6	82.4	82	82	82.1	82.4	82.7	82.5	82.6
Round 61-70	82.9	82.8	82.9	82.7	82.6	82.9	82.6	83.1	83	83.3
Round 71-80	83.2	83.6	83.6	83.9	83.9	83.8	83.7	83.8	83.9	83.4
Round 81-90	83.9	83.8	83.8	83.9	83.7	83.6	83.4	83.9	83.4	83.5
Round 91-100	84.9	84.3	84.7	84.9	84.6	84.7	84.1	84	84.4	84.3

Table 5.6: Cifar-10 global model accuracy with FedAvg algorithm

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	56.5	56.6	56.9	56.1	56.9	57.1	57.1	56.9	56.8	56.9
Round 11-20	57.5	57.8	57.8	57.6	57.2	57.6	57.3	57.9	57.8	57.9
Round 21-30	58.1	57.9	58.2	58.6	58.5	58.8	58.8	58.9	58.8	58.9
Round 31-40	58.8	58.7	59.1	59.3	59.4	59.6	59.4	59.1	59.2	59.3
Round 41-50	59.6	59.9	59.8	59.7	59.8	59.8	60.1	60.2	60.4	60.5
Round 51-60	60.3	60.5	60.9	61	61.3	61.8	61.9	61.7	61.9	61.9
Round 61-70	62.2	62.6	62.4	62.8	62.9	62.8	63	63.1	63.3	63.2
Round 71-80	63.4	63.6	63.8	63.9	63.3	63	63.4	63.7	63.5	63.7
Round 81-90	63.9	63.8	64.1	64.2	64.1	64.5	64.9	64.8	64.7	64.7
Round 91-100	64.5	64.8	64.7	64.2	64.5	64.3	64.4	64.8	64	64.1

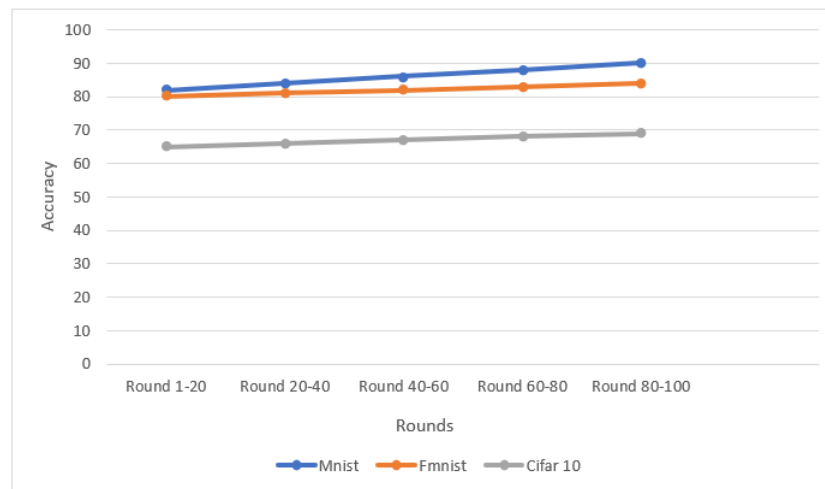


Figure 5.1: Testing accuracy of Mnist, FMnist and Cifar-10

## 5.5.2 Experiments with FedAvg Algorithm with Credit-Based Client Selection

In these experiments, we assessed the accuracy of the clients on three datasets: Mnist, FMnist, and CIFAR-10 (Tables 5.8, 5.9, and 5.10, respectively). Implementing the credit-based client selection method under normal circumstances showed almost identical results to the normal Federated Averaging (with less than 2% deviation). This

Table 5.7: Cifar-10 global model accuracy with VGG-1 with FedAvg algorithm

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	61.8	61.7	61.5	61.6	61.4	61.3	61.2	60.9	60.6	61.1
Round 11-20	62.9	62.8	62.6	62.9	62.5	62.3	62.4	62.7	62.8	62.8
Round 21-30	64.5	63.9	63.8	63.5	63.6	63.4	63.7	63.6	63.2	63
Round 31-40	64.9	64.8	64.7	64.5	64.8	64.8	64.5	64.8	64.8	64.2
Round 41-50	64.9	64.8	65.1	65.2	65	64.8	64.9	64.9	64.8	64.7
Round 51-60	66.7	66.8	66.9	66.5	66.3	66.5	65.9	65.7	65.6	65.6
Round 61-70	67.5	67.2	67.3	67.1	66.9	66.8	66.9	66.8	66.9	66.7
Round 71-80	67.9	68.1	68.1	67.4	67.5	67.8	67.5	67.3	67.5	67.6
Round 81-90	69.1	68.8	68.4	68.5	68.6	68.4	68.4	68.5	68.5	68.2
Round 91-100	69.3	69.1	69.7	69.3	69	68.9	69.2	69.8	68.9	69.7

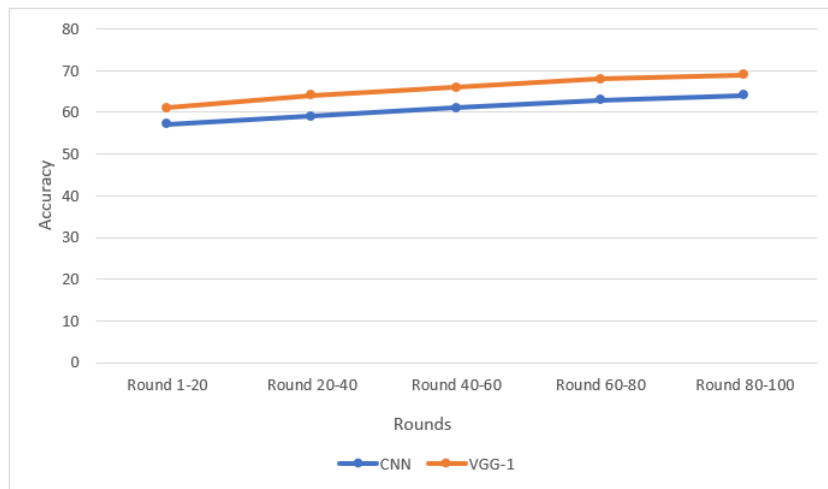


Figure 5.2: CNN and VGG-1 for Cifar-10

significant finding highlights one of the key contributions of our work, which is the preservation of model fidelity. Unlike many other defense approaches, where model performance typically decreases by more than 10%, our proposed method preserves the model’s accuracy. Furthermore, in Figure 5.3, we visually demonstrate the accuracy of these datasets using the credit-based client selection algorithm. The comparison of accuracy with and without the implementation of our approach is shown in Figures 5.4, 5.5, and 5.6, further illustrating the fidelity of our experimental results.

Table 5.8: Mnist global model accuracy with FedAvg algorithm with credit-based client selection

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	78.6	78.5	78.5	77.6	78.9	79.7	79.7	77.4	78.4	78
Round 11-20	79.9	80.8	80.8	81.6	81.1	81.8	82	81.3	81	79.5
Round 21-30	82.5	82.9	82.2	82.1	82.4	83.6	81.6	81.8	82.5	79.6
Round 31-40	82.3	82.1	82.6	81.8	82.7	82.4	82.6	82.9	82.1	81
Round 41-50	82.9	83.1	83.4	83.8	83.7	83.6	83.8	83.2	83.7	83.6
Round 51-60	84.2	84.8	84.3	84.3	84.1	84.5	84.8	84.8	85	84.9
Round 61-70	84.8	85.1	85.5	85.9	85	85.4	85.6	85.5	85.6	85.6
Round 71-80	85.5	861	86.3	86.6	85.6	85.2	85.4	85.1	85.6	85.9
Round 81-90	86.3	86.1	86.5	86.4	86.7	86.1	86.6	86.3	86.1	86.4
Round 91-100	87.2	87.5	87.8	87.8	86.6	86.6	87.7	87.6	87.9	87.5

Table 5.9: FMnist global model accuracy with FedAvg algorithm with credit-based client selection

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	77.8	77.8	77.6	77.9	77.5	77.6	77.5	77.6	77.8	77.8
Round 11-20	77.8	77.7	78.1	78.5	77.9	78.9	78.7	78.8	78.6	78.1
Round 21-30	78.1	78.7	78.8	78.6	78.7	78.9	78.3	78.5	78.8	78.4
Round 31-40	79	78.9	79.4	79.4	79.8	79.4	79.4	79.5	79.7	79.5
Round 41-50	79.8	80	79.9	79.6	79.7	80.3	80.1	80	80.1	80.1
Round 51-60	80.2	80.6	80.8	80.4	80	80.1	80.5	80.7	80.5	80.7
Round 61-70	80.8	80.8	80.5	80.7	80.5	80.9	80.6	81.1	81	81.5
Round 71-80	81.1	81.6	81.4	81.9	81.9	81.8	81.7	81.8	81.9	81.2
Round 81-90	81.8	81.8	81.9	81.9	81.2	81.6	81.4	81.9	81.1	81.1
Round 91-100	82.8	82.3	82.7	82.9	82.6	82.7	82.1	82	82.4	82.9

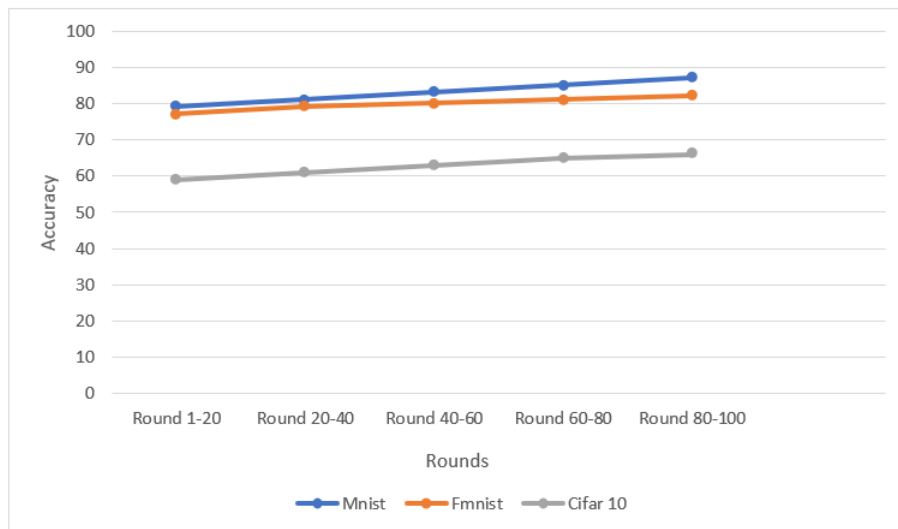


Figure 5.3: Testing accuracy of Mnist, FMnist and Cifar-10 with credit-based client selection

Table 5.10: Cifar-10 global model accuracy with FedAvg algorithm with credit-based client selection

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	59.4	59.7	59.5	59.6	59.4	59.3	59.2	58.9	58.6	59.3
Round 11-20	60.5	60.8	60.6	60.9	60.5	60.3	60.4	60.7	60.8	60.6
Round 21-30	63.7	61.9	61.8	61.5	61.6	61.4	61.7	61.6	61.2	61.3
Round 31-40	62.6	62.5	62.5	62.5	62.6	62.8	62.6	62.8	62.5	62.2
Round 41-50	62.1	62.2	62.2	62.3	63.6	63.8	63.7	63.6	63.9	63.1
Round 51-60	64.7	64.9	64.1	64.1	64.3	64.3	63.3	63.4	63.5	63.6
Round 61-70	65.4	65.2	65.3	65.3	64.9	64.1	64.9	64.2	64.9	64.6
Round 71-80	65.3	66.1	66.2	65.4	65.5	65.5	65.5	65.3	65.4	65.3
Round 81-90	67.2	66.4	66.4	66.5	66.6	66.2	66.4	66.3	66.5	66.1
Round 91-100	67.9	67.8	67.7	67.7	67	66.8	67.2	67.8	66.9	67.2

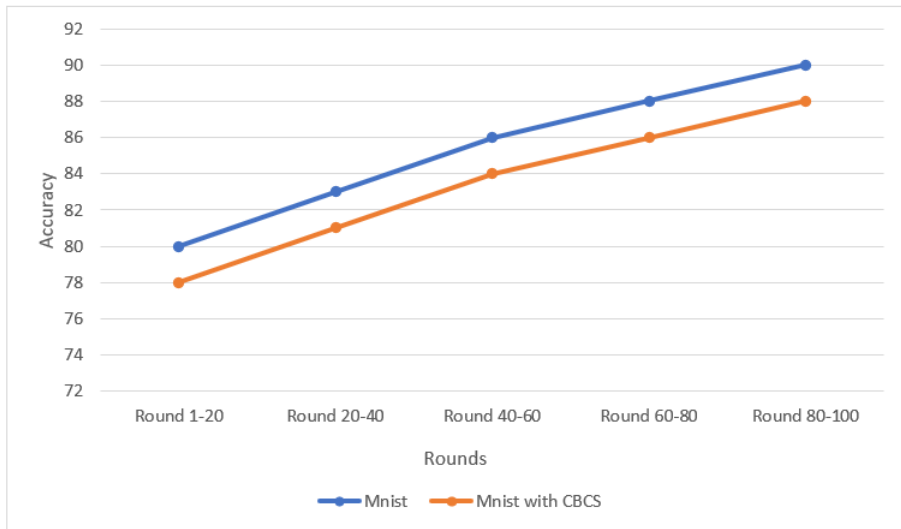


Figure 5.4: Testing accuracy of Mnist with and without credit-based client selection

### 5.5.3 Experiments with FedAvg Algorithm under our Attack Model without Credit-Based Client Selection

In this section, we present a comprehensive evaluation of our attack model’s impact on the accuracy of the global model over 100 rounds. The results showcased in the tables below (Mnist (Tables: 5.11 and 5.12), FMnist (Tables: 5.13 and 5.14) and Cifar-10 (Tables: 5.15 and 5.17)) demonstrate the effectiveness of our attack in decreasing the global accuracy for all three datasets: Mnist, FMnist, and CIFAR-10. We conducted experiments with two different malicious weights ( $\omega$ ) for each dataset, with the value of  $\omega = 0.3$  and  $\omega = 10$ . The results clearly indicate that the higher positive value of the malicious weight ( $\omega$ ) leads to a greater impact on the final round. In other words, a higher malicious weight enables the attacker to exert more influence on the global model, resulting in a more pronounced decrease in accuracy.

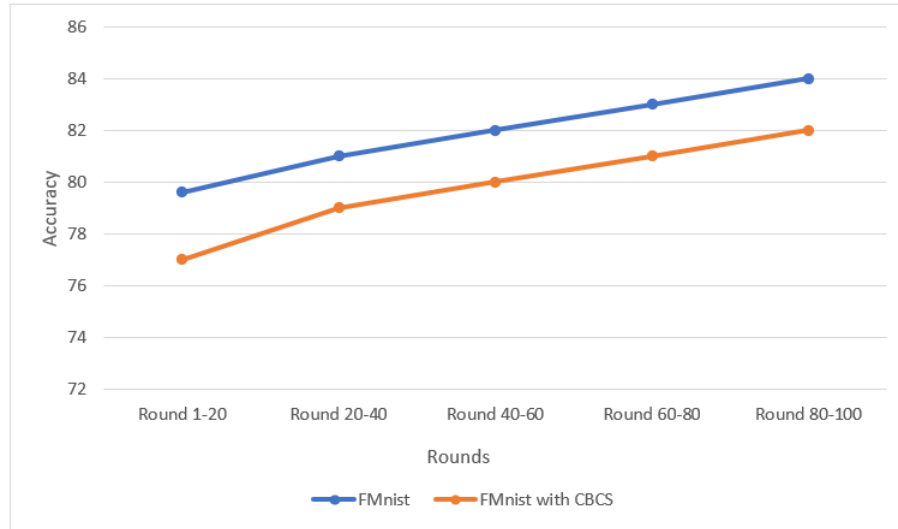


Figure 5.5: Testing accuracy of FMnist with and without credit-based client selection

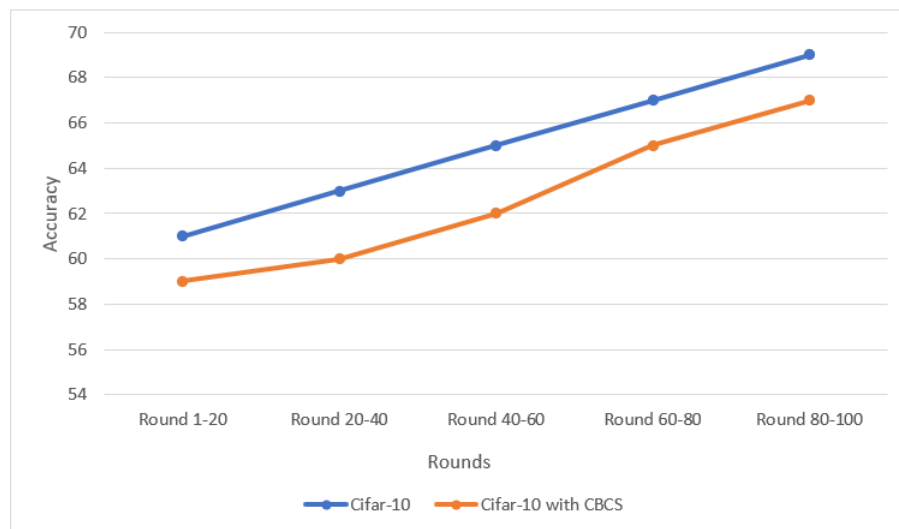


Figure 5.6: Testing accuracy of Cifar-10 with and without credit-based client selection

To implement our attack, we strategically chose one of the poor-performing clients from each dataset as the poisoned dataset. Since multiple clients in most datasets have poor data that can potentially degrade the global model accuracy, we leveraged this flaw by evaluating the accuracy of each client over 100 rounds and selecting those with the lowest accuracy for poisoning as our poisoned data. In addition, we demonstrate a comparison of the impact of different malicious weights for each dataset through Figures 5.7, 5.8, and 5.9. These visual representations illustrate how the malicious weight  $\omega$  affects the accuracy, emphasizing the increasing impact with higher weight

values.

Table 5.11: Testing accuracy of Mnist with our attack model ( $\omega = 0.3$ ) in %

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	50.3	49.6	48.7	49.3	49.8	50.1	50.3	50.7	49.4	49.2
Round 11-20	40.1	40.6	40.8	40.6	40.4	40	40.7	40.5	40.8	40.1
Round 21-30	38.5	38.6	38.4	38.6	38.4	35.6	38.1	38.2	37.6	37.5
Round 31-40	36	36.5	36.7	35.6	31.6	33.8	33.7	35.1	36.5	36.1
Round 41-50	33.6	33.5	34.5	36.1	35.1	32.1	32.9	33.6	33.7	31.2
Round 51-60	31.5	31.6	31.5	31.6	31.8	31.9	31.7	30.1	30.2	30.4
Round 61-70	29.6	29.4	28.6	28.6	28.1	28.7	28.1	29.3	27.6	25.9
Round 71-80	27.3	27.6	28.1	28.3	27.6	27.1	26.3	26.1	27	26.7
Round 81-90	26.9	26.6	26.4	26.1	27.5	26.3	27.9	27.8	27.6	27.1
Round 91-100	27.3	27.6	27.6	27.1	27.2	27.4	27.5	27.4	27.9	27.1

Table 5.12: Testing accuracy of Mnist with our attack model ( $\omega = 10$ ) in %

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	45.7	45.6	45.3	45.6	45.7	45.1	44.9	44.1	44.8	44.4
Round 11-20	40.1	40.6	40.3	40.9	40.6	40.9	40.1	39.1	39.1	39
Round 21-30	40.9	39.1	39.2	37	36.1	36.8	36.5	36.8	35.1	34.1
Round 31-40	27.6	27.9	27.1	27.6	27	26.6	26.1	26.8	26.2	26.7
Round 41-50	25.7	25.8	25.6	25.1	25.7	25.1	27.9	27	24.3	27.1
Round 51-60	30.5	31.6	28.8	29.9	28	25	24.9	24.1	25.3	26.8
Round 61-70	21.6	24.3	22.4	23.1	22.6	22.1	21.5	24.6	24.2	24.1
Round 71-80	27.2	25.1	27.6	24.3	27.9	27.6	21.6	20.9	20.4	20.7
Round 81-90	25.4	24.9	25.6	25.4	25.9	25.5	24.8	25	25.1	25.6
Round 91-100	24.3	24.3	24.8	24.6	25.1	23.9	24.6	24.5	24.4	24

Table 5.13: Testing accuracy of FMnist with our attack model ( $\omega = 0.3$ ) in %

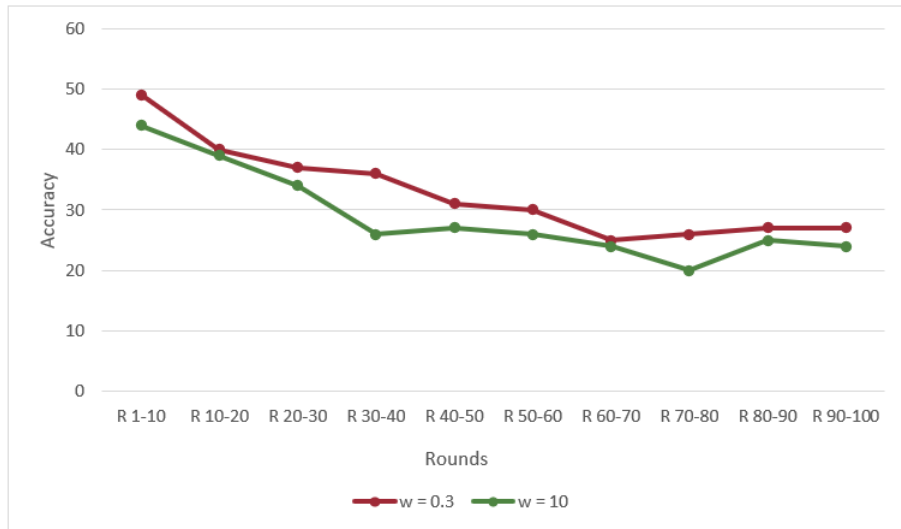
Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	48.2	48.5	48.2	48.3	48.8	48.9	48.1	48.3	48.1	48
Round 11-20	34.3	34.2	34.1	34.8	34.1	34.8	34.8	33.1	34.6	33.1
Round 21-30	29.5	29.1	28.5	28.9	27.5	25	27.9	28.1	28.8	27.6
Round 31-40	27.2	27.1	27.6	27.1	27.9	27.6	27.1	27.6	27.1	27.6
Round 41-50	32.3	32.6	32.5	32.6	32.8	32.6	30.8	30.9	31.6	30.9
Round 51-60	26.3	26.8	26.7	26.3	26.7	26.8	26.1	26.7	26.9	26.1
Round 61-70	30.2	30.6	30.7	30.5	30.7	30.6	30.3	30	30	30.1
Round 71-80	28.9	28.6	28.1	28.5	28.1	28.6	28.1	28.6	28.3	28
Round 81-90	25.4	25.6	25.1	25.9	25.1	25.3	25.7	25.3	25.1	25.7
Round 91-100	27.1	27.6	27.6	27	26.6	27.9	27.7	27.8	27.1	26.2

Table 5.14: Testing accuracy of FMnist with our attack model ( $\omega = 10$ ) in %

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	55.3	55.6	55.1	55.8	55.5	55	54.1	55.2	54.9	54.2
Round 11-20	28.2	28.6	28.1	28.3	28.1	28.7	28.6	28.1	28.6	28.3
Round 21-30	27.2	27.6	27.1	27.5	27.6	27.1	27.9	27.1	27.6	27.6
Round 31-40	24.6	24.9	24.6	24.8	24	24.6	23.9	24.7	24	24.1
Round 41-50	24.8	24.6	24.6	24.8	23.9	24.1	23.8	23.3	24.1	24
Round 51-60	24.6	23.6	23.9	23.7	23.5	22.1	22.9	22.7	22.5	22
Round 61-70	27.1	26.5	27.1	27.6	27.1	27.9	27.2	27.6	27.8	27.1
Round 71-80	20.3	20.6	20.6	20.9	20.1	20.7	20.6	20.5	19.7	18.9
Round 81-90	19.1	18.6	17.6	19.2	19.9	18.5	17.6	17.9	17.8	17.6
Round 91-100	17.3	17.8	17.6	17.1	17.5	17.8	17.1	17.3	17.5	17.8

Table 5.15: Testing accuracy of Cifar-10 with our attack model ( $\omega = 0.3$ ) in %

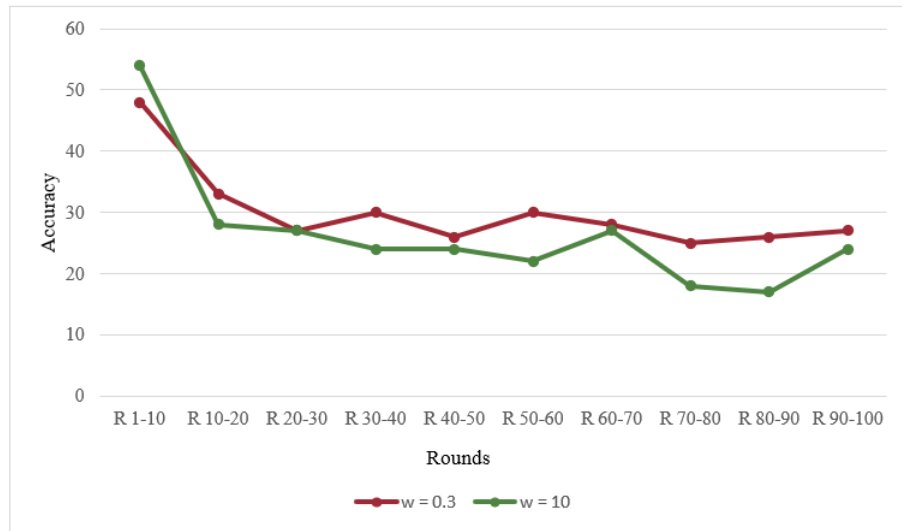
Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	30	29.6	29.7	29.5	30.1	30.5	29.9	29.4	29.3	29.8
Round 11-20	28.3	28.1	28.9	27.9	29.1	28.6	28.1	28.1	28	27.9
Round 21-30	26.9	26.8	26.1	26.7	26.8	26.1	26	25.7	25.1	25.5
Round 31-40	23.2	24.1	23.5	23.1	24.1	23.9	24.1	24.5	23.4	24.2
Round 41-50	19.6	19.8	18.4	17.5	16.8	17.7	16.8	17	16.9	16.7
Round 51-60	13.4	12.9	12.9	12	12.7	13.7	13.5	13.1	12.8	13
Round 61-70	11.5	11.6	11.8	11.7	11.7	11.8	11.7	11.8	11.9	11.2
Round 71-80	12.6	12	12	13.4	12.4	12.5	13.1	12.9	12.1	11.9
Round 81-90	11.6	11.5	11.6	11.4	11.6	11.7	11.8	11.5	11.6	11.8
Round 91-100	10.3	11.1	11.8	11.5	10.5	10.7	10.8	10.6	10.7	10

Figure 5.7: Testing accuracy of Mnist under attack ( $\omega = 0.3$  and  $\omega = 10$ )

Furthermore, we evaluate the performance of each dataset under three different scenarios: the normal Federated Averaging algorithm, the Federated Averaging algorithm with credit-based client selection, and under the attack model. Figures 5.10, 5.11, and 5.12 provide a comprehensive comparison of these scenarios, demonstrat-

Table 5.16: Testing accuracy of Cifar-10 with our attack model ( $\omega = 10$ ) in %

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	27.5	27.1	27.6	27.8	27.9	27.1	27.8	27.2	27.1	27
Round 11-20	20.6	19.3	19.8	20.8	20.7	20.3	20.5	20.9	20.4	19.8
Round 21-30	20.3	20.8	20.2	20.7	20.6	20.6	20	20	21.1	20.7
Round 31-40	19.6	19.8	19.6	19.7	19.2	19.3	19.7	19.3	19.7	19.2
Round 41-50	14.2	14.8	14.6	14.2	14.9	14.1	14.3	15.1	14.8	14.9
Round 51-60	15.6	15.7	15.6	15.7	15.9	15.1	15.3	15.2	15.7	15.3
Round 61-70	12.5	12.1	11.9	12.5	12.5	12.9	12	12.3	12.1	12.8
Round 71-80	11.3	11.8	11.8	11.7	11.9	11.1	11.6	11.8	10.9	11.1
Round 81-90	10.3	10.5	10.9	10.8	9.9	9.7	9	10.1	10.2	10.8
Round 91-100	8.1	7.5	8.3	8.6	8.7	8	8	7.5	7.1	8.2

Figure 5.8: Testing accuracy of FMnist under attack ( $\omega = 0.3$  and  $\omega = 10$ )

ing the effectiveness of our attack in downgrading the model’s accuracy, while also showcasing the benefits of credit-based client selection in maintaining model fidelity under normal circumstances.

Overall, our evaluation sheds light on the success of our attack model in reducing the global model’s accuracy and highlights the significance of malicious weight  $\omega$  in determining the level of impact. Additionally, the comparison among different scenarios underscores the robustness of our proposed approach in adversarial settings and its potential to enhance the security of federated learning systems.



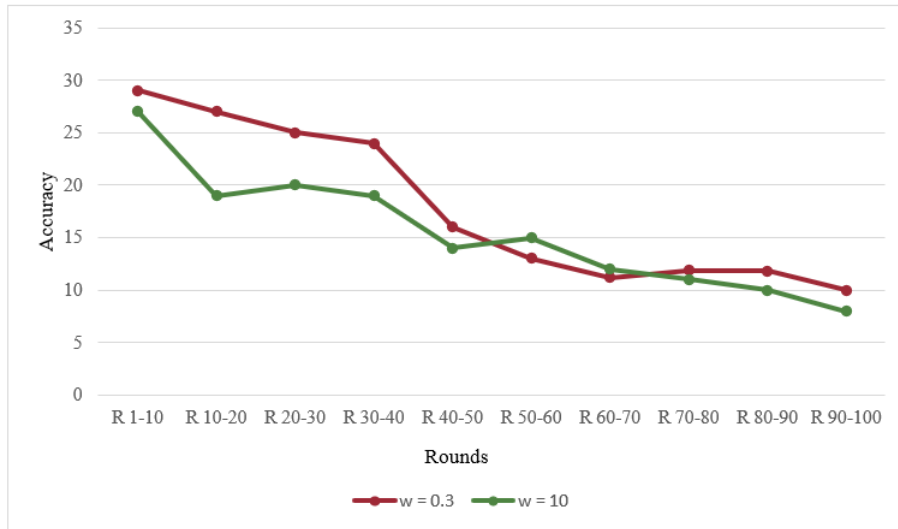


Figure 5.9: Testing accuracy of Cifar-10 under attack ( $\omega = 0.3$  and  $\omega = 10$ )

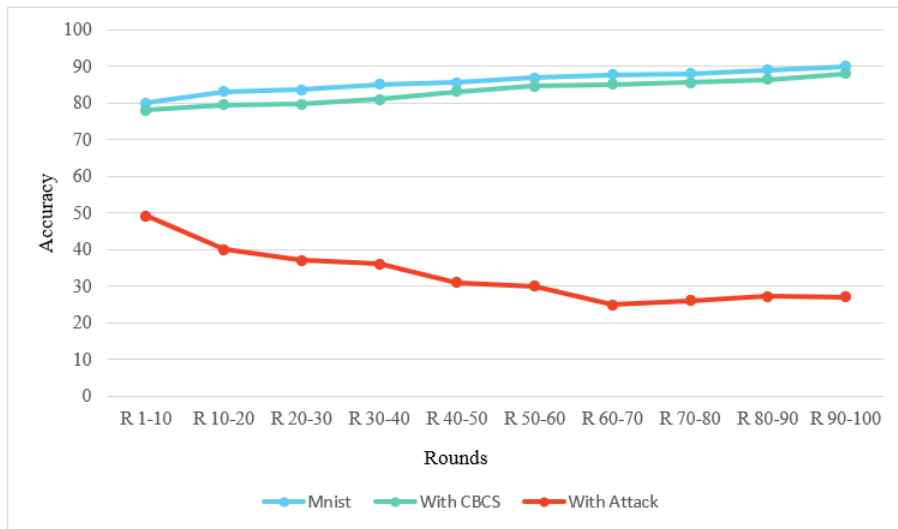


Figure 5.10: Testing accuracy of Mnist, Mnist with CBCS, Mnist under attack

#### 5.5.4 Experiments with FedAvg Algorithm and our Attack Model with Credit-Based Client Selection

In this section, we present the evaluation of our defense strategy under the most challenging conditions, wherein we employ poisoning attacks in every round with the highest malicious weight, denoted as  $\omega = 10$ . To assess our defense strategy, we conduct experiments by varying the loss parameter  $\zeta$  with two different values,  $\zeta = 0.1$  and  $\zeta = 10$ , on three datasets: Mnist, FMnist, and CIFAR-10. Tables

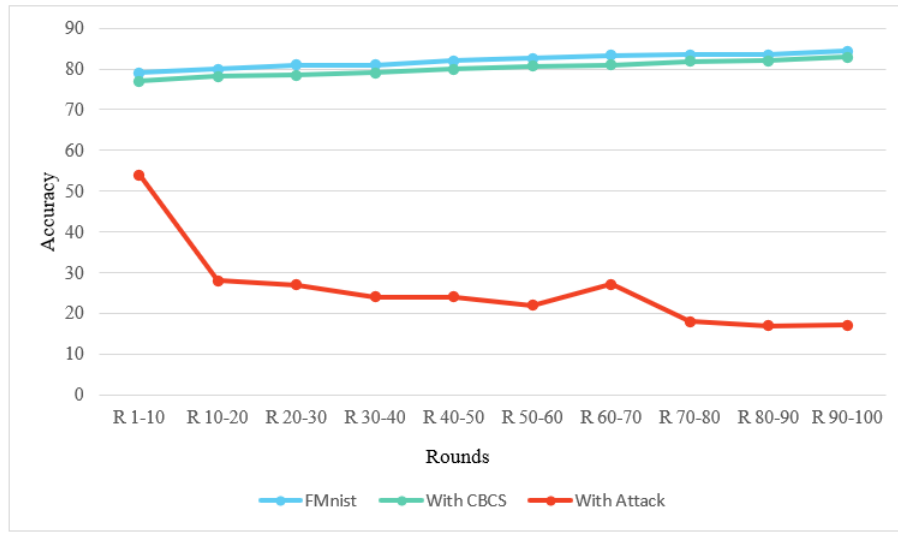


Figure 5.11: Testing accuracy of FMnist, FMnist with CBCS, FMnist under attack

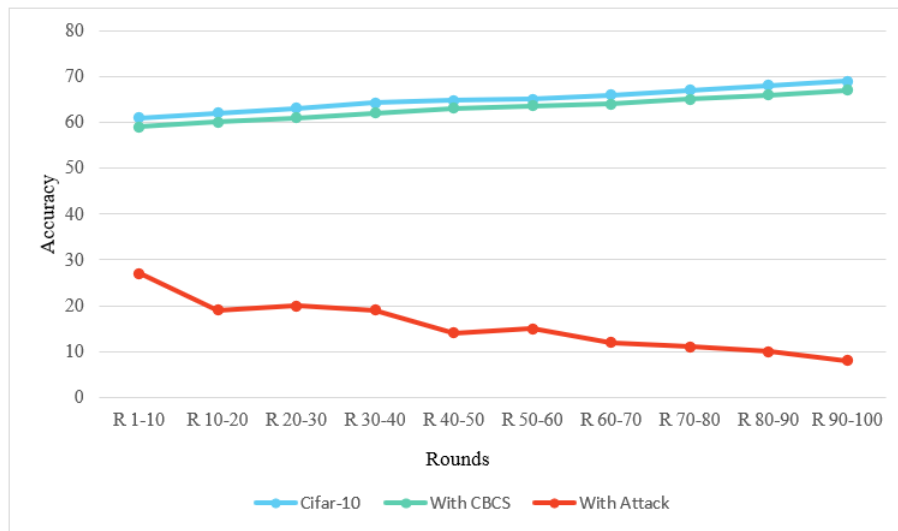


Figure 5.12: Testing accuracy of Cifar-10, Cifar-10 with CBCS, Cifar-10 under attack

5.17 and 5.18 illustrate the results for the Mnist dataset, while Tables 5.19 and 5.20 depict the outcomes for FMnist, and Tables 5.21 and 5.22 represent the results for CIFAR-10.

Table 5.17: Mnist global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 0.1$ )

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	78.5	77.5	78.5	77.6	77.9	78.7	79.7	77.4	78.4	78
Round 11-20	79.9	80.8	80.8	81.6	81.1	81.8	81	81.3	81	79.5
Round 21-30	82.5	82.9	81.2	82.1	82.4	81.6	81.6	81.8	82.5	79.6
Round 31-40	82.3	82.0	82.5	81.8	82.7	82.4	82.6	82.9	82.1	81
Round 41-50	82.9	82.1	82.1	83.8	82.7	83.6	83.8	83.2	83.7	82.6
Round 51-60	84.2	83.8	83.1	84.2	84.1	84.5	84.8	84.8	84	83.9
Round 61-70	84.6	85.1	84.4	84.9	84	85.4	85.6	84.5	84.6	83.6
Round 71-80	85.5	86.1	86.2	86.6	85.6	85.2	85.4	85.1	85.6	84.9
Round 81-90	85.3	86.1	86.5	85.4	85.7	86.1	86.6	86.3	85.1	86.4
Round 91-100	88.2	88.5	88.8	87.8	86.6	86.6	87.7	87.6	87.9	87.5

Table 5.18: Mnist global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 10$ )

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	78.1	77.5	78.5	77.6	78.9	78.7	78.7	76.4	78.4	77
Round 11-20	79.1	79.8	80.8	81.6	81.1	81.8	82	81.3	81	78.5
Round 21-30	82.8	81.9	82.2	82.1	82.4	82.6	81.6	81.8	82.5	78.6
Round 31-40	82.6	82.1	82.6	81.8	82.7	82.4	82.6	82.9	82.1	80
Round 41-50	82.6	83.1	82.4	83.8	83.7	83.6	83.8	83.2	83.7	82.6
Round 51-60	84.1	83.8	83.3	84.3	84.1	83.5	84.8	83.8	85	83.9
Round 61-70	84.4	84.1	84.5	83.9	83	83.4	84.6	85.5	84.6	84.6
Round 71-80	85.1	85.1	86.3	85.6	84.6	85.2	85.4	85.1	85.6	84.9
Round 81-90	85.2	85.1	85.5	85.4	85.7	85.1	85.6	85.3	85.1	84.4
Round 91-100	85.1	85.5	85.8	85.8	85.6	85.6	85.7	85.6	85.9	85.5

From our findings, two key insights emerge. Firstly, decreasing the value of  $\zeta$  maintains high accuracy levels and yields more stable results. This observation validates the importance of incorporating the loss parameter in our credit-based score calculation, as it indicates the model’s performance on the given dataset. Higher loss values for clients in the model would result in significant differences between  $\zeta = 0.1$  and  $\zeta = 10$ . However, the marginal difference in our results demonstrates the robustness of our model and highlights the significance of the loss parameter in determining the best-performing clients.

Additionally, we compare our credit-based client selection method under different  $\zeta$  values in the presence of the attack. Figure 5.13 showcases the results for the Mnist dataset, Figure 5.14 for FMnist, and Figure 5.15 for CIFAR-10. The slight differences observed in the accuracies for different  $\zeta$  values emphasize the consistency of our approach.

Table 5.19: FMnist global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 0.1$ )

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	80.7	79.8	79.6	79.9	80.5	80.6	80.5	80.6	80.8	80.8
Round 11-20	80.7	80.7	80.1	80.5	80.9	80.9	81.7	81.8	81.6	80.1
Round 21-30	80.6	81.7	78.8	78.6	81.7	81.9	81.3	81.5	80.8	80.4
Round 31-40	81.5	81.9	81.4	81.4	81.8	80.4	81.4	81.5	82.7	81.5
Round 41-50	79.4	80	79.9	81.6	81.7	80.3	80.1	80	80.1	82.1
Round 51-60	80.3	80.6	80.8	80.4	80	80.1	80.5	80.7	80.5	82.7
Round 61-70	80.7	80.8	80.5	81.7	81.5	79.9	79.6	79.1	79	82.5
Round 71-80	80.1	79.6	80.4	80.9	80.9	80.8	80.7	80.8	80.9	83.2
Round 81-90	80.9	81.8	81.9	81.9	81.2	81.6	81.4	80.9	82.1	83.1
Round 91-100	83.9	83.3	83.7	82.9	82.6	83.7	83.1	83	82.4	83.9

Table 5.20: FMnist global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 10$ )

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	77.7	77.8	77.6	77.9	77.5	77.6	77.5	77.6	77.8	77.8
Round 11-20	78.7	77.7	78.1	78.5	77.9	78.9	78.7	78.8	78.6	78.1
Round 21-30	78.2	78.7	78.8	78.6	78.7	78.9	78.3	78.5	78.8	78.4
Round 31-40	79.3	78.9	79.4	79.4	79.8	79.4	79.4	79.5	79.7	79.5
Round 41-50	79.7	80	79.9	80.6	79.7	80.3	80.1	80	80.1	80.1
Round 51-60	80.7	80.6	80.8	80.4	80	80.1	80.5	80.7	80.5	80.7
Round 61-70	81.4	81.8	81.5	81.7	80.5	81.9	80.6	81.1	81	81.5
Round 71-80	81.2	81.6	80.4	81.9	81.9	80.8	81.7	81.8	81.9	81.2
Round 81-90	81.4	81.8	81.9	81.9	81.2	81.6	81.4	81.9	81.1	81.1
Round 91-100	80.5	81.3	81.7	82.9	81.6	82.7	82.1	82	80.4	81.9

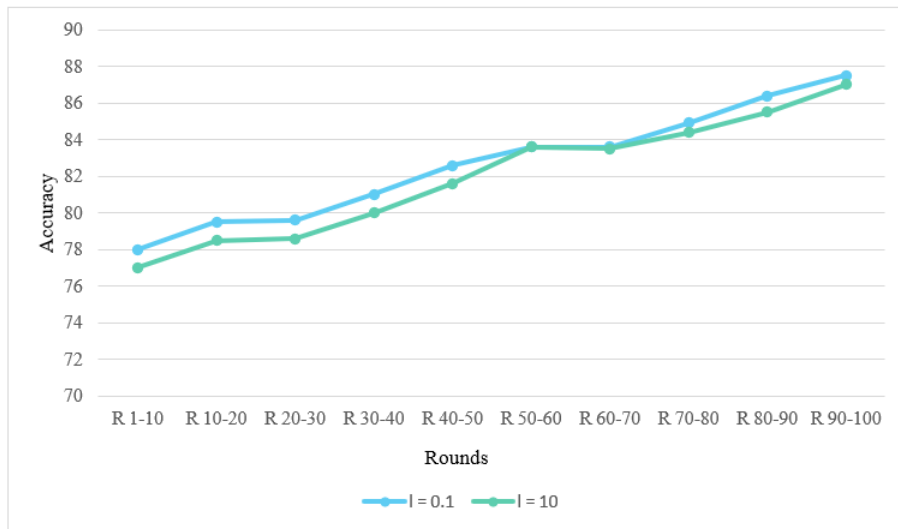


Figure 5.13: Comparison of accuracy of Mnist under attack ( $\omega = 10$ ) using our credit-based client selection defence ( $\zeta = 0.1$  and  $\zeta = 10$ )

Finally, we present a comparison of three conditions: (1) Training our model with the Federated Averaging (FedAvg) algorithm without any attack or client selection approach, (2) Training our model with federated averaging and the client selection

Table 5.21: Cifar-10 global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 0.1$ )

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	59.6	59.7	59.5	59.6	59.4	59.3	59.2	58.9	58.6	59.3
Round 11-20	60.4	60.8	60.6	60.9	60.5	60.3	60.4	60.7	60.8	60.6
Round 21-30	61.3	60.9	61.8	60.5	61.6	61.4	61.7	61.6	60.2	61.3
Round 31-40	61.4	62.5	62.5	62.5	62.6	62.8	62.6	62.8	61.5	62.2
Round 41-50	61.7	62.2	62.2	62.3	63.6	61.8	62.7	62.6	62.9	62.1
Round 51-60	62.8	62.9	26.1	63.1	63.3	63.3	62.3	63.4	63.5	63.6
Round 61-70	63.1	65.2	63.3	63.3	64.9	63.1	64.9	63.2	64.9	63.6
Round 71-80	64.3	64.1	64.2	65.4	65.5	65.5	65.5	64.3	65.4	64.3
Round 81-90	65.4	65.4	65.4	65.5	66.6	65.2	66.4	66.3	66.5	65.1
Round 91-100	66.7	66.8	66.7	66.7	65	66.8	66.2	65.8	66.9	66.2

Table 5.22: Cifar-10 global model accuracy under attack ( $\omega = 10$ ) with credit-based client selection ( $\zeta = 10$ )

Rounds	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8	Client 9	Client 10
Round 1-10	59.3	59.7	59.5	59.6	59.4	59.3	59.2	58.9	58.6	59.3
Round 11-20	59.7	59.8	60.6	59.9	59.5	59.3	60.4	60.7	59.8	59.6
Round 21-30	59.2	60.9	61.8	59.5	61.6	60.4	61.7	60.6	61.2	60.3
Round 31-40	59.7	60.5	60.5	59.5	60.6	61.8	60.6	60.8	60.5	60.2
Round 41-50	60.2	60.2	59.2	60.3	60.6	60.8	60.7	60.6	60.9	61.1
Round 51-60	61.5	61.9	60.1	60.1	61.3	61.3	60.3	61.4	60.5	60.6
Round 61-70	61.6	61.2	61.3	61.3	61.9	61.1	61.9	61.2	61.9	61.6
Round 71-80	62.1	62.1	61.2	63.4	62.5	62.5	61.5	61.3	62.4	62.3
Round 81-90	63.9	62.4	62.4	62.5	63.6	62.2	62.4	62.3	62.5	63.1
Round 91-100	63.3	63.8	64.7	64.7	63	64.8	63.2	64.8	64.9	63.2

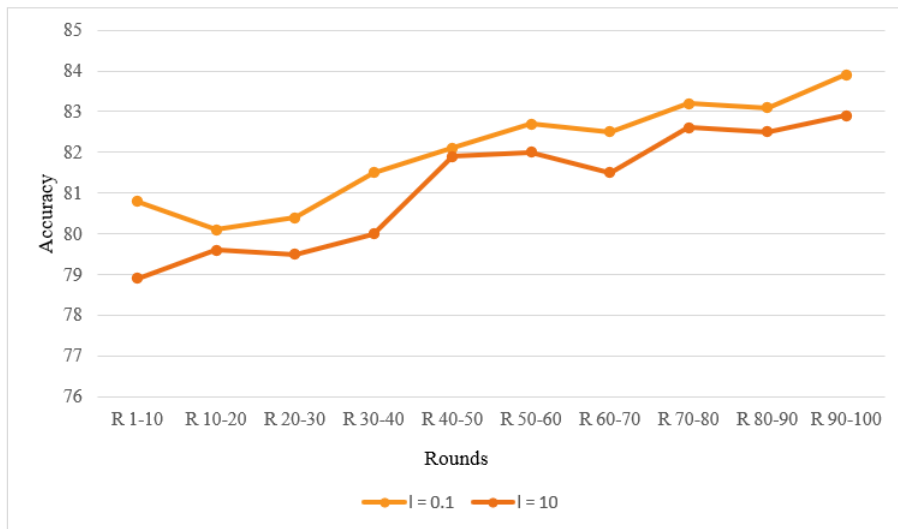


Figure 5.14: Comparison of accuracy of FMnist under attack ( $\omega = 10$ ) using our credit-based client selection defence ( $\zeta = 0.1$  and  $\zeta = 10$ )

method, and (3) Training our model under attack using credit-based client selection. Figures 5.16, 5.17, and 5.18 showcase the results for each dataset. This comparison underscores the effectiveness of our credit-based client selection method and high-

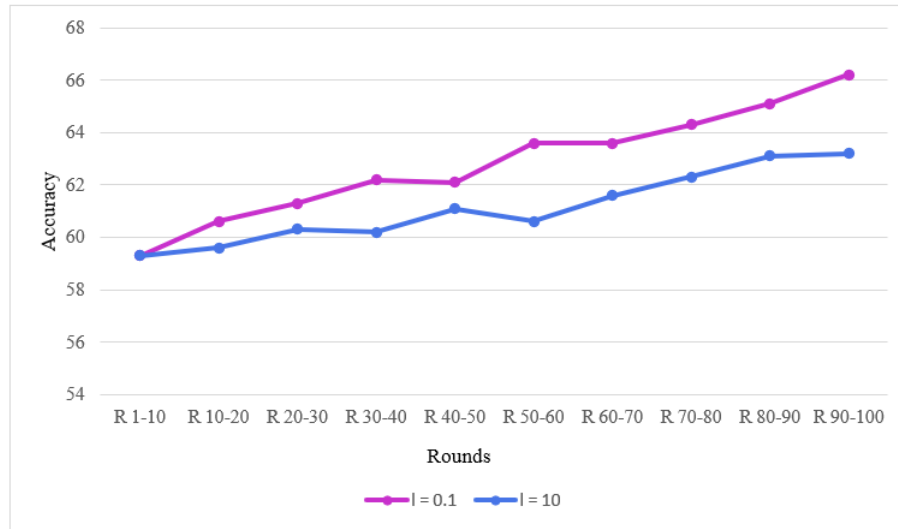


Figure 5.15: Comparison of accuracy of Cifar-10 under attack ( $\omega = 10$ ) using our credit-based client selection defence ( $\zeta = 0.1$  and  $\zeta = 10$ )

lights its significance in improving model performance and mitigating the impact of poisoning attacks.

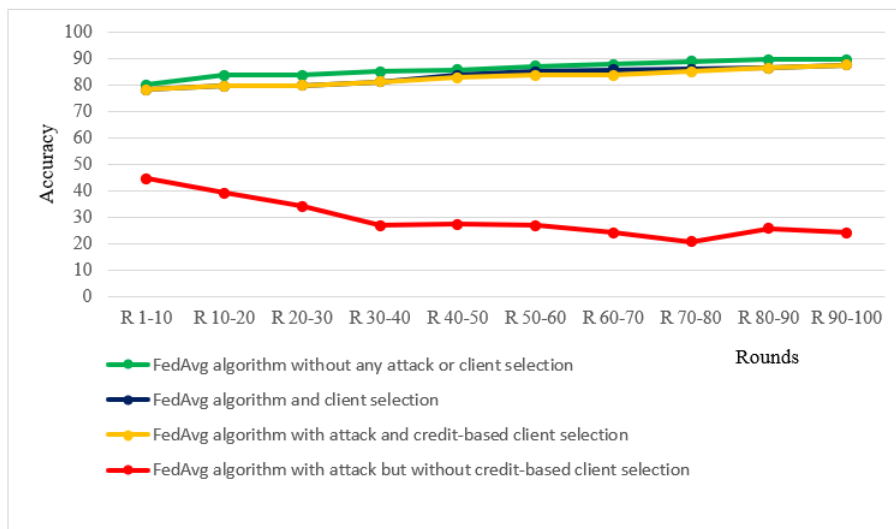


Figure 5.16: Comparison of Mnist with FedAvg, Mnist with CBCS, Mnist with only the attack and Mnist with attack + CBCS ( $\omega = 10$  and  $\zeta = 0.1$ )

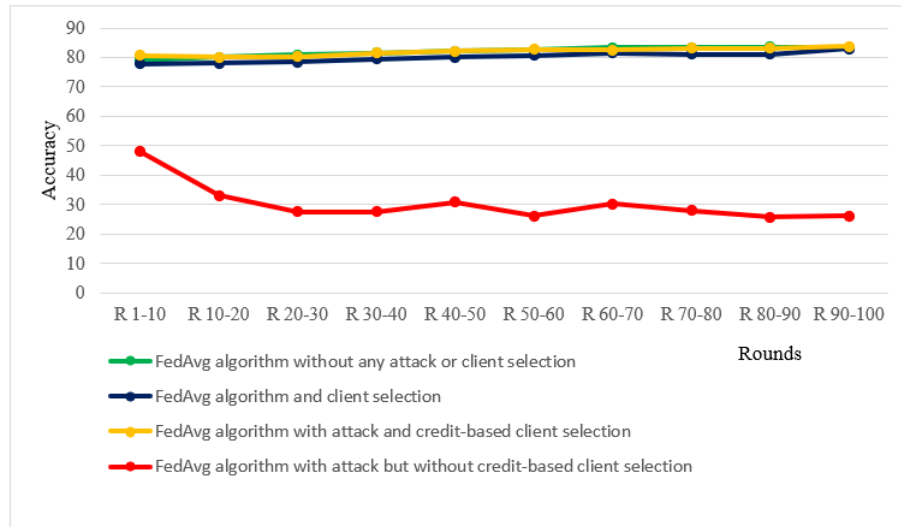


Figure 5.17: Comparison of FMnist with FedAvg, FMnist with CBCS, FMnist with only the attack and FMnist with attack + CBCS ( $\omega = 10$  and  $\zeta = 0.1$ )

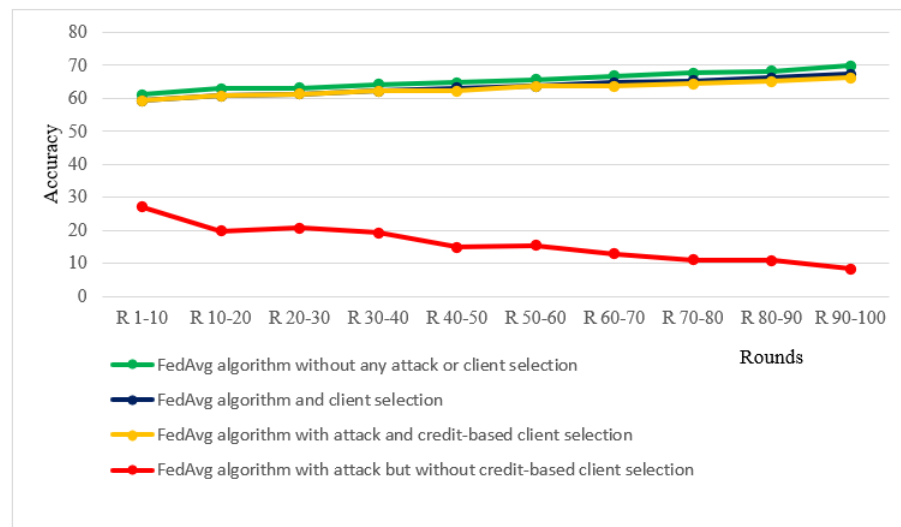


Figure 5.18: Comparison of Cifar-10 with FedAvg, Cifar-10 with CBCS, Cifar-10 with only the attack and Cifar-10 with attack + CBCS ( $\omega = 10$  and  $\zeta = 0.1$ )

## Chapter 6

# Conclusion and Future Work

In this thesis, we introduce a credit-based client selection method for federated learning, aimed at enhancing model performance while effectively countering the impact of poisoning attacks. Our approach involves calculating a credit score that considers both the accuracy and loss metrics of each client’s local model. By strategically eliminating poorly performing models and potential attackers, we achieve a more robust and accurate global model. Extensive experiments conducted on three diverse datasets, namely Mnist, FMnist, and CIFAR-10, demonstrate the effectiveness and stability of our approach even in the presence of non-IID and imbalanced data distributions.

The key highlight of our research lies in the ability of our credit-based client selection to maintain the fidelity of the global model, even under challenging attack scenarios. The incorporation of loss as a crucial factor in the credit score calculation proves instrumental in identifying the most reliable and high-performing clients. Moreover, our method exhibits remarkable consistency across diverse parameter settings, rendering it a reliable defense strategy. However, our contributions extend beyond the introduction of credit-based client selection. We conducted an in-depth evaluation of the performance of the widely-used Federated Averaging (FedAvg) algorithm under various scenarios. This allowed us to understand its limitations and pave the way for the development of our new defense strategy.

As we look towards the future, our credit-based client selection holds great promise, this method offers a promising solution to enhance federated learning, striking a delicate balance between security and efficiency., but there are still opportunities for further exploration and enhancements. An exciting avenue involves testing the robustness of our defense strategy against more advanced adversarial attacks, including



membership inference attacks. Additionally, optimizing the selection of hyperparameters like the loss parameter  $\zeta$  and malicious weight  $\omega$  can potentially amplify the performance of our approach.

Furthermore, scalability remains a critical consideration, and we plan to extend our evaluation to larger-scale federated learning settings with a diverse range of clients and data distributions. Real-world applicability is also a crucial aspect to explore, especially in resource-constrained environments. Addressing any computational overheads and evaluating the efficiency of our approach in practical scenarios will be valuable for its broader adoption.

It is important to acknowledge that federated learning is a rapidly evolving field [9]. As we draw inspiration from recent research in the area, it becomes evident that our innovative approach holds immense potential. With technology advancing rapidly, we anticipate exciting developments and new opportunities to explore federated learning in various domains, including privacy-sensitive applications and edge computing environments.

# Appendix A

## Selected Code Snapshots

In this appendix, we provide different sections of our Python code for demonstration purposes. These sections showcase key functionalities and algorithms implemented in our code-base.

### A.1 Attack Model

As previously mentioned, the provided code demonstrates the implementation of our attack model, which incorporates the attacker weight and the poisoned data of the FMnist dataset. This allows us to showcase the effectiveness of our attack strategy and its impact on the model’s performance as shown in figures A.1 and A.2.

```
# Federated averaging function with credit-based client selection
def federated_avg(models, client_weights, attacker_weight=1):
    new_model = create_model()
    num_models = len(models)
    model_weights_shape = [w.shape for w in new_model.get_weights()]
    weights_sum = [np.zeros(shape) for shape in model_weights_shape]

    total_weights = sum(client_weights)

    for i, model in enumerate(models):
        weight = client_weights[i] / total_weights
        model_weights = model.get_weights()
        weights_sum = [w + weight * model_weight for w, model_weight in zip(weights_sum, model_weights)]

    if attacker_weight is not None:
        attacker_weight_resaped = [np.reshape(w, shape) for w, shape in zip(attacker_weight, model_weights_shape)]
        weights_sum = [w + attacker_w for w, attacker_w in zip(weights_sum, attacker_weight_resaped)]

    averaged_weights = [w / num_models for w in weights_sum]
    new_model.set_weights(averaged_weights)
    return new_model
```

Figure A.1: Python code for the implementation of the attack model

```

# Updated federated learning process with model poisoning and improved credit-based client selection
def federated_learning_with_poisoning(num_rounds=10, num_clients=10, poison_round=1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10):
    models = [create_model() for _ in range(num_clients)]
    attacker_model = create_model() # Attacker creates a model to generate malicious weights
    attacker_data = federated_train_data[0] # Data from the poisoned client

    attacker_model_weights = 1 # Initialize the attacker's model weights

    client_weights = [1.0] * num_clients # Initialize client weights with equal weight

    for round_num in range(num_rounds):
        print(f"Round {round_num + 1}/{num_rounds}")
        # Sort clients based on their current weights (higher weights are better)
        sorted_clients = np.argsort(client_weights)[::-1]

        for client_id in range(num_clients):
            if round_num != poison_round:
                selected_client_id = sorted_clients[client_id]
                train_on_shard(federated_train_data[selected_client_id], models[selected_client_id])
            else:
                X_attacker, y_attacker = zip(*attacker_data)
                X_attacker, y_attacker = np.array(X_attacker), np.array(y_attacker)
                attacker_model.fit(X_attacker, y_attacker, epochs=1, verbose=0) # Train the attacker model
                attacker_model_weights = attacker_model.get_weights() # Attacker's malicious model weights

```

Figure A.2: Python code for the implementation of the attack model (continued)

## A.2 Calculation of Credit-Based Client Selection

In the provided Figures A.3 and A.4, we illustrate the step-by-step implementation of our defense strategy, including the calculation of our credit score and the incorporation of the sorting mechanism. This visualization highlights the key components of our credit-based client selection method and showcases how we determine the credit of each client and select the most reliable ones for aggregation into the global model.

```

# Calculate accuracy and loss on test data
def test_model_accuracy_loss(model, test_data):
    X_test, y_test = zip(*test_data)
    X_test, y_test = np.array(X_test), np.array(y_test)
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
    return accuracy, loss

# Update client weights based on credit (accuracy and loss on test data)
def update_client_weights(models, test_data):
    client_weights = []
    for model in models:
        accuracy, loss = test_model_accuracy_loss(model, test_data)
        # Modify the credit mechanism to combine accuracy and loss
        credit_score = accuracy - (0.1 * loss)
        client_weights.append(credit_score)
    return client_weights

```

Figure A.3: Python code for the implementation of our credit-based client selection

```

# Aggregate models using federated averaging with credit-based client selection
global_model = federated_avg(models, client_weights, attacker_model_weights if round_num == poison_round else None)
models = [global_model] * num_clients

# Update client weights based on performance difference between consecutive rounds
test_data = list(zip(test_images, test_labels))
prev_accuracy = [1.0] * num_clients

for client_id in range(num_clients):
    prev_accuracy[client_id] = test_model_accuracy_loss(models[client_id], federated_train_data[client_id])[0]

for client_id in range(num_clients):
    train_on_shard(federated_train_data[client_id], models[client_id])
    curr_accuracy = test_model_accuracy_loss(models[client_id], federated_train_data[client_id])[0]

    # Update the client weight based on the difference in accuracy between consecutive rounds
    performance_diff = curr_accuracy - prev_accuracy[client_id]
    client_weights[client_id] += performance_diff

# Normalize client weights to ensure they sum up to 1
total_weights = sum(client_weights)
client_weights = [weight / total_weights for weight in client_weights]

# Calculate and print accuracy
print(f"Global Model Accuracy after Round {round_num + 1}: {test_model_accuracy_loss(global_model, test_data)[0]}")
for client_id in range(num_clients):
    print(f"Client {client_id + 1} Accuracy after Round {round_num + 1}: {test_model_accuracy_loss(models[client_id], federated_train_data[client_id])[0]}")

# Run federated learning with model poisoning and improved credit-based client selection
federated_learning_with_poisoning()

```

Figure A.4: Python code for the implementation of our credit-based client selection (continued)

## A.2.1 Softmax Function

In our work, the softmax function is used for the evaluation of the model’s predictions and the calculation of the loss during the training process. The softmax function is commonly used in multi-class classification tasks, which is the case for the datasets we utilized (e.g., Mnist, FMnist, CIFAR-10). It converts the model’s raw output scores (logits) into probabilities, assigning a probability value to each class, representing the model’s confidence in its prediction. In the context of our defense strategy, the softmax function plays a vital role in computing the cross-entropy loss, which is a standard loss function used in classification tasks. The loss quantifies how different the predicted probabilities are from the ground truth labels. By minimizing this loss during training, the model learns to make more accurate predictions. Moreover, the softmax function is used in evaluating the accuracy of the local models of individual clients. It helps measure the certainty of each client’s predictions and is a crucial factor in calculating their credit scores. Clients with higher accuracy and lower loss values will have higher credit scores, making them more likely to be selected for model aggregation. Overall, the softmax function shown in figure A.5 is an integral part of our thesis, as it contributes to the model’s training, evaluation, and the calculation of the credit scores for our credit-based client selection method.

The Softmax function,  $h_{\theta}(x)$ , is defined as follows:

$$h_{\theta}(x) = \begin{pmatrix} P(y = 1|x; \theta) \\ P(y = 11|x; \theta) \\ \cdot \\ \cdot \\ P(y = k|x; \theta) \end{pmatrix} = \frac{1}{\sum_{j=1}^k \exp(\theta_j^T x)} \begin{pmatrix} \exp(\theta_1^T x) \\ \exp(\theta_2^T x) \\ \cdot \\ \cdot \\ \exp(\theta_k^T x) \end{pmatrix} \quad \text{Equation B-1}$$

Figure A.5: Softmax Function

# Bibliography

- [1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.
- [2] Krizhevsky Alex. Learning multiple layers of features from tiny images. pages 32–33, 2009.
- [3] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. *Advances in Neural Information Processing Systems*, page 31, 2018.
- [4] Alejandro Baldominos, Yago Saez, and Pedro Isasi. A survey of handwritten character recognition with mnist and emnist. *Applied Sciences*, 9(15), 2019.
- [5] Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *CoRR*, abs/1902.06156, 2019.
- [6] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 634–643. PMLR, 09–15 Jun 2019.
- [7] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *31st International Conference on Neural Information Processing Systems, NIPS'17*,, page 118–128, New York,USA, 2017. Curran Associates Inc.
- [8] KALLISTA BONAWITZ, PETER KAIROUZ, BRENDAN MCMAHAN, and DANIEL RAMAGE. Federated learning and privacy: Building privacy-preserving systems for machine learning and data science on decentralized data. *Queue*, 19:87–114, 11 2021.

- [9] Parimala Boobalan, Swarna Priya Ramu, Quoc-Viet Pham, Kapal Dev, Sharnil Pandya, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Thien Huynh-The. Fusion of federated learning and industrial internet of things: A survey. *Computer Networks*, 212:109048, 2022.
- [10] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *CoRR*, abs/2012.13995, 2020.
- [11] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients. 2018.
- [12] Bipin Chhetri, Saroj Gopali, Rukayat Olapojoye, Samin Dehbash, and Akbar Siami Namin. A survey on blockchain-based federated learning and data privacy. 2023.
- [13] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017.
- [14] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Fedavg with fine tuning: Local updates lead to representation learning. 2022.
- [15] Datta H. Deshmukh, Tushar Ghorpade, and Puja Padiya. Improving classification using preprocessing and machine learning algorithms on nsl-kdd dataset. In *2015 International Conference on Communication, Information and Computing Technology (ICCICT)*, pages 1–6, 2015.
- [16] Rachid Guerraoui El Mahdi El Mhamdi and Sébastien Rouault. The Hidden Vulnerability of Distributed Learning in Byzantium. In *35th International Conference on Machine Learning*, page 3518–3527. PMLR, 2018.
- [17] Andreas Veit Eugene Bagdasaryan, Deborah Estrin Yiqing Hua, and Vitaly Shmatikov. How to backdoor federated learning. *CoRR*, abs/1807.00459, 2018.
- [18] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. *USENIX*, 2020.
- [19] Michael Feil. On the drawbacks of fedavg and the benefits of posterior averaging. July 2021.

- [20] Joaquin Delgado Fernandez, Martin Brennecke, Tom Barbereau, Alexander Rieger, and Gilbert Fridgen. Federated learning: Organizational opportunities, challenges, and adoption strategies. 2023.
- [21] Carlos Vladimiro Gonzalez Zelaya. Towards explaining the effects of data pre-processing on machine learning. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2086–2090, 2019.
- [22] Rémi Gosselin, Loïc Vieu, Faiza Loukil, and Alexandre Benoit. Privacy and security in federated learning: A survey. 2022.
- [23] Filip Granqvist, Matt Seigel, Rogier van Dalen, Aine Cahill, Stephen Shum, and Matthias Paulik. Improving on-device speaker verification using federated learning with privacy. *arXiv preprint arXiv:2008.02651*, 2020.
- [24] Tianmei Guo, Jiwen Dong, Henjian Li, and Yunxing Gao. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 721–724, 2017.
- [25] Andrew Hard, Kanishka Rao, Rajiv Mathews, Fran coise, Beaufays Sean, Augenstein Hubert, Eichner, Chlo´e Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *CoRR*, *abs/1811.03604*,, 2018.
- [26] Chao Huang, Jianwei Huang, and Xin Liu. Cross-silo federated learning: Challenges and opportunities. 2022.
- [27] Asmida Ismail, Siti Anom Ahmad, Azura Che Soh, Khair Hassan, and Hazreen Haizi Harith. Improving convolutional neural network (cnn) architecture (minivggnet) with batch normalization and learning rate decay factor for image classification. *International Journal of Integrated Engineering*, 11(4), Sep. 2019.
- [28] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *Proceedings - 2018 IEEE Symposium on Security and Privacy, SP 2018*, Proceedings - IEEE Symposium on Security and Privacy, pages 19–35, United States, jul 2018. Institute of Electrical and Electronics Engineers Inc.



- [29] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Towards understanding biased client selection in federated learning. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 10351–10375. PMLR, 28–30 Mar 2022.
- [30] Malhar Jere, Tyler Farnan, and Farinaz Koushanfar. A taxonomy of attacks on federated learning. *IEEE Security & Privacy*, 19:20–28, 2021.
- [31] V L Helen Josephine, A.P. Nirmala, and Vijaya Lakshmi Alluri. Impact of hidden dense layers in convolutional neural network to enhance performance of classification model. *IOP Conference Series: Materials Science and Engineering*, 1131(1):012007, apr 2021.
- [32] Konency, McMahan Yu, and Suresh Bacon. Federated learning: strategies for improving communication efficiency. 2017. 1610.05492.
- [33] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.
- [34] Alex Krizhevsky. The cifar-10 and cifar-100, 2017.
- [35] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar 10 dataset, 20121.
- [36] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Deep learning cnn for fashion-mnist clothing classification, 2020.
- [37] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [38] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
- [39] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. THE MNIST DATABASE of handwritten digits, 1999.
- [40] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. 2020.
- [41] The MIT License. Fashion mnist dataset, an alternative to mnist, 2021.

- [42] Tianyi Liu, Shuangfang Fang, Yuehui Zhao, Peng Wang, and Jun Zhang. Implementation of training convolutional neural networks. *CoRR*, abs/1506.01195, 2015.
- [43] B. McMahan, Eider More, Dani Ramag, Seth Hampson, and Blaise Aguera. Communication efficient learning of deep networks from decentralized data. *AISTATS*, 2018.
- [44] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. 2018.
- [45] Jed Mills, Jia Hu, and Geyong Min. Faster federated learning with decaying number of local sgd steps. 2023.
- [46] Mohammad Moshawrab, Mehdi Adda, Abdenour Bouzouane, Hussein Ibrahim, and Ali Raad. Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives. *Electronics*, 12(10), 2023.
- [47] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. 2017.
- [48] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.
- [49] Kim Phil. *Convolutional Neural Network*, pages 121–147. Apress, Berkeley, CA, 2017.
- [50] Qammar, Attia, Ding, Jianguo, Ning, and Huansheng. Federated learning attack surface: taxonomy, cyber defences, challenges, and future directions. *Artificial Intelligence Review*, 55, 06 2022.
- [51] Rhythm. Data preprocessing for cifar 10 dataset, 2020.
- [52] Sarvepallia and Sarat Kumar. Deep learning in neural networks: The science behind an artificial brain. 2015.
- [53] Zalando SE. Fashion mnist an mnist-like dataset, 2017.

- [54] Neha Sharma, Vibhor Jain, and Anju Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018. International Conference on Computational Intelligence and Data Science.
- [55] Virat Shejwalkar and A. Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. *NDSS*, 2021.
- [56] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on federated learning. *CoRR*, abs/2108.10241, 2021.
- [57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
- [58] Lili Su, Jiaming Xu, and Pengkun Yang. A non-parametric view of fedavg and fedprox: Beyond stationary points. 2022.
- [59] Mohamed Suliman and Douglas Leith. Two models are better than one: Federated learning is not private for google gboard next word prediction. 2022.
- [60] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized federated averaging. *CoRR*, abs/2104.11375, 2021.
- [61] Siham Tabik, Daniel Peralta, Andrés Herrera-Poyatos, and Francisco Herrera. A snapshot of image pre-processing for convolutional neural networks: case study of mnist. *Int. J. Comput. Intell. Syst.*, 10:555–568, 2017.
- [62] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. *CoRR*, abs/2007.08432, 2020.
- [63] Nguyen Binh Truong, Kai Sun, Siyao Wang, Florian Guitton, and Yike Guo. Privacy preservation in federated learning: Insights from the GDPR perspective. *CoRR*, abs/2011.05411, 2020.
- [64] Majid Vafadar. A convolutional neural network solution for mnist dataset. 02 2018.
- [65] Yuwei Wang and Burak Kantarci. A novel reputation-aware client selection scheme for federated learning within mobile environments. In *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6, 2020.

- [66] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.
- [67] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd. 2018.
- [68] Jean-Paul A. Yaacoub, Hassan N. Noura, and Ola Salman. Security of federated learning with iot systems: Issues, limitations, challenges, and solutions. *Internet of Things and Cyber-Physical Systems*, 3:155–179, 2023.
- [69] Timoth Yang, Galn Andrew, Hubert Eichn, Haichng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Fran coise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv:1812.02903*, 2018.
- [70] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantinerobust distributed learning: Towards optimal statistical rates. In *35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research*, page 5650–5659. PMLR, July 2018.
- [71] Y. Zhan, J. Zhang, Z. Hong, L. Wu, P. Li, and S. Guo. A survey of incentive mechanism design for federated learning. *IEEE Transactions on Emerging Topics in Computing*, 10(02):1035–1044, apr 2022.