



University of
Zurich^{UZH}

Improving the Model Robustness of DFL in Non-IID Environment

Zien Zeng
Zurich, Switzerland
Student ID: 21-741-558

Supervisor: Chao Feng
Date of Submission: August 6, 2024

Zusammenfassung

Federated Learning (FL) ist anfällig für Vergiftungsangriffe. Während einige Abwehrmechanismen vorgeschlagen wurden, gehen sie überwiegend von IID-Daten aus, was in realen Szenarien normalerweise nicht der Fall ist. In der Praxis sind Nicht-IID-Datenverteilungen häufiger und stellen zusätzliche Herausforderungen für die Abwehr dar, da die heterogenen Daten zwischen den Teilnehmern die Erkennung schädlicher Knoten erschweren. Diese Studie befasst sich mit diesen Herausforderungen, indem sie einen auf Moving Target Defense (MTD) basierenden Mechanismus vorschlägt, der zum Schutz vor Vergiftungsangriffen in FL unter Nicht-IID-Szenarien entwickelt wurde. Der vorgeschlagene Verteidigungsmechanismus umfasst ein duales Reputationssystem, das aus Modellähnlichkeit und Nachbarmodellverlust besteht; dynamische Reputationsschwellen unter Verwendung von DBSCAN; Topologie MTD; und Aggregationsalgorithmus MTD.

Die Wirksamkeit dieses Abwehrmechanismus wurde rigoros anhand von Benchmark-Datensätzen für verschiedene Giftangriffe und Bedrohungsstufen bewertet. Insgesamt wurden 1020-Experimente durchgeführt und gezeigt, dass der Mechanismus sehr robust gegen Label Flipping und Modellvergiftungsangriffe ist. Zwar zeigte es Resilienz gegen Backdoor-Angriffe bei niedrigen Vergifteten Knotenverhältnissen (PNR), aber es gibt noch Raum für Verbesserungen, insbesondere wenn die Bedrohung steigt. Diese Studie untersuchte auch den Einfluss verschiedener Netzwerktopologien und Aggregationsalgorithmen auf die Gesamtmodellleistung. Die Ergebnisse zeigten, dass die Sterntopologie und der Krum-Algorithmus im Vergleich zu anderen Konfigurationen eine höhere Instabilität aufwiesen. Insgesamt verbessert diese Forschung die Abwehr von FL-Systemen gegen Giftangriffe in Nicht-IID-Umgebungen und liefert Erkenntnisse zur Optimierung von Verteidigungsstrategien mit verschiedenen Netzwerktopologien und Aggregationsalgorithmen.

Abstract

Federated Learning (FL) is vulnerable to poisoning attacks. While some defense mechanisms have been proposed, they predominantly assume IID data, which is usually not the case in real-world scenarios. In practice, non-IID data distributions are more common, posing additional challenges to defenses, as the heterogeneous data across participants increases the difficulty of detecting malicious nodes. This study addresses these challenges by proposing a Moving Target Defense (MTD) based mechanism designed to protect against poisoning attacks in FL under non-IID scenarios. The proposed defense mechanism includes a dual reputation system composed of model similarity and neighbor model loss; dynamic reputation thresholds using DBSCAN; topology MTD; and aggregation algorithm MTD.

The effectiveness of this defense mechanism was rigorously evaluated using benchmark datasets across various poisoning attacks and threat levels. A total of 1020 experiments were conducted and demonstrated that the mechanism is highly robust against label flipping and model poisoning attacks. While it exhibited resilience against backdoor attacks at low poisoned node ratios (PNR), there is still room for improvement, especially as threat levels increase. This study also investigated the impact of different network topologies and aggregation algorithms on overall model performance. The results indicated that the star topology and the Krum algorithm exhibited higher instability compared to other configurations. Overall, this research advances the defense of FL systems against poisoning attacks in non-IID environments and provides insights regarding optimizing defense strategies with diverse network topologies and aggregation algorithms.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Chao Feng, for his expert guidance and support throughout my master thesis. His insightful suggestions, critical feedback, and profound expertise in federated learning are invaluable, ensuring that I stayed on the right track and improving my work.

Furthermore, I am also grateful to Prof. Dr. Burkhard Stiller for providing me the opportunity to undertake my master thesis in an area that aligns with my interests. Conducting research at the Communication Systems Group has been an enriching and rewarding experience.

Contents

Zusammenfassung	i
Abstract	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	2
1.2 Description of Work	2
1.3 Thesis Outline	3
2 Background	5
2.1 Federated Learning	5
2.1.1 Centralized Federated Learning	5
2.1.2 Decentralized Federated Learning	6
2.2 Poisoning Attack	7
2.2.1 Attack Objective	8
2.2.2 Attack Strategy	9
2.3 Non Independent and Identically Distributed Data	10
2.3.1 Non-IID Data in Federated Learning	10
2.3.2 Non-IID Data Influence on Federated Learning	12
2.4 Moving Target Defense	13
2.4.1 What to move	14
2.4.2 How to move	16
2.4.3 When to move	17

3	Related Work	19
3.1	Non-IID	19
3.1.1	Data-based Approach	20
3.1.2	Algorithm-based Approach	21
3.1.3	Optimization-based Approach	23
3.2	Moving Target Defense	24
3.2.1	MTD in Federated Learning	24
3.2.2	MTD in Other Domains	25
3.3	Motivation	26
4	Architecture	29
4.1	Workflow	29
4.2	Reputation System	31
4.2.1	Model Similarity	31
4.2.2	Neighbor Model Loss	32
4.2.3	Reputation Threshold	33
4.3	Moving Target Defense	34
4.3.1	Topology MTD	34
4.3.2	Aggregation Algorithm MTD	36
4.4	Non-IID Data Handling	37
5	Implementation	39
5.1	Reputation System	39
5.1.1	Model Similarity	39
5.1.2	Neighbor Model Loss	40
5.1.3	Reputation Threshold	40
5.2	Moving Target Defense	41
5.2.1	Topology MTD	42
5.2.2	Aggregation Algorithm MTD	42
5.3	Non-IID Data Handling	43

<i>CONTENTS</i>	ix
6 Evaluation	47
6.1 Experiment Setup	47
6.1.1 Datasets and Models	47
6.1.2 Threat Model	48
6.1.3 Experiment Configuration	49
6.2 Results	51
6.2.1 IID and Non-IID	51
6.2.2 Untargeted Label Flipping	52
6.2.3 Untargeted Model Poisoning	55
6.2.4 Backdoor	57
6.2.5 Network Topology	60
6.2.6 Aggregation Algorithm	65
6.2.7 FedProx	68
6.3 Discussion	69
7 Summary and Conclusions	71
7.1 Conclusion	71
7.2 Limitations and Future Work	72
Bibliography	75
Abbreviations	81
List of Figures	81
List of Tables	84

Chapter 1

Introduction

With the rapid development of the Internet of Things (IoT), an increasing number of devices are connected to the Internet. These devices generate vast amounts of data, posing significant challenges in data processing and analysis. Initially, traditional machine learning approaches were employed, where data collected by edge devices was uploaded to a central server for aggregation and model training. However, this centralized approach raises data privacy [1] and scalability concerns, as it gathers sensitive information from diverse sources into a single server, increasing the risk of data breaches and causing difficulties in managing large-scale datasets efficiently.

To address these privacy issues, Federated Learning (FL) was proposed [2]. FL, encompassing Centralized Federated Learning (CFL) and Decentralized Federated Learning (DFL), is a collaborative approach that allows clients to independently train models on their local data. These local models are then aggregated to form a global model, alleviating the privacy concerns associated with traditional machine learning [3]. Despite this advantage, CFL still relies on a central server for global model aggregation, making it susceptible to single point of failure and potential network bottlenecks.

DFL, on the other hand, eliminates the need for a central server by adopting a decentralized framework. In DFL, all nodes are considered equal and can function as both trainers and aggregators simultaneously. This not only improves system robustness but also further enhances data privacy [4]. Nevertheless, the decentralized nature of DFL makes it vulnerable to adversarial attacks, especially poisoning attacks. In these attacks, malicious participants intentionally corrupt their data or models and share them with neighbors, undermining the global model's integrity and robustness. Anomaly detection becomes crucial in this context.

However, the heterogeneous environments in which the devices operate lead to non-Independent and Identically Distributed (non-IID) data distributions across clients. This causes variations in their models, making it more challenging to distinguish between benign and malicious nodes and resulting in biases and inaccuracies in the aggregated global model [5]. Consequently, effectively mitigating attacks and enhancing the robustness of DFL models in non-IID environments is a vital area of research.

1.1 Motivation

To enhance the security and robustness of FL systems, researchers have explored various defense mechanisms. One promising strategy is Moving Target Defense (MTD), which dynamically adjusts system components to increase uncertainty and complexity for potential attackers, making it more difficult for them to exploit system vulnerabilities and launch attacks [6]. While MTD has been researched in the IoT domain, its application in the context of FL, particularly in DFL, remains relatively limited. The unique characteristics of DFL, including the ability of nodes to perform aggregation and communicate directly with neighbors, provide additional possibilities for the use of MTD. Current research on MTD in FL has primarily focused on reactive approaches and Independent and Identically Distributed (IID) data settings. However, evaluating defense strategies in non-IID scenarios is crucial for realistic assessments of system resilience.

Overall, proactive strategies aimed at enhancing DFL’s robustness under non-IID conditions have received less attention to date. There is a clear need to develop and evaluate a proactive defense strategy specifically tailored for non-IID data settings. This study aims to address this gap by proposing an MTD-based defense mechanism for DFL in non-IID environments.

1.2 Description of Work

This research involves designing, implementing, and evaluating both proactive and reactive MTD-based defense mechanisms to counteract poisoning attacks in DFL under non-IID conditions. To achieve this objective, the following tasks were completed:

(1) Conducted extensive research on federated learning, poisoning attacks, non-IID data characteristics, and MTD strategies. This involved a thorough literature review on non-IID data handling techniques and MTD applications in various domains.

(2) Proposed an MTD-based aggregation protocol that integrates several key components:

- Dual Reputation System: used model similarity (measured by Euclidean distance) and neighbor model loss on local test datasets to assess neighbor trustworthiness comprehensively.
- Dynamic Reputation Thresholds: utilized DBSCAN algorithm to dynamically set thresholds for detecting malicious models and initiating reactive MTD strategies.
- Topology MTD and Aggregation Algorithm MTD: implemented MTD strategies in both proactive and reactive modes to enhance system resilience.

(3) A total of 1020 experiments were conducted to validate the performance of the proposed defense mechanism. Evaluations were conducted using MNIST, FashionMNIST, and CIFAR-10 datasets under various poisoning attack scenarios (targeted and untargeted) with varying poisoned node ratios (10%, 30%, 50%, and 70%). Different initial

aggregation algorithms (FedAvg, Krum, Median, and TrimmedMean), network topologies (ring, star, and fully connected), non-IID severity levels ($\alpha = 100, 1$, and 0.5), and combinations of MTD strategies (topology MTD only, aggregation algorithm MTD only, and both) were tested. Performance metrics such as F1-score and Attack Success Rate (ASR) were employed to assess the defense mechanism's effectiveness.

1.3 Thesis Outline

This thesis is structured as follows: Chapter 2 introduces background information essential to this study, including topics such as FL, poisoning attacks, non-IID data, and MTD. Chapter 3 conducts a comprehensive literature review on non-IID data handling techniques and MTD applications across various domains, offering valuable insights for the system architecture design. Chapter 4 illustrates the workflow and components of the proposed defense mechanism, including the dual reputation system, dynamic reputation thresholds, MTD strategies, and specialized techniques for handling non-IID data distributions. Chapter 5 focuses on the detailed implementation of the defense mechanism, outlining the integration of aforementioned components into a cohesive framework. Chapter 6 is dedicated to defining the experimental setup and evaluating the performance of the defense mechanism through extensive experiments. Finally, Chapter 7 concludes the study by synthesizing the findings, discussing their implications, and providing directions for future research.

Chapter 2

Background

FL has gained popularity due to its elimination of the need for a central server to store data, thus addressing data privacy issues inherent in traditional machine learning methods. However, the decentralized nature of FL and the heterogeneous environments in which the devices operate introduce complexities that are not present in traditional machine learning.

The following sections of this chapter first provide a theoretical background of FL, then delve deeper into the challenges faced by FL, including poisoning attacks and non-IID data distributions. Finally, a novel defense mechanism, Moving Target Defense, is introduced.

2.1 Federated Learning

FL [2] is a collaborative and decentralized privacy-preserving technology designed to address the challenges of data silos and data sensitivity [7]. Unlike traditional machine learning models that rely on centralized servers storing large datasets, which cause significant data leakage concerns, FL operates by allowing individual devices to locally train models using their own data. These local models are then aggregated to form a global model without exposing raw data. By utilizing the computational power of edge devices, FL enhances both data privacy and efficiency, making the learning process scalable [8]. FL is typically classified into two main categories: Centralized Federated Learning and Decentralized Federated Learning.

2.1.1 Centralized Federated Learning

Centralized Federated Learning (CFL) is the predominant FL approach, characterized by its reliance on a central server to orchestrate the entire learning process. The central server coordinates the creation, distribution, and aggregation of a global model among all participating clients. As shown in the left of Figure 2.1, CFL process contains the following steps:

- The server initializes the model and distributes it to each client.
- Each client independently trains the received model on its local data, ensuring the raw data remains private.
- After local training, each client sends its locally trained model to the server.
- The server aggregates these local models to update the global model, and then redistributes to the clients for further training.

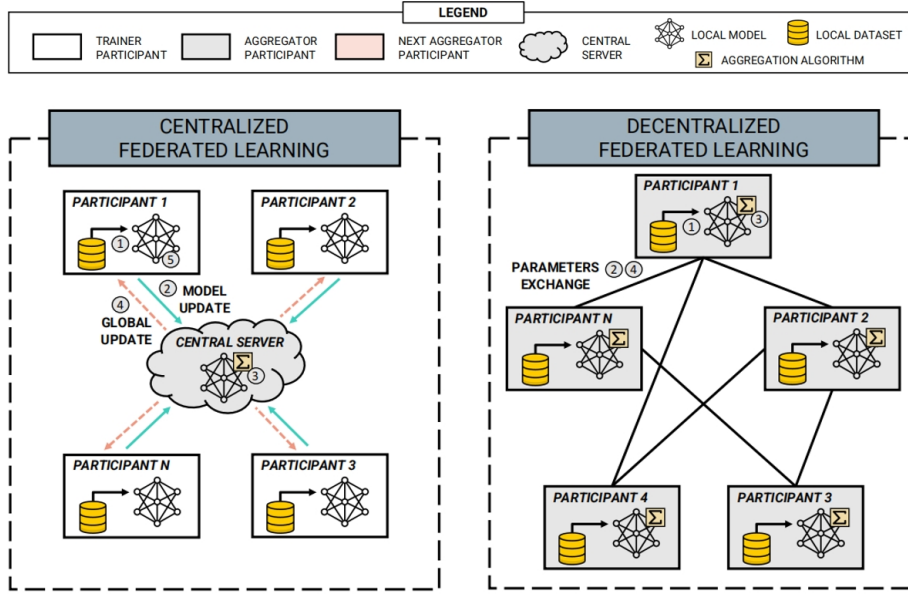


Figure 2.1: Centralized Federated Learning (left) and Decentralized Federated Learning (right) [4]

This cycle of local training and model aggregation continues until the global model converges. While CFL enhances privacy by transmitting model parameters instead of raw data, it still depends on a central server, making it vulnerable to a single point of failure. If the central server is compromised, the entire learning process would be disrupted, posing a significant security risk. Additionally, scalability concerns arise as the number of participating clients increases. The server can become a bottleneck, struggling to manage the communication and aggregation tasks efficiently, leading to increased network traffic.

2.1.2 Decentralized Federated Learning

Decentralized Federated Learning (DFL), which emerged in 2018, enables clients to communicate directly without relying on a central server. In DFL architectures, each node acts as both a trainer and an aggregator, managing communications with other nodes and independently aggregating model parameters. As shown in the right of Figure 2.1, DFL process involves the following steps:

- Clients train their local models on their local datasets.
- Clients send local model parameters to their neighbors and receive model parameters from them.
- Clients aggregate their own local models with those received from neighbors to update their models.

Compared to CFL, DFL has notable advantages. Firstly, by allowing clients to communicate directly with their neighbors and eliminating the need for a central coordinator, DFL mitigates the risk of a single point of failure and enhances fault tolerance. Secondly, DFL facilitates more efficient resource utilization by distributing the computing power required for model aggregation across all participating nodes, rather than relying on a single central server [4]. Additionally, DFL supports diverse network configurations such as star, ring, and fully connected topologies, enhancing its flexibility and adaptability to different environments.

Despite these advancements, DFL face several challenges [3]:

- (1) Privacy Risk: While federated learning enhances privacy by sharing model parameters instead of raw data, the communication of these parameters can still potentially leak sensitive information, such as through reverse engineering of gradient information [9].
- (2) Communication Cost: Communication remains a significant bottleneck in federated learning, especially in networks with massive edge devices. The extensive communication required during model training, particularly in fully connected networks, can impede efficiency and scalability.
- (3) Unreliable Model Exchange: FL is vulnerable to attacks where malicious participants send faulty model parameters, disrupting the aggregation process and compromising the integrity of the global model.
- (4) Data Heterogeneity: Real-world data collected from various devices or institutions often exhibits disparities in size and distribution patterns [7], which can significantly impact the performance of federated learning models trained on such heterogeneous data.

2.2 Poisoning Attack

In federated learning, the privacy of participants' data and the independence of their training processes increase the difficulty for nodes to verify each other's trustworthiness. This lack of transparency make it easier for malicious clients to carry out poisoning attacks, where they attempt to corrupt the learning process by tampering with data or models [10].

Poisoning attacks can be categorized based on their objectives and strategies. From the perspective of attack objectives, poisoning attacks can be classified into three types:

untargeted, targeted and backdoor attacks. From the perspective of attack strategies, poisoning attacks fall into two primary categories: data poisoning and model poisoning. Figure 2.2 provides a taxonomy of poisoning attacks, and the following sections explain each category in detail.

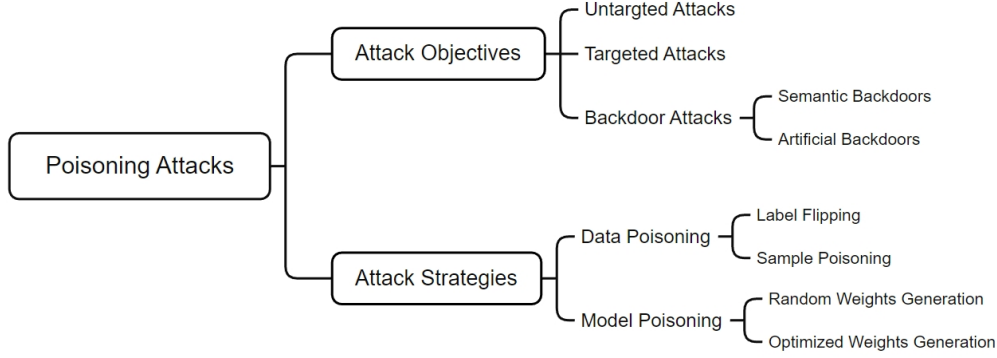


Figure 2.2: Taxonomy of Poisoning Attacks

2.2.1 Attack Objective

Based on attack objectives, poisoning attacks can be classified into three categories: untargeted, targeted and backdoor attacks. Figure 2.3 illustrates how these different types of attacks operate.

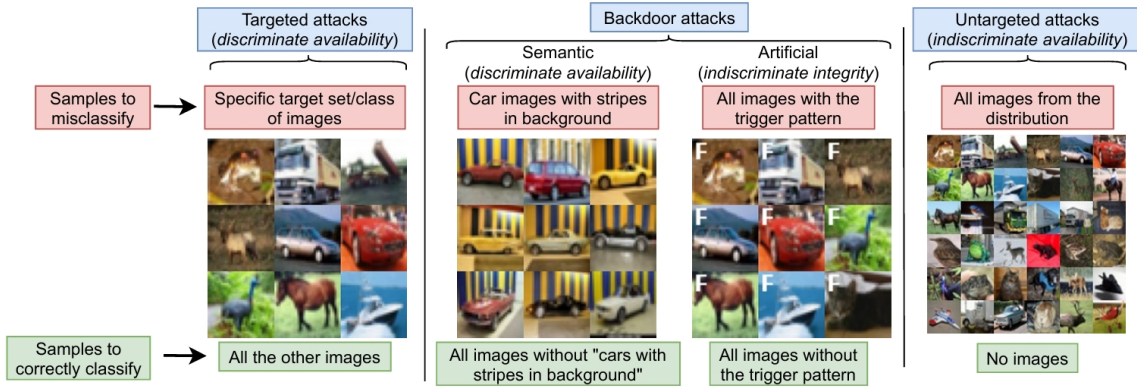


Figure 2.3: Classification of Poisoning Attacks Based on Attack Objectives [11]

(1) Untargeted Poisoning Attacks

Untargeted attacks aim to degrade the overall performance of the model without targeting at specific instances or classes [12]. For example, attackers can randomly alter the labels of any images, resulting in widespread misclassification throughout the dataset.

(2) Targeted Poisoning Attacks

Targeted attacks are designed to cause the model to misclassify specific instances or classes while maintaining high performance on the remaining dataset [13]. For example, an attacker might change the labels of images from “cat” to “horse”, causing the model to misclassify cats as horses while still correctly identifying images of other classes.

(3) Backdoor Attacks

Backdoor attacks degrade the performance of a specific subtask while maintaining the main task’s performance. These attacks involve injecting a hidden trigger into the model, which activates specific behaviors only when the trigger is present in the input. These attacks are insidious because they do not significantly affect the model’s performance during normal operation, making them harder to detect. Based on the nature of the trigger, backdoor attacks can be classified into:

- **Semantic Backdoors:** The trigger is a naturally occurring feature, such as a specific background or object in an image [14]. In Figure 2.3, the semantic trigger for car images is the stripes in the background.
- **Artificial Backdoors:** The trigger is an artificial feature added by the attacker, such as a specific pattern or watermark embedded in the data, which might not be visible to the human eye [15]. In Figure 2.3, the artificial trigger is the character “F” at the upper left corner.

2.2.2 Attack Strategy

Based on the attack strategies, poisoning attacks can also be classified into two main types: data poisoning and model poisoning.

(1) Data Poisoning Attacks

In data poisoning attacks, the adversary manipulates their own local data to influence the model training process. Depending on whether they alter data features or data labels, these attacks can be further subdivided into:

- **Label Flipping:** The attacker changes the labels of specific data points while keeping the feature values intact [16]. For example, in the MNIST dataset, an attacker might change the labels of all digit “1” instances to label “7”, causing the model to misclassify digits “1” as “7” during testing.
- **Sample Poisoning:** The attacker alters the features of the data points to degrade the model’s performance [17]. For instance, they might modify pixel values in images to corrupt their class characteristics without changing their labels.

(2) Model Poisoning Attacks

In model poisoning attacks, the adversary directly modifies the parameters of the locally trained model, such as weights or gradients, before sharing them with others. This manipulation can occur without changing the underlying data. Common types of model poisoning attacks include:

- **Random Weights Generation:** The attacker generates random weights with dimensions similar to the original model's weights [14]. These random weights are then used to manipulate the local model's gradients, compromising the integrity of the global model.
- **Optimized Weights Generation:** The attacker employs optimization techniques to carefully craft the weights, ensuring they resemble past benign updates while introducing impactful changes to the model's behavior [12]. This approach makes the attacks difficult to detect and mitigate, ultimately forcing the main model to produce attacker-specified outputs.

2.3 Non Independent and Identically Distributed Data

Traditional machine learning often assumes Independent and Identically Distributed (IID) data, yet real world data distributions usually deviate from this ideal scenario. Instead, Non-Independent and Identically Distributed (Non-IID) data settings are more prevalent, where data across clients may not exhibit independence or sampled from identical distribution.

2.3.1 Non-IID Data in Federated Learning

Non-IID data poses a significant challenge to federated learning, as it can lead to weight divergence among local models, ultimately affecting overall model performance [18]. Consider a supervised learning task on client k , where each data sample (x, y) , comprises input attributes or features x and its corresponding label y , following a local distribution denoted as $P_k(x, y)$. In federated learning, data heterogeneity implies discernible distinctions between P_i and P_j for different clients i and j . Data collected by each client is heavily influenced by the specific usage patterns and hardware configurations of the local device, thus introducing potentially substantial variations in data distribution among clients.

Data heterogeneity in federated learning manifests in diverse forms, including feature distribution skew, label distribution skew, same label different features, same feature different labels and quantity skew [19], as shown in Figure 2.4. Each type of data heterogeneity will be discussed in detail in the following.

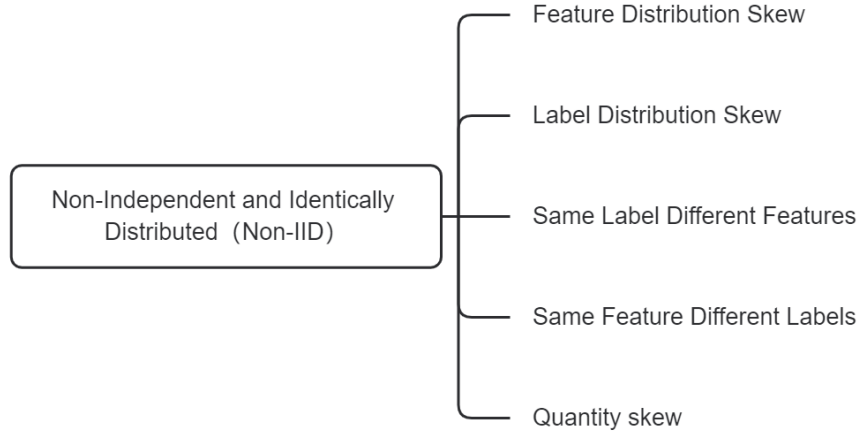


Figure 2.4: Classification of Non-IID Data

(1) Feature Distribution Skew

Feature distribution skew indicates variation of the feature distribution $P(x)$ among clients, while the conditional label distributions, $P(y|x)$, remain consistent. For instance, client A prefers to write the number "3" in bold, whereas client B prefers a slender style. Consequently, the probability of observing the bold "3" is higher with client A than with client B, indicating a disparity between $P_A(x)$ and $P_B(x)$. Nevertheless, given the condition $x = \text{bold}$, the probability of different clients utilizing this feature to predict the label $y = 3$ is the same, implying $P_A(y|x) = P_B(y|x)$.

(2) Label Distribution Skew

Label distribution skew indicates variation of the label distribution $P(y)$ among clients, while the conditional feature distribution, $P(x|y)$, remains consistent. For example, A's dataset comprises 80% male and 20% female samples, while client B's dataset consists of 90% male and 10% female samples, highlighting difference between $P_A(y)$ and $P_B(y)$. However, if the condition $y = \text{female}$ is specified, the probability of different clients predicting the feature x as female remains consistent.

(3) Same Label Different Features

The scenario of same label different features implies that although $P(y)$ remains constant, the conditional distribution, $P(x|y)$, may vary across clients. For example, the architectural style in Western countries, denoted by $P_W(x|y)$, may differ from that in Eastern countries, $P_E(x|y)$, despite both being categorized as buildings, meaning $P(y)$ is the same.

(4) Same Feature Different Labels

The scenario of same feature different labels, also known as label preference skew, suggests that although $P(x)$ remains constant, the conditional distribution, $P(y|x)$, may vary across clients. Different users may annotate different labels for the same data sample based on individual preferences. For instance, an image of a dog wearing a hat might be labeled as "like" by some individuals and "dislike" by others. Such discrepancies are common in crowd-sourced data, where the same picture is tagged with varied labels by different individuals.

(5) Quantity Skew

Quantity skew refers to variations in the amount of data among clients, which is very common in decentralized settings. For example, client A has collected only 100 data records, while client B has collected 2,000,000 data records, illustrating significant difference in data volume across clients.

2.3.2 Non-IID Data Influence on Federated Learning

Non-IID data has impact on the accuracy and convergence speed of federated learning models. In the extreme scenario where each client possesses only one class label, the global model fails to converge [20].

The presence of non-IID data introduces client drift, as depicted in Figure 2.5. Let x^* represents the global optimum of $f(x)$, and x_i^* denotes the local optimum of the loss function $f_i(x)$ for client i . Assuming all clients initiate training from the same starting point, they progressively move towards their respective local optima within each training round. The average of client updates (the server update) converges towards $\frac{1}{N} \sum x_i^*$, indicated by the black points in the figure.

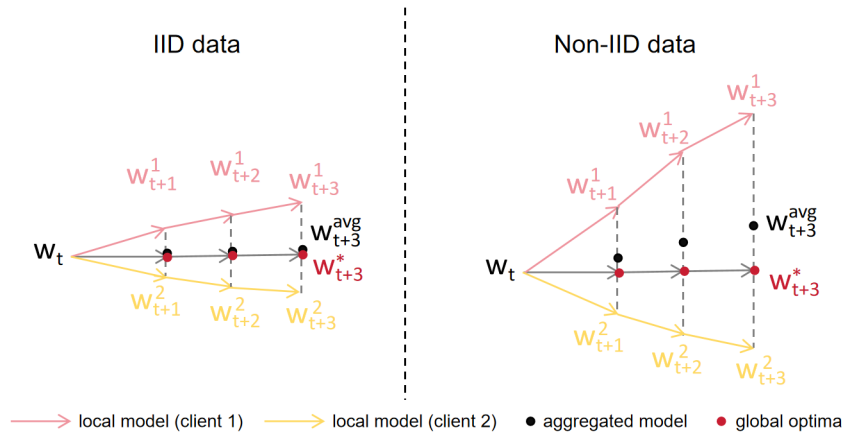


Figure 2.5: Model Divergence for IID and Non-IID Data

In an IID data scenario, where each client’s local optimum closely aligns with both other client’s and the global optimum, the average of client updates may approximate the global optimum x^* . However, in a non-IID data context, due to discrepancies between the local and global data distributions, the objectives of individual parties do not align with the global optima. Consequently, there is a drift in the local updates, with each x_i^* being distant from both one another and the global optimum x^* , leading to significant divergence between the average of client updates and the true optimum [21]. This phenomenon, where the updates from clients drift away from the global optimum due to differing data distributions, is referred to as client drift, leading to suboptimal performance of the federated learning model.

This discrepancy tends to magnify with increasing training epochs [20]. As a result, the difference between the averaged local parameters and the global parameters becomes increasingly larger in non-IID data scenarios compared to IID data, with the divergence accumulating over successive communication rounds.

2.4 Moving Target Defense

Traditional defense mechanisms exhibit inherent limitations due to their static nature concerning network and configuration. Firstly, attackers can exploit this static nature to gather information about the target environment, uncover existing vulnerabilities, and plan their attacks accordingly [22]. Secondly, most traditional defense mechanisms operate reactively, taking action only after detecting intrusion, and therefore limiting the ability to prevent attacks in advance. Thirdly, zero-day attacks developed using the information gathered during the reconnaissance phase, can render traditional defenses ineffective.

To mitigate the shortcomings of the traditional defense mechanisms, Moving Target Defense (MTD) was proposed, which proactively defend against adaptive adversaries. MTD dynamically controls the attack surface by continuously shuffling various system components, increasing the unpredictability and complexity for attackers. This diminishes the asymmetric advantages that attackers inherently possess against traditional defense mechanisms [23], ensuring the information gathered by attackers in the reconnaissance phase becomes outdated during the attack phase, as the defender moved to a new configuration within that time.

One crucial aspect of MTD is its dynamic nature, requiring built-in randomness to remain effective. A deterministic shifting mechanism would enable attackers to learn and predict movements over time and design attacks accordingly, nullifying the advantages of MTD. The incorporation of randomness makes it challenging for attackers to identify vulnerable system components and increases the cost for them to launch successful attacks.

MTD techniques are built on existing technologies, minimizing development costs and ensuring compatibility with current systems without the need for developing new security mechanisms. However, reliance on legacy technologies can be a double-edged sword; if these technologies are not robust against attacks, the additional MTD layer may be constrained by vulnerabilities inherent in the legacy systems, potentially leading to increased defense costs.

An MTD system can be described using a three tuple $\langle C, M, T \rangle$, where C represents the configuration space or the set of configurations between which the system switches, M represents the movement strategy that directs the system action, and T denotes the timing function governing when a switch action occurs [24]. These three elements pose fundamental questions that must be addressed during MTD design [6], namely: (1) What to move? (2) How to move? and (3) When to move? Answers to these questions help categorize MTD techniques, as illustrated in Figure 2.6.

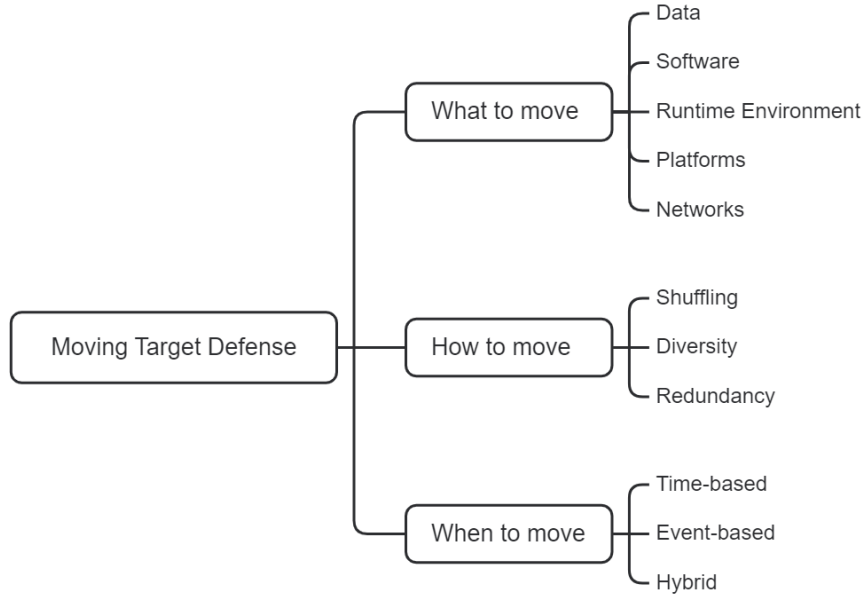


Figure 2.6: Classification of Moving Target Defense

2.4.1 What to move

“What to move” refers to which elements of the system configuration can be altered to confuse potential attackers. Common system configuration attributes include instruction sets, address space layouts, IP addresses, port numbers, proxies, virtual machines, operating systems, and software programs [25]. The configuration space must be broad, meaning the possible values of these configuration attributes should be extensive, preventing attackers from easily brute-forcing future values.

Based on the answer to the question of what to move, MTD mechanisms can be categorized into five groups: dynamic data, dynamic software, dynamic runtime environments, dynamic platforms, and dynamic networks.

(1) Dynamic Data

Dynamic data-based MTD aims to dynamically alter the representation of data [26], such as its format, syntax, or encoding. For instance, changing “Gender = Female” to “Gender = F” alters the data representation while preserving the semantic meaning. This diversity

in data representation helps prevent unauthorized access to the content while maintaining the semantic integrity of the data.

(2) Dynamic Software

Dynamic software-based MTD involves modifying an application's code dynamically [27] to enhance the heterogeneity of software applications while maintaining their functionality. This can include altering program instructions, reordering them, changing their grouping, or modifying their format. By creating multiple execution duplicates with equivalent functionality but different structures, and periodically switching among these application replicas, the attack surface becomes more uncertain and diverse. This unpredictability makes it more difficult for attackers to exploit vulnerabilities, as they cannot rely on a consistent code structure.

(3) Dynamic Runtime Environment

Dynamic runtime environment-based MTD focuses on dynamically altering the execution environment to increase unpredictability. These techniques can be further subdivided into Address Space Randomization (ASR) and Instruction Set Randomization (ISR).

ASR refers to techniques that alter the memory layout dynamically to defend against address attacks. This can include randomizing the locations of program code, libraries, stack/heap, and individual functions. With these changing memory locations, it is hard for attackers to predict where specific code or data resides, thus thwarting exploit attempts that rely on known memory addresses.

ISR refers to techniques that dynamically alter the instruction set [28], which is the interface presented by the operating system to applications. This interface includes processor instructions and system calls used to manipulate I/O devices. The alteration can be achieved by encrypting vulnerable system instructions at load time and decrypting them just before execution, or by introducing a unique set of instructions for each executing program [29]. With these methods, the code injection attacks would fail because the injected code may not match the expected instruction set.

(4) Dynamic Platforms

Dynamic platform-based MTD dynamically changes platform properties. These properties include the operating system version, CPU architecture, virtual machine instance [30], storage system, communication channel, and more. Running the same application in parallel across different instances with varied architectural contexts or migrating applications between machines can increase the complexity for launching attacks because attackers must exploit diverse platforms.

(5) Dynamic Networks

Dynamic network-based MTD focuses on changing network properties dynamically. These properties include protocols, IP addresses [31], port numbers, network topology [32], communication protocols, and more. With frequently changed IP addresses and port numbers, it is difficult for attackers to identify and target specific systems, contributing to the improved system security.

2.4.2 How to move

“How to move” refers to the principles used to alter system configuration elements, aiming to increase unpredictability and confuse attackers. Based on their nature of operation, MTD mechanisms can be categorized into shuffling, diversity, and redundancy [33].

(1) Shuffling

This approach involves randomizing system configuration elements, such as changing IP addresses at the TCP/IP layer, switching port numbers [31], or shuffling network topology [32]. The primary goal of shuffling-based MTD techniques is to create confusion and uncertainty for attackers, rendering the information they gather obsolete and making it more difficult for them to identify vulnerabilities.

(2) Diversity

This approach equips systems with components that differ in implementation but provide the same functionalities [27]. For example, it can involve designing diverse paths for routing or changing platforms that consist of different implementations of software components or migrating between various platforms. Diversity-based MTD aims to enhance resilience; even if an attacker is present in the system, the system can continue providing service by offering multiple alternatives.

(3) Redundancy

This approach involves implementing multiple replicas of system components, such as multiple paths between nodes in a network layer or multiple software components [34] providing the same functionality at the application layer. The goal of redundancy-based MTD techniques is to increase system reliability and mitigate insider threats by providing stable service through redundant components when some network nodes or services are compromised.

Shuffling, diversity, and redundancy are interrelated and impact each other. The way system components are rearranged can affect the degree of system diversity. Conversely,

the inherent diversity of a target can influence the effectiveness of shuffling through the size of the shuffling space. If a system attribute can only change between two states, the performance achievable by the shuffling operation is inherently limited. Similarly, if redundancy is high but diversity is low, attackers can exploit it by targeting alternative ways into the system [33], such as attacking another server or reaching a target through a different path. Therefore, shuffling, diversity, and redundancy should be considered in combination to enhance system robustness.

2.4.3 When to move

After determining the configuration set that the defender can switch among, the next crucial thing to consider is the timing, or "when to move." In an MTD system, "when to move" refers to the optimal moment to initiate an adaptation, switching from the current state to another. Deciding on the best timing is a complex task because frequent changes to the attack surface can be costly and may reduce service availability to users. For instance, changing IP addresses too frequently can lead to network communication failures, increase the likelihood of becoming unreachable to other nodes, and disrupt service provision [35]. Since MTD aims to provide an affordable defense by balancing the effectiveness of the strategy with reasonable costs, the trade-off between security and resource consumption is of vital importance.

MTD strategies can be categorized into three groups based on how the timing for multiple switches is determined: time-based, event-based, and hybrid.

(1) Time-based

This approach involves initiating MTD adaptations based on a certain time interval, which can be either fixed or variable [36]. In this method, the MTD mechanism changes the attack surface according to a predetermined schedule, keeping the surface unchanged during the remaining time interval.

Choosing the right time interval is critical for the effectiveness and efficiency of the defense mechanism. If the interval is too long, attackers have ample time to gather information about the system, identify vulnerabilities, and intrude into it, leading to potential security breaches. Conversely, if the interval is too short, any information the attackers gain becomes obsolete quickly, making it harder for them to exploit the system. However, changing the attack surface too frequent can lead to resource wastage and service instability, degrading performance even when attackers absent.

Finding an optimal time interval is challenging but essential, as it directly influences the balance between security and system performance. The interval must be long enough to avoid excessive resource consumption and instability, yet short enough to minimize the window of opportunity for attackers.

(2) Event-based

The event-based approach triggers MTD adaptations in response to specific events or alerts, such as the detection of malware or suspicious activity. When such an event occurs, the system reacts by switching to a new configuration [37]. This approach relies on advanced monitoring and the ability to detect malicious behavior accurately and promptly. Immediate detection and reaction help minimize the potential damage from attacks and quickly render any information gathered by attackers obsolete.

(3) Hybrid

A hybrid approach combines elements of both time-based and event-based strategies [38], providing a comprehensive defense mechanism. In this approach, adaptations can be launched regularly at predefined time intervals to proactively address potential and undetected security threats. Also, the system is equipped to make immediate adaptations in response to detected threats or events.

Selecting the right timing strategy is crucial for maintaining the effectiveness and efficiency of an MTD system. The goal is to maximize security benefits while minimizing disruptions and costs, making the system robust to attacks without compromising service quality.

Chapter 3

Related Work

This chapter provides a thorough review of the literature in the fields of non-IID data handling in federated learning and MTD strategies. It begins with exploring the technologies used to address non-IID data in federated learning, categorizing the approaches into data-based, algorithm-based, and optimization-based methods. Following this, the chapter discusses the application of MTD strategies across various domains, including federated learning, web applications, and cloud computing. The chapter concludes by summarizing key insights obtained from the comprehensive literature review.

3.1 Non-IID

Non-IID data is one of the challenges faced by federated learning, and researchers have developed various strategies to handle it, which can be broadly categorized into data-based, algorithm-based, and optimization-based approaches, as shown in Figure 3.1. This section investigates each of these methodologies, outlining their respective merits and limitations.

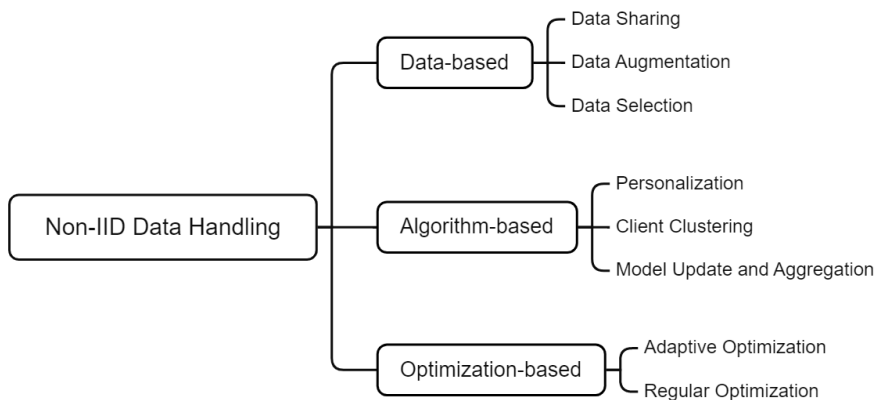


Figure 3.1: Non-IID Data Handling Techniques Classification

3.1.1 Data-based Approach

In federated learning, the degradation of model performance often stems from heterogeneous data distributions, prompting researchers to explore solutions from a data-centric perspective. Data-based techniques for mitigating non-IID data in federated learning primarily involve data sharing, data augmentation, and data selection.

(1) Data Sharing

[39] illustrated that model training on highly skewed non-IID data, with accuracy decreasing by up to 55%, is attributed to weight divergence between the local data distribution on each client and the population distribution. To alleviate this degradation, the Fed-Share mechanism was proposed. Initially, a warm-up model is trained on a globally shared dataset G , which is a uniformly distributed dataset across classes, and then sent to clients with a random α portion of dataset G . Subsequently, clients train their local models on this shared dataset along with their private data and transmit the local models to the server for aggregation. The experimental results showed that with 5% of globally shared data, model accuracy can increase by 30% on CIFAR-10.

[40] employed heuristic algorithms to select a limited number of clients to upload their local data to the server, in order to construct an IID dataset. The server then trained the model on the data gathered from these clients and aggregate it with other clients' local models. Experiment showed that with only 1% of data-uploading clients, a 13.5% higher classification accuracy can be achieved under a non-IID scenario.

While data sharing can enhance model accuracy, it has some drawbacks. Firstly, data sharing techniques may lead to data leakage, violating the intention of federated learning, which prioritize data privacy. Secondly, determining the proportion of data to be shared presents a challenge. Thirdly, acquiring a high-quality global dataset, such as a uniformly distributed dataset, is difficult because the server lacks knowledge of the data distributions among clients.

(2) Data Augmentation

Data augmentation serves as a technique to synthesize additional data to address data scarcity and enhance model performance. In the context of federated learning, data augmentation can also mitigate data imbalance issues across clients.

Generative Adversarial Networks (GANs) are prominent approaches for data augmentation in machine learning. In the FAug framework, edge devices upload local samples to the server, which then trains a generative model and distributes to each client. With this received generator, clients can augment their local samples to replenish and balance their training data [41]. To preserve data privacy and adapt GANs to decentralized federated learning, [42] proposed training generators and discriminators locally, and periodically aggregated by a randomly elected coordinator.

In addition to GANs, researchers have explored other techniques for federated data augmentation. [43] utilized up-sampling, specifically SMOTE, by using all K nearest neighbors found by KNN for linear interpolation of samples in the minority class to synthesize data points. A portion of these synthetic data points is then shared with neighboring clients to aid in their training.

Data augmentation techniques enhance model performance in non-IID scenarios through replenishing local imbalanced data with augmented samples. However, many data augmentation strategies rely on data sharing, raising data privacy concerns.

(3) Data Selection

To address the data heterogeneity challenge in federated learning, [44] proposed an analytics-driven client selection framework. In this framework, clients first extract insights from their local data while safeguarding sensitive information. The server uses these insights to assess the variability in clients' data distributions with Hoeffding's inequality. Then, a dueling bandit algorithm is employed to intelligently select clients with more balanced datasets, forming an optimal client pool.

[45] formulated client selection in federated learning as a deep reinforcement learning problem and trained a reinforcement learning agent using a deep Q-learning network. This agent dynamically selects a subset of devices in each communication round solely based on model weights information, with the aim of enhancing the global model. This strategy helps mitigate bias stemming from non-IID data.

Compared to data sharing and data augmentation, data selection selects the highest quality clients to contribute in each round of federated learning and does not alter clients' data and offers higher data security. However, this approach may exacerbate the model shift problem caused by data heterogeneity and introduce unfairness into federated learning. The model would miss valuable information and lose generalizability if only clients with slight data heterogeneity are selected.

3.1.2 Algorithm-based Approach

Algorithms play a crucial role in machine learning, prompting researchers to address non-IID data challenges in federated learning through algorithmic solutions. Algorithm-based techniques primarily include personalization, client clustering, and model update and aggregation.

(1) Personalization

While the aim of federated learning is to collaboratively train a global model across decentralized clients, this shared model may not always benefit local training, resulting in model performance degradation [20]. To tackle this issue, researchers have proposed

personalization techniques, such as multitask learning and the base + personalization layer approach.

Federated multitask learning (FMTL) integrates federated learning with multitask learning by assigning distinct tasks to each participating client, allowing customization of its local model. This approach preserves privacy while uncovering underlying relationships. [46] demonstrated the efficacy of multitask learning in addressing statistical challenges and introduced MOCHA, a notable FMTL framework employing primal-dual optimization to generate individual yet related models for each client.

In the base + personalization layer technique, each client incorporates personalized layers into its neural network models. Each client’s model consists of base layers and personalization layers, with only the base layer exchanged in the federated learning network. For instance, FedPer [47] collaboratively trains base layers using federated averaging, while personalization layers are trained solely on local data using SGD. Research suggests that personalization layers can effectively alleviate the adverse effects of statistical heterogeneity in federated learning.

While personalization techniques offer the potential for improving model performance on non-IID data, they introduce complexity into the system, which may lead to increased computational overhead during training.

(2) Client Clustering

To address varying data distributions among clients, clustered federated learning has emerged as a solution. Some researchers group similar clients into clusters and aggregate based on these clusters or maintain multiple global models. [48] introduced the Personalized Adaptive Neighbor Matching (PANM) algorithm, which operates in two stages. First, clients utilize Monte Carlo methods to select neighbors based on local pairwise similarity measurements, aligning with peers sharing similar objectives. Second, by employing a Gaussian Mixture Model for neighbor augmentation, clients can discover additional peers with similar objectives.

Federated learning is susceptible to malicious attacks, where harmful updates from malicious nodes disrupt model convergence or introduce bias. [5] identified deficiencies in current anti-poisoning techniques, especially in their handling of minority class data. These techniques often misclassify legitimate minority data as outliers or malicious, neglecting their importance and leading to reduced model performance, particularly when data is not uniformly distributed. To remedy this issue, [5] proposed a balanced approach that combats poisoning while accommodating diversity, aiming for fairer and less discriminatory models. This method enhances the process of update modeling by identifying similarities among outliers, which can indicate the presence of minority groups, and only discard truly isolated updates. They introduce three fair attack detection methods, based on microaggregation, Gaussian mixture models, and DBSCAN respectively, enhancing both fairness and model quality by integrating updates from minority groups.

While employing similarity clustering can promote fairness in federated learning and alleviate the effects of data heterogeneity on model performance by grouping clients with

similar data distributions into clusters, the integration of clustering steps may escalate communication expenses. The increased communication expenses can not be neglect, especially in scenarios involving thousands or millions of edge devices engaged in federated learning.

(3) Model Update and Aggregation

The heterogeneity among clients' local datasets and computational speeds leads to discrepancies in the amount of local updates executed by each client during each communication round. Simply aggregating these models may result in objective inconsistency, where the global model converges to a stationary point of a mismatched objective function, deviating from the true objective. To tackle this issue, [49] introduced FedNova, a normalized averaging approach. FedNova computes the average of normalized local gradients rather than cumulative ones. This framework allows clients to have varying local steps and local solvers, as it would dynamically adjust the aggregated weight and effective local steps based on local progress, alleviating inconsistency.

Federated Labels (FedLbl) [50] offers a method for aggregating client model updates in federated learning. In contrast to conventional methods that weigh local updates solely based on the data volume, FedLbl considers the number of classes or labels each client possesses. The method categorizes local models into two groups based on the heterogeneity level of their respective dataset, determined by the number of classes or labels. If a client's dataset contains fewer labels than a predefined threshold, its local model is assigned to a group; otherwise, it is assigned to another group. This approach improves the aggregation process by accounting for the diversity in data distributions across clients' datasets.

3.1.3 Optimization-based Approach

Optimization-based techniques are crucial for addressing non-IID data challenges in FL, which mainly includes adaptive optimization and regular optimization. Through combining algorithmic strategies with optimization techniques, FL systems can effectively confront the challenges posed by non-IID data.

(1) Adaptive Optimization

Researchers develop various adaptive optimization algorithms which can be applied to either clients or servers, aiming to empower clients with the ability to adapt to local data and enhance the model's resilience to the influence of data heterogeneity. Adaptive optimizers like ADAGRAD, ADAM, and YOGI have been adapted for federated learning settings in [51], proved to improve convergence. These adaptive optimization methods dynamically adjust learning processes based on clients' data characteristics, fine-tuning learning rates, regularization parameters, and other hyperparameters to accommodate data heterogeneity.

In medical scenarios, where data originate from diverse sites with varying scanner settings, the issue of statistical heterogeneity becomes particularly pronounced. To address this issue, [52] introduced Inverse Distance Aggregation (IDA), a novel adaptive weighting scheme for aggregating clients' models. IDA calculates weights based on statistical meta-information, prioritizing clients whose data closely aligns with the global average. This approach mitigates inconsistency among updated local parameters induced by non-IID data.

(2) Regular Optimization

Regular optimization is a widely adopted technique in machine learning, aiming to expedite model convergence and improve generalization capabilities. By integrating regular optimization into both local and global objective functions, models can potentially become more robust to data heterogeneity with minimal computational overhead.

In Federated Averaging (FedAvg), client drift is a notable challenge in the presence of heterogeneous data, resulting in unstable and slow convergence. To address this issue, [53] introduced a novel algorithm called Stochastic Controlled Averaging (SCAFFOLD). SCAFFOLD employs a client-variance reduction technique to rectify the observed client drift in local updates. Additionally, SCAFFOLD utilizes the similarity among client data to achieve faster convergence.

FedProx [54], a modification on FedAvg, introduces a proximal term into the objective function to minimize the gap between local and global models. This adjustment alleviates the adverse effects of system heterogeneity by allowing devices to perform variable amounts of work locally based on their available system resources. The server aggregates the partial solutions transmitted by these devices, leading to more stable convergence.

3.2 Moving Target Defense

Moving Target Defense emerges as a novel technology to overcome the static nature of traditional defense mechanisms by continually altering the attack surface of a protected system. This dynamic approach aims to confuse attackers, making it more challenging and costly to target specific system components and launch successful attacks. MTD layers are typically built on existing technologies, eliminating the need for developing new security mechanisms and reducing both development costs and the risk of incompatibility. Due to its convenience and enhanced security benefits, MTD has been adopted across various domains, including federated learning, web applications, and cloud computing.

3.2.1 MTD in Federated Learning

Robust aggregation rules and secure federated ensemble strategies are two commonly adopted defense approaches in FL. However, most of these aggregation rules and ensemble

strategies were originally designed for CFL, rather than DFL [37]. After recognizing the growing importance of security in FL and the gradual spread of FL techniques, researchers have recently started exploring the application of MTD mechanisms on FL to mitigate cyberattacks and enhance system robustness.

Voyager [37], a reactive MTD-based aggregation protocol, is proposed to enhance the resilience of DFL against poisoning attacks by manipulating network topology connectivity. Voyager comprises three stages, each with a key component: an anomaly detector, a network topology explorer, and a connection deployer. The anomaly detector compares received models from neighbors with the local model to identify potential anomalies. Upon detecting a malicious model, a trigger message is sent, prompting the network topology explorer to identify more trustworthy participants within the network. The connection deployer then alters the network topology to fortify system security. Experimental results demonstrate that Voyager effectively mitigates various poisoning attacks without imposing significant resource and computational burdens on participating nodes. However, Voyager’s efficiency has only been analyzed under IID settings.

Similar to Voyager, [32] proposed an MTD mechanism that continuously shuffles network topology but in a proactive manner. [32] introduced a security module, integrating MTD techniques with encryption in DFL. This security module utilizes symmetric encryption to safeguard model exchanges and asymmetric encryption for efficient key management, helping to prevent potential breaches and enhance data confidentiality. The security module integrates a proactive MTD layer, incorporating random neighbor selection and IP/port hopping. By continually altering communication routes and nodes, these techniques increase difficulties for potential attackers to establish a foothold in the system due to increased complexity and resource requirements. Evaluation results indicate that this security module effectively mitigates risks associated with eavesdropping or eclipse attacks.

CyberForce [55] is an innovative collaborative MTD selection mechanism that harnesses Federated and Reinforcement Learning (FRL) to collectively and securely determine the most effective MTD techniques for countering a broad spectrum of zero-day attacks. With behavioral fingerprinting and anomaly detection, CyberForce can swiftly identify zero-day threats and provide real-time feedback to its FRL-based agent. This agent utilizes Deep-Q Learning to strategically select optimal MTD mechanisms tailored to specific attack patterns. CyberForce encompasses four crucial MTD techniques, from which it can select: (i) IP shuffling, which dynamically migrates the private IP address of the targeted victim, effectively countering C&C attacks; (ii) Ransomware trap, which creates dummy files that are subsequently encrypted by ransomware attacks; (iii) File randomization, which alters file format extensions, concealing the files from manipulation; and (iv) Library sanitation, which shuffles shared system libraries across different sets and cleans associated links.

3.2.2 MTD in Other Domains

MTD techniques are not confined solely to federated learning. They have been implemented across diverse domains, showcasing their versatility and effectiveness in safeguarding against malicious attacks.

Except for the manipulation of network topology and IP addresses, MTD can be applied to various system attributes, such as instruction sets, address space layouts, virtual machines, operating systems, and software programs. This adaptability results in a wide range of MTD mechanisms. For instance, [28] applied Kerckhoff’s principle to create randomized and unique instruction sets for running process, protecting against code-injection attacks. This approach is equally applicable to machine-code executables and interpreted code. An attacker without knowledge of the randomization algorithm will inject code that is incompatible with the execution environment and invalid for the randomized processor, leading to runtime exceptions. For example, injecting x86 machine code into a process running on a SUN/SPARC system would likely to cause the process to crash, effectively thwarting the attack.

MTD can be applied to every layer of web applications to prevent from web application exploits. [27] implemented two MTD approaches to enhance security. One implementation involves source code language diversification at the logic layer. By developing a translator to automatically switch the web application’s programming language, this technique mitigates code injection exploits, as some vulnerability classes are specific to certain programming languages. Another technique is database dialect diversification at the storage layer, which shifts the database implementation to prevent SQL injection exploits.

[30] combined shuffle and diversity MTD techniques to address security issues in cloud computing. They implemented Virtual Machine Live Migration (VM-LM), allowing VMs to migrate from one physical host to another and generate VM replicas. They found that properly combining these MTD techniques not only improves the cloud’s security posture but also enhances its reliability. Using Graphical Security Models (GSMs) and various security metrics, they demonstrated the effectiveness of this hybrid MTD approach in the cloud environment.

3.3 Motivation

This chapter provides a comprehensive review on the non-IID data handling techniques in FL and explores MTD strategies, briefly listed in Table 3.1 and Table 3.2. While FL frameworks often assume data to follow an IID pattern, real-world data distributions usually deviate from this ideal scenario. This discrepancy results in suboptimal performance of FL models when confronted with non-IID data distributions. To improve the performance and adaptability of FL to real-world non-IID settings, this study adjusts the clients’ objective functions to tackle challenges associated with non-IID data.

Security is of vital importance within FL systems. MTD techniques offer a promising avenue for enhancing resilience of FL systems against adversarial attacks. Despite their potential, there is a notable dearth of research on the application of MTD in FL, especially in DFL environments. To bridge this gap, this work designs and implements an MTD strategy specifically tailored to non-IID DFL environments, aiming to safeguard FL systems and mitigate evolving adversarial threats.

Table 3.1: Non-IID Data Handling Techniques

Category	Method	Reference	Year	Detail
Data	Data Sharing	[39]	2018	Hold a globally shared IID dataset
	Data Sharing	[40]	2020	Partial clients upload data to construct an IID dataset
	Data Augmentation	[41]	2018	Server trains a GAN model using clients' local datasets
	Data Augmentation	[42]	2022	Clients train GANs locally, and aggregated by a coordinator
	Data Augmentation	[43]	2021	Using K-SMOTE to synthesize data samples
	Data Selection	[44]	2021	Server selects clients with more balanced datasets
Algorithm	Data Selection	[45]	2020	Reinforcement learning agent selects optimal subsets of clients
	Personalization	[46]	2017	Federated multitask learning
	Personalization	[47]	2019	Base + Personalization Layer
	Client Clustering	[48]	2022	Select neighbors with similar objectives
	Client Clustering	[5]	2023	Identify similarities among outliers to detect minority groups
	Model Aggreaction	[49]	2020	FedNova, computes the average of normalized local gradients
Optimization	Model Aggreaction	[50]	2023	Categorize local models based on their data heterogeneity level
	Adaptive Optimization	[51]	2020	ADAGRAD, ADAM, and YOGI
	Adaptive Optimization	[52]	2020	Calculate weights based on statistical meta-information
	Regular Optimization	[53]	2020	SCAFFOLD, employs a client variance reduction technique
	Regular Optimization	[54]	2020	FedProx, adds a proximal term to the objective function

Table 3.2: MTD Applications in Various Domains

Reference	Year	Domain	Method	When	Threat
[37]	2023	DFL	Shuffling (network topology)	Reactive	Poisoning Attack
[32]	2024	DFL	Shuffling (network topology + IP address), encryption	Proactive	Communication Security
[55]	2023	CFL	Hybrid: Shuffling (IP address + library sanitation), Diversity (file randomization), Redundancy (ransomware trap)	Reactive	Zero-Day Attack
[28]	2003	Software	Diversity (instruction set)	Proactive	Code Injection Attack
[27]	2015	Web Application	Diversity (programming language)	Proactive	Code Injection Attack & SQL Injection Attack
[30]	2020	Cloud Computing	Hybrid: Shuffling (virtual machine), Diversity (operating system)	Proactive	Cyber Attack
This work	2024	DFL	Shuffling (network topology + aggregation algorithm)	Proactive & Reactive	Poisoning Attack

Chapter 4

Architecture

This chapter presents the design choices and architecture of the defense system developed for DFL with non-IID data. The system comprises three primary components: the reputation system, MTD strategies, and non-IID data handling approaches, each of which will be explained in detail in the following sections.

4.1 Workflow

The workflow of the defense mechanism is illustrated in Figure 4.1. Each training round consists of the following steps:

- (1) Local Training: Each node trains local model on its respective local dataset.
- (2) Exchange Model Parameters: Each node sends its local model parameters to neighboring nodes and receives model parameters from them.
- (3) Calculate Reputation Score: Each node evaluates the reputation score of the received models based on model similarity and loss on its local test dataset. This evaluation helps in detecting suspicious nodes.
- (4) Apply MTD Strategies:
 - Reactive Mode: When a malicious model is detected, it triggers network topology and aggregation algorithm MTD strategies.
 - Proactive Mode: MTD strategies are applied regardless of whether there is a suspicious node.
- (5) Model aggregation: Nodes aggregate models received from neighbors and their local model to update and improve their own model.

This workflow ensures that the defense system can dynamically adapt to potential threats, maintaining the integrity and performance of the decentralized federated learning process.

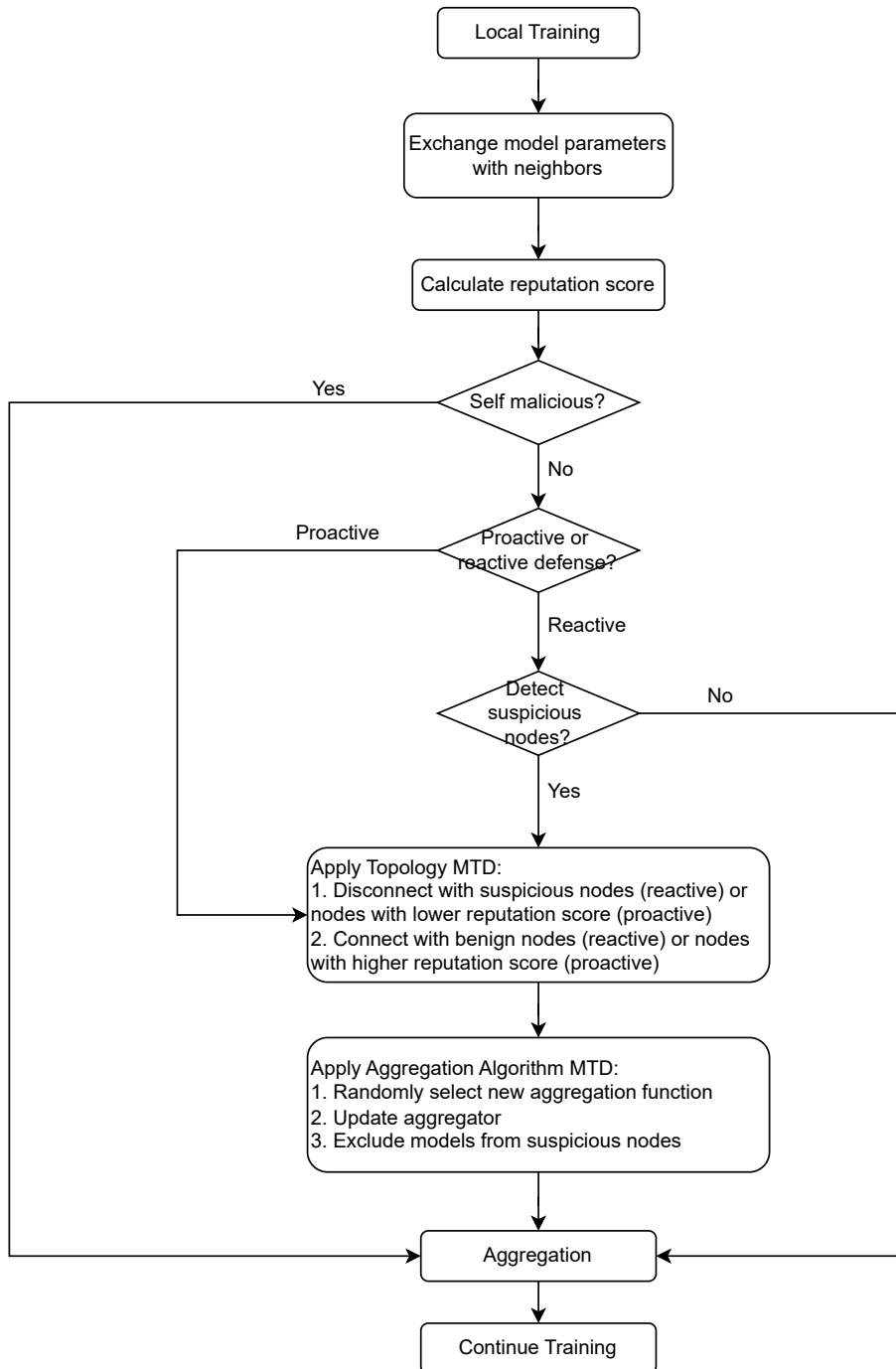


Figure 4.1: Workflow of Defense Mechanism

4.2 Reputation System

This study employs a reputation scoring mechanism that evaluate both model similarity and loss on the test dataset to provide a comprehensive assessment of each node's trustworthiness. Through reputation scores, the system can identify malicious or underperforming nodes.

4.2.1 Model Similarity

To quantify the similarity between local and received neighbor models, cosine similarity and Euclidean distance are two commonly used metrics:

(1) Cosine Similarity

Cosine similarity measures the cosine of the angle between two non-zero vectors defined in an inner product space. It disregards magnitudes of the vectors, and depend solely on their angle. The cosine similarity always belongs to the interval $[-1, +1]$. Cosine similarity is defined as:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \quad (4.1)$$

where A and B are the model parameter vectors.

(2) Euclidean Distance

Euclidean distance measures the straight-line distance between two points in Euclidean space, emphasizing magnitude differences between corresponding vector elements. Euclidean distance is defined as:

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^n (A_i - B_i)^2} \quad (4.2)$$

where A_i and B_i are the corresponding elements of the model parameter vectors.

To determine which metric can better identify malicious models from benign models, a preliminary experiment was conducted in the DFL network context. The experiment involved ten nodes, with nodes 2, 4, and 8 subjected to untargeted label flipping attacks, while other nodes operated normally as benign nodes. The results, as shown in Figure 4.2, indicated that the Euclidean distance between malicious and benign models is more distinguishable compared to cosine similarity. Therefore, this study quantifies model similarity by computing the layer-wise Euclidean distance between each node's local model and the models received from neighboring nodes in the DFL network. The Euclidean distance-based model similarity algorithm is outlined in Algorithm 1.

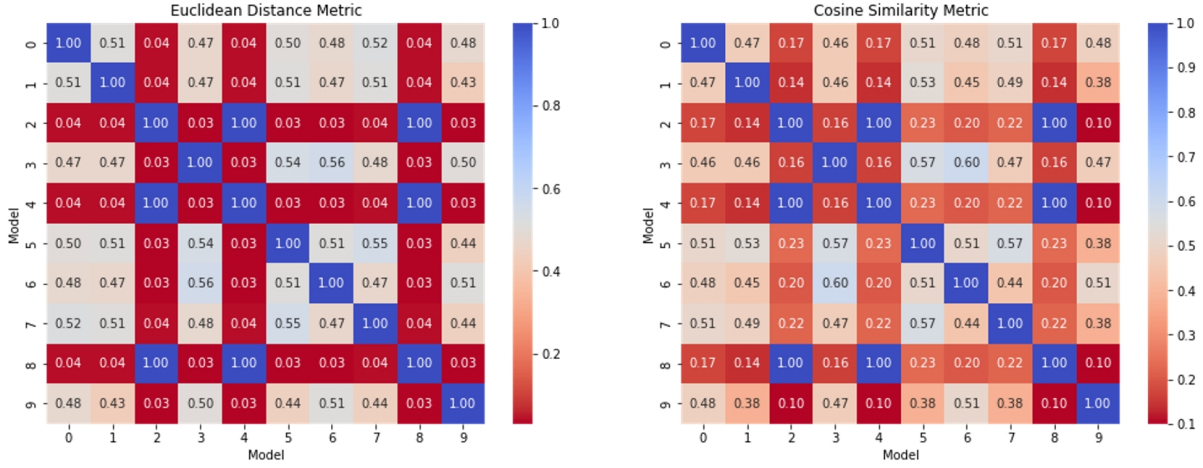


Figure 4.2: Model Similarity Measurement Using Euclidean Distance and Cosine Similarity

Algorithm 1 Euclidean Distance

Require: m_i : local model, m_j : neighbor model

- 1: Initialize $distance \leftarrow 0$
 - 2: **for** l_i, l_j in m_i, m_j **do**
 - 3: $diff \leftarrow l_i - l_j$
 - 4: $squared_diff \leftarrow diff^2$
 - 5: $distance \leftarrow distance + squared_diff$
 - 6: **end for**
 - 7: $distance \leftarrow \sqrt{distance}$
 - 8: **return** $distance$
-

4.2.2 Neighbor Model Loss

In addition to model similarity, the loss of the received model on the local test dataset is also evaluated. This loss measurement helps in identifying models that do not perform well on the local data, further aiding in the identification of potentially malicious nodes. The loss evaluation algorithm is outlined in Algorithm 2.

Algorithm 2 Neighbor Model Loss

Require: m_j : neighbor model, D_i : local test dataset

- 1: Initialize $L \leftarrow 0$
 - 2: **for** each $(x, y) \in D_i$ **do**
 - 3: $y_{pred} \leftarrow \text{Predict with } m_j \text{ on } x$
 - 4: $L \leftarrow L + l(y, y_{pred})$ ▷ Compute loss for the current sample and accumulate
 - 5: **end for**
 - 6: $L \leftarrow L / |D_i|$ ▷ Compute average loss
 - 7: **return** L
-

4.2.3 Reputation Threshold

Typically, building an anomaly detector involves setting a fixed threshold based on the results of the node reputation. However, such fixed thresholds are unsuitable in non-IID settings where model similarity and loss evaluation differ because of the variation in local data distributions. To address this issue, a dynamic reputation threshold mechanism, utilizing the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [56] algorithm, is adopted.

Algorithm 3 Dynamic Reputation Threshold

Require: M'_i : neighbor models, m_i : local model, *sensitive*: sensitivity parameter, D_i : local test dataset

- 1: Initialize *trigger* $\leftarrow$ False, *similarity_threshold* $\leftarrow$ 0, *loss_threshold* $\leftarrow$ 10
- 2: *nei_similarity_score* $\leftarrow$ []
- 3: *nei_loss_score* $\leftarrow$ []
- 4: **for** each $m_j \in M'_i$ **do**
- 5: $s_{ij} \leftarrow \text{EuclideanDistance}(m_i, m_j)$
- 6: Append s_{ij} to *nei_similarity_score*
- 7: $Loss_{ij} \leftarrow \text{reputation_loss}(m_j, D_i)$
- 8: Append $Loss_{ij}$ to *nei_loss_score*
- 9: **end for**
- 10: Apply DBSCAN on *nei_similarity_score* with $\text{eps} = \text{sensitive}$ to get clusters
- 11: **if** number of unique clusters > 1 **then**
- 12: *trigger* $\leftarrow$ True
- 13: Calculate *similarity_threshold* as the mean of the highest lower bound and the lowest upper bound of the clusters
- 14: **end if**
- 15: Apply DBSCAN on *nei_loss_score* with $\text{eps} = \text{sensitive}$ to get clusters
- 16: **if** number of unique clusters > 1 **then**
- 17: *trigger* $\leftarrow$ True
- 18: Calculate *loss_threshold* as the mean of the highest lower bound and the lowest upper bound of the clusters
- 19: **end if**
- 20: **if** only one unique cluster in both similarity scores and loss scores **then**
- 21: *loss_threshold* $\leftarrow$ loss score of m_i
- 22: **end if**
- 23: **return** *similarity_threshold, loss_threshold, trigger*

DBSCAN is a popular unsupervised machine learning algorithm used for clustering data based on density. Unlike other clustering algorithms, DBSCAN does not require specifying the number of clusters a priori. Instead, it groups together points that are closely packed, marking regions of high density, while identifying points in sparse areas as outliers. DBSCAN operates with two primary parameters: ϵ (eps), which defines the maximum distance between two points to be considered neighbors, and minPts, the minimum number of points required to form a dense region. Iteratively expanding clusters from points that have a sufficient number of neighbors, DBSCAN is particularly effective for discovering clusters of arbitrary shape and for handling noisy datasets. This makes it well-suited

for applications where the data distribution is complex or unknown, and where outlier detection is crucial.

In this study, after computing the reputation scores of neighboring nodes, the DBSCAN algorithm is applied to cluster these scores and determine the number of clusters. The similarities between benign models, which are trained on non-IID data but follow a legitimate distribution, typically form a single cluster due to their inherent consistency despite data variations. Similarly, the loss values for these models on local test data tend to cluster together. However, malicious models, introduced by adversaries, deviate significantly in their parameters or performance, creating distinct clusters.

Benign nodes, even with non-IID data, will generally exhibit similar model similarities and loss values. In contrast, malicious nodes behave abnormally, often generating outliers that form distinct clusters. Therefore, if multiple clusters are detected either in model similarity or loss evaluation, it indicates the presence of malicious nodes. Then, the system can dynamically adjust reputation thresholds based on these clusters' boundaries, enhancing the detection and mitigation of attacks. This procedure is illustrated in Algorithm 3.

4.3 Moving Target Defense

MTD enhances the security of the DFL network by dynamically altering the system's configuration, making it more difficult for attackers to predict and exploit vulnerabilities. While IP addresses and port numbers are commonly chosen attributes for switching in traditional systems, DFL provides more possibilities for MTD strategy design due to its distinctive characteristics. Compared to CFL, DFL allows each node to perform aggregation and enables direct communication between nodes. These unique features make the network topology and aggregation algorithm potential attributes for MTD strategies. This study utilizes these distinct characteristics of DFL and applies MTD strategies to both the network topology and the aggregation algorithm.

4.3.1 Topology MTD

Topology MTD dynamically alter communication links between nodes, aiming to disrupt predictable patterns that attackers could exploit, prevent adversaries persistently targeting specific nodes or communication paths. The topology MTD strategy is outlined in Algorithm 4 and can be summarized as follows:

- (1) Reputation Score Computing: The process begins with computing the similarity between neighboring models and local model, and evaluating the loss on the local test dataset.
- (2) Reputation Score Clustering: With the DBSCAN algorithm, the network clusters models based on their similarity and loss scores to identify potential malicious nodes.
- (3) Proactive Topology MTD:

- When clusters of model similarity or loss indicate the presence of malicious nodes (more than one cluster), reputation thresholds are dynamically set based on these clusters' boundaries.
- Nodes then compare the reputation scores of their neighbors against these thresholds. They disconnect from nodes scoring below the similarity threshold or above the loss threshold and attempt to establish connections with those scoring above the similarity threshold and below the loss threshold.
- If only one cluster exists, suggesting no malicious nodes detected, the local model's loss on the local test dataset becomes the loss threshold. Nodes disconnect from neighbors with losses higher than this threshold and seek connections with nodes having lower losses. If the predefined connection amount is not met, nodes prioritize connecting with nodes having higher model similarities.

Algorithm 4 Topology MTD

Require: L : neighbor nodes, C : all nodes in the network, κ_n : neighbor amount threshold, nei_loss_score : neighbor model loss on local test dataset, $nei_similarity_score$: similarity between neighbor model and local model, κ_l : loss threshold, κ_s : similarity threshold

```

1: for each  $c_i \in C$  do
2:   if  $c_i \in L$  then
3:     if  $nei\_loss\_score[c_i] > \kappa_l$  or  $nei\_similarity\_score[c_i] < \kappa_s$  then
4:        $L \leftarrow L - c_i$ 
5:     end if
6:   else
7:     if  $nei\_loss\_score[c_i] \leq \kappa_l$  and  $nei\_similarity\_score[c_i] \geq \kappa_s$  then
8:        $L \leftarrow L + c_i$ 
9:     end if
10:  end if
11: end for
12: if mode is proactive and  $current\_neighbor < \kappa_n$  then
13:    $sorted\_nodes \leftarrow \text{sort } C \text{ by } nei\_similarity\_score \text{ in descending order}$ 
14:   for each  $c_i \in sorted\_nodes$  do
15:     if  $c_i \notin L$  then
16:        $L \leftarrow L + c_i$ 
17:       if  $current\_neighbor \geq \kappa_n$  then
18:         break
19:       end if
20:     end if
21:   end for
22: end if

```

(4) Reactive Topology MTD:

- Triggered when clusters of model similarity or loss suggest the presence of malicious nodes (more than one cluster).

- Similar to proactive mode, nodes set reputation thresholds based on the boundaries of identified clusters and compare neighbor scores against these thresholds.
- Nodes disconnect from neighbors that do not meet the threshold requirements and establish connections with those that do, minimizing exposure to potentially compromised nodes.
- If only one cluster exists, indicating no malicious node detected, reactive topology MTD will not be triggered.

4.3.2 Aggregation Algorithm MTD

Aggregation algorithm MTD periodically switches the aggregation algorithms used by nodes. Different aggregation algorithms have distinct strengths and varying degrees of robustness to different attacks. Continuously shuffling the aggregation strategy might keep the system robust under various types of attacks, and reduce the success rate of attacks. This study maintains an aggregation algorithm pool that includes:

- FedAvg [2]: Performs a weighted average of the updates, commonly used in federated learning.
- Krum [57]: Selects the update that is closest to the average of the remaining updates, excluding the outliers.
- Median: Uses the median of the updates, which is less sensitive to outliers.
- Trimmed Mean: Computes the mean of the updates after discarding a fixed proportion of the largest and smallest values.

The aggregation algorithm MTD strategy is outlined in Algorithm 5 and can be summarized as follows:

(1) Aggregation Algorithm Pool: The system maintains a pool of aggregation algorithms that nodes can switch between during the federated learning process.

(2) Modes of Aggregation Algorithm MTD:

- Proactive Mode: If a node itself is not malicious, it will randomly select an aggregation algorithm from the pool in each round.
- Reactive Mode: If a node itself is not malicious and detects a malicious node through the reputation system, it will randomly select an aggregation algorithm from the pool in response to the threat and exclude the malicious nodes from the aggregation process.

Algorithm 5 Aggregation Algorithm MTD

Require: A : aggregation algorithm pool, M : aggregated models and weights, N_m : malicious nodes

Ensure: Updated Aggregator

```

1:  $A \leftarrow ["Krum", "Median", "TrimmedMean"]$  ▷ Set of aggregation algorithms
2:  $A\_target \leftarrow \text{random choice}(A)$  ▷ Select new aggregation algorithm
3:  $A\_current \leftarrow A\_target$  ▷ Change the aggregator
4:  $A\_current.set\_nodes\_to\_aggregate(self.trainSet)$ 
5:  $A\_current.setRound(self.round)$ 
6: for  $s \in M.keys()$  do
7:   if  $s \notin N_m$  then
8:      $(m_s, w_s) \leftarrow M[s]$  ▷ Get model and weights
9:      $A\_current.addModel(s, m_s, w_s)$  ▷ Add model
10:  end if
11: end for

```

4.4 Non-IID Data Handling

The non-IID nature of data across different nodes is a significant challenge in FL. This data heterogeneity can result in skewed model performance and convergence difficulties. To address the issue, this study utilizes FedProx to mitigate the impact of non-IID data.

FedProx introduces a proximal term into the loss function, which is defined as follows:

$$\mathcal{L}_{\text{FedProx}}(w) = \mathcal{L}_{\text{local}}(w) + \frac{\mu}{2} \|w - w^t\|^2 \quad (4.3)$$

where $\mathcal{L}_{\text{local}}(w)$ is the local loss function at node i , w represents the local model parameters, w^t denotes the global model parameters at the previous iteration, and μ is a regularization parameter that controls the trade-off between local data integrity and global model consistency.

With this penalty term, FedProx could reduce the discrepancies caused by non-IID data distributions among nodes, contributing to more stable model updates and improving the overall performance of the DFL system.

Chapter 5

Implementation

In this chapter, the detailed implementation of the proposed defense mechanism is presented, focusing on three primary components: the reputation system, MTD strategies, and approaches for handling non-IID data. Each component is crucial for maintaining the robustness and security of the DFL network.

5.1 Reputation System

The reputation system in this study comprises three key elements: model similarity, neighbor model loss, and dynamic reputation thresholds. These components form the basis of the MTD strategies and work together to evaluate the trustworthiness of participating nodes.

5.1.1 Model Similarity

Listing 1 demonstrates the process of calculating the Euclidean distance between a local model and its neighboring models to measure model similarity. The procedure involves flattening the model tensors and optionally standardizing them. The function then iterates through each layer of the local model, calculating the Euclidean distance to the corresponding layer in each neighboring model. After computing the distances for all layers, the function returns their average. This resulting Euclidean distance provides a measure of how similar the models are, with smaller distances indicating higher similarity.

The Euclidean distance is effective for distinguishing between benign models, which typically have smaller distances due to legitimate updates, and potentially malicious models, which are likely to have larger distances due to adversarial manipulation.

Listing 1 Model Similarity Measured By Euclidean Distance [58]

```

1 def euclidean_metric(model1: OrderedDict[str, torch.Tensor],
2                       model2: OrderedDict[str, torch.Tensor],
3                       standardized: bool = False,
4                       similarity: bool = True) -> Optional[float]:
5     if model1 is None or model2 is None:
6         return None
7     distances = []
8
9     for layer in model1:
10         if layer in model2:
11             l1 = model1[layer].flatten().to(torch.float32)
12             l2 = model2[layer].flatten().to(torch.float32)
13             if standardized:
14                 l1 = (l1 - l1.mean()) / l1.std()
15                 l2 = (l2 - l2.mean()) / l2.std()
16
17             distance = torch.norm(l1 - l2, p=2)
18             if similarity:
19                 norm_sum = torch.norm(l1, p=2) + torch.norm(l2, p=2)
20                 similarity_score = 1 - (distance / norm_sum if norm_sum != 0 else 0)
21                 distances.append(similarity_score.item())
22             else:
23                 distances.append(distance.item())
24
25     if distances:
26         avg_distance = torch.mean(torch.tensor(distances))
27         return avg_distance.item()
28     else:
29         return None

```

5.1.2 Neighbor Model Loss

As shown in Listing 2, the performance of each neighboring model is evaluated by computing the loss on the local test dataset. The function iterates through the test dataset, passing the inputs through the neighboring model to obtain outputs, and calculates the cross-entropy loss with the true labels. These losses are then accumulated and averaged over the dataset. Finally, the average loss for each neighboring model is stored and constitutes part of its reputation score.

This process helps in identifying poorly performing models, which could indicate either malicious intent or inadequate generalization capability.

5.1.3 Reputation Threshold

To dynamically set reputation thresholds, the system employs the DBSCAN clustering algorithm to analyze both model similarity and neighbor model loss. The implementation details are outlined in Listing 3.

Listing 2 Neighbor Model Loss on Local Test Dataset

```

1 def cal_reputation(self):
2     nei_loss_score = {}
3     current_round_nei_models = self.nei_model_record[self.current_round]
4     for nei in current_round_nei_models:
5         nei_model_param = current_round_nei_models[nei]
6         avg_loss = 0
7         sum_loss = 0
8         neighbour_model = MNISTModelMLP()
9         neighbour_model.load_state_dict(nei_model_param)
10
11         with torch.no_grad():
12             for inputs, labels in self.test_dataset:
13                 outputs = neighbour_model(inputs)
14                 loss = F.cross_entropy(outputs, labels)
15                 sum_loss += loss.item()
16             avg_loss = sum_loss / len(self.test_dataset)
17             nei_loss_score[nei] = avg_loss
18     return nei_loss_score

```

The process begins by initializing a trigger flag to False. DBSCAN clustering is then applied to the similarity scores and loss scores separately, with the sensitivity parameter controlling the granularity of clustering. If multiple clusters are identified in either the model similarity or loss scores, the trigger flag is set to True, indicating the potential presence of malicious activity.

Reputation thresholds are computed based on the clustering results. Specifically, the thresholds are determined by averaging the highest lower bound and the lowest upper bound of the identified clusters. This calculation provides thresholds that are used to effectively identify and filter out potential anomalies. In cases where no malicious nodes are detected, the loss threshold is set directly to the node's own loss score. This ensures that nodes with poor performance are excluded from further consideration.

This dual analysis enables a dynamic and responsive reputation management framework, where the system can detect malicious nodes through the identification of anomalies in both model similarities and loss evaluations.

5.2 Moving Target Defense

MTD strategies in this study include both topology-based and aggregation algorithm-based approaches, each of which is implemented in proactive and reactive modes. These MTD strategies collectively aim to strengthen network security against evolving threats in distributed learning environments. They introduce variability and unpredictability in both the network topology and the aggregation process, preventing attackers from predicting and exploiting vulnerabilities.

5.2.1 Topology MTD

Topology MTD shuffles communication links between nodes based on calculated reputation thresholds. Nodes with high model similarity and low losses are selectively connected, while those failing to meet these criteria are disconnected to mitigate potential threats. This study has implemented both proactive and reactive modes of operation for topology MTD:

- (1) Proactive Mode: In each round, nodes proactively switch communication partners while maintaining a predefined minimum number of neighbor connections.
- (2) Reactive Mode: When reputation score clustering indicates the presence of malicious nodes (i.e., more than one cluster is detected), the topology is reconfigured to isolate these nodes.

The implementation details of topology MTD are shown in Listing 4. This process involves recomputing reputation scores in each round, applying the DBSCAN algorithm for clustering, and calculating thresholds. Neighbor nodes are removed from the list if their loss score exceeds the loss threshold or their similarity score falls below the similarity threshold. If no malicious nodes are detected (i.e., trigger is False) and the number of current connections is below the target, nodes are sorted by similarity scores and high-scoring nodes are added to the neighbor list to maintain an adequate amount of neighbors. This iterative approach enables timely adaptation to network dynamics and effective mitigation of security risks.

5.2.2 Aggregation Algorithm MTD

Aggregation algorithm MTD periodically alters the aggregation algorithms used by nodes to enhance resilience against attacks. This study has implemented both proactive and reactive modes of operation for aggregation MTD:

- (1) Proactive Mode: Each node randomly selects an aggregation algorithm from a predefined pool (Krum, Median, TrimmedMean) in each round.
- (2) Reactive Mode: Upon detecting multiple clusters, the reactive MTD strategy is triggered, and the benign nodes randomly select an aggregation algorithm to mitigate the impact of malicious models.

The implementation for aggregation algorithm MTD is illustrated in Listing 5. It involves maintaining a pool of aggregation algorithms and selecting one based on the operation mode.

Listing 6 demonstrates the aggregation process, which integrates several key techniques. Initially, it calculates reputation scores for neighboring nodes using the Euclidean metric

and neighbor model loss. Based on these scores, it determines the appropriate thresholds and triggers. Depending on whether the system is operating in proactive or reactive mode, it applies MTD techniques. In proactive mode, the network topology and aggregation algorithms are dynamically updated in each round. In contrast, in reactive mode, updates are triggered only upon the detection of malicious nodes. This shuffling-based MTD strategy provides robust defense throughout the aggregation process.

Listing 6 Aggregation Process

```

1 def aggregation(self, testing:bool=True):
2     current_round_nei_models = self.nei_model_record[self.curren_round]
3     nei_models_list = []
4
5     if self.curren_round > 0:
6         # calculate the reputation score
7         nei_similarity_score, nei_loss_score = self.cal_reputation(euclidean_metric)
8         # get the reputation thresholds
9         sim_threshold, loss_threshold, trigger =
10         ↪ self.get_threshold_trigger(nei_similarity_score, nei_loss_score)
11
12         # MTD
13         if self.is_proactive:
14             # proactive
15             if self.dynamic_agg:
16                 self.dynamic_aggregation()
17             if self.dynamic_topo:
18                 self.dynamic_topology(trigger, nei_similarity_score, nei_loss_score,
19                 ↪ sim_threshold, loss_threshold)
20         else:
21             # reactive
22             if trigger:
23                 if self.dynamic_agg:
24                     self.dynamic_aggregation()
25                 if self.dynamic_topo:
26                     self.dynamic_topology(trigger, nei_similarity_score,
27                     ↪ nei_loss_score, sim_threshold, loss_threshold)
28
29         ..... # further aggregation steps

```

5.3 Non-IID Data Handling

To handle non-IID data distributions among participating nodes, this study employs FedProx. FedProx incorporates a proximal term into the local objective function, which stabilizes model updates and minimizes divergence between local and global models. As a result, local updates remain close to the parameters of the previous aggregated global model. The detailed implementation is shown in Listing 7.

Listing 7 FedProx Technique for Non-IID Data Handling

```

1 def step(self, batch, phase):
2     images, labels = batch
3     images = images.to(self.device)
4     labels = labels.to(self.device)
5     y_pred = self.forward(images)
6     loss = self.criterion(y_pred, labels)
7
8     if phase == "Train" and self.previous_global_model_params is not None:
9         prox_term = 0.0
10        current_params = self.named_parameters()
11        for name, param in current_params:
12            if name in self.previous_global_model_params:
13                prev_param = self.previous_global_model_params[name]
14                prox_term += ((param - prev_param) ** 2).sum()
15        # 0.1 is the proximal term coefficient
16        loss += 0.5 * 0.1 * prox_term

```

As Listing 7 demonstrates, during the training phase, when previous global model parameters are available, the proximal term is computed. This term measures the squared difference between the current local model parameters and the previous global model parameters. This difference is accumulated across all parameters that are present in both the local and global models. The proximal term is then scaled by a coefficient (0.1 in this case) and added to the loss function. This adjustment helps maintain consistency between local and global models by penalizing deviations from the global model parameters, which stabilizes the training process in non-IID environments.

Listing 3 Dynamic Reputation Thresholds with DBSCAN

```

1 def get_threshold_trigger(self, nei_similarity_score, nei_loss_score, sensitive=0.1):
2     trigger = False
3     similarity_threshold = 0
4     loss_threshold = 10
5
6     simList = list(nei_similarity_score.values())
7     lossList = list(nei_loss_score.values())
8     max_value = int(max(simList))
9     if max_value==1:
10         simList.remove(max_value)
11
12     X = np.array(list(simList)).reshape(-1, 1)
13     db = DBSCAN(eps=sensitive, min_samples=1)
14     clusters = db.fit_predict(X.reshape(-1,1))
15     unique_labels = set(clusters)
16
17     X2 = np.array(list(lossList)).reshape(-1, 1)
18     db2 = DBSCAN(eps=sensitive, min_samples=1)
19     clusters2 = db2.fit_predict(X2.reshape(-1,1))
20     unique_labels2 = set(clusters2)
21
22     if len(unique_labels) > 1:
23         print(f"[MTD] malicious detected [model similarity]!")
24         trigger = True
25         lowerbound = []
26         upperbound = []
27
28         for label in unique_labels:
29             cluster_points = X[clusters == label]
30             lowerbound.append(min(cluster_points))
31             upperbound.append(max(cluster_points))
32
33         # the threshold is the average of lowerbound of the first class and
34         ↪ upperbound of the last class
35         lowerbound.sort()
36         upperbound.sort()
37         similarity_threshold = np.mean([lowerbound[-1], upperbound[0]])
38
39     if len(unique_labels2) > 1:
40         print(f"[MTD] malicious detected [loss]!")
41         trigger = True
42         lowerbound = []
43         upperbound = []
44
45         for label in unique_labels2:
46             cluster_points = X2[clusters2 == label]
47             lowerbound.append(min(cluster_points))
48             upperbound.append(max(cluster_points))
49
50         # the threshold is the average of lowerbound of the first class and
51         ↪ upperbound of the last class
52         lowerbound.sort()
53         upperbound.sort()
54         loss_threshold = np.mean([lowerbound[-1], upperbound[0]])
55
56         # only used in proactive mode
57         if len(unique_labels) == 1 and len(unique_labels2) == 1:
58             loss_threshold = nei_loss_score[self.node_id]
59
60     return similarity_threshold, loss_threshold, trigger

```

Listing 4 Topology Moving Target Defense

```

1  def dynamic_topology(self, trigger, nei_similarity_score, nei_loss_score,
    ↪ similarity_threshold, loss_threshold):
2      if self.attack_type.lower() == 'no attack':
3          original_nei = len(self.neiList)
4          connected_node = len(self.neiList)
5          node_list = list(nei_similarity_score.keys())
6
7      for node_id in node_list:
8          if node_id in self.neiList:
9              if (nei_loss_score[node_id] > loss_threshold and
    ↪ nei_loss_score[node_id] > nei_loss_score[self.node_id]) or
    ↪ nei_similarity_score[node_id] < similarity_threshold:
10                 self.remove_nei_from_neiList(node_id)
11                 connected_node -= 1
12                 print(f"[dynamic_topology] in {self.node_id}: remove {node_id}
    ↪ from nei list")
13             else:
14                 if nei_loss_score[node_id] <= loss_threshold and
    ↪ nei_similarity_score[node_id] >= similarity_threshold:
15                     self.add_nei_to_neiList(node_id)
16                     connected_node += 1
17                     print(f"[dynamic_topology] in {self.node_id}: add {node_id} to
    ↪ nei list")
18
19         if trigger == False:
20             target_connections = original_nei
21             # Handle full connection case (original_nei == len(node_list))
22             if original_nei == len(node_list):
23                 target_connections = int(7 / 10 * original_nei)
24
25             # If still not enough connections, sort nei_similarity_score in
    ↪ descending order
26             if connected_node < target_connections:
27                 sorted_by_reputation = sorted(node_list, key=lambda x:
    ↪ nei_similarity_score[x], reverse=True)
28                 for node_id in sorted_by_reputation:
29                     if node_id not in self.neiList:
30                         self.add_nei_to_neiList(node_id)
31                         connected_node += 1
32                         print(f"[dynamic_topology] in {self.node_id}: add {node_id}
    ↪ to nei list based on model similarity")
33                     if connected_node >= target_connections:
34                         break
35

```

Listing 5 Aggregation Algorithm Moving Target Defense

```

1  def dynamic_aggregation(self):
2      aggregation_list = [krum, median, trimmedMean]
3      random_index = random.randint(0, len(aggregation_list)-1)
4      self.curr_aggregation = aggregation_list[random_index]

```

Chapter 6

Evaluation

This chapter evaluates the performance of the Moving Target Defense mechanism, which integrates topology MTD and aggregation algorithm MTD, across different datasets. This chapter first outlines the experiment setup, then presents the experimental results, and concludes with a discussion of the observations.

6.1 Experiment Setup

The following sections specify the datasets, their corresponding models, the threat model, and the experiment configurations used in the evaluations.

6.1.1 Datasets and Models

This study chose three datasets and corresponding deep learning models to evaluate the performance of the newly proposed MTD-based defense mechanism. Experiments were conducted under both IID and non-IID settings. The datasets were distributed according to the Dirichlet distribution with alpha parameters set to 100, 1, and 0.5, representing IID, medium non-IID, and severe non-IID, respectively.

- **MNIST** [59] is a widely-used handwritten digits (0-9) dataset for image recognition, consisting of 60,000 training images and 10,000 testing images, each in grayscale and sized 28x28 pixels. A Multilayer Perceptron (MLP) model is employed to learn the MNIST dataset, consisting of three fully connected layers: the first layer maps the 784-pixel flattened input to 256 neurons, the second layer reduces this to 128 neurons, and the final layer maps to the 10 output classes. ReLU activation functions follow the first two layers, and the final output passes through a softmax function. The model is optimized using Adam with a learning rate of $1e-3$, and cross-entropy loss is applied. The MLP is trained for 3 epochs per federated round.

- **FashionMNIST (FMNIST)** [60] is a dataset created as a more challenging alternative to MNIST for machine learning benchmarks. It contains 60,000 training and 10,000 testing grayscale images, each sized 28x28 pixels, across 10 fashion categories (e.g., trousers, dress, coat). A small convolutional neural network (CNN) is employed to learn FashionMNIST dataset. It comprises two convolutional layers: the first with 32 filters of size 3x3 and the second with 64 filters of size 3x3, each followed by ReLU activation. The output is then flattened and passed through two fully connected layers with 128 neurons and 10 neurons, respectively, ending with a log softmax function. Cross-entropy loss is used for classification, optimized by the Adam optimizer with a learning rate of $1e-3$. This CNN model is trained for 3 epochs per round.
- **CIFAR-10** [61] is another widely-used benchmark dataset for machine learning. It consists of 50,000 training and 10,000 testing images of size 32x32 pixels across 10 different classes (e.g., airplane, automobile, bird). Its colorful images and diverse classes make CIFAR-10 a more challenging classification task compared to MNIST and FashionMNIST. The CNN model designed for CIFAR-10 includes multiple convolutional layers with varying channels, stride values, and kernel sizes to extract features, followed by batch normalization and ReLU activation function. It concludes with a global adaptive average pooling layer to reduce spatial dimensions and a fully connected layer to produce classification logits. The model is trained using the Adam optimizer with a learning rate set to $1e-3$. This CNN model is trained for 5 epochs per round.

6.1.2 Threat Model

This study utilized untargeted label flipping, untargeted model poisoning and backdoor to evaluate the performance of the newly proposed MTD-based defense mechanism. Malicious participants are randomly selected from all clients. In label flipping, malicious participants alter a predefined proportion of data samples' labels to incorrect values. For example, in MNIST, the digit "4" may be labeled as digit "8"; in FashionMNIST, images originally labeled as "dress" may change to "sneaker"; and in CIFAR-10, images with label "cat" may change to label "horse." In model poisoning, malicious participants inject noise into their local model updates and share these corrupted model parameters with their neighbors. Backdoor attacks embed malicious behavior into a model, which can be triggered by specific inputs that the attacker controls. To characterize these poisoning attacks, the following parameters are used:

- **Poisoned Node Ratio (PNR)**: The percentage of malicious participants in the federated learning network.
- **Poisoned Sample Ratio (PSR)**: The percentage of data samples or labels modified by an attacker.
- **Noise Ratio (NR)**: The amount of noise injected into local data or models by an attacker.

This study focuses on the impact of varying PNRs on model performance, setting PSR and NR to 100% for all attacks and using PNR values of 10%, 30%, 50%, and 70% to represent low, medium, and highly poisoned environments. Variations in PNR provide a thorough assessment of the mechanism’s robustness.

6.1.3 Experiment Configuration

The following paragraphs describe the detailed configurations of the experiments. The experiments were performed using a total of 10 nodes. The initial network topology varied among fully-connected, ring, and star structures, as depicted in Figure 6.1. During training, the network topology could change dynamically if the topology MTD was enabled.

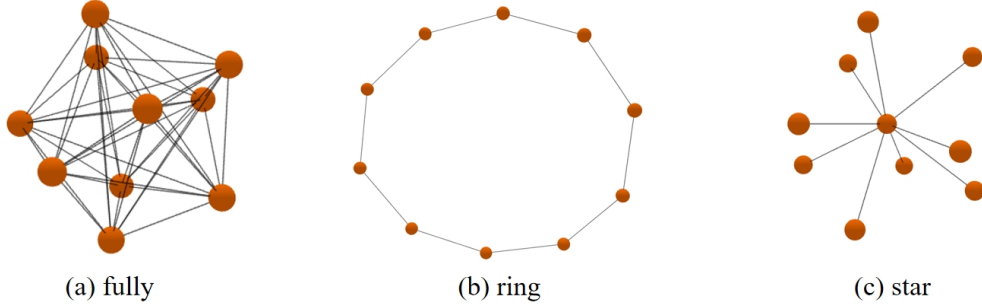


Figure 6.1: Initial Network Topology for Decentralized Federated Learning

Each node acted as a trainer and an aggregator. Different initial aggregation algorithms were tested, including FedAvg, Krum, Median, and Trimmed Mean, with all nodes using the same aggregation algorithm at any given time. During the training process, benign nodes could switch their aggregation algorithm based on whether the aggregation MTD was enabled and its operational mode (proactive or reactive). Three datasets, MNIST, FashionMNIST, CIFAR-10 were used. These data were distributed according to Dirichlet distributions with alpha values set to 100, 1, and 0.5 to represent IID, moderate non-IID, and severe non-IID scenarios, respectively. A 10% validation set was created from the training data for all experiments.

To ensure comparability of the experimental results, the selection of malicious participants was consistent across experiments using the same dataset and PNR. For instance, if nodes 0, 3, and 5 were designated as malicious in one experiment, this configuration was saved to a file and reused in subsequent experiments under the same conditions. Similarly, data distribution and initial node interconnections were kept consistent. For example, if experiment A and experiment B both started with a ring topology, each node’s neighbors remained the same at the beginning of both experiments.

The federated learning was configured to run for 10 rounds in all experiments. Round 0 executed a model diffusion, resulting in a total of 11 rounds, although local training occurred only in the subsequent 10 rounds.

Table 6.1: Experiment Variables and Options

Experiment Variable	Options
Dataset	MNIST, FashionMNIST, CIFAR-10
α	100, 1, 0.5
Attack	Label Flipping, Model Poisoning, Backdoor
PNR	10%, 30%, 50%, 70%
Topology	Fully-Connected, Ring, Star
Aggregation Algorithm	FedAvg, Krum, Median, Trimmed Mean
MTD	Reactive Topology MTD Only, Reactive Aggregation MTD Only, Reactive Topology & Aggregation MTD, Proactive Topology & Aggregation MTD

Table 6.2: Experimental Setup. R-T, R-A, R-TA, and P-TA represent reactive topology MTD, reactive aggregation MTD, reactive topology and aggregation MTD, and proactive topology and aggregation MTD, respectively.

Experiment	Experiment Setup	Amount
No Attacks	dataset = [MNIST, FashionMNIST, CIFAR-10] α = [100, 1, 0.5] topology = [fully, star, ring] init_aggregation = [FedAvg, Krum, TrimmedMean, Median]	108
Attacks Without Defense	dataset = [MNIST, FashionMNIST, CIFAR-10] α = [0.5] topology = [fully, star, ring] init_aggregation = [FedAvg, Krum, TrimmedMean, Median] attack_type = [label flipping, model poisoning, backdoor] PNR = [10, 30, 50, 70]	432
Attacks With MTD	dataset = [MNIST, FashionMNIST, CIFAR-10] α = [0.5] topology = [fully, star, ring] init_aggregation = [FedAvg] attack_type = [label flipping, model poisoning, backdoor] PNR = [10, 30, 50, 70] MTD = [R-T, R-A, R-TA, P-TA]	432
Attacks With FedProx	dataset = [MNIST, FashionMNIST, CIFAR-10] α = [0.5] topology = [fully] init_aggregation = [FedAvg] attack_type = [label flipping, model poisoning] PNR = [10, 30, 50, 70] MTD = [R-T, N/A]	48

To summarize the experiment settings, for each dataset (MNIST, FMNIST, CIFAR-10),

the initial settings included three data distributions (α values of 100, 1, and 0.5), three network topologies (fully-connected, ring, and star), and four aggregation algorithms (FedAvg, Krum, Median, and Trimmed Mean). Experiments were conducted with varying PNR values of 10%, 30%, 50%, and 70%, with a fixed NR or PSR of 100%. In all poisoning scenarios, five defense configurations were tested: no defense; proactive topology and aggregation MTD defense; reactive topology and aggregation MTD defense; reactive topology MTD defense only; and reactive aggregation MTD defense only. The no-defense serves as the baseline. The configuration options used in this work are listed in Table 6.1. A total of 1020 experiments were conducted, categorized into four groups: no attack, attacks without defense, attacks with MTD, and attacks with FedProx. Detailed experiment setups are provided in Table 6.2.

6.2 Results

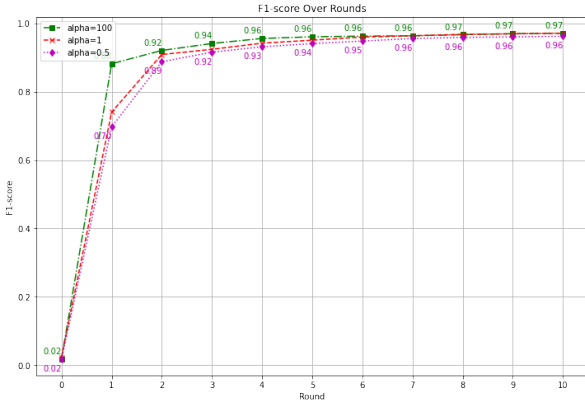
This section evaluates the effectiveness and robustness of the MTD-based defense mechanism under poisoning attacks. First, the impacts of IID and non-IID data on model performance are presented. Next, the effects of different types of poisoning attacks (untargeted label flipping, untargeted model poisoning, and backdoor) are analyzed individually across three datasets (MNIST, FashionMNIST, and CIFAR-10). These results serve as baselines to evaluate the effectiveness of topology MTD and aggregation algorithm MTD in preventing poisoning attacks. Then, the influence of initial topology and initial aggregation algorithm on the FL process is discussed. Finally, FedProx is introduced into the FL system to mitigate client drift caused by non-IID data, and its efficacy is evaluated.

6.2.1 IID and Non-IID

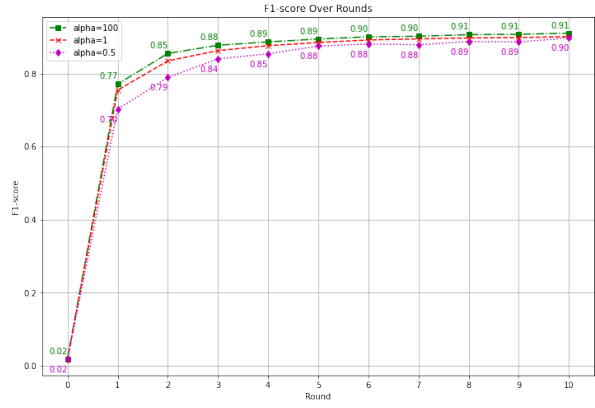
In this study, data were distributed using Dirichlet distributions with alpha values of 100, 1, and 0.5, corresponding to IID, moderate non-IID, and severe non-IID scenarios, respectively. Figure 6.2 shows the model performance, measured by F1-score, under different types of data distribution.

Model performance with IID data ($\alpha = 100$) is superior to that with non-IID data, with performance degrading as the non-IID degree increases. Specifically, performance with $\alpha = 100$ is better than with $\alpha = 1$, and performance with $\alpha = 1$ is better than with $\alpha = 0.5$. This phenomenon becomes more pronounced as the dataset complexity increases. With the MNIST and FashionMNIST datasets, the model demonstrates better performance in the first few rounds with IID data. The performance gap between IID and non-IID data decreases in the subsequent rounds, eventually becoming negligible by the 10th round.

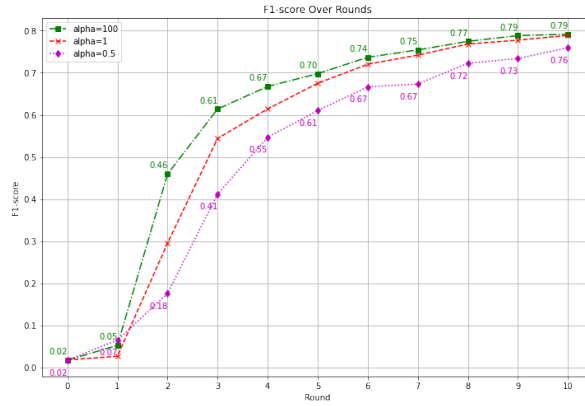
However, with the more complex CIFAR-10 dataset, even after multiple training rounds, the model’s performance with IID data remains significantly better than with non-IID data. This indicates that as the complexity of the dataset increases, the advantages of having IID data become more substantial, highlighting the need to address non-IID data challenges in federated learning.



(a) MNIST F1-score



(b) FashionMNIST F1-score



(c) CIFAR-10 F1-score

Figure 6.2: Model Performance with IID and Non-IID Data

6.2.2 Untargeted Label Flipping

In untargeted label flipping attacks, malicious participants alter the labels of samples to incorrect ones, resulting in a poisoned model that is shared with neighboring nodes afterwards. In these experiments, the PSR remained at 100% across all configurations, with PNRs of 10%, 30%, 50%, and 70%, using a fully connected topology and FedAvg aggregation algorithm as the initial settings. Figure 6.3 shows the model performance, measured by F1-score, under untargeted label flipping attacks at various PNR levels, with no MTD applied serving as the baseline.

For the MNIST dataset, when the PNR is 10%, the performance of the baseline model is comparable to that of models with MTD applied. This suggests that a 10% PNR represents a relatively mild attack, which does not substantially affect the overall federated learning network. However, the application of MTD might cause benign nodes to inadvertently disconnect from each other, which could explain why proactive MTD performs slightly worse than the baseline when PNR is low. When the PNR increases to 30%, the model performance with reactive topology MTD becomes slightly better than the baseline. As the PNR increases further, the advantages of topology MTD become more apparent. At a PNR of 50%, the model with reactive topology MTD achieves an F1-score of 0.95,

while the baseline performance drops to 0.76. The performance gap widens at a PNR of 70%, with the reactive topology MTD achieving an F1-score of approximately 0.92, whereas the baseline performance falls dramatically to 0.24. This disparity is expected, as a higher proportion of malicious models aggregated without security measures leads to a drastic decline in overall model performance.

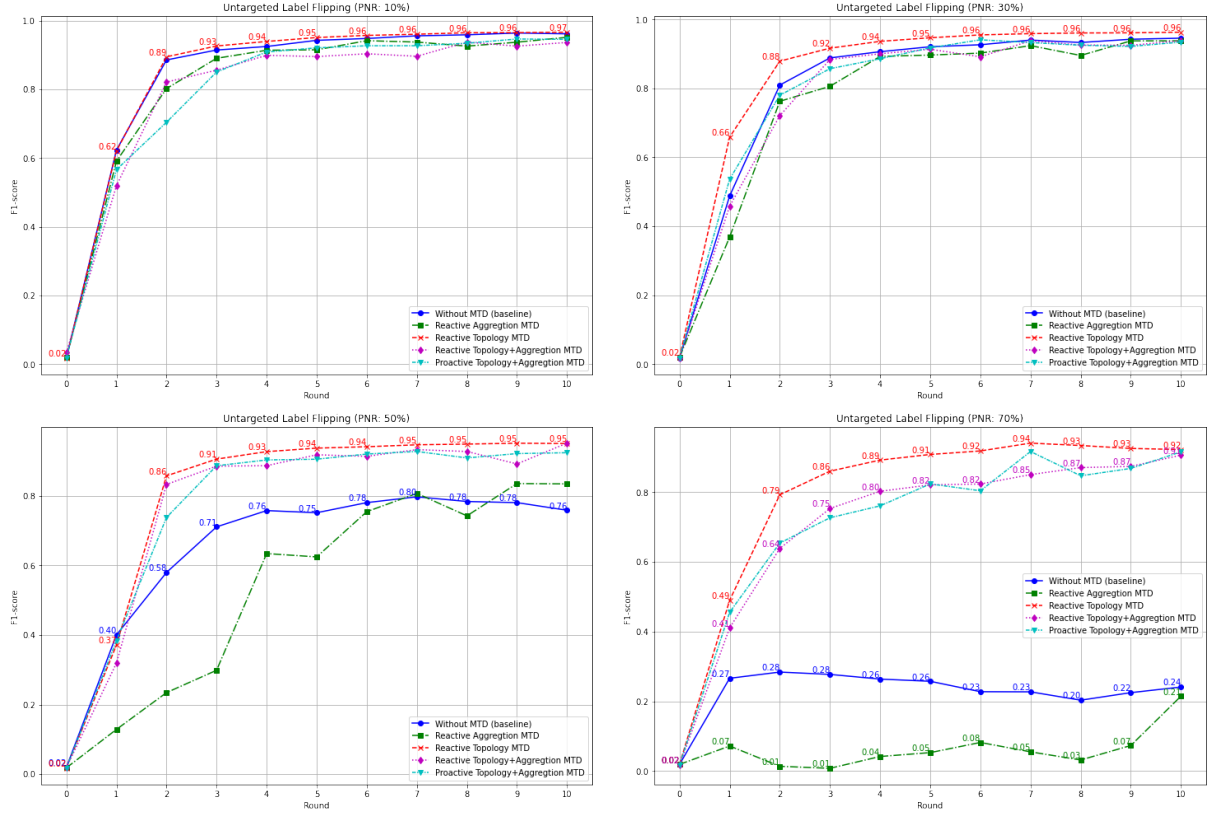


Figure 6.3: Performance under untargeted label flipping for MNIST with varied PNRs

Across all PNR configurations, models using topology MTD consistently outperform the baseline, as well as those using only aggregation algorithm MTD or combinations of topology and aggregation MTD. This indicates that topology MTD is an effective strategy for mitigating label flipping attacks by isolating malicious clients while maintaining connections with trustworthy ones to gather valuable information.

The aggregation algorithm MTD demonstrates effectiveness at lower PNR levels but becomes ineffective, performing similarly to or worse than the baseline at higher PNRs. For the MNIST dataset, the aggregation algorithm MTD is effective at PNRs of 10% and 30%, then performs significantly worse at a PNR of 50%, with a further dramatic decline at higher PNRs. Since different aggregation algorithms are proficient in defending against different types of attacks, further experiments are needed to evaluate their effectiveness in other attacking scenarios. These findings are consistent for the FashionMNIST and CIFAR-10 datasets as well, as shown in Figure 6.4.

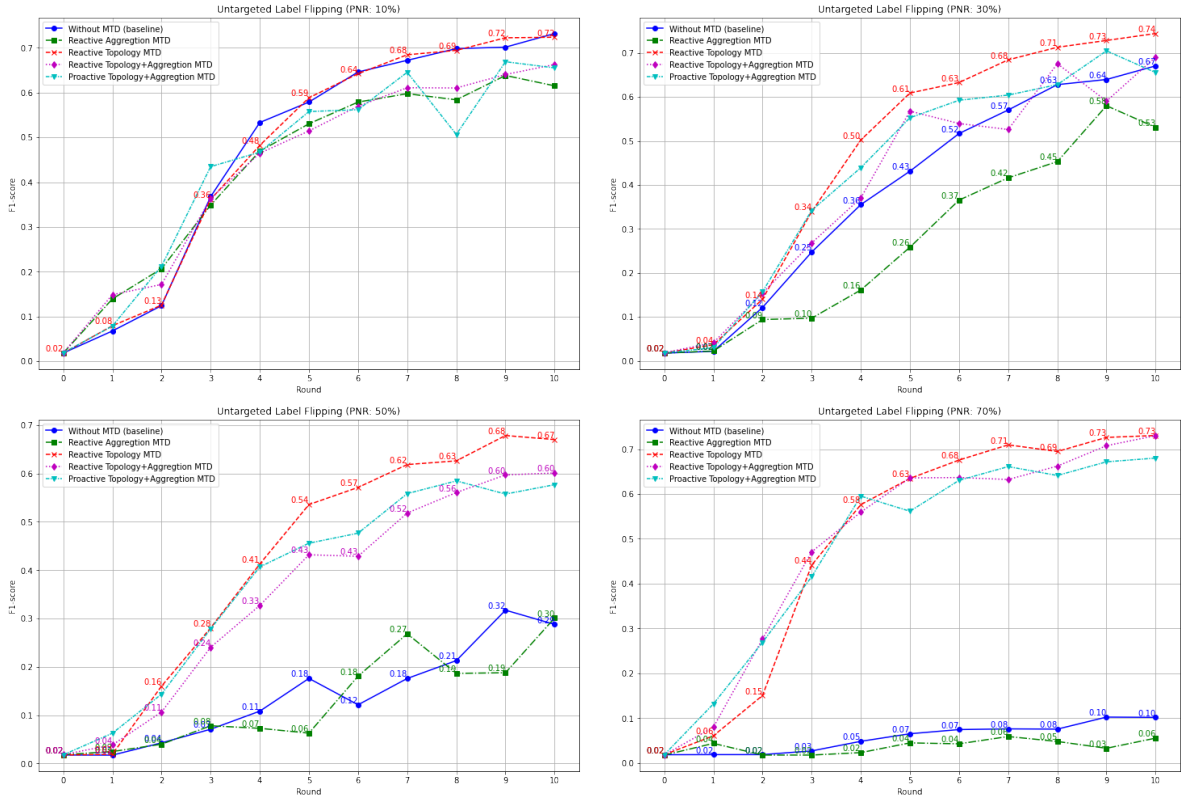
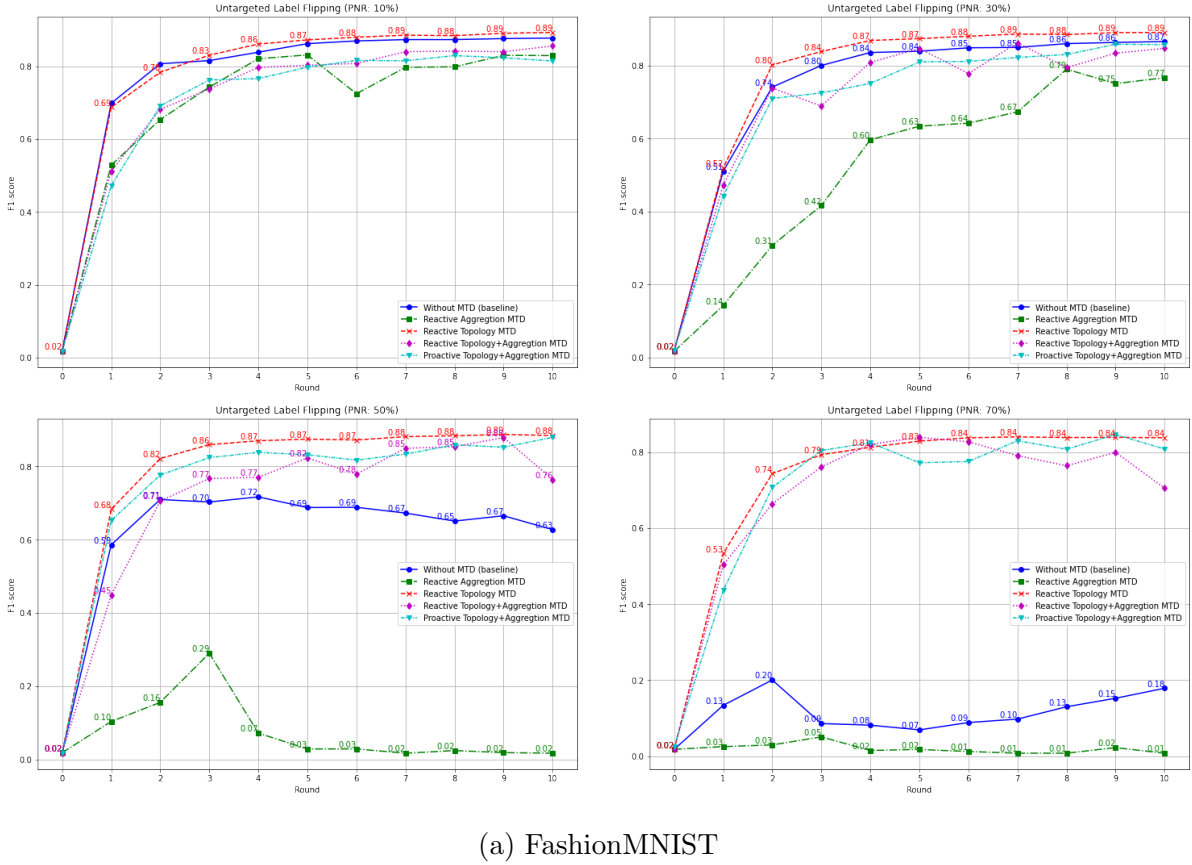


Figure 6.4: Performance under untargeted label flipping for FashionMNIST and CIFAR-10 with varied PNRs

6.2.3 Untargeted Model Poisoning

In untargeted model poisoning attacks, malicious participants introduce noise to their local models, which is then shared with neighboring nodes, adversely affecting the overall FL process. Figure 6.5 illustrates the MNIST model performance under untargeted model poisoning with PNRs of 10%, 30%, 50%, and 70%. Several key observations can be made from this figure:

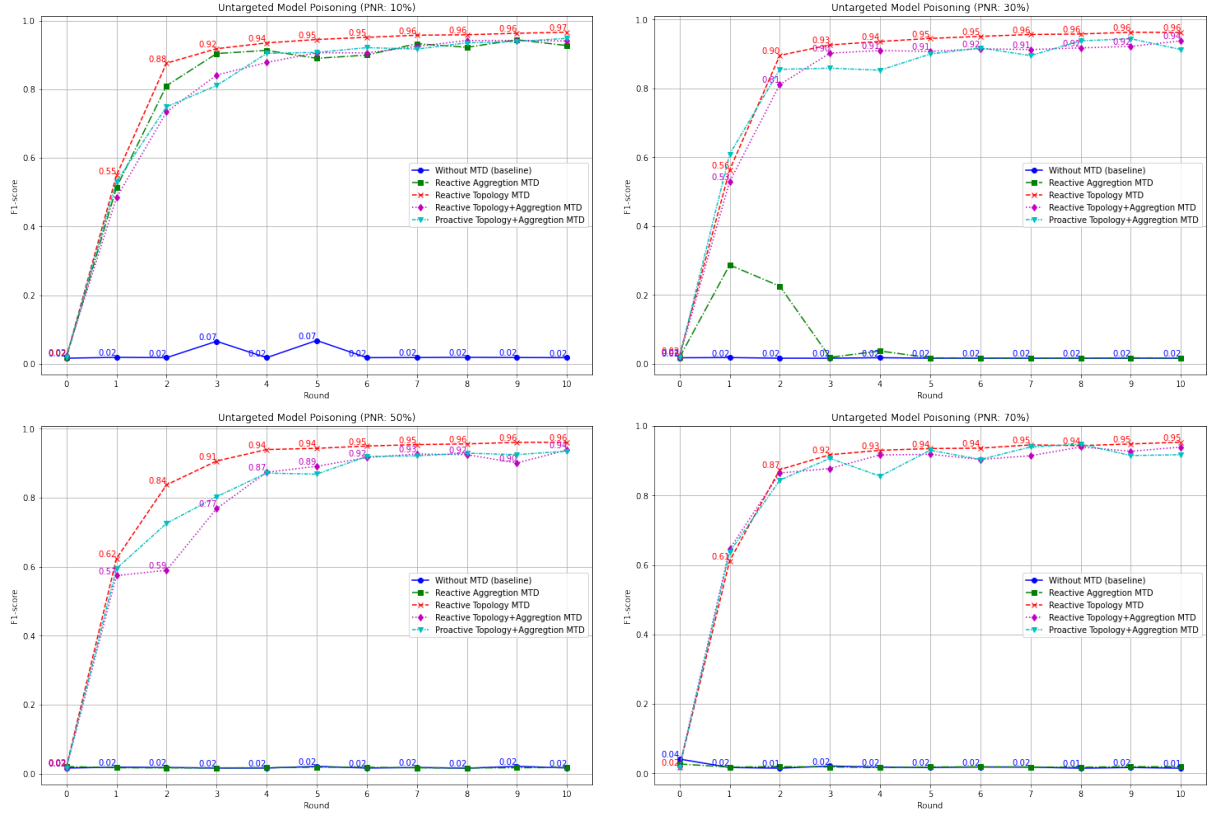


Figure 6.5: Performance under untargeted model poisoning for MNIST with varied PNRs

- FedAvg's Vulnerability to Model Poisoning:** FedAvg exhibits poor resilience to model poisoning. In label flipping attacks, the baseline model is comparable to models employing MTD strategies when the PNR is 10%. However, under model poisoning attacks, the baseline performance deteriorates significantly at a PNR of 10%, with the F1-score dropping to around 0.02. This deterioration occurs because model poisoning substantially distorts the attacked nodes' models, and the averaging process, without any defense measures, exacerbates the impact on the aggregated model. The vulnerability of FedAvg is further supported by the evidence that, at a 10% PNR, the reactive aggregation algorithm MTD performs better than the baseline. In the reactive aggregation algorithm MTD, FedAvg is not included in the aggregation pool, indicating that alternative methods such as Krum, Median, and Trimmed Mean are more robust to model poisoning than FedAvg.

- **Effectiveness of Reactive Topology MTD:** The reactive topology MTD strategy evidently outperforms the baseline and is highly robust, maintaining an F1-score of approximately 0.95 across varying PNRs.
- **Ineffectiveness of Aggregation Algorithm MTD in High PNRs:** Similar to the findings in Section 6.2.2, the aggregation algorithm MTD is ineffective in high PNRs scenarios. Specifically, when the PNR reaches 50%, its performance deteriorates to a level comparable to the baseline.
- **Comparable Performance of Reactive and Proactive MTD Strategies:** The lines representing reactive and proactive topology + aggregation MTD strategies are intertwined, indicating similar performance between these two strategies, though they may differ in resource consumption.

Figure 6.6 and Figure 6.7 further demonstrates the impact of model poisoning on the FashionMNIST and CIFAR-10 datasets. The results for these datasets align with those observed for the MNIST dataset. Overall, the findings indicate: 1) Topology MTD strategy effectively mitigate model poisoning attacks, whereas the aggregation algorithm MTD is less effective. 2) The reactive and proactive topology + aggregation MTD strategies exhibit similar efficacy, suggesting that the choice between these strategies may depend more on resource constraints than performance differences.

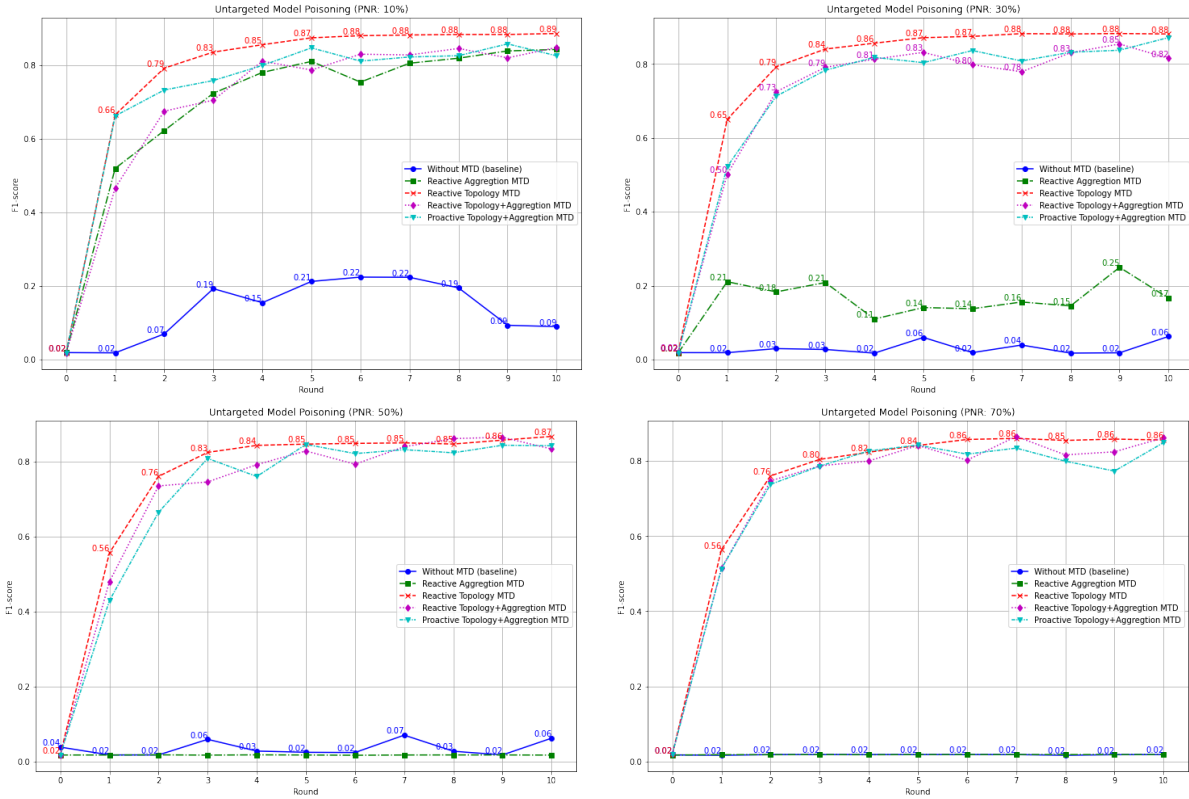


Figure 6.6: Performance under untargeted model poisoning for FashionMNIST with varied PNRs

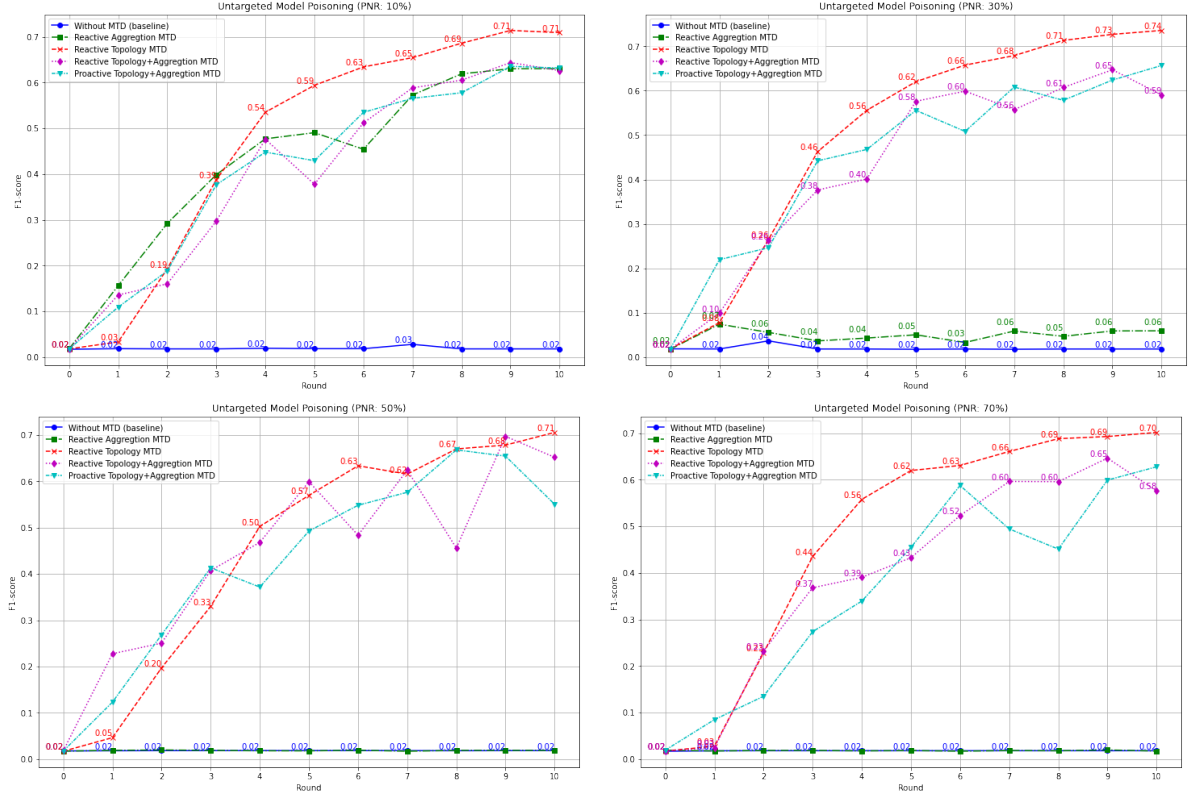


Figure 6.7: Performance under untargeted model poisoning for CIFAR-10 with varied PNRs

6.2.4 Backdoor

Backdoor attack involves embedding a specific trigger into a subset of images, causing the model to classify these images as the targeted label when the trigger is present, regardless of the actual content. When the trigger is absent, the model performs as expected, maintaining its general classification accuracy. This study investigated backdoor attacks by adding a 10×10 pixels 'X' pattern at the top-left corner of images that are labeled with the targeted label. Figure 6.8 illustrates that using a test dataset without any embedded triggers, both the baseline and MTD approaches maintain high F1-scores under backdoor attacks on MNIST, even when PNR is 70%.

The F1-score is a standard metric for measuring a model's general classification capability. However, it is not suitable for evaluating the effectiveness of backdoor attacks. Instead, backdoor attacks are assessed through the Attack Success Rate (ASR). ASR measures the ratio at which the model classifies images containing the trigger as the targeted label. The formula for calculating ASR is as follows:

$$\text{ASR} = \frac{\text{Number of successfully attacked samples}}{\text{Total number of samples}} \quad (6.1)$$

where the total number of samples refers to the number of all adversarial samples generated by the attacker, and the number of successfully attacked samples refers to the number of

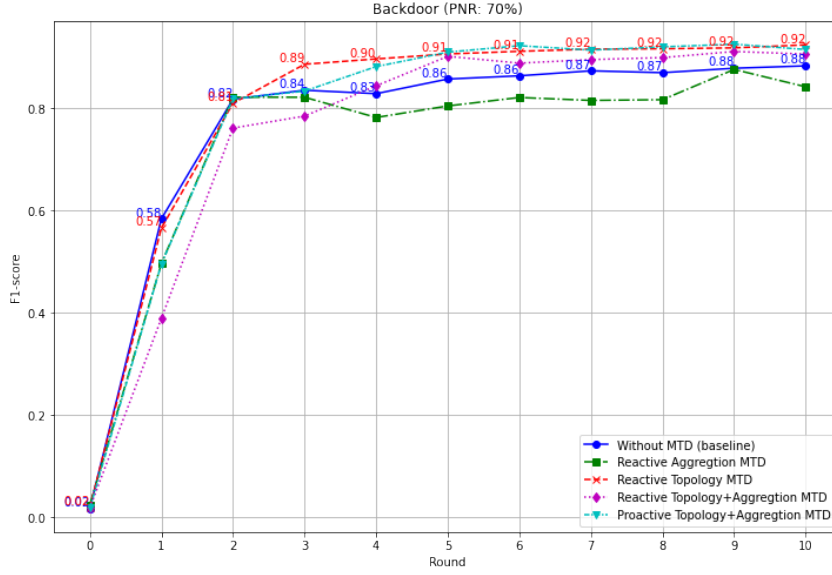


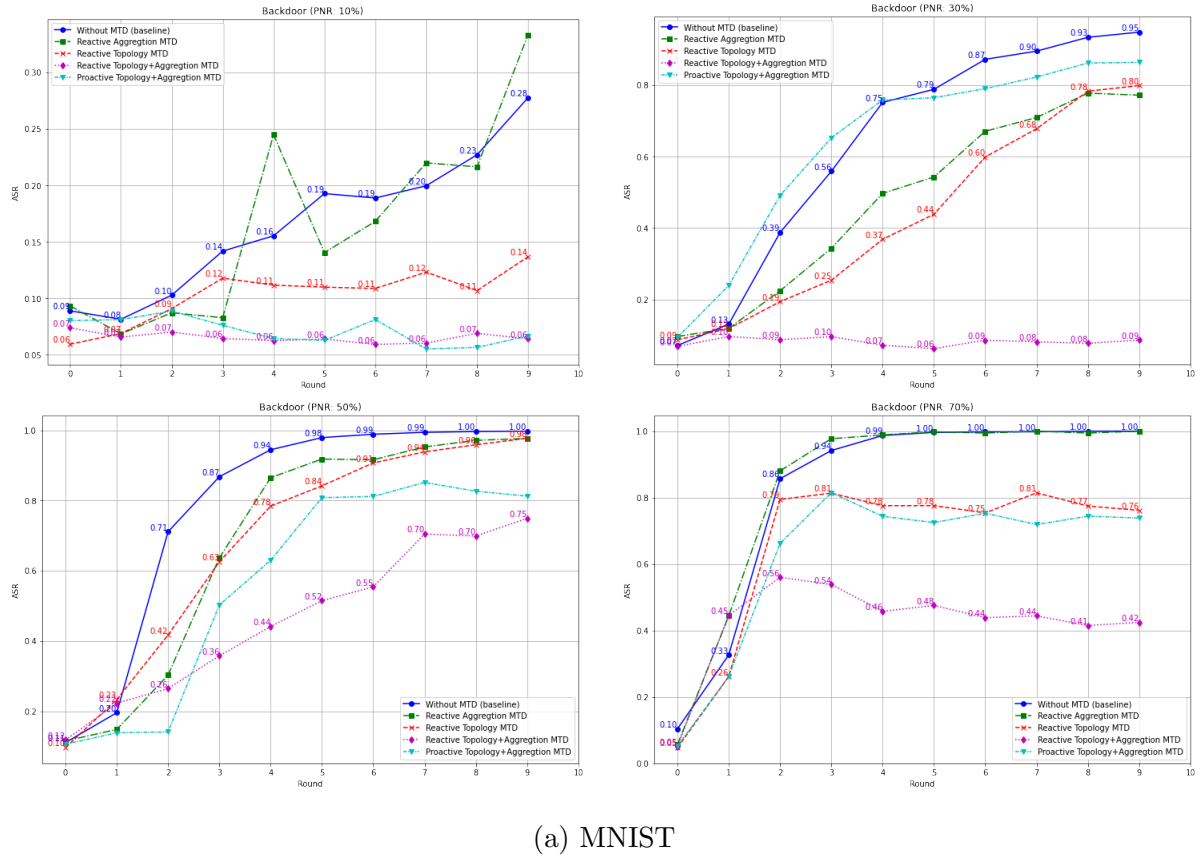
Figure 6.8: Model performance measured by F1-score on MNIST under backdoor attack with 70% PNR using a test dataset without triggers

adversarial samples that the attacker successfully deceived the classifier to produce the targeted label. Note that samples originally labeled as the targeted label and predicted as the targeted label are excluded from both the numerator and the denominator to ensure the ASR accurately reflects the impact of the backdoor attack.

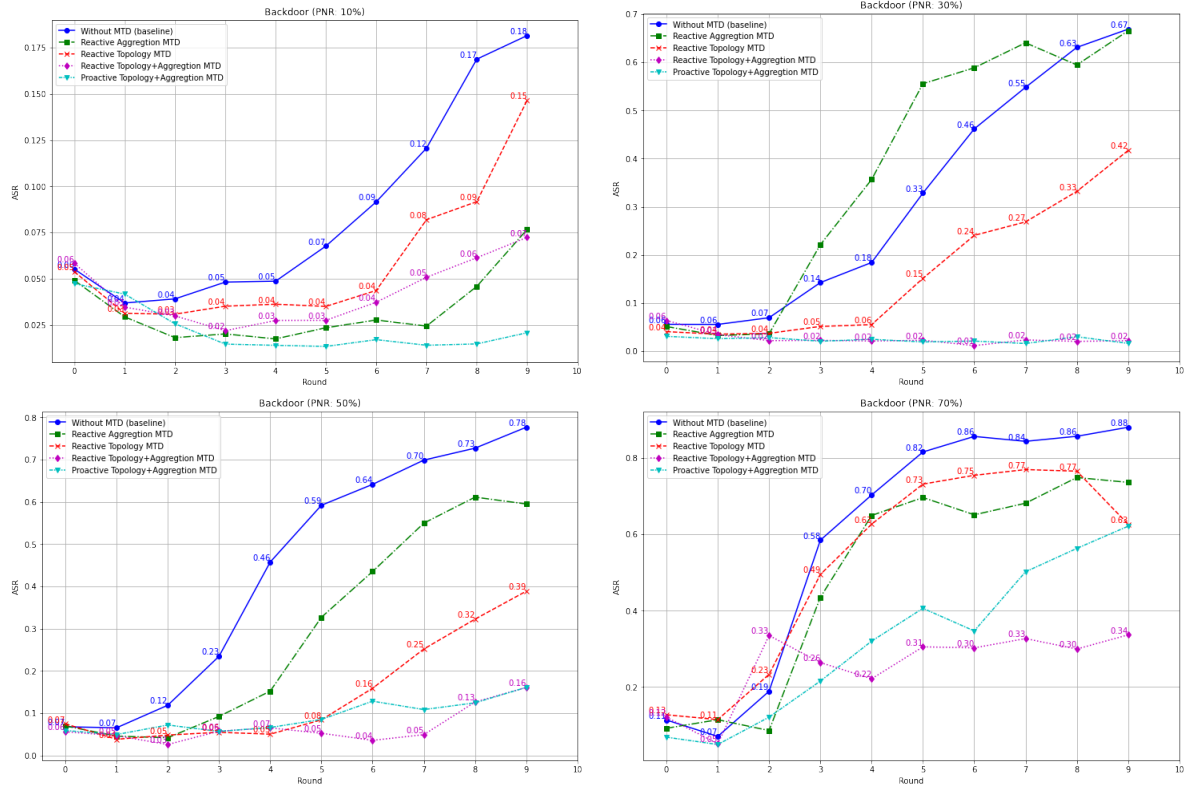
ASR is crucial for evaluating backdoor attacks as it quantifies the model’s susceptibility to the implanted trigger. A lower ASR indicates a more robust defense mechanism. In this study, the target label is set to 3. Figure 6.9 shows the MNIST model performance under backdoor attack. The baseline ASR values are 0.28, 0.95, 1.00, and 1.00 for PNRs of 10%, 30%, 50%, and 70%, respectively. The reactive topology MTD shows some effectiveness against backdoor attacks but not ideal, with ASR values of 0.14, 0.80, 0.98, and 0.76. In contrast, the combination of reactive topology and aggregation MTD achieves better performance, with ASR values of 0.06, 0.09, 0.75, and 0.42. Figures 6.10 shows the backdoor confusion matrix for two experiments on MNIST, each with a PNR of 30%. One experiment uses reactive topology MTD, while the other uses a combination of reactive topology and aggregation MTD. This figure further confirms that the combination of reactive topology and aggregation MTD outperforms reactive topology MTD alone.

While the reactive topology and aggregation MTD show some promise, the MTD strategies overall have limited effectiveness in combating backdoor attacks, especially with high PNRs. This ineffectiveness may stem from the reliance on model similarity and neighbor model loss as reputation scores. Neighbor model loss is calculated on a test dataset without triggers, which makes this indicator invalid for detecting backdoor. Additionally, backdoor attacks are often insidious and may only cause minor adjustments to the model’s parameters, reducing the effectiveness of the model similarity indicator.

Reputation-based indicators remain limited usefulness due to the inherent subtleties of backdoor attacks. Further research is needed to develop systems that can accurately detect backdoor attacks.



(a) MNIST



(b) FashionMNIST

Figure 6.9: Model performance measured by ASR on MNIST and FashionMNIST under backdoor attacks with varied PNRs using a test dataset with triggers

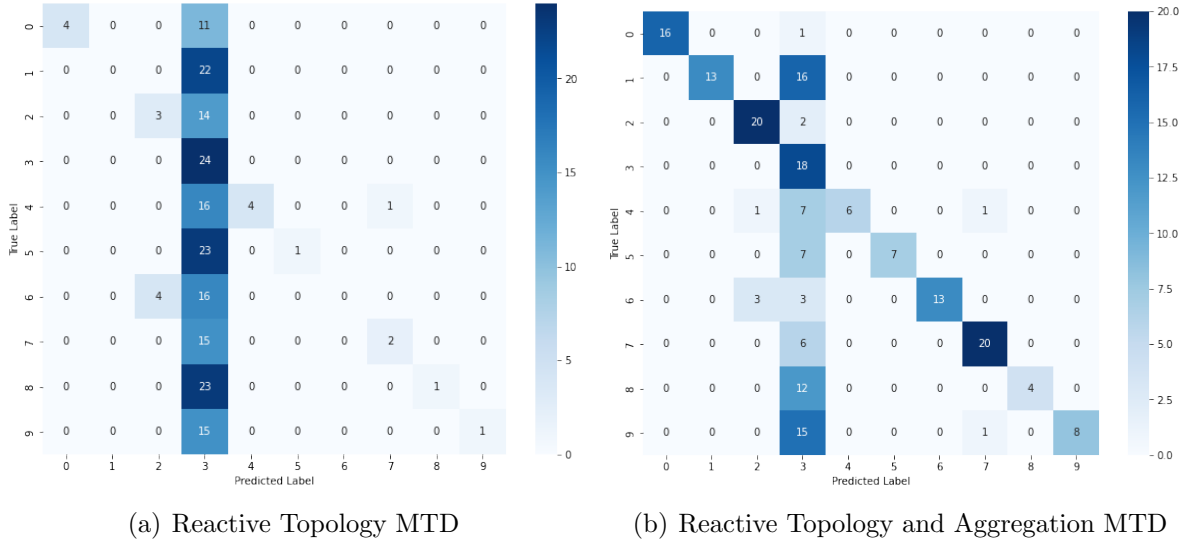


Figure 6.10: Confusion matrix at round 10 for participant node 9 with two MTD mechanisms. Evaluated on the test dataset with triggers under backdoor attacks on MNIST. $\text{PNR} = 30\%$.

6.2.5 Network Topology

This section examines the impact of different network topologies on model performance under three conditions: no attacks, attacks without a defense mechanism, and attacks with an MTD mechanism.

(1) Performance Without Attacks

Figure 6.11 shows the model performance, measured by the F1-score, for different network topologies: fully connected, ring, and star. The performance is evaluated across three datasets: MNIST, FashionMNIST, and CIFAR-10. For the MNIST dataset, the model performances are relatively uniform across the topologies. The fully connected topology achieves an F1-score of 0.96, the star topology scores 0.93, and the ring topology scores 0.94. Similarly, for the FashionMNIST dataset, performance differences among the topologies are small. This suggests that for simpler datasets, the choice of network topology does not significantly impact the model performance. One possible explanation is that information from a few nodes may be sufficient for these models to learn.

As dataset complexity increases, the influence of network topology becomes more apparent. For the CIFAR-10 dataset, the fully connected topology significantly outperforms the others with an F1-score of 0.76. The star topology scores 0.55, and the ring topology performs the worst with an F1-score of 0.20. This discrepancy might be due to the nature of information propagation, where information gradually diminishes with each transmission. In a ring topology, information passes through multiple nodes, and with each hop,

there is a loss of information, resulting in the least effective transmission. In a star topology, all edge nodes are connected to a single central node. This intermediary connection reduces the overall information loss, maintaining lower information attenuation. In contrast, a fully connected topology allows direct communication between all nodes, ensuring maximum information exchange.

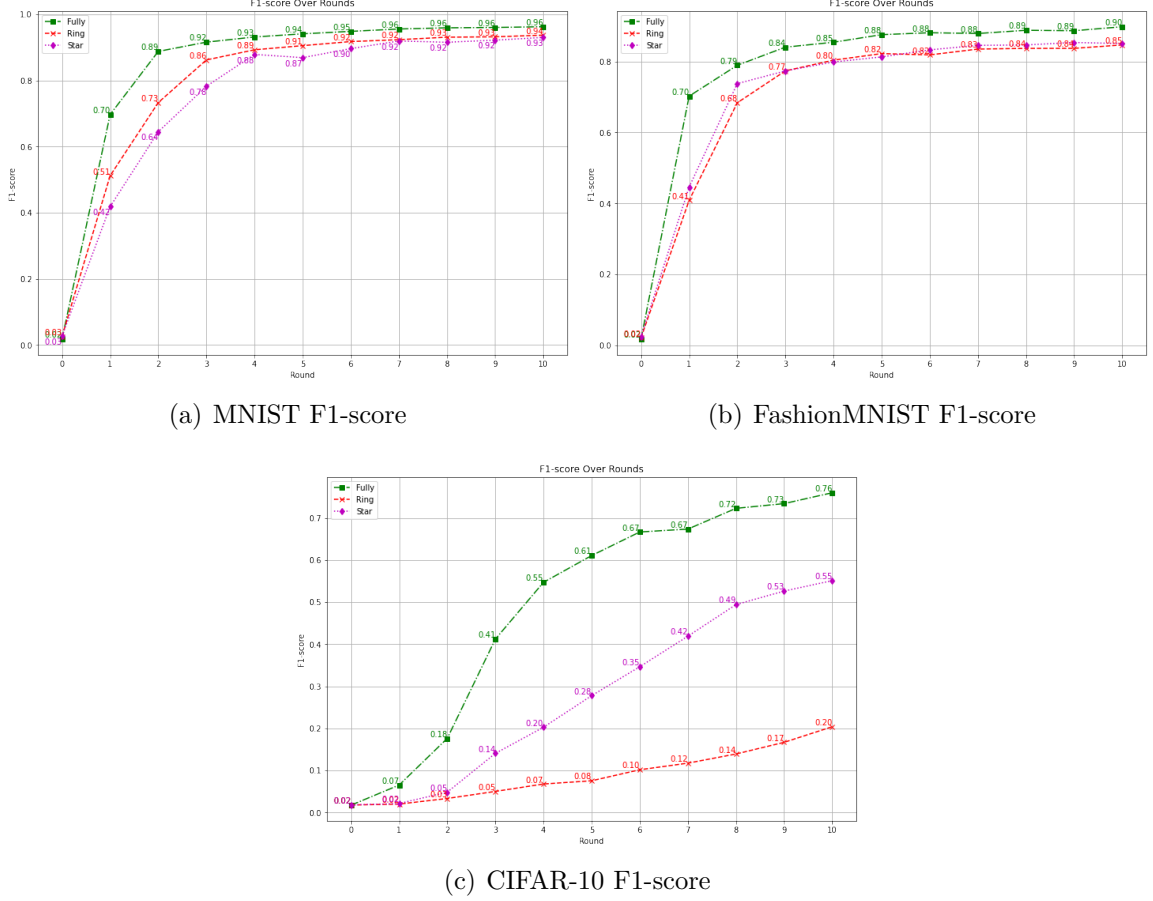


Figure 6.11: Performance with different network topologies in the absence of attacks

(2) Performance Under Attacks Without Defense

Figure 6.12 shows the model performance under label flipping attacks for MNIST with varied PNRs without MTD applied. At 10% and 30% PNRs, the fully connected topology outperforms the ring and star topologies. This might be because, in these scenarios, benign nodes outnumber malicious ones, allowing the fully connected topology to gather more useful information through numerous connections with benign nodes. However, at 50% PNR, the advantage of the fully connected topology diminishes, performing between the ring and star topologies. At 70% PNR, where malicious nodes outnumber benign ones, the fully connected topology aggregates more malicious models, significantly degrading overall performance.

An interesting observation is that at 10% PNR, the star topology achieves an F1-score of 0.92, but at 30% PNR, the score drops sharply to 0.63, then rise again to 0.90 at 50%

PNR, and subsequently decreases to 0.58 at 70% PNR. After examining the logs, it was found that at 30% and 70% PNRs, the central node in the star topology was malicious, directly impacting all nodes due to its central position. This highlights a significant risk related to the star topology: if the central node is attacked, it can adversely affect the entire network. The corresponding star topologies are depicted in Figure 6.13, where red nodes represent malicious nodes and blue nodes represent benign nodes.

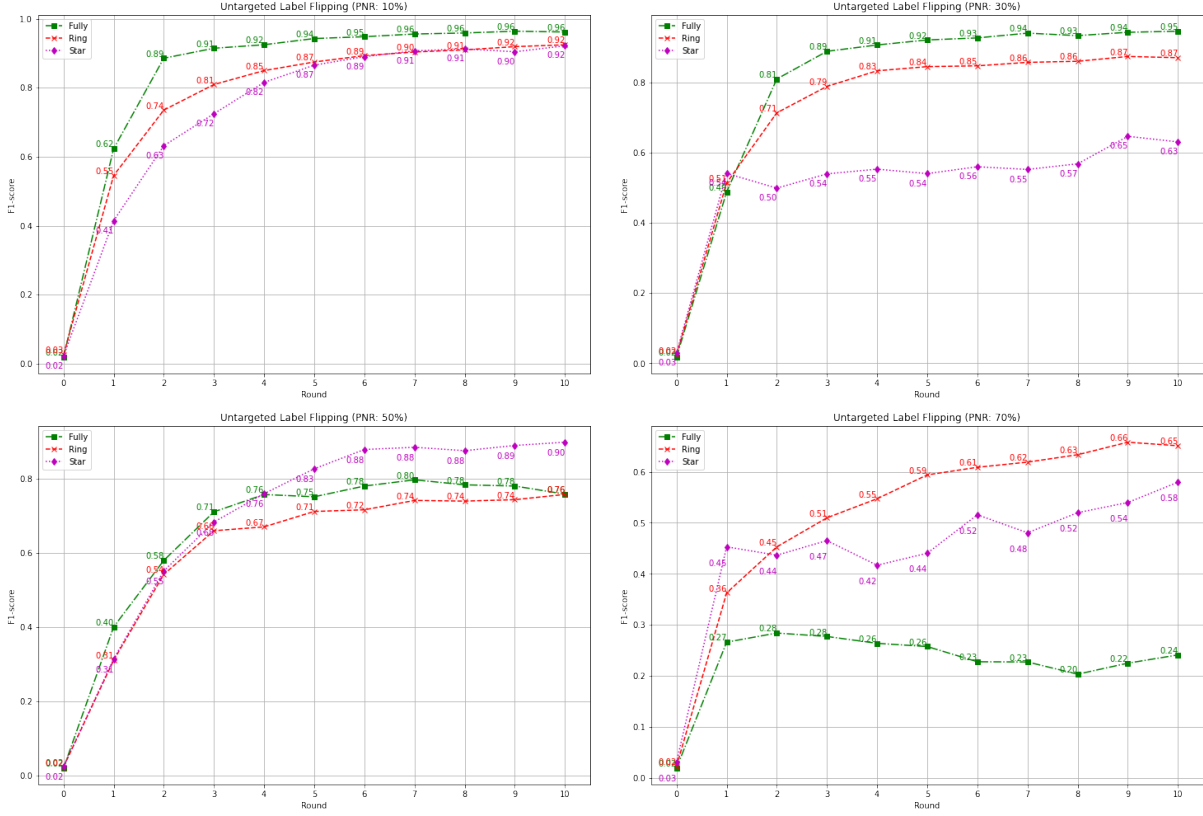


Figure 6.12: Performance with different network topologies under label flipping attacks for MNIST without defense

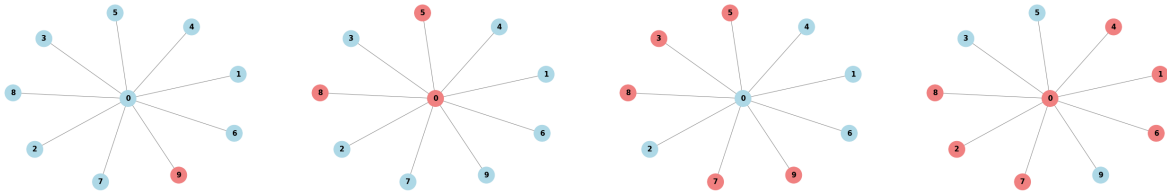
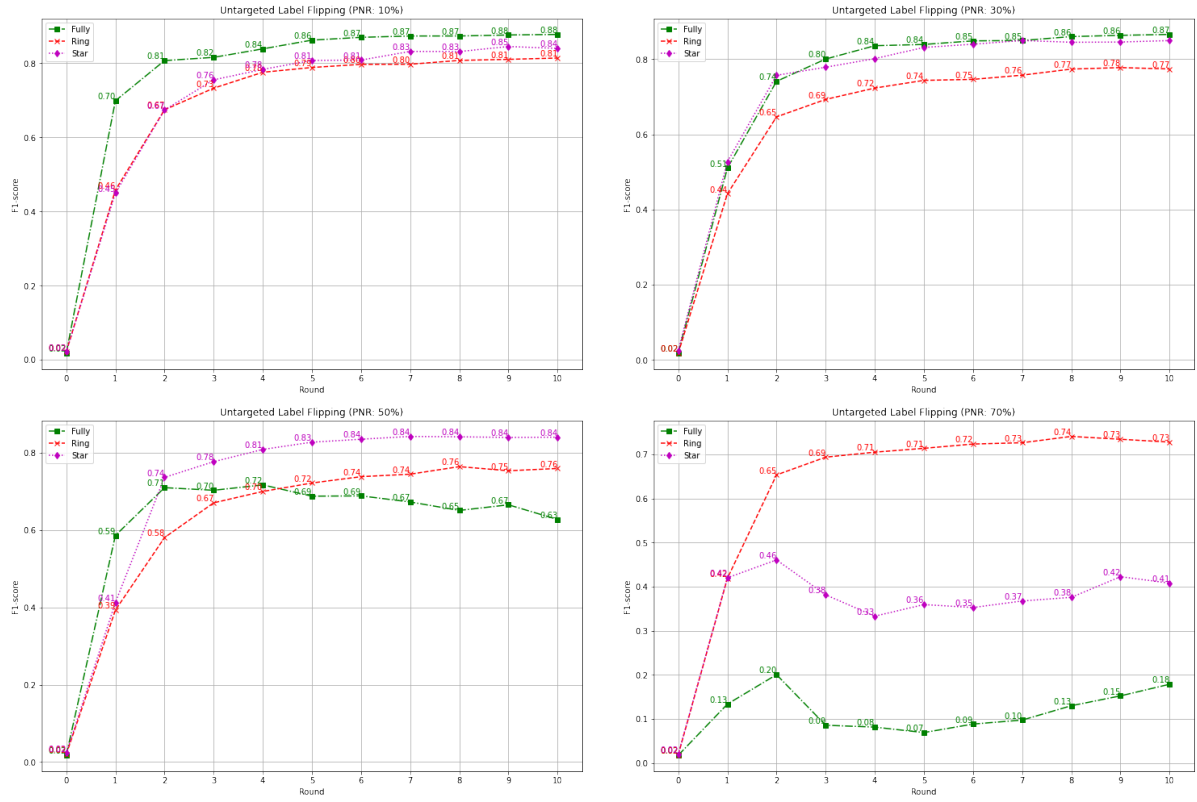
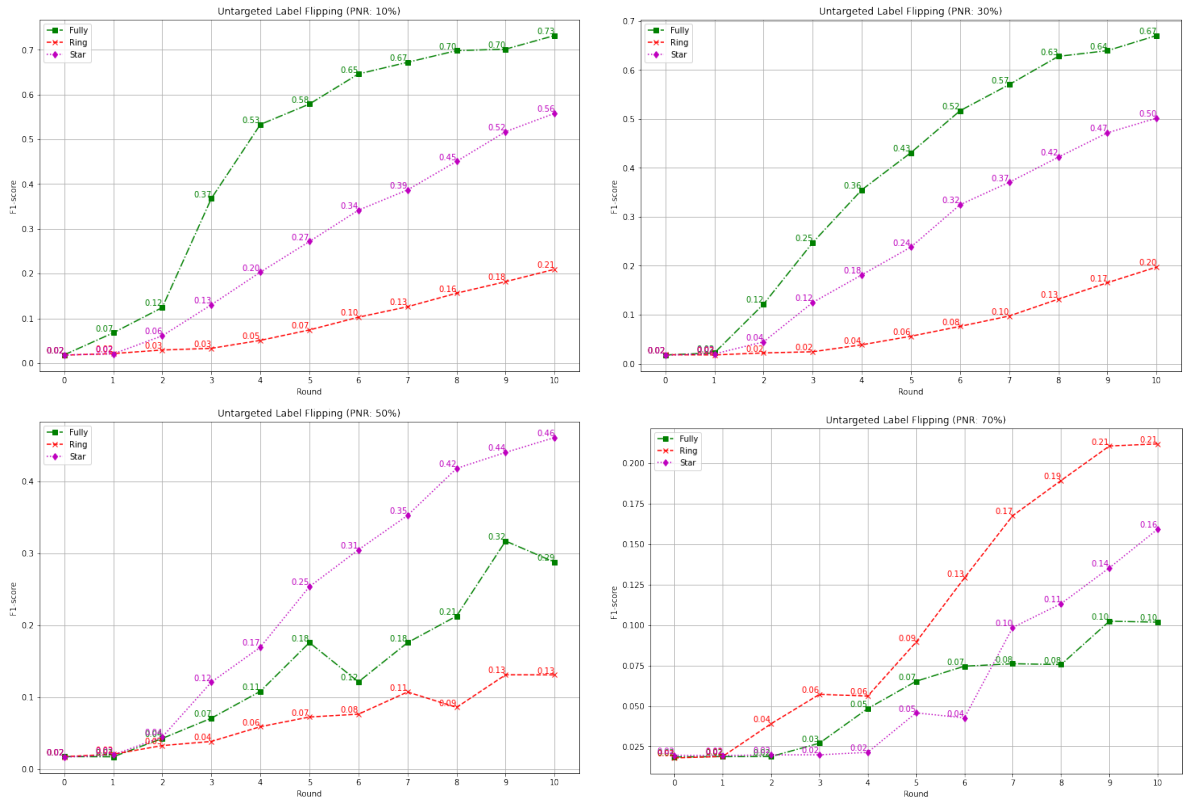


Figure 6.13: Star topology of MNIST under label flipping attacks, where red nodes represent malicious nodes and blue nodes represent benign nodes.

Figure 6.14 shows the model performance on the FashionMNIST and CIFAR-10 datasets. The findings derived from these datasets are consistent with those observed in the MNIST dataset.



(a) FashionMNIST



(b) CIFAR-10

Figure 6.14: Performance with different network topologies under label flipping attacks for FashionMNIST and CIFAR-10 without defense

(3) Performance Under Attacks With MTD

When applying topology MTD, the impact of different initial network topologies becomes negligible. Figure 6.15 shows the model performance under label flipping attacks on MNIST using reactive topology MTD. Models starting with ring topologies perform slightly worse than those with fully connected and star topologies. This could be attributed to information attenuation. After about three rounds, model performances become nearly identical. This is because, with the reputation system and topology MTD, benign nodes eventually connect to each other and isolate malicious nodes.

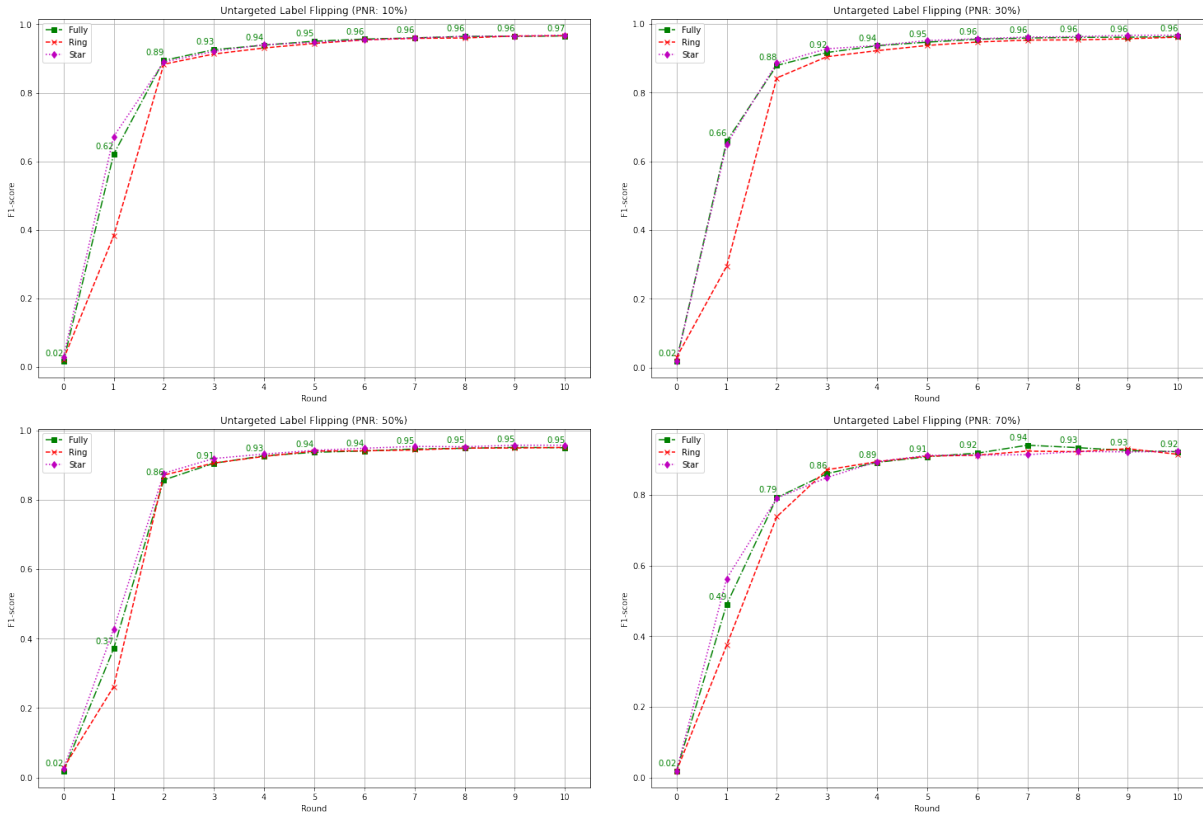


Figure 6.15: Performance with different network topologies under label flipping attacks for MNIST with MTD

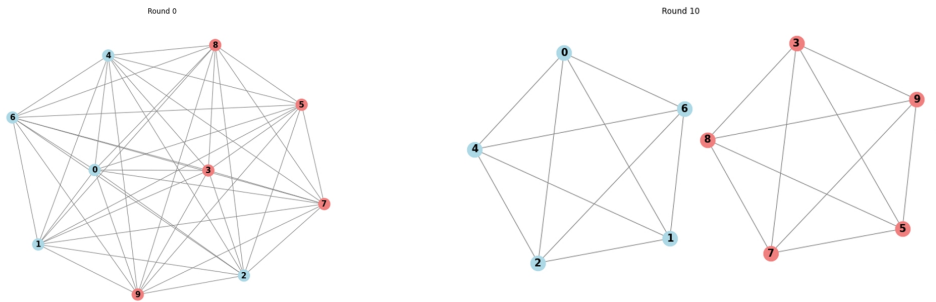


Figure 6.16: Network topology in round 0 and round 10, where red nodes represent malicious nodes and blue nodes represent benign nodes.

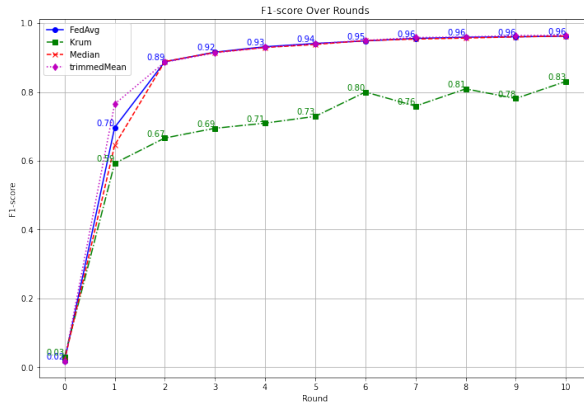
Figure 6.16 illustrates the change in topology over time. In round 0, the graph was a fully connected network with 10 nodes. By round 10, after applying the topology MTD, benign nodes were only fully connected with other benign nodes.

6.2.6 Aggregation Algorithm

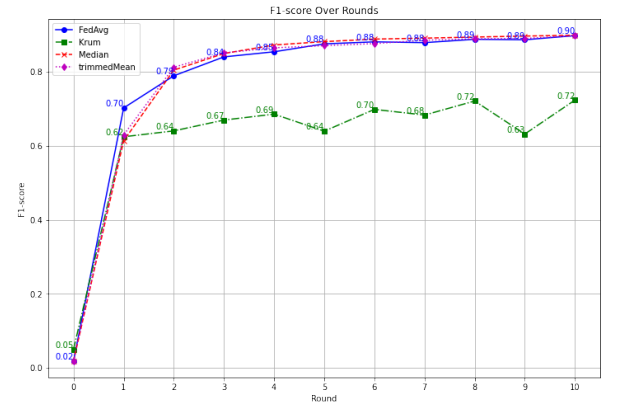
This section investigates the impact of various aggregation algorithms on model performance under two conditions: no attacks and attacks without a defense mechanism.

(1) Performance Without Attacks

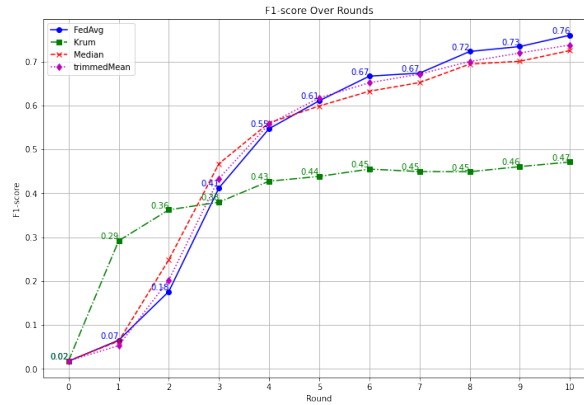
Figure 6.17 shows the model performance with FedAvg, Krum, Median, and Trimmed Mean aggregation algorithms on the MNIST, FashionMNIST, and CIFAR-10 datasets. For all three datasets, FedAvg, Median, and Trimmed Mean perform similarly, while Krum consistently underperforms. This performance disparity increases with dataset complexity.



(a) MNIST F1-score



(b) FashionMNIST F1-score



(c) CIFAR-10 F1-score

Figure 6.17: Performance with different aggregation algorithms in the absence of attacks

For the MNIST dataset, FedAvg, Median, and Trimmed Mean achieve an F1-score of 0.96, whereas Krum only reaches 0.83. The model converges quickly with all algorithms, indicating simplicity of the dataset. On the FashionMNIST dataset, FedAvg, Median, and Trimmed Mean achieve an F1-score of 0.90, while Krum falls further behind with an F1-score of 0.72. The difference between Krum and the other algorithms is more evident, highlighting the impact of increased complexity. This gap widens further with the CIFAR-10 dataset, where FedAvg, Median, and Trimmed Mean achieve F1-scores of 0.76, 0.72, and 0.74 respectively, while Krum drops significantly to 0.47. Additionally, all algorithms converge more slowly on CIFAR-10 due to its higher complexity.

(2) Performance Under Attacks Without Defense

Figure 6.18 shows the model performance under label flipping attacks with PNRs of 10%, 30%, 50%, and 70%. At 10% and 30% PNRs, federated learning exhibits some resilience; however, model performance degrades significantly when the PNR increases to 50%. For the MNIST dataset with a 70% PNR, FedAvg, Median, and Trimmed Mean achieve F1-scores of 0.24, 0.17, and 0.25, respectively, while Krum drops to 0.03.

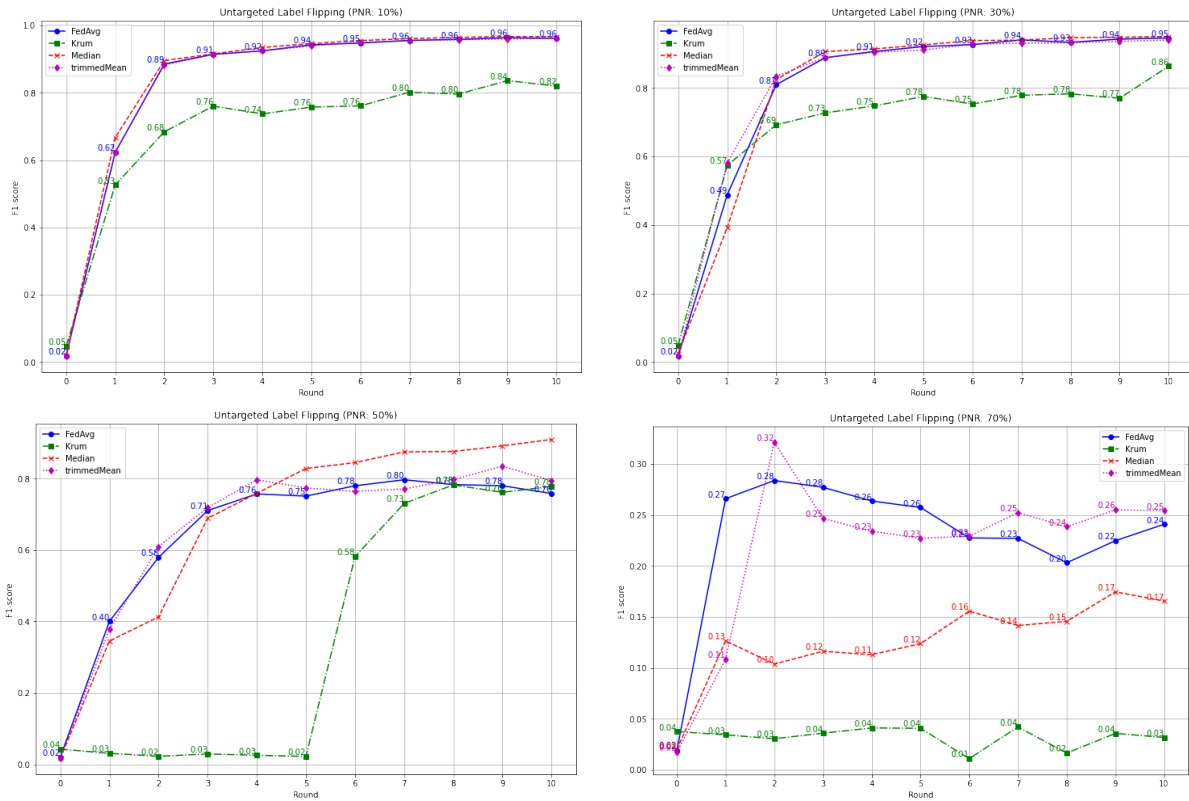


Figure 6.18: Performance with different aggregation algorithms under label flipping attacks for MNIST without defense

Krum's performance is notably unstable; for instance, at 50% PNR on the MNIST dataset, its F1-score remains about 0.03 for the first five rounds, but it unexpectedly increases to 0.58 in the 6th round and then further rises to 0.73 in the 7th round. A more erratic performance is observed with Krum on the FashionMNIST dataset at 70% PNR, as shown

in Figure 6.19. This sporadic performance is likely due to Krum occasionally identifying a correct gradient, but generally struggling to maintain consistency. Krum still performs the worst among the four aggregation algorithms on CIFAR-10 dataset, as shown in Figure 6.20.

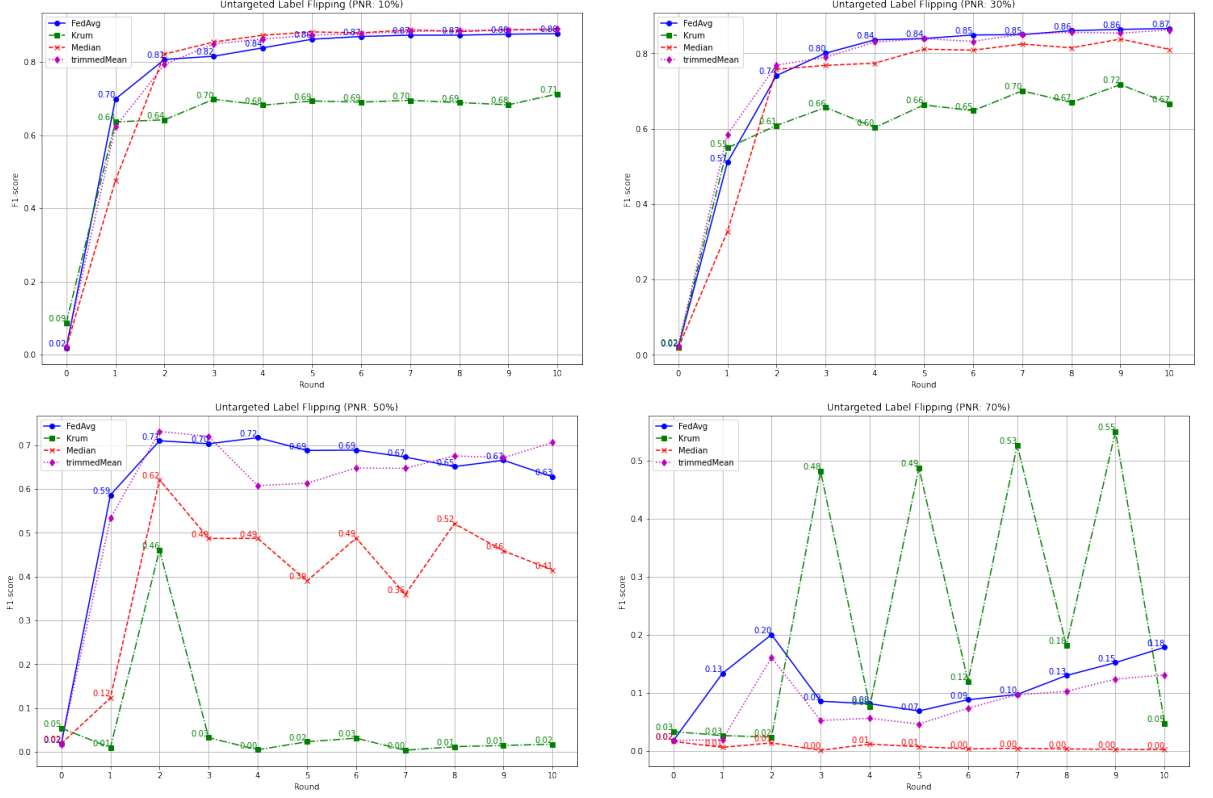


Figure 6.19: Performance with different aggregation algorithms under label flipping attacks for FashionMNIST without defense

Krum’s poor performance can be attributed to several factors:

- (1) **Assumption of Small Gradient Variance Fails:** Krum assumes that correct gradients have a small variance. However, in non-IID settings, the variance of benign gradients increases due to the data heterogeneity across clients. This larger variance makes it difficult for Krum to accurately distinguish between benign and faulty gradients.
- (2) **Assumption of Large Gradient Magnitude Fails:** Krum relies on the assumption that correct gradients have a relatively large magnitude compared to their variance. This assumption becomes less valid as training progresses and gradients approach zero, leading to difficulties in distinguishing correct gradients from faulty ones.

In summary, Krum’s reliance on assumptions about gradient variance and magnitude often fails, making it unsuitable for non-IID data settings. As dataset complexity increases, these limitations become more evident, resulting in a significant decline in model performance compared to other aggregation algorithms like FedAvg, Median, and Trimmed Mean.

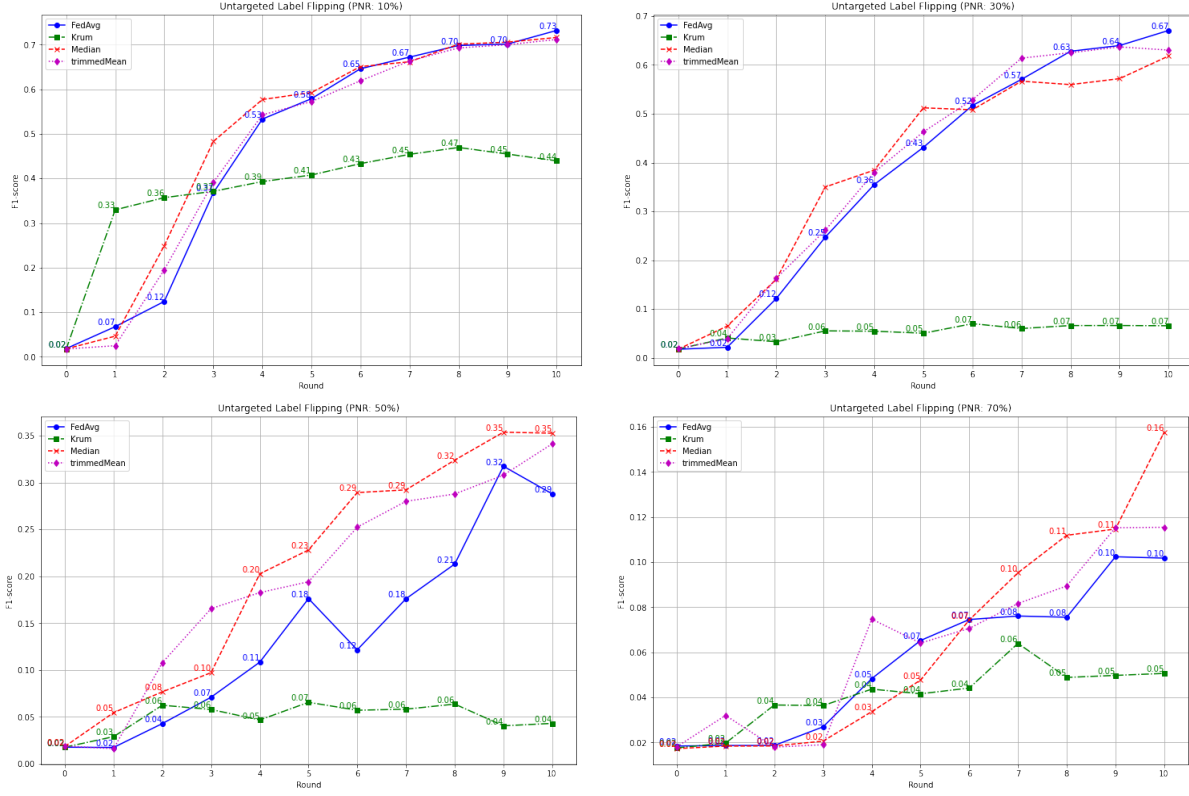


Figure 6.20: Performance with different aggregation algorithms under label flipping attacks for CIFAR-10 without defense

6.2.7 FedProx

FedProx is a modification of FedAvg, designed to mitigate the issues of data and computational resource heterogeneity among participants. This section investigates whether incorporating FedProx into the reactive topology MTD enhances model performance.

Figure 6.21 shows the MNIST model performance with reactive topology MTD and FedProx under label flipping attacks. The results indicate that FedProx does not improve performance. The ineffectiveness may be attributed to the fact that the reactive topology MTD already addresses the challenges associated with non-IID data. Specifically, the reactive topology MTD mechanism employs a reputation system to identify nodes with high model similarity and low neighbor model loss. It then modifies the network topology to cluster benign nodes together while isolating malicious nodes or nodes with significantly different data distributions. As a result, after applying reactive topology MTD, benign nodes predominantly connect with other benign nodes.

In this context, adding FedProx, which introduces a regularization term in the loss function to penalize deviations from the global model parameters, might be counterproductive. Because the local model updates are constrained, leading to less information being learned locally, which could negatively impact overall model performance. Nonetheless, these observations verify the practicality of the proposed topology MTD in non-IID environments.

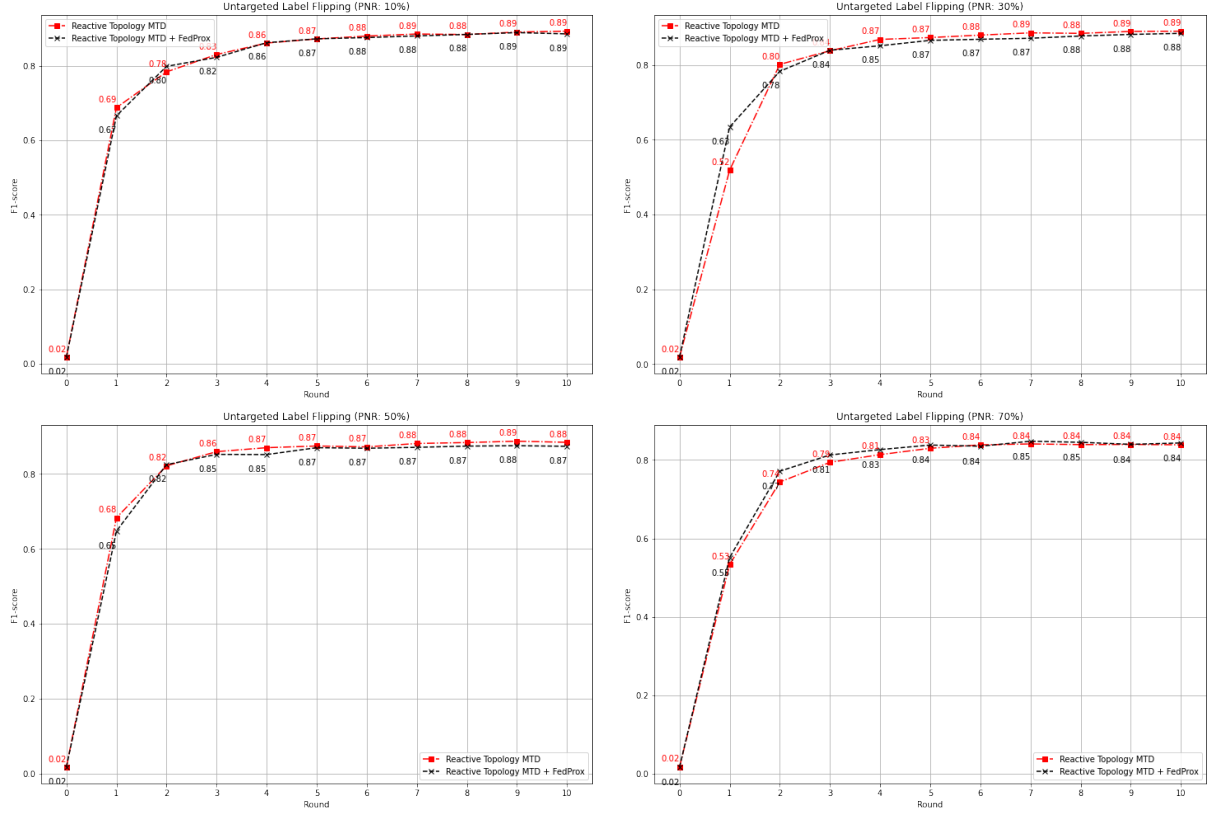


Figure 6.21: MNIST model performance with reactive topology MTD and FedProx under label flipping attacks

6.3 Discussion

The results presented in Section 6.2 highlight two significant challenges in FL. First, model performance deteriorates when working with non-IID data compared to IID data, particularly as the dataset complexity increases. Second, FL without any defensive measures is vulnerable to poisoning attacks, with model performance declining as the PNR rises.

To address these issues, this study proposes a defense mechanism integrating topology and aggregation algorithm MTD. After extensive experiments, here are some key findings:

- (1) The proposed defense approach demonstrates effectiveness in countering untargeted label flipping and model poisoning attacks, with the reactive topology MTD showing notable resilience.
- (2) The reactive and proactive topology combined with aggregation MTD offer comparable performance. Therefore, the selection between these strategies may depend more on resource requirements than performance differences.
- (3) The reactive aggregation algorithm MTD proves to be efficient primarily in low poisoning environments (low PNRs). As the PNR increases up to 50%, its performance tends to align with that of the baseline model.

- (4) The proposed defense mechanism shows limited effectiveness against backdoor attacks due to the inherent subtleties of these attacks and the inadequacy of reputation measuring indicators for detecting backdoor activities.
- (5) The star topology poses a risk if the central node is malicious, as it can significantly impact the entire federated learning network. In contrast, when the central node is benign, the star topology often outperforms the ring topology due to the characteristics of information propagation in FL. In a ring topology, communication between remote nodes must traverse multiple nodes, with information loss occurring at each hop, resulting in less information being obtained. Conversely, in a star topology, each edge node connects directly to a central node, thus reducing information loss with fewer hops.
- (6) Krum is less stable compared to FedAvg, Mean, and Trimmed Mean, as it occasionally identifies a correct gradient, but generally struggles to maintain accuracy. This instability arises because Krum relies on assumptions about small gradient variance, which fails in non-IID environments.
- (7) Integrating FedProx into the reactive topology MTD does not enhance model performance; instead, it degrades performance. The MTD mechanism already addresses non-IID issues effectively. Adding the proximal term to the local objective function constrains local model updates, resulting in less information being learned locally and negatively impacting overall model performance.

Chapter 7

Summary and Conclusions

This chapter provides a comprehensive summary of this study, synthesizes the key findings, and explores potential directions for future research to enhance the security and robustness of FL systems.

7.1 Conclusion

Federated learning is vulnerable to poisoning attacks, necessitating the adoption of robust defense measures. Existing defense mechanisms for FL often assume IID data among participants. However, non-IID data distributions are more common and introduce additional challenges, as data heterogeneity among participating nodes complicates the detection of malicious nodes. Therefore, the primary goal of this study is to establish defense mechanisms against poisoning attacks in FL under non-IID conditions. To achieve this goal, the study was structured into several key phases.

At the beginning, this work introduced the essential background knowledge on FL, poisoning attacks, the characteristics of non-IID data, and the novel defense approach known as Moving Target Defense. Following this, the study presented a comprehensive literature review on non-IID data handling techniques and MTD applications across various domains. Insights from this literature review led to the development of a defense mechanism architecture comprising three key components: a dual reputation system, topology MTD, and aggregation algorithm MTD.

In the implementation phase, these three key components were developed and deployed. The reputation system was built with model similarity measured by Euclidean distance and neighbor model loss evaluation. After calculating the reputation scores, the DBSCAN algorithm was employed to set dynamic thresholds, which are more suited for non-IID data compared to static thresholds. Besides, a set of MTD mechanisms were implemented, including reactive topology MTD, reactive aggregation algorithm MTD, reactive topology and aggregation MTD, and proactive topology and aggregation MTD.

A total of 1020 experiments were conducted to evaluate the effectiveness of the proposed MTD mechanisms under untargeted label flipping, untargeted model poisoning, and backdoor attacks with PNRs of 10%, 30%, 50%, and 70%. The results demonstrated that:

- The reactive topology MTD was highly robust against label flipping and model poisoning attacks but had limited effectiveness against backdoor attacks.
- The reactive topology and aggregation MTD showed resilience to backdoor attacks under low PNRs.
- Without defense mechanisms, a fully connected topology performed better than ring and star topologies under attacks with low PNRs. However, its performance dropped significantly at high PNRs, becoming worse than ring and star topologies.
- The star topology exhibited unstable performance, highly dependent on whether the central node was malicious or benign.
- The Krum algorithm showed instability compared to FedAvg, Mean, and Trimmed Mean, primarily due to its assumption of small gradient variance, which fails in non-IID environments.
- Integrating FedProx was found to degrade model performance, indirectly validating the practicality of the proposed defense mechanisms, which already thoroughly considered non-IID issues.

This study contributes to the security of FL by developing a set of MTD defense strategies that are effective in non-IID environments, supported by extensive experimental validation. It also evaluates the impact of different network topologies and aggregation algorithms on model performance, providing empirical support for related research.

7.2 Limitations and Future Work

This study proposed a set of MTD defense mechanisms and demonstrated their effectiveness against label flipping and model poisoning attacks. However, several areas present opportunities for further research and improvements.

Firstly, although this study addressed label flipping and model poisoning attacks, backdoor attacks remain an area requiring more investigation. Future research could focus on enhancing the reputation system or developing additional auxiliary systems to better detect backdoor attacks.

Additionally, the evaluation of the defense mechanisms was limited to only three types of attacks. Expanding this evaluation to include a broader spectrum of attack scenarios would provide a more comprehensive assessment of the robustness of these defenses.

While the defense mechanisms were tested in non-IID environments, the study ensured that each node had at least 10 samples per class to avoid extreme cases. Future research

could examine the robustness of these mechanisms under more challenging conditions, where nodes might lack samples from certain classes.

Addressing these areas will advance the development of more robust and versatile defense mechanisms capable of handling a wider range of adversarial threats, particularly in non-IID environments.

Bibliography

- [1] P. Regulation, “Regulation (eu) 2016/679 of the european parliament and of the council”, *Regulation (eu)*, vol. 679, p. 2016, 2016.
- [2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data”, in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.
- [3] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning”, *Knowledge-Based Systems*, vol. 216, p. 106 775, 2021.
- [4] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, 2023.
- [5] A. K. Singh, A. Blanco-Justicia, and J. Domingo-Ferrer, “Fair detection of poisoning attacks in federated learning on non-iid data”, *Data Mining and Knowledge Discovery*, vol. 37, no. 5, pp. 1998–2023, 2023.
- [6] R. Zhuang, S. A. DeLoach, and X. Ou, “Towards a theory of moving target defense”, in *Proceedings of the first ACM workshop on moving target defense*, 2014, pp. 31–40.
- [7] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, “A review of applications in federated learning”, *Computers & Industrial Engineering*, vol. 149, p. 106 854, 2020.
- [8] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, “Federated learning for the internet of things: Applications, challenges, and opportunities”, *IEEE Internet of Things Magazine*, vol. 5, no. 1, pp. 24–29, 2022.
- [9] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, “Inverting gradients-how easy is it to break privacy in federated learning?”, *Advances in neural information processing systems*, vol. 33, pp. 16 937–16 947, 2020.
- [10] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, “Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients”, in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2545–2555.
- [11] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, “Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning”, in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 1354–1371.

- [12] M. Fang, X. Cao, J. Jia, and N. Gong, “Local model poisoning attacks to {byzantine-robust} federated learning”, in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 1605–1622.
- [13] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, “Analyzing federated learning through an adversarial lens”, in *International conference on machine learning*, PMLR, 2019, pp. 634–643.
- [14] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning”, in *International conference on artificial intelligence and statistics*, PMLR, 2020, pp. 2938–2948.
- [15] C. Xie, K. Huang, P.-Y. Chen, and B. Li, “Dba: Distributed backdoor attacks against federated learning”, in *International conference on learning representations*, 2019.
- [16] M. S. Jere, T. Farnan, and F. Koushanfar, “A taxonomy of attacks on federated learning”, *IEEE Security & Privacy*, vol. 19, no. 2, pp. 20–28, 2020.
- [17] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara, “Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges”, *Information Fusion*, vol. 90, pp. 148–173, 2023.
- [18] M. Ye, X. Fang, B. Du, P. C. Yuen, and D. Tao, “Heterogeneous federated learning: State-of-the-art and research challenges”, *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–44, 2023.
- [19] X. Ma, J. Zhu, Z. Lin, S. Chen, and Y. Qin, “A state-of-the-art survey on solving non-iid data in federated learning”, *Future Generation Computer Systems*, vol. 135, pp. 244–258, 2022.
- [20] H. Zhu, J. Xu, S. Liu, and Y. Jin, “Federated learning on non-iid data: A survey”, *Neurocomputing*, vol. 465, pp. 371–390, 2021.
- [21] Q. Li, Y. Diao, Q. Chen, and B. He, “Federated learning on non-iid data silos: An experimental study”, in *2022 IEEE 38th international conference on data engineering (ICDE)*, IEEE, 2022, pp. 965–978.
- [22] J. Zheng and A. S. Namin, “A survey on the moving target defense strategies: An architectural perspective”, *Journal of Computer Science and Technology*, vol. 34, pp. 207–233, 2019.
- [23] G.-l. Cai, B.-s. Wang, W. Hu, and T.-z. Wang, “Moving target defense: State of the art and characteristics”, *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 11, pp. 1122–1153, 2016.
- [24] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, “A survey of moving target defenses for network security”, *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.
- [25] B. C. Ward, S. R. Gomez, R. Skowrya, *et al.*, “Survey of cyber moving targets second edition”, *MIT Lincoln Laboratory Lexington United States, Tech. Rep.*, 2018.
- [26] A. Nguyen-Tuong, D. Evans, J. C. Knight, B. Cox, and J. W. Davidson, “Security through redundant data diversity”, in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, IEEE, 2008, pp. 187–196.

- [27] M. Taguinod, A. Doupé, Z. Zhao, and G.-J. Ahn, “Toward a moving target defense for web applications”, in *2015 IEEE international conference on information reuse and integration*, IEEE, 2015, pp. 510–517.
- [28] G. S. Kc, A. D. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization”, in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 272–280.
- [29] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, “Moving target defense techniques: A survey”, *Security and Communication Networks*, vol. 2018, no. 1, p. 3759626, 2018.
- [30] H. Alavizadeh, D. S. Kim, and J. Jang-Jaccard, “Model-based evaluation of combinations of shuffle and diversity mtd techniques on the cloud”, *Future Generation Computer Systems*, vol. 111, pp. 507–522, 2020.
- [31] Y.-B. Luo, B.-S. Wang, X.-F. Wang, X.-F. Hu, G.-L. Cai, and H. Sun, “Rpah: Random port and address hopping for thwarting internal and external adversaries”, in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 1, 2015, pp. 263–270.
- [32] E. T. Martínez Beltrán, P. M. Sánchez Sánchez, S. López Bernal, *et al.*, “Mitigating communications threats in decentralized federated learning through moving target defense”, *Wireless Networks*, pp. 1–15, 2024.
- [33] J.-H. Cho, D. P. Sharma, H. Alavizadeh, *et al.*, “Toward proactive, adaptive defense: A survey on moving target defense”, *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.
- [34] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari, “Architecture-based self-protecting software systems”, in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, 2013, pp. 33–42.
- [35] T. E. Carroll, M. Crouse, E. W. Fulp, and K. S. Berenhaut, “Analysis of network address shuffling as a moving target defense”, in *2014 IEEE international conference on communications (ICC)*, IEEE, 2014, pp. 701–706.
- [36] S. Venkatesan, M. Albanese, G. Cybenko, and S. Jajodia, “A moving target defense approach to disrupting stealthy botnets”, in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, 2016, pp. 37–46.
- [37] C. Feng, A. H. Celdran, M. Vuong, G. Bovet, and B. Stiller, “Voyager: Mtd-based aggregation protocol for mitigating poisoning attacks on dfl”, *arXiv preprint arXiv:2310.08739*, 2023.
- [38] R. Zhuang, S. Zhang, S. A. DeLoach, X. Ou, A. Singhal, *et al.*, “Simulation-based approaches to studying effectiveness of moving-target network defense”, in *National symposium on moving target research*, Citeseer, vol. 246, 2012.
- [39] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data”, *arXiv preprint arXiv:1806.00582*, 2018.
- [40] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, “Hybrid-fl for wireless networks: Cooperative learning mechanism using non-iid data”, in *ICC 2020-2020 IEEE International Conference On Communications (ICC)*, IEEE, 2020, pp. 1–7.

- [41] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, “Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data”, *arXiv preprint arXiv:1811.11479*, 2018.
- [42] Z. Wang, S. Duan, C. Wu, *et al.*, “Generative data augmentation for non-iid problem in decentralized clinical machine learning”, in *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, IEEE, 2022, pp. 336–343.
- [43] H. Wang, L. Muñoz-González, D. Eklund, and S. Raza, “Non-iid data re-balancing at iot edge with peer-to-peer federated learning for anomaly detection”, in *Proceedings of the 14th ACM conference on security and privacy in wireless and mobile networks*, 2021, pp. 153–163.
- [44] Z. Wang, Y. Zhu, D. Wang, and Z. Han, “Fedacs: Federated skewness analytics in heterogeneous decentralized data environments”, in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, IEEE, 2021, pp. 1–10.
- [45] H. Wang, Z. Kaplan, D. Niu, and B. Li, “Optimizing federated learning on non-iid data with reinforcement learning”, in *IEEE INFOCOM 2020-IEEE conference on computer communications*, IEEE, 2020, pp. 1698–1707.
- [46] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, “Federated multi-task learning”, *Advances in neural information processing systems*, vol. 30, 2017.
- [47] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, “Federated learning with personalization layers”, *arXiv preprint arXiv:1912.00818*, 2019.
- [48] Z. Li, J. Lu, S. Luo, *et al.*, “Towards effective clustered federated learning: A peer-to-peer framework with adaptive neighbor matching”, *IEEE Transactions on Big Data*, 2022.
- [49] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, “Tackling the objective inconsistency problem in heterogeneous federated optimization”, *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.
- [50] D. M. S. Bhatti and H. Nam, “A robust aggregation approach for heterogeneous federated learning”, in *2023 Fourteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2023, pp. 300–304.
- [51] S. Reddi, Z. Charles, M. Zaheer, *et al.*, “Adaptive federated optimization”, *arXiv preprint arXiv:2003.00295*, 2020.
- [52] Y. Yeganeh, A. Farshad, N. Navab, and S. Albarqouni, “Inverse distance aggregation for federated learning with non-iid data”, in *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning: Second MICCAI Workshop, DART 2020, and First MICCAI Workshop, DCL 2020, Held in Conjunction with MICCAI 2020, Lima, Peru, October 4–8, 2020, Proceedings 2*, Springer, 2020, pp. 150–159.
- [53] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, “Scaffold: Stochastic controlled averaging for federated learning”, in *International conference on machine learning*, PMLR, 2020, pp. 5132–5143.
- [54] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks”, *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

- [55] C. Feng, A. H. Celdran, P. M. S. Sanchez, *et al.*, “Cyberforce: A federated reinforcement learning framework for malware mitigation”, *arXiv preprint arXiv:2308.05978*, 2023.
- [56] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise”, in *kdd*, vol. 96, 1996, pp. 226–231.
- [57] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent”, *Advances in neural information processing systems*, vol. 30, 2017.
- [58] E. T. M. Beltrán, Á. L. P. Gómez, C. Feng, *et al.*, “Fedstellar: A platform for decentralized federated learning”, *Expert Systems with Applications*, vol. 242, p. 122 861, 2024.
- [59] Y. LeCun and C. Cortes, “Mnist handwritten digit database”, 2010.
- [60] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms”, *arXiv preprint arXiv:1708.07747*, 2017.
- [61] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images”, 2009.

Abbreviations

FL	Federated Learning
CFL	Centralized Federated Learning
DFL	Decentralized Federated Learning
IID	Independent and Identically Distributed
Non-IID	Non-Independent and Identically Distributed
MTD	Moving Target Defense
IoT	Internet of Things
ASR	Address Space Randomization
ISR	Instruction Set Randomization
FMTL	Federated Multitask Learning
SGD	Stochastic Gradient Descent
FedAvg	Federated Averaging
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
ASR	Attack Success Rate
PNR	Poisoned Node Ratio
PSR	Poisoned Sample Ratio
NR	Noise Ratio

List of Figures

2.1	Centralized Federated Learning (left) and Decentralized Federated Learning (right) [4]	6
2.2	Taxonomy of Poisoning Attacks	8
2.3	Classification of Poisoning Attacks Based on Attack Objectives [11]	8
2.4	Classification of Non-IID Data	11
2.5	Model Divergence for IID and Non-IID Data	12
2.6	Classification of Moving Target Defense	14
3.1	Non-IID Data Handling Techniques Classification	19
4.1	Workflow of Defense Mechanism	30
4.2	Model Similarity Measurement Using Euclidean Distance and Cosine Similarity	32
6.1	Initial Network Topology for Decentralized Federated Learning	49
6.2	Model Performance with IID and Non-IID Data	52
6.3	Performance under untargeted label flipping for MNIST with varied PNRs	53
6.4	Performance under untargeted label flipping for FashionMNIST and CIFAR-10 with varied PNRs	54
6.5	Performance under untargeted model poisoning for MNIST with varied PNRs	55
6.6	Performance under untargeted model poisoning for FashionMNIST with varied PNRs	56
6.7	Performance under untargeted model poisoning for CIFAR-10 with varied PNRs	57
6.8	Model performance measured by F1-score on MNIST under backdoor attack with 70% PNR using a test dataset without triggers	58

6.9	Model performance measured by ASR on MNIST and FashionMNIST under backdoor attacks with varied PNRs using a test dataset with triggers .	59
6.10	Confusion matrix at round 10 for participant node 9 with two MTD mechanisms. Evaluated on the test dataset with triggers under backdoor attacks on MNIST. $\text{PNR} = 30\%$	60
6.11	Performance with different network topologies in the absence of attacks . .	61
6.12	Performance with different network topologies under label flipping attacks for MNIST without defense	62
6.13	Star topology of MNIST under label flipping attacks, where red nodes represent malicious nodes and blue nodes represent benign nodes.	62
6.14	Performance with different network topologies under label flipping attacks for FashionMNIST and CIFAR-10 without defense	63
6.15	Performance with different network topologies under label flipping attacks for MNIST with MTD	64
6.16	Network topology in round 0 and round 10, where red nodes represent malicious nodes and blue nodes represent benign nodes.	64
6.17	Performance with different aggregation algorithms in the absence of attacks	65
6.18	Performance with different aggregation algorithms under label flipping attacks for MNIST without defense	66
6.19	Performance with different aggregation algorithms under label flipping attacks for FashionMNIST without defense	67
6.20	Performance with different aggregation algorithms under label flipping attacks for CIFAR-10 without defense	68
6.21	MNIST model performance with reactive topology MTD and FedProx under label flipping attacks	69

List of Tables

3.1	Non-IID Data Handling Techniques	27
3.2	MTD Applications in Various Domains	27
6.1	Experiment Variables and Options	50
6.2	Experimental Setup. R-T, R-A, R-TA, and P-TA represent reactive topology MTD, reactive aggregation MTD, reactive topology and aggregation MTD, and proactive topology and aggregation MTD, respectively.	50