



**University of
Zurich^{UZH}**

Design and Implementation of an Information Metrics-based Anomaly Model Detector in Decentralized Federated Learning

*Xiao Chen
Zürich, Switzerland
Student ID: 21-742-820*

Supervisor: Chao Feng, Dr. Alberto Huertas Celdran
Date of Submission: July 2, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland



Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: <http://www.csg.uzh.ch/>

Abstract

Federated Learning (FL) remains vulnerable to poisoning attacks, which are malicious attempts to manipulate the learning process by injecting poisoned data during training. As a solution to the traditional client-server architecture of FL, Decentralized FL completes communication without a central server. However, ensuring the security of the entire system has become more complex since each client has access to the model parameters. Existing defense mechanisms may lose their effectiveness against poisoning attacks when the data is non-IID. Therefore, it is necessary to develop an endogenous and data-independent defense mechanism.

This work proposes a defense approach called “AIDFL” for mitigating untargeted data poisoning attacks in Decentralized Federated Learning. AIDFL utilizes information theory-based metrics, which have the advantage of being independent of the data distribution, to define a detection defense mechanism consisting of k-means clustering, estimation of mutual information and conditional entropy, and anomaly detection. In addition, AIDFL was evaluated on different datasets under different configurations of poisoning parameters. It is verified that AIDFL achieves outstanding results in defending against untargeted data poisoning (label flipping and sample poisoning), which promotes the research on defense under non-IID setting in decentralized federated learning. Finally, the results demonstrate the necessity for further research on defense against model poisoning in the same settings.

Zusammenfassung

Föderiertes Lernen bleibt anfällig für Vergiftungsangriffe, bei denen es sich um böswillige Versuche handelt, den Lernprozess durch Einspeisung von vergifteten Daten während des Trainings zu manipulieren. Als Lösung für die traditionelle Client-Server-Architektur von FL ermöglicht dezentralisiertes FL die Kommunikation ohne einen zentralen Server. Allerdings ist die Sicherstellung der Sicherheit des gesamten Systems komplizierter geworden, da jeder Client Zugriff auf die Modellparameter hat. Bestehende Abwehrmechanismen können ihre Wirksamkeit gegen Vergiftungsangriffe verlieren, wenn die Daten nicht unabhängig und identisch verteilt (non-IID) sind. Daher ist es notwendig, einen endogenen und datenunabhängigen Abwehrmechanismus zu entwickeln.

Diese Arbeit schlägt einen Verteidigungsansatz namens „AIDFL“ vor, um nicht gezielte Datenvergiftungsangriffe im dezentralisierten föderierten Lernen zu mildern. AIDFL nutzt informationstheoretische Metriken, die den Vorteil haben, unabhängig von der Datenverteilung zu sein, um einen Erkennungsabwehrmechanismus zu definieren, der aus K-Means Clustering, der Schätzung von wechselseitiger Information und bedingter Entropie sowie Anomalieerkennung besteht. Darüber hinaus wurde AIDFL auf verschiedenen Datensätzen unter verschiedenen Konfigurationen von Vergiftung Parametern evaluiert. Es wurde bestätigt, dass AIDFL herausragende Ergebnisse bei der Abwehr von nicht gezielten Datenvergiftungen (Label-Flippen und Datenvergiftung) erzielt, was die Forschung zur Abwehr von Vergiftungsangriffen unter nicht-IID-Bedingungen im dezentralisierten föderierten Lernen fördert. Schließlich zeigen die Ergebnisse die Notwendigkeit weiterer Forschung zur Abwehr von Modellvergiftungen unter den gleichen Bedingungen.

Acknowledgments

First and foremost, I would like to express my gratitude to Chao Feng, a PhD student at the Communication Systems Group at the University of Zurich, for mentoring my Master thesis. In our bi-weekly meetings, Chao patiently and promptly resolved any questions I had, providing guidance to keep me on the right track. I would also like to extend special thanks to Lisai Cao and Zhuqiang Li from Jilin University for lending me the remote server to complete my experiments. My appreciation also goes to my friend Shushi Luo, who inspired the naming of the algorithm, and to my fitness instructor, Mr. Fish, who helped relieve my stress during tense and anxious times. Lastly, I am grateful to Prof. Dr. Stiller for allowing me to complete my Master thesis at CSG.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	2
1.3 Thesis Outline	2
2 Background	4
2.1 Federated Learning	4
2.1.1 Federated Optimization Algorithm	4
2.1.2 Decentralized Federated Learning	6
2.1.3 DFL Approaches: non-IID Applications	7
2.2 Poisoning Attacks	8
2.2.1 Attack Surfaces	8
2.2.2 Attack Goals	11
2.3 Information Theory	12
2.3.1 Entropy	13
2.3.2 Mutual Information	13
2.3.3 Information Bottleneck (IB) Method	15
2.3.4 Applications in Machine Learning	15

3 Related Work	17
3.1 Robust Aggregation	17
3.2 Anomaly Detection	19
3.3 Perturbation Mechanism	20
3.4 Hybrid Defenses	21
3.5 Information theory-based Defense Approaches	21
3.6 Research Motivation	23
4 Design	26
4.1 Evaluation Metrics	26
4.2 Attack Specification	28
4.3 Defense Desgin	28
4.3.1 Pre-Design	29
4.3.2 AIDFL	30
5 Implementation	36
6 Evaluation	41
6.1 Experiment Setup	41
6.1.1 Datasets and Models	41
6.1.2 Selected Reference Approaches	43
6.1.3 Threat Models	44
6.2 Results	45
6.2.1 Baseline Performance	45
6.2.2 Untargeted Label Flipping	49
6.2.3 Untargeted Sample Poisoning	55
6.2.4 AIDFL: Compare performance between IID and non-IID settings .	65

7 Summary and Conclusions	70
7.1 Conclusions	70
7.2 Limitations	71
7.3 Future Work	71
Bibliography	72
Abbreviations	78
List of Figures	78
List of Tables	81
List of Algorithms	83

Chapter 1

Introduction

Typical Machine Learning (ML) algorithms rely on the centralized processing and analysis of large amounts of data. When confronted with limited resource access, concerns about data privacy, and issues related to national policies and regulations [1], these techniques may face problems such as the leakage of sensitive information and violations of privacy. This situation is common in a few industries, especially in fields involving sensitive data such as healthcare, financial sectors, and government departments. Therefore, Federated learning (FL), a privacy-preserving distributed ML paradigm, is explored and developed in order to allow multiple participants to collaboratively train ML models on their local devices and send updates to a central server without sharing data [2].

One of the limitations of FL systems is their vulnerability to poisoning attacks, which are malicious attempts to manipulate the learning process by injecting poisoned or adversarial data during training [3] due to the centralized orchestration structure, and they essentially apply non-independent and identically distributed (non-IID) data in real world cases [4]. Decentralized FL (DFL) has emerged as a solution to the traditional client-server architecture of FL and data heterogeneity [5]. Nevertheless, ensuring the security of the entire system has become more complex since each client has access to the model parameters [6].

1.1 Motivation

Existing defense mechanisms against poisoning attacks primarily focus on exogenous factors [7]. However, these data-driven processes may lead to misidentification of outliers, potentially overlooking real poisoning attacks due to differences in the training data distribution. Therefore, developing an endogenous and data-independent defense mechanism, especially under non-IID conditions is crucial.

Previous researches introduced theoretical approaches that use layer-wise information metrics [8] such as mutual information to generate an information plane, which quantifies and analyzes DNN performance [9]. Defense Mechanisms utilizing information-theoretic characteristics to endogenously quantify and explain the orderliness of the model, as well as interconnections between layers in neural networks. These methods are potential in

non-IID environments due to their independent of data distribution. This work aims to analyze, design, and prototype an information theory-based defense mechanism, targeting a broader range of poisoning attacks and enhancing the adaptability and robustness of defense systems in DFL environments, particularly under non-IID conditions.

1.2 Description of Work

In the first stage, the state-of-the-art concepts, technologies, and systems involved in the project are reviewed and documented. During the design phase, this paper proposes an anomaly detection method named “AIDFL”, presenting a proposal outlining the design and structure of the novel defense mechanism, improving the ability to detect untargeted data poisoning attacks under non-IID environments.

AIDFL is defined as a three-phase anomaly detection protocol used to identify potential malicious updates. After splitting the data in non-IID, gradient descent is performed, followed by the feature extraction. Next, K-means clustering is applied to the sample set of each client. After estimating mutual information and conditional entropy, thresholds are set based on the mean and standard deviation for anomalies filtering. Finally, the normal weights are aggregated and normalized to be updated. The updated model will reduce the impact of data poisoning attacks. AIDFL combines anomaly detection and robust aggregation in a hybrid defense mechanism and sets up a series of poisoning configurations to validate the effectiveness of the defense against poisoning attacks for subsequent evaluation.

Lastly, the proposed defense mechanism and its implementation have been evaluated, analyzed, and discussed across various dimensions by measuring a set of metrics to illustrate the validity of the algorithm in non-IID environments. Currently, there are extensive experiments on MNIST, FMNIST, and CIFAR10 implemented to evaluate AIFDL. Simulations include different poisoning nodes ratio (PNR), poisoning sample ratio (PSR), and noise ratio (NR) under untargeted data poisoning attacks. Overall, AIDFL has demonstrated outstanding performance, advancing the robustness research of DFL in non-IID environments.

1.3 Thesis Outline

The overview and structure of this thesis are as follows: Chapter 2 provides a detailed introduction to the basic theories and concepts involved in this thesis. Then it lists related works, focusing on the classification of defense mechanisms in non-IID environments at the current stage, with special emphasis on those mechanisms based on information theory in Chapter 3. Next, Chapter 4 proposes a new defense mechanism based on the summary and improvement of related work. It also defines two types of data poisoning attacks and specifies metrics to be used as an evaluation later on. Chapter 5 will outline the implementation details and technical specifications of the defense strategy. An extensive evaluation of the defense mechanisms proposed against the poisoning attacks specified in

Chapter 4 will be provided in Chapter 6. Finally, Chapter 7 summarizes the results and observations of the experiments, and presents the limitations during the implementation process, together with the direction of future work.

Chapter 2

Background

The following sections introduce the background of FL and DFL with their inherent opportunities and vulnerabilities, as well as taxonomy according to the targets of poisoning attacks. In practical scenarios, data distribution varies across different clients. The entire thesis from this chapter forward will focus on studying DFL under non-IID conditions.

2.1 Federated Learning

The conventional centralized data processing workflow involves aggregating data collected from various sources to a single server or data center [10] for further unified processing and analysis. This method is widely utilized by researchers for its effectiveness in applying ML algorithms on large datasets to create predictive models, but it may raise concerns related to data security and the requirement for substantial computational resources [1].

FL utilizes the concept of edge computing [11], where data processing and model training only occur on local devices rather than a central server. In FL, each client independently uses its own data for learning and sends updates to the cloud or central server [10], which plays a key orchestrating role by collecting and integrating these updates to optimize and improve the overall global model [6]. FL reduces reliance on centralized processing methods by allowing multiple devices to collaboratively train models. This approach reduces data transmission costs and holds significant potential in addressing data privacy related concerns.

2.1.1 Federated Optimization Algorithm

The server-client architecture in FL allows the server to aggregate model updates trained locally on clients to optimize the global model. The updated models are then sent back to the clients for further training [12]. This process does not require central data storage, thus preserving the privacy of the data. Based on this characteristic, [12] proposed the

FederatedAveraging (FedAvg) algorithm, which is regarded as the standard framework of FL and widely applied today. This method builds on the original FL framework by splitting data into batches and performing stochastic gradient descent (SGD) on each client. The central server then takes charge of calculating the weighted average of the parallel model updates.

The detailed process and steps of the FedAvg algorithm typically involve 5 stages [13]:

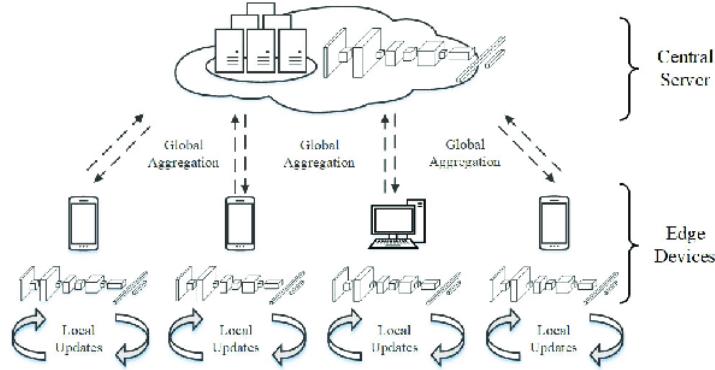


Figure 2.1: Structure of FedAvg. [14]

- **Initialization.** The server initializes the global model w_0 and distributes it to all participating clients. Define the number of epochs as E and there are K clients.
- **Local Training.** Each client k independently trains the model on their local data using SGD.
- **Model Updates.** Each client sends their model updates parallelly to the server after training in each epoch. The updated model is given by:

$$w \leftarrow w - \eta \nabla \ell(w; b) \quad (2.1)$$

where η is the learning rate.

- **Aggregation.** After each round, the server aggregate these updates by computing the weighted average:

$$m_t \leftarrow \sum_{k \in S_t} n_k \quad (2.2)$$

$$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k \quad (2.3)$$

where n_k is the number of data points at client k , and m_t is total number of data points.

- **Iteration.** The server distributes the updated parameters back to clients and repeat the above processes round by round until the global model (server) converges or meets specific performance criteria [12] [13].

Compared to other Federated optimization algorithms, especially those applying SGD, FedAvg uses less rounds of communication to train models by effectively combining local learning with global aggregation, reducing the need for frequent data transmission and helping to maintain data privacy. Security of the system can still be guaranteed even if there are sets of clients drop out during training time [12] [15].

However, as the number of clients k and epochs E increases, whether FedAvg directly drops out the models of clients that cannot complete training on time or then uses unfinished models for aggregation, it will impact the convergence effect of the final global model. This also reflects the system heterogeneity of FedAvg, which solves the problem of non-variable local work due to network scale and system constraints by directly dropping out unfinished models. From the perspective of data heterogeneity, distribution of data generated by different devices will vary greatly. Therefore, the local models will be skewed from the original global model under non-IID data settings. [16]

2.1.2 Decentralized Federated Learning

FL, like other client-server systems, can probably face single point of failure issues. To address this, DFL eliminates central orchestration, reducing overload, enhancing transmission speed, and mitigating the effects of data heterogeneity. DFL operates on a P2P basis without a central server, positioning all nodes as equals. Each client shares its model updates with immediate neighbors, who then aggregate these updates locally, leading to a decentralized model improvement process across the network. This setup is illustrated in Figure 2.2, showcasing the structural differences between traditional FL and DFL [6] [2].

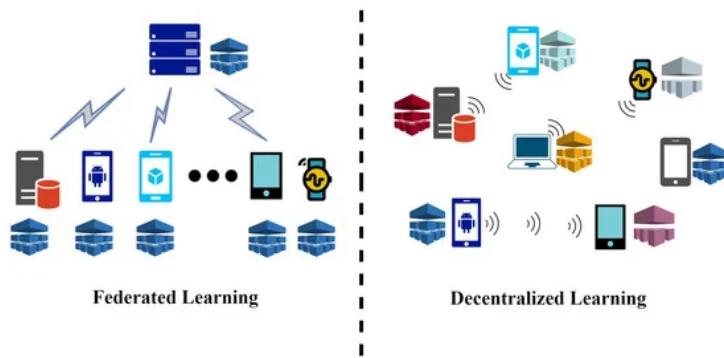


Figure 2.2: Difference between (Centralized)FL and DFL. [17]

DFL Approaches can generally be classified based on key characteristics such as *scalability* [6] (resilience to node and new participants join failures), *communication architecture* (P2P networks capacities), *data distribution handling* (methods to tackle non-IID data), and *privay & security measures* (blockchain integration or secure aggregation protocols), which reflect the strategies DFL employs to address challenges in decentralized environments.

2.1.3 DFL Approaches: non-IID Applications

This thesis focuses on creating a defense system for handling non-IID data within DFL, emphasizing the challenge of data heterogeneity. The main objective is to ensure robust model performance in distributed environments, addressing the difficulties FL methods face with non-IID data. The distribution, varying widely across clients, can introduce model bias and reduce efficiency, underscoring the need for innovative solutions to achieve robustness in model training and performance in DFL settings. [18].

PENS. As an innovative method in DFL, Performance-Based Neighbor Selection (PENS) is tailored for enhancing model accuracy in scenarios with non-IID data. It involves clients selecting neighbors based on performance metrics, indicating similar data characteristics. This selection enables clients to collaborate effectively by evaluating and sharing training losses, thus facilitating a model learning process that is better suited to the specific data distributions of each client. However, this method requires comparing performance metrics of clients to assess their similarity without compromising data privacy. Additionally, managing dynamic networks, ensuring scalability of PENS, and efficiently addressing communication overhead are significant challenges. Successfully overcoming these issues is crucial for PENS to effectively improve model accuracy in non-IID data scenarios [4].

AsyNG. To reduce communication cost, Chen *et al.* [19] proposed an algorithm which incorporates asynchronous operations, neighbor selection, and gradient push mechanisms to improve model training performance. For each edge node, gradients are only pushed to a subset of neighbors, allowing for dynamic neighbor selection for each node. Theoretically, AsyNG uses convergence analysis and a priority-based selection algorithm to dynamically optimize communication and model performance. Its effectiveness and accuracy have been demonstrated through extensive experimental validation. This innovative approach helps mitigate the impact of data and system heterogeneity, making a valuable contribution to DFL research. Since the article does not explicitly mention the implementation challenges of the method, this section will outline a key challenge that asynchronous methods in DFL may encounter: managing stale updates resulting from asynchronous communication, which could potentially slow the convergence of models. This highlights a critical aspect of asynchronous methods, focusing on the balance between efficiency and the risk of outdated information affecting learning progress.

Def-KT. [20] Decentralized Federated Learning via Mutual Knowledge Transfer (Def-KT) is designed to tackle data heterogeneity in DFL within IoT systems, where IoT clients collaboratively train models for image classification tasks in a structure without a central server, thereby reducing the risk of privacy data leakage. It consists of two main steps:

- (i) **Model Updating:** Fix number of clients train their models on private data using SGD and then share their fine-tuned models with another randomly selected client.

- (ii) **Model Fusing [21]:** Receiving clients integrate knowledge from both their local model and the received one through Mutual Knowledge Transfer (MKT) rather than model averaging [22] [23].

`Def-KT` mitigates client-drift and enhances model generalization across heterogeneous datasets by iterating the steps. This approach, while demonstrating outstanding effectiveness in addressing data heterogeneity in DFL, may face certain challenges and opportunities for future work. For example, it may involve optimizing process for efficiency across diverse and dynamic networks, and exploring fault tolerance and scalability in other larger decentralized networks. Future work might also explore bandwidth utilization [6] of `Def-KT` and conduct a theoretical analysis to deepen the understanding of its potential applications [20].

DFL has presented substantial advantages due to the structural feature of no centralized server, allowing data to be processed on local devices, which significantly further improves communication efficiency. Moreover, it also improves system resilience and scalability by eliminating centralized bottlenecks, which facilitates a more robust and efficient learning environment across distributed networks. The decentralized structure of DFL, while offering numerous benefits over FL, also introduces specific challenges. In addition to budget constraints and peer trust common in classic P2P systems, DFL is particularly susceptible to poisoning attacks. The complexity of ensuring the security of the entire system increases since each client has access to the model parameters. Classification of poisoning attacks, focusing on non-IID data will be introduced in detail in the following section [2] [6] [24].

2.2 Poisoning Attacks

In FL, poisoning attacks are where attackers intentionally manipulating models or data [25] during the aggregation process, resulting in wrong updates of the global model [26], degrading the overall performance and integrity of the model. Since poisoning attacks occur during the training phase, in a training environment without a centralized server, some clients may execute these attacks undetected. It can be inferred that DFL also faces similar threats [2] [6].

Poisoning attacks can be classified based on the poisoned surfaces or the objectives of adversaries. The poisoned attack areas refer to malicious interference during the training process or aggregation phase. The attack goals clarifies if there is a specific target behind the attack, shaping the strategy and potential impact on the learning system. The following sections will explore the concepts and specific implementation methods for each category of poisoning attacks.

2.2.1 Attack Surfaces

Based on the area of attack, poisoning attacks can be categorized into two types: data poisoning attacks and model poisoning attacks. Data poisoning attack involves malicious

actors manipulating the training data on their local devices [25] before it is used to update the global model. FL relies on aggregating updates from many distributed sources, even a small number of compromised devices can significantly impact the overall model integrity. Model poisoning attack refers to malicious participants deliberately altering their model updates before aggregation to corrupt the global model [27]. Recent studies [28] [29] have indicated that model poisoning attacks can be particularly severe as they directly manipulate the aggregated model without detection, causing immediate and widespread degradation in system performance. Compared to model poisoning, data poisoning might require more effort to impact the model due to aggregation and averaging processes in FL.

Data Poisoning Attacks. Generally, an attacker manipulates the training data on their local device to include incorrect or misleading information [26]. This can involve altering labels, injecting malicious samples, or distorting features within the dataset [30]. The poisoned data, when used for local model training, produces skewed updates that, once aggregated into the global model. The goal of the attacks is to manipulate the model into producing incorrect outcomes, leading to inaccurate predictions or decisions in future tasks. Specific strategies of data poisoning attacks, including label flipping attacks and poisoning samples attacks, will be detailed later. The attackers exploit the trust FL systems place in local updates, and the distributed nature of the learning process to introduce corruption subtly.

- **Label-Flipping Attacks:** Adversaries with access to training data alter the labels of certain data points [25]. For example, they might maliciously label multiple instances of "1" as "7" [31] as shown in Figure 2.3, while keeping the data features and content intact, to deceive the FL models. Label flipping can be random or targeted, aiming to compromise the model's learning process [30].

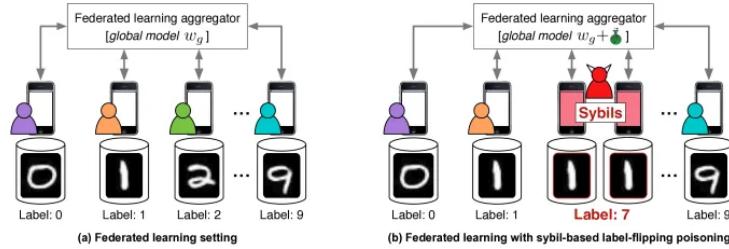


Figure 2.3: Label-Flipping Attacks: Adversaries alter labels. [31]

- **Poisoning Samples Attacks:** In this type of attack, adversaries modify the training data by introducing malicious patterns or adding noise to create poisoned samples [25] [30]. Unlike label flipping, poisoned samples alter the features of the data without changing the labels. The majority of methods in recent researches for generating poisoned samples utilize Generative Adversarial Networks (GANs) [32]. Attackers disguised as honest clients train a GAN to create synthetic data that closely mimics genuine training samples. They then use this simulated data to generate poisoned samples, which are subsequently sent during the aggregation phase to introduce poisoned updates aimed at compromising the global model [30].

Model Poisoning Attacks. Data poisoning attacks ultimately lead to model poisoning, hence they can be regarded as a subset of model poisoning attacks [25]. Model poisoning attacks methods mentioned in this paper typically manipulate the local model parameters directly before they are sent for aggregation. The attackers attempt to deliberately adjusting the weights or gradients of their model to introduce errors or biases [25] [26] [30]. As shown in Figure 2.4, the initial global model is defined as ω_0 . After completing the k^{th} round of local model training, malicious clients modify the model update δ_i^k , where δ_i^k represents the weight difference for each client between rounds k and $k - 1$ [27]. When the updates are aggregated, it can reduce efficiency, slow down the convergence of the global model and make inaccurate predictions. Attackers craft these manipulations carefully to avoid detection [26]. Especially in non-IID settings, it becomes difficult for the central server to detect these malicious updates [27].

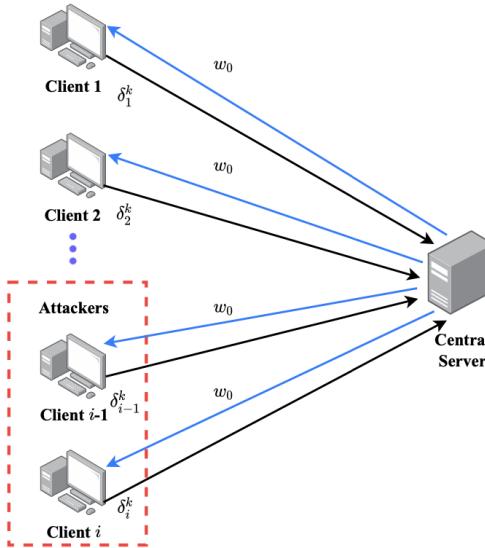


Figure 2.4: Model poisoning attack directly manipulate the local models in FL. [27]

The model poisoning attacks can be categorized into the following two types through the different generation methods of the modified parameters:

- **Optimization Methods:** Optimization methods maximize the impact of the model poisoning attacks while minimizing the difference between the original model from the previous round and the poisoned model. This approach significantly enhances stealth of the attack mechanisms, making it difficult for the central server to detect. Therefore, optimization methods are particularly effective in targeted attacks, especially backdoor attacks, by fine-tuning the attack to evade detection while achieving the intended malicious outcomes [25] [30] [33].
- **Random Weights Generation:** In model poisoning attacks, the strategy of generating random weights involves attackers randomly creating a set of weights that match the dimension of the original weights of the model [25]. Based on their understanding of the global model, adversaries deduce a numerical interval $[-R, R]$ and

generate input data within this range [30]. When these specially crafted inputs are incorporated into the training process, it leads to the adoption of randomly assigned weights, thereby reducing the performance of the global model and compromising its functionality. [33]

2.2.2 Attack Goals

Poisoning attacks can indeed be categorized based on the attack objectives into targeted attacks, which aim at manipulating specific model outcomes, backdoor attacks, and untargeted attacks, which seek to degrade the overall performance of the model.

Targeted Attacks. In FL, targeted poisoning attacks are designed to cause the model to misclassify specific types of inputs. This is achieved by manipulating the training set or local models, leading to incorrect predictions by the model [25] [34]. Since this type of attacks only generates poisoned updates through the injection of specific malicious clients and does not affect the overall performance for other inputs, it is harder to detect compared to untargeted attacks [30]. Most targeted poisoning attacks have been categorized into data poisoning and model poisoning attacks previously. Besides, Xingchen et al. proposed a targeted model poisoning attack algorithm [35] based on capacity of the neural network. Adversaries inject poisoned neurons into the model’s redundant space After understanding the model capacity. This method has achieved significant results in terms of stealth and persistence.

Backdoor Attacks. Backdoor attacks typically occur during the training phase, where adversaries manipulate part of the training dataset by injecting hidden patterns or triggers to manipulate the model training process [25]. This type of attack causes samples with triggers to behave anomalously, while other benign clients remain unchanged [36]. These attacks can be categorized primarily into two types:

1. Semantic attacks: The targeted inputs possess natural properties, such as specific pixels, striped triggers, or word sequences. The attackers do not modify the original features of the samples but will flip their labels [37] [38].
2. Artificial attacks: This type within backdoor poisoning attacks specifically refer to scenarios where attackers artificially insert synthetic triggers into training data to manipulate the behavior of the trained model. Unlike semantic attacks, artificial attacks additionally assign patterns to target samples in the testing phase. For example in the Figure 2.5, comprised clients are artificially added with global triggers or tuned partial triggers to modify the pictures [37] [38].

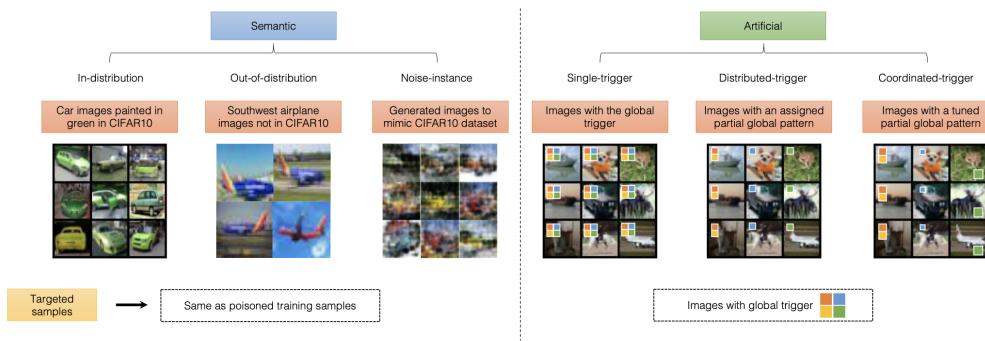


Figure 2.5: Semantic attacks and artificial attacks in backdoor attacks. [37]

In the classification method described in this paper, backdoor attacks are considered as a subclass of targeted attacks, which means they do not affect the overall performance of the model. This characteristic makes backdoor attacks difficult to detect because they are designed to activate only under specific conditions, leaving the general functionality of the model unaffected for benign clients. [25] [30]

Untargeted Attacks. Different from the two targeted attacks mentioned earlier, untargeted attacks aim to degrade the integrity and performance of the learning process [25] [26]. For example, samples are attacked indiscriminately in a classification task, leading to their incorrect classification across the board. One classic scenario of is the Byzantine attack [25] [39], where participants maliciously share harmful information by introducing incorrect updates or gradients. Common untargeted attack methods include noise injection—introducing random noise into data and model parameters [26], which can be categorized as model poisoning strategies (discussed in 2.2.1). Another approach is to influence the convergence of the model by manipulating the weights of the global model [26] [39]. Additionally, data poisoning techniques like untargeted label flipping are employed. The purpose of these methods is to reduce the accuracy and reliability of the model without targeting specific outcomes or classifications. Therefore, untargeted attacks are easier to detect because it only requires comparing model updates to identify anomalies [30].

2.3 Information Theory

Information theory, as founded by Claude Shannon in the 1940s, is a mathematical framework for quantifying, storing, and communicating information. Core concepts of information theory are entropy, which measures data uncertainty, and mutual information, quantifying the shared information between input variables and resulting outcomes. These principles have been widely applications, and have been instrumental in advancements across science domains and technology.

Information theory is considered because the main objective of this paper is to design a defense strategy using intrinsic model variables, independent of uneven data distribution.

Information theory has become a powerful tool for developing defense mechanisms that do not rely directly on the characteristics of the data, but rather on the underlying information processing principles of the models themselves.

Information theory can quantify attacks by evaluating the impact on the entropy or the mutual information. Significant changes in these metrics can reveal the fact that the model has been tampered with by an attack, as it suggests changes to the information processing behavior of the model. This approach provides a quantitative framework for detecting and analyzing the effects of attacks on a model's performance and reliability. The remaining sections of this chapter will cover some foundational concepts of information theory and their application methods in ML.

2.3.1 Entropy

Entropy is a measure of the uncertainty or randomness in a dataset, which is represented as the expected amount of information [40]. For a discrete random variable X that follows probability P , the entropy H is given by:

$$H(X) = - \sum_i p(x_i) \log p(x_i) \quad (2.4)$$

where $p(x_i)$ is the probability mass function (p.m.f.) of the random variable X for each discrete outcome x_i .

Similarly for continuous random variables X , $p(x)$ is the probability density function (p.d.f.) of X , then the differential entropy is given by the formula:

$$H(X) = - \int_{-\infty}^{\infty} p(x) \log p(x) dx \quad (2.5)$$

Negative logarithms and expectations are used in the definition of entropy in information theory. Logarithm functions are applied because they can convert the product of probabilities into a summation, which is essential for additivity of entropy across independent random variables.

Typically, it is assumed that frequent events carry less information than rare events because rare events provide more "surprise" and thus contain more information when they occur. Therefore, a negative sign is used to establish a decreasing relationship between the probabilities and their associated information content, ensuring that entropy remains positive [40]. Entropy is used in ML to quantify the amount of information required to describe the state of a system or to predict the outcome of a random variable.

2.3.2 Mutual Information

While entropy is used to formulate the quantity of information, mutual information quantifies the amount of information shared between two random variables. It reflects how much understanding one variable aids in reducing the uncertainty of another, thereby revealing the mutual dependence or correlation between the two variables.

- **Joint Entropy:** Combined with relevant concepts from statistics, joint entropy indicates the entropy of a joint distribution of two variables X and Y , quantifying the average amount of information needed to describe their joint outcomes.

Given the joint probability distribution of two discrete randoms X and Y as $p(x, y)$, the joint entropy $H(X, Y)$ [40] is defined as:

$$H(X, Y) = - \sum_x \sum_y p(x, y) \log p(x, y) \quad (2.6)$$

For continuous cases, $H(X, Y)$ will be:

$$H(X, Y) = - \int_x \int_y p(x, y) \log p(x, y) dx dy \quad (2.7)$$

From the concept of entropy, joint entropy can be inferred to represent the total uncertainty or the amount of information contained in a pair of random variables.

- **Conditional Entropy:** To calculate the shared information between two variables, the amount of information obtained by the two variables has already been determined in the previous paragraph. Therefore, it is necessary to further calculate the remaining uncertainty of one variable given the knowledge of another to accurately measure the degree of information sharing between them. Based on conditional probability in probability theory, $H(Y|X)$ is the conditional entropy of Y given X , representing the average uncertainty in Y when X is known [40]:

$$H(Y|X) = - \sum_x \sum_y p(x, y) \log \left(\frac{p(x, y)}{p(x)} \right) = - \sum_x \sum_y p(x, y) \log p(y|x) \quad (2.8)$$

for discrete random variables X and Y , and

$$H(Y|X) = - \int_x \int_y p(x, y) \log \left(\frac{p(x, y)}{p(x)} \right) dx dy = - \int_x \int_y p(x, y) \log p(y|x) dx dy \quad (2.9)$$

for continuous random variables. $p(x, y)$ represents the joint probability distribution of X and Y , and $p(x)$ represents the marginal probability distribution of X . This formula calculates the expected value of the logarithmic ratio of the joint probability to the marginal probability, summing or integrating over X and Y .

Therefore, information of Y given X , $H(Y|X)$, is equal to information obtained by X and Y together subtract information in X , and the same applies to Y as well [40]:

$$H(Y|X) = H(X, Y) - H(X) \quad (2.10)$$

$$H(X|Y) = H(X, Y) - H(Y)$$

- **Mutual Information:** Intuitively, the mutual information is defined in terms of joint entropy $H(X, Y)$ and the individual entropies $H(X)$ and $H(Y)$:

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \quad (2.11)$$

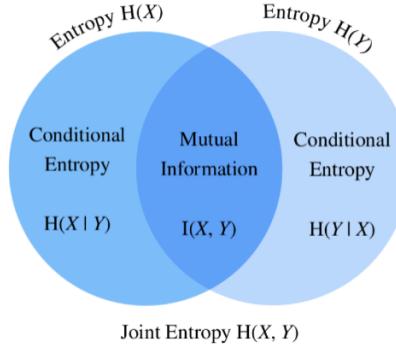


Figure 2.6: The relationship between mutual information, conditional entropy, and joint entropy. [40]

Alternatively, as shown in Figure 2.6, mutual information is the intersection of total information contained in both X and Y . Specifically, it can be expressed as the sum of information in X and Y minus the information exclusively contained in X and not in Y , and the information exclusively contained in Y but not in X . This measures the amount of information shared between X and Y :

$$I(X, Y) = H(X, Y) - H(X|Y) - H(Y|X) \quad (2.12)$$

2.3.3 Information Bottleneck (IB) Method

The Information Bottleneck (IB) method is a principle from information theory that seeks to find a compact representation of an input variable X that preserves as much information as possible about an output variable Y . Essentially, it is a method for extracting the part of the input that is most relevant for predicting the output, filtering out irrelevant parts of the input data.

The IB method formalizes this as an optimization problem:

$$\max_T I(T, Y) - \beta I(T, X) \quad (2.13)$$

where T is the transformed variable, representing the compressed version of X , and β is a Lagrange multiplier that trades off between compression and prediction accuracy.

The goal of IB optimal representation is to compress X by mapping it to T such that T retains as much information about Y as possible, while also being as small as possible. The parameter β controls the trade-off between the complexity of the representation and the amount of information about Y that is preserved. [41] [42] [43]

2.3.4 Applications in Machine Learning

Information theory provides a quantification method for the uncertainties that arise during the research process. The primary objective of ML is to extract insights from data and

make critical predictions. As a result, ML models trained with information theory leverage its principles to optimize data representation and decision processes. For example, the cross-entropy loss, commonly used in ML models, originates from concepts rooted in information theory. [40]

Common applications include feature selection, model evaluation, regularization, etc. With respect to the endogenous characteristics of models, information theory could help in understanding the intrinsic properties of data and the communication channels through which data is processed, leading to models that are not only more efficient in learning from data but also in making more informed predictions.

One of the typical applications is utilizing information theory to understand the learning dynamics of models. During the training of Deep Neural Networks (DNN), the transition from disordered to ordered feature extraction plays a crucial role in enabling the network to understand the meaning and abstract concepts in the data gradually. This eventually leads to the creation of a structured data representation, which improves the accuracy of identification tasks. Yu *et al.* [8] had introduced theoretical approaches that use layer-wise information metrics to examine the interactions and dynamics between different layers in DNNs.

Shwartz-Ziv *et al.* [9] proposed the concept of the information plane, an idea used in the analysis of DNNs within the context of information theory. It provides a graphical way to understand how information flows through a network during training. The plane is typically represented by plotting two quantities for each layer of the network: the horizontal axis $I(X; T)$, represents the mutual information between the input and the hidden layers T , and $I(Y; T)$ on the vertical coordinates, refers to the mutual information output (decision of the neural network) and the hidden layers. The position on the information plane reflects how much information about X and Y is preserved by different hidden layers. This visualization technique is influenced by the idea of IB theory.

In conclusion, this mechanism is based on principles from information theory and the inherent characteristics of the model, making it independent of data distribution. Consequently, the impact of data features on identification results is not significant. These methods have the potential to be applied in non-IID environments.

Overall, information theory provides mathematical tools and theoretical foundations for machine learning, enabling most research to theoretically analyze and optimize the performance of ML algorithms. Particularly in handling large-scale data and complex model structures, the role of information theory becomes especially important.

Chapter 3

Related Work

This chapter will discuss existing defense mechanisms, specifically their contributions and gaps in DFL. Existing research primarily focuses on centralized structures as they are more prone to malicious attacks. The current defense strategies can be classified into three types: robust aggregation, anomaly detection and perturbation mechanism. Each of them has unique applications in DFL, effectively addressing different kinds of attacks and threats. Understanding the strengths and limitations of these defense strategies enables this research to fill in the gaps in DFL, implementing a more robust and efficient security mechanism.

3.1 Robust Aggregation

In FL, robust aggregation is one type of the defense strategies that can mitigate poisoning attacks by altering the aggregation method of the global model during the training phase. The main objective of this approach is to ensure that the training of the entire model is not compromised by anomalous or malicious attacks. Most robust aggregations are mainly implemented through geometric methods or regularization. This chapter will provide a detailed introduction to several aggregations designed by geometric methods, which can be specifically divided into vector-wise and dimension-wise filtering. The following two mean-based methods utilize dimension filtering, by excluding extreme clients. Unlike Krum, which applies vector filtering to select vectors that are close to the majority [25] [26] [38]. Current robust aggregations could more effectively defend against untargeted attacks. Future research will focus on the targeted attacks which are hard to detect.

TrimmedMean. This method [44] is calculated by removing certain percentage of the highest and lowest updates before calculating the mean. Specifically, suppose there are values $a_1 < a_2 < a_3 < \dots < a_n$, and $2\beta < n$. Then the β -trimmed mean m_β of parameters $a_1, a_2, a_3, \dots, a_n$ is defined as:

$$m_\beta = \frac{\sum_{i=\beta+1}^{n-\beta} a_i}{n - 2\beta} \quad (3.1)$$

Median. Median (Coordinate-wise) is another Byzantine-robust aggregation defense strategy that applies dimension-wise filtering [45]. For the j^{th} global model parameter, the server sorts the j^{th} parameter from m computing nodes and takes the median as the j^{th} parameter for the global model. For the even cases, median is the average of the two middle parameters. Due to its insensitivity to skewed distributions [46], this robust aggregation is verified to be effective in model replacement attacks.

Similar to TrimmedMean, Median aggregation arrives to the optimal order error rate when the objective function is strongly convex. Furthermore, a common assumption under these two robust algorithms is that most clients are benign [38]. In addition to algorithms for dimension-based filtering, the following will continue with vector-wise filtering aggregation defenses: Krum and Bulyan, a variant of Krum and TrimmedMean combined approach.

Krum. Krum is a well-known aggregation method in FL for preventing malicious attackers from influencing the global model by manipulating the local model weights and it is robust against Byzantine attacks. The algorithm was first proposed by Blanchard *et al.* in 2017 [47]. The core idea of the Krum algorithm is to perform a special ranking and selection of participants' local model weights at the end of each training round. Specifically, Krum follows the steps below:

1. Calculate the distance between model weights: for each pair of participants i and j , calculate the Euclidean distance between their local model weight vectors.
2. Calculate the sum of distance for each participant: there are n participants in total, and for each participant i , assuming that there are f attackers, then calculate the sum between the participant's model weights and the other nearest $n - f - 1$ participants.
3. Select the model with the smallest sum of distance as the aggregated model.

In this way, Krum is able to build a consensus among the participants, filtering out abnormal weights that may be influenced by malicious attacks, and thus preserving the robustness of the global model.

Overall, Krum can be applied to various types of FL scenarios. However, the limitations are that it has a high computational complexity: Krum needs to calculate the distance between each pair of participants with a computational complexity of $O(n^2)$, where n is the number of participants. Besides, in the case of a large number of participants, the computational burden may be heavier, while Krum leads to higher communication overhead: Krum needs to transmit model weights and distance measures between participants, which may lead to larger communication overhead. In environments with limited network bandwidth or unstable communication, the efficiency of FL may be affected.

Bulyan. Mhadi *et al.* [48] combined the mechanisms of Krum and TrimmedMean to propose an algorithm called Bulyan, which has significant advantages in ensuring model robustness and stability. Firstly, Bulyan iteratively applies Krum to select $\theta < n - 2f$

model parameter updates, where n is the number of clients and f is the number of Byzantine clients. Then TrimmedMean is applied to these candidate parameters, removing the largest and smallest β values and calculating the corrected mean for the remaining parameters to produce the global aggregated model. Bulyan performs Krum multiple times in each iteration and use it to calculate pairwise distances (nearest neighbors) between the node data. Overall, Bulyan eliminates the effect of a portion of anomalous parameters in Krum.

The assumption in Krum that honest client updates are closer, while distance bias exists in malicious attacks does not hold in the non-IID setting of this thesis, as the unbalanced distribution of the data can lead to a large deviation in benign updates as well. Research has shown that implementing Krum in non-IID environment would lead to a degradation of model performance. The above results have been validated in FL [31] [49] [50], and it can be inferred that in DFL, relying only on the communication between clients further decreasing model performance due to the absence of a central server. This is due to the fact that in DFL, the communication between clients is more complex and unstable, and malicious participants are more likely to influence the global model.

3.2 Anomaly Detection

Defense methods for anomaly detection are typically independent of the aggregation process and have no dependency on the aggregation rules themselves. By analyzing updates or data from each client, the algorithms identify and remove anomalous results to maintain system stability. The next chapters will mention mechanisms that combine anomaly detection and robust aggregation to make the defense process more complete [25] [30].

LoMar. In LoMar[51], the authors argue that existing studies only considered malicious updates as global anomalies in FL, and have not analyzed the local feature patterns of malicious remote updates. This method first uses Kernel Density Estimation (KDE) to measure the distribution among neighboring clients, then calculates the malicious score using dynamic thresholds and transforms it into a Boolean vector to classify benign and malicious clients. It has been validated that LoMar shows strong robustness when dealing with non-IID scenarios. Furthermore, since this method does not rely on the distribution of training data, it inherently preserves the privacy characteristics of FL.

FoolsGold. As one of the defense mechanisms against security threats, Fung *et al.* [31] proposed an algorithm called FoolsGold that identifies Sybil attacks by monitoring and adjusting the weights of participants' contributions. First of all, FoolsGold assumes that benign participants produce more diverse updates, whereas updates from malicious participants tend to be similar. The method measures the similarity of participants' contributions by calculating the cosine similarity between the updates. When the updates are too similar, malicious Sybil groups are recognized. Overall, experiments have shown that FoolsGold performs stably and robustly against multiple attack patterns and works effectively with non-IID data distributions [30] [38] [52].

Spectral Anomaly Detection. Since FL architectures are vulnerable to be attacked by malicious clients, which can lead to an overall degradation of model performance, Li et al. proposed an approach that uses spectral anomaly detection [53] to identify malicious client updates. The framework creates an encoder-decoder architecture that embeds model updates into a low-dimensional space where normal and abnormal updates can be clearly identified based on their essential features, and the decoder is thus used to reconstruct model updates [38]. In addition, spectral anomaly detection applies dynamic thresholds to distinguish between benign and malicious updates. After considering all client updates in training rounds and excluding updates above the threshold from aggregation ensures that the negative impact of these updates is minimized, maintaining the integrity and performance of the global model. In the experimental phase, the method effectively mitigates both untargeted and targeted (artificial backdoor) attacks under non-IID condition [53] [52].

3.3 Perturbation Mechanism

Perturbation mechanisms apply privacy-preserving techniques, especially when dealing with shared model parameters or the local dataset of each client, to ensure that the privacy will not be comprised during training phase. The core of this type of mechanism is to add a certain amount of perturbation (noise) to the data or model to reduce useful information that can be extracted by an attacker, thus protecting the user's privacy while maintaining the utility of the data [25] [30] [54].

Differential Privacy. One of the most common method in perturbation mechanism is differential privacy, which is based on rigorous mathematical theory. In recent researches, it has been widely applied in different areas of data privacy protection [55].

Du *et al.* presents a method to improve the performance of anomaly detection and backdoor attack detection using differential privacy techniques [56]. The authors show how to enhance the detection capabilities to recognize abnormal data samples by adding random noise to the data or to the aggregation process. [25] [54]

FLAME. Existing differential privacy methods may degrade the performance of benign clients, therefore FLAME [57] has been proposed to utilize model clustering and weight-clipping methods to minimize the amount of noise required to eliminate backdoor attacks [25]. Specifically, FLAME identifies potentially poisoned updates through clustering analysis and eliminates outliers in participant updates by limiting the maximum threshold for the weights. Additionally, FLAME injects estimated noise into the weights to further reduce potential backdoor effects. Compared to conventional differential privacy methods, FLAME effectively eliminates backdoor while minimizing the impact on benign participants [38] [57].

3.4 Hybrid Defenses

The Hybrid Mechanisms, by combining robust aggregation and anomaly detection techniques, can enhance the overall performance and security of systems more effectively in certain scenarios than using a single technique alone. This combination leverages the strengths of both techniques to more effectively confront data tampering and malicious attacks [38].

Anomaly detection identifies unusual patterns in data or model updates, while robust aggregation reduces the impact of these outliers on the final global model. However, applying anomaly detection alone may ignore subtle anomalies, while robust aggregation may not be sufficient to handle large-scale coordinated attacks. Overall, in combined algorithms, robust aggregation can keep model performance when facing malicious attacks, and potential threats can be identified and responded to earlier by implementing anomaly detection, thus intervening earlier to prevent malicious data from affecting the model training process. [25] [30]

Trusted DFL. According to the research proposed by Gholami *et al.*, Trusted DFL [58] discusses enhancing security and trust relationships in DFL by implementing a trust metric. Each node in this framework is evaluated based on their behaviour scores, which reflects performance contribution and update consistency of each client [38]. The score of each node is computed based on the distance metric and in the framework of DFL, the client can compute the global trust score by broadcasting the local trust score to communicate with its neighbors. After the training phase, the global trust scores of the neighbors are served as weighting factors in the aggregation. Trusted DFL performs well against untargeted modeling attacks in non-IID scenarios, especially when dealing with random weight generation. For future directions of work, this includes continuing to evaluate the effectiveness of the defense mechanism against targeted backdoor attacks [38] [58].

3.5 Information theory-based Defense Approaches

After discussing three types of poisoning attack defense mechanisms in the previous sections, this section will introduce the information theory-based defense approaches. Information theory is the core concept of the algorithm design in this paper, hence the following two methods will involve metrics related to information theory and serve as the foundation of the design in this thesis.

VFedAD. The method presented in the article involves the Information Bottleneck (IB) principle (Section 2.3.3). Specifically, the paper addresses the challenge of data poisoning attacks in Vertical Federated Learning (VFL). To defend against these attacks, Lai *et al.* proposed a method called VFedAD [59], which is based on the information-theoretic mechanisms issuing vertical federated data poisoning attacks.

VFedAD uses an unsupervised learning approach to learn semantic client data representations. It does this through contrastive learning and cross-client prediction tasks, which are designed to maximize mutual information between representations of different clients while simultaneously minimizing conditional entropy to discard irrelevant information [59]. This aligns with the core concept of the IB method, which aims to retain as much relevant information about a target variable as possible while compressing the input variable.

For contrastive learning, the optimization objective is represented as [59]:

$$\mathcal{L}_{CL} = \sum_{k \neq l} \frac{1}{N} \sum_{i=1}^N \log \frac{e^{\text{sim}(Z_i^k, Z_i^l)}}{\sum_{j=1}^N e^{\text{sim}(Z_i^k, Z_j^l)}} \quad (3.2)$$

By minimizing the contrastive loss, VFedAD seeks to capture essential semantic information that is useful for detecting anomalies generated by data poisoning attacks, ensuring that the information relevant for making correct predictions is retained.

While discarding irrelevant information, the authors introduce cross-client prediction tasks, which is to minimize conditional entropy between representations of client k and client l , Z^k and Z^l . Therefore, minimizing cross-client prediction loss function for multiple clients is equivalent to [59]:

$$\mathcal{L}_{CP} = \sum_{k \neq l} \mathcal{L}_{CP}^{k|l} = \frac{1}{N} \sum_{i=1}^N \sum_{k \neq l} \|Z_i^k - g^{l \rightarrow k}(Z_i^l)\|_2^2 \quad (3.3)$$

Overall, optimizing the combination of the two loss functions will capture the most amount of information [59]:

$$\mathcal{L} = \mathcal{L}_{CL} + \lambda \mathcal{L}_{CP} \quad (3.4)$$

Although VFedAD utilizes mutual information and conditional entropy to filter information, it is essentially categorized as an anomaly detection mechanism. Therefore, after the loss function converges, the algorithm designs an anomaly scoring function that calculates the neighbor consistency score $S_{NC}(i)$ and client consistency score $S_{CC}(i)$ to identify and remove anomalous samples with higher scores, effectively defending against data poisoning in VFL [59].

Sageflow. The authors of Sageflow argue that existing defenses are insufficient to address stragglers and malicious attacks. Therefore, SageFlow is proposed to solve both issues simultaneously. The approach includes strategies such as staleness-aware grouping, entropy-based filtering, and loss-weighted averaging. By integrating these methods, malicious and honest clients are effectively distinguished, and the losses caused by malicious attacks are minimized [30] [60].

For staleness-aware grouping, SageFlow allows results sent from stragglers to be aggregated in later global rounds. This not only minimizes the effect of the delay, but also provides a good platform to counter the adversaries.

1. **Entropy-based filtering:** The Shannon entropy of client k is computed using a portion of the public data collected in the server after it receives updates from each device [60]:

$$E(k) = \frac{1}{n_{pub}} \sum_{j=1}^{n_{pub}} E_{x_{pub,j}}(k) \quad (3.5)$$

which represents the average of the Shannon entropy of sample $x_{pub,j}$ in the k^{th} client. Figure 3.1, shows that there is a gap between entropy of malicious clients and benign ones when confronting model poisoning attacks, a threshold E_{th} thus can easily be set and filter out anomalous models.

2. **Loss weighted averaging:** Sageflow performs loss weighted averaging by replacing the FedAvg in aggregation, compensating for the fact that entropy filtering does not work well in data poisoning [60].

The global model weight w_{t+1} for the next round $t + 1$ is computed by aggregating the model weights w_t of all clients k in the current round t . w_t is weighted by its corresponding weighting factor $\beta_t^{(k)}(\delta)$. In loss weighted averaging, $\beta_t^{(k)}(\delta)$ is inversely proportional to the performance of each client model on the public dataset, and the sum of all weights will be normalized to 1. The averaging algorithm can be computed as the following expressions [60]:

$$w_{t+1} = \sum_{k \in S_t} \beta_t^{(k)}(\delta) w_t^{(k)} \quad (3.6)$$

For data poisoning attacks, Sageflow reduces impact of adversaries via loss:

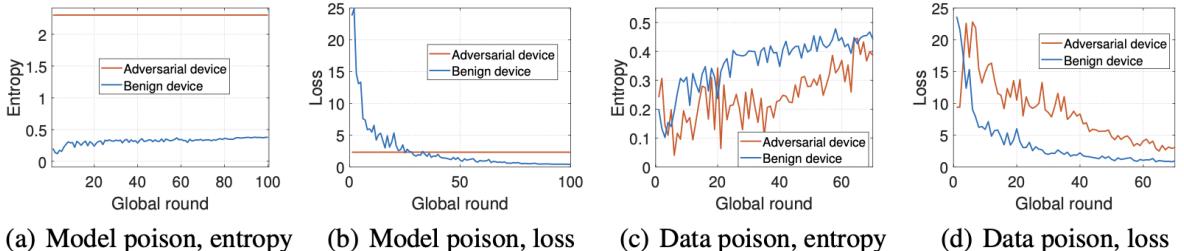


Figure 3.1: According to the type of attacks, two methods are used separately to filter out malicious models and mitigate the damage. [60]

Defending against both stragglers and adversaries by applying the above two approaches at different stages creates a hybrid defense mechanism that can deal with different types of poisoning attacks [30] [60].

3.6 Research Motivation

Most of the current researches focus on CFL, while relatively little has been explored on defense mechanisms in DFL, which removes the central server and transmits model

updates only through direct communication between clients, making the structure more vulnerable to malicious attackers. Despite the widespread interest in the prospects of DFL, there is still a lack of relevant research in this area. Therefore, it is particularly important to develop a decentralized algorithm that does not rely on a central server, especially one that can operate effectively in non-IID environment, which is much closer to practical application scenarios. This will be of great significance to promote the research progress of DFL in security and privacy protection.

The information-theoretic characteristics can endogenously quantify and explain the orderliness of the model and are independent of the data distribution, making it more suitable for application in non-IID environment. The thesis will use this idea to design a defense mechanism.

The Table 3.1 below summarize the previous work in CFL and DFL. It highlights the gaps in DFL research, strongly demonstrating the current need for new defense algorithms. In addition, the table outlines the method proposed in this paper, which is based on information theory and investigate defense mechanisms against data poisoning attacks in DFL under non-IID settings.

Table 3.1: Classification of defense approaches against poisoning attacks. For each defense listed, the type of attack (targeted/untargeted) and the scope (data/model) are indicated. The technique describes the main methods applied by each approach. This table primarily summarizes the defense strategies for DFL environment, with the information-theoretic defenses being based on VFL. Adopted from [25] [30].

	Category	Method	Objective	Technique	Architecture
Robust Aggregation	Robust Aggregation	Krum	D;M	Untargeted	Euclidean distance
		TrimmedMean	D;M	Untargeted	Filtered mean
		Median	D;M	Untargeted	Coordinate-wise median
	Bulyan	D;M	Untargeted	Krum and TrimmedMean	CFL (non-IID)
Current Poisoning Attack	Anomaly Detection	LoMar	D/M	Targeted	Kernel density estimation
		FoolsGold	Model	U/T	Cosine similarity
		Li <i>et al.</i>	D/M	U/T	Spectral Anomaly Detection
	Perturbation Mechanism	Du <i>et al.</i>	Data	Untargeted	Differential Privacy
Defense Mechanisms	Perturbation Mechanism	FLAME	Data	Targeted	Clustering;
					Adaptive clipping
	Hybrid Defenses	Trusted DFL	Model	Untargeted	Trusted aggregation
	Anomaly Detection	VFedAD	Data	Untargeted	Mutual information; Conditional entropy
Defenses Based on Information Theory	Robust Aggregation	Sageflow	D/M	Untargeted	Entropy filtering;
					Loss weighted averaging
	This work (Hybrid)	AIDFL	Data	Untargeted	Information theory; k-nearest neighbors

Chapter 4

Design

The previous chapters have provided a detailed introduction and summary of the current situation faced by DFL and the existing defense strategies. Based on the current research gaps, the theoretical foundation and baseline methods for this work are established. This chapter will discuss the design framework for the mechanism and simulated poisoning attack patterns, starting by selecting several types of poisoning attacks as the standard for evaluating the effectiveness of the algorithms, followed by an introduction to the algorithm processes and related concepts.

4.1 Evaluation Metrics

The following metrics are used to evaluate the effect of the defense, including three attack parameters and one measure of model performance [61]:

- **Poisoned Node Ratio (PNR)** is defined as the ratio of the number of poisoned nodes to the total number of clients participating in the network.

$$PNR(\%) = \frac{\text{number of poisoned nodes}}{\text{number of total clients}} \quad (4.1)$$

Understanding the PNR is crucial for assessing the threat level in a distributed system. It is an essential parameter for simulating attack scenarios and testing defense strategies under different levels of system compromise.

- **Poisoned Sample Ratio (PSR)** refers to the number of maliciously modified sample data or labels as a percentage of the total number of samples. This metric is used to assess the extent to which a dataset has been altered and is often used to analyze the impact of data poisoning attacks.

$$PSR(\%) = \frac{\text{number of poisoned samples}}{\text{number of total samples}} \quad (4.2)$$

- **Noise Ratio (NR)** used in data poisoning attacks is to control the degree or proportion of noise added, which reflects the noise intensity or coverage of the tampered samples in the dataset. There will be three noise types simulated in this algorithm design:

1. Salt Noise: It randomly sets the values of certain pixel points in an image to white (usually 255), simulating the highest brightness interfering pixels. For each pixel point (x, y) in the image, the probability $p = NR$ is set to 255, keeping the original pixel value with the probability of $1 - p$.
2. Gaussian Noise: The amplitude of Gaussian noise follows normal distribution. The larger the variance, the stronger the noise and the blurrier or more distorted the image. The adversary completes the attacks by adding to each pixel in the image a normally distributed random variable n with $n \sim N(0, \sigma^2)$, where $\sigma^2 = NR$. The new pixel point will be:

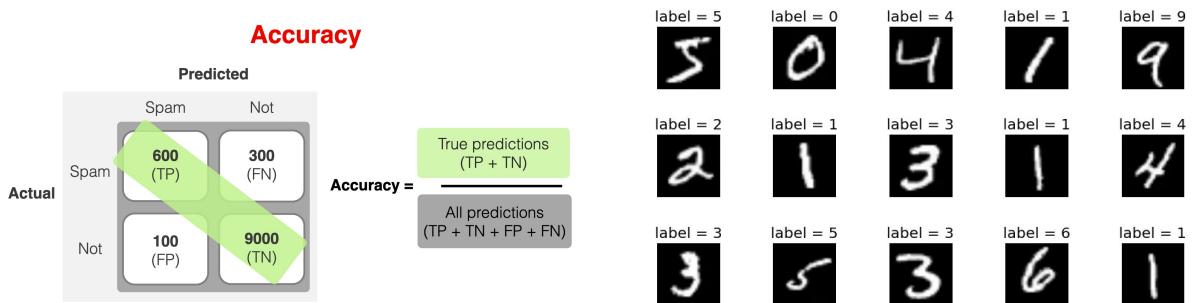
$$I'(x, y) = I(x, y) + n \quad (4.3)$$

3. Salt & Pepper Noise: Salt and pepper noise while randomly selecting pixels in the image and setting their values to the lowest (0) or highest (255):

$$I'(x, y) = \begin{cases} 0 & p = \frac{NR}{2} \\ 255 & p = \frac{NR}{2} \\ I(x, y) & \text{otherwise} \end{cases} \quad (4.4)$$

- **Accuracy** is a measure of the performance of a model, indicating the proportion of true results (both True Positives (TP) and True Negatives (TN)) among the total number of cases tested, as shown in Figure 4.1a:

Figure 4.1: Classification Accuracy with MNIST Digit Samples



(a) Understanding Accuracy in Binary Classification [62].

(b) (Multi-classes) Image Classification on MNIST Dataset [63].

Specifically, in the case of multi-class classification such as image classification tasks (Figure 4.1b) under poisoning attacks, accuracy indicates the proportion of labels that the model predicts correctly, that is, the number of images the model are classified correctly out of the total number of images. The formula for calculating accuracy is:

$$Accuracy(\%) = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \quad (4.5)$$

In scenarios involving data poisoning, accuracy is particularly important because it helps evaluate whether the attack has successfully degraded the model's performance. Therefore, on the other hand, the effectiveness of the defense mechanism can be reasonably examined by observing the change in the accuracy rate of the unattacked clients.

4.2 Attack Specification

The research background of this paper is based on data with non-IID distributions, which makes the data more susceptible to attacks. Therefore, the mechanism proposed in this paper are specifically designed to defend against data poisoning. Two types of poisoning attacks are chosen to analyze and compare current state of research and the novel approach.

Untargeted Label Flipping. Targeted label-flipping attacks are mentioned in Section 2.2.1 that several instances of a label is replaced by another certain label. In the context of this work, poisoning attacks are introduced by simulating untargeted label-flipping, where adversaries change the label of any data to a random label. Both original and new labels are in the set of all labels known to the attackers [38]. The number of flipped labels can be represented by $PSR \in [0, 100]$, where 0 indicates no labels are flipped, and 100 indicates all labels are attacked. In a distributed network, the number of compromised nodes is represented by the PNR, also ranging from 0 to 100. By combining these two metrics, poisoning attack is introduced and the impact of the untargeted label flipping attacks on benign clients is evaluated using accuracy.

Untargeted Sample Poisoning. Another type of data poisoning attacks is poisoning samples attacks, under the taxonomy in Section 2.2.1. Untargeted Sample Poisoning are implemented mainly by randomly adding different types of noise to a certain percentage of samples in the dataset. The term "untargeted" here means that the attack does not target specific categories or labels, but randomly selects data samples to add noise to. $PSR \in [0, 100]$ in untargeted sample poisoning indicates the percentage of data samples to be tampered with, and $NR \in [0, 100]$ is used to show the proportion and degree of noise addition.

4.3 Defense Design

The main structure of CFL consists of a server and multiple clients, where the server receives updates from participants in each round, which can be classified into benign updates and malicious updates. However, DFL does not have a fixed centralized server. Instead, updates are shared among distributed nodes which can both send and receive updates, making the system potentially more vulnerable to various security threats by malicious updates. The research background of this work is based on a unbalanced distributed

setting, that is to distribute the dataset in a non-IID manner across multiple clients. Data is predominantly non-iid distributed in reality, and with the dynamic and diverse network structures characteristic of DFL, traditional defense methods that rely on data distribution are often inadequate. This highlights the necessity for a mechanism that is independent of data distribution. Information theory provides a robust framework to develop such a mechanism. By leveraging intrinsic metrics derived from information theory, it is possible to devise a defense strategy that remains effective regardless of the underlying data distribution. This approach allows for a more adaptable and resilient defense mechanism in the face of the complex and variable environments typical of distributed networks. In the following sections, a defense mechanism called AIDFL(Anomalous Information Metrics-based Detection in DFL) will be proposed that adopts the aforementioned properties of DFL. [38]

Table 4.1: Configuration overview for the two selected attacks. NR is not applicable to the label flipping attacks.

Attack Type	Parameter			Metric
	PNR [%]	PSR [%]	NR [%]	
Untargeted Label Flipping	[0, 100]	[0, 100]	-	Accuracy
Untargeted Sample Poisoning	[0, 100]	[0, 100]	[0, 100]	Accuracy

4.3.1 Pre-Design

The design begins by grouping the data according to class, then randomly shuffling the indices within each class, and evenly distributing these indices to each client. This allocation strategy ensures that each client may receive samples that are highly concentrated in one or a few categories, rather than a representative sample of the entire dataset. Finally, a dataloader is created for each client’s subset of indices, which is used for loading data during model training. This non-IID distribution of data simulates real-world scenarios where different devices or users might only have access to a subset of the data, helping to study and understand how models train and generalize under such conditions.

An experimental evaluation is conducted prior to the introduction of the network structure of DFL with the aim of demonstrating that the entropy of the poisoning model is higher than that of benign. As shown in the Figure 4.2, the entropy of the model parameters increases as the PSR grows, indicating that the data is getting more disordered. It also side-steps the fact that attacks will disrupt the intrinsic information of the model.

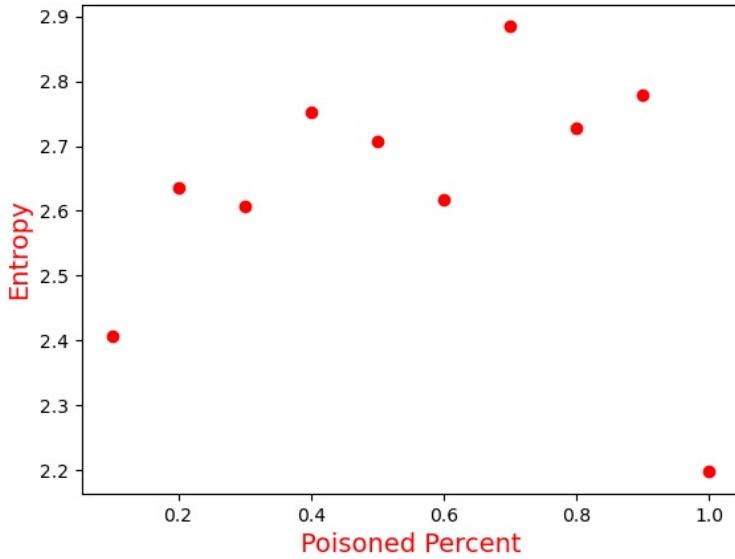


Figure 4.2: Relationship between entropy and PSR.

4.3.2 AIDFL

AIDFL is a defense mechanism that can adapt to the dynamic connectivity network of DFL. It does not require a central server to receive model updates, and it utilizes information theory metrics, which do not depend on data distribution, thus it will not be affected by non-IID data. Each client, integrating their local data, employs K -means clustering and information theoretic metrics to obtain as much shared information as possible from its neighbors and discard irrelevant information [59]. Algorithm 1 defines three-phase anomaly detection approach consisting of (1) K -means clustering, (2) Mutual information / Conditional Entropy Estimation, (3) Anomaly detection. The following paragraphs will provide detailed explanations for each step. Figure 4.3 indicates an overview of the detection process in AIDFL.

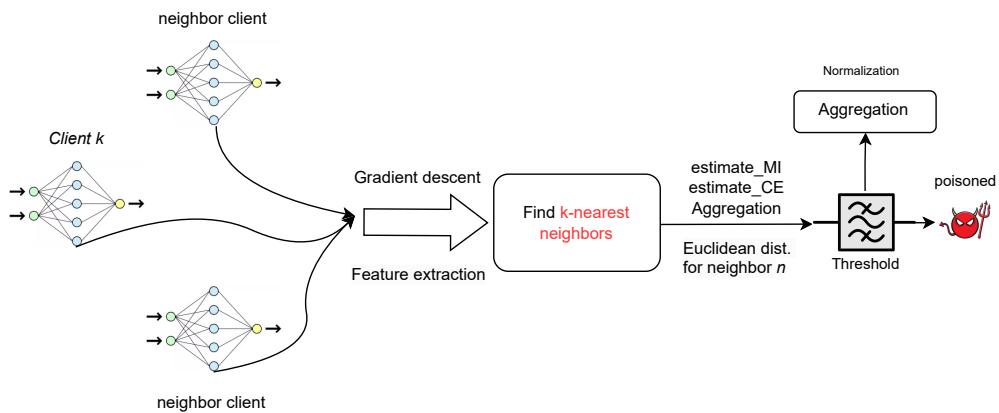


Figure 4.3: High-level overview of the detection process in AIDFL.

(1) K-means Clustering. Each client independently trains their local model on their dataset without sharing any raw data, conducts gradient descent during training, and extracts feature representations from local data using the intermediate layers of the models. Then it comes to the first step of AIDFL. As shown in Algorithm 2, K -means clustering is initialized with a specified number of clusters K . For each sample x in client k , the algorithm tries to find cluster centers that minimize the variance between the data points and their respective cluster centers.

Algorithm 1 AIDFL Aggregation Algorithm

```

1: Input: Initialized model  $w_0^{(k)}$ 
2: Output: Aggregated weights  $w_t^{(k)}$ 
3: for  $t = 0$  to  $T - 1$  do
4:   for each client  $k$  in parallel do
5:      $w_t^{(k)} = w_{t-1}^{(k)} - \eta \nabla L(w_{t-1}^{(k)}, D^{(k)})$             $\triangleright$  Gradient descent
6:      $F^{(k)} = w_t^{(k)} * D^{(k)}$                                           $\triangleright$  Feature extraction
7:     for each sample  $x$  in  $D^{(k)}$  do
8:        $KmeansClustering(x, K)$                                       $\triangleright$  (1) K-means Clustering
9:        $MutualInformationEstimation(x_{np}, A)$                        $\triangleright$  (2.1) MI Estimation
10:       $ConditionalEntropyEstimation(x_{np}, A)$                        $\triangleright$  (2.2) CE Estimation
11:    end for
12:     $MI(k) = \frac{1}{|D^{(k)}|} \sum_{x \in D^{(k)}} MI_x^{(k)}$            $\triangleright$  Aggregate estimations for client  $k$ 
13:     $CE(k) = \frac{1}{|D^{(k)}|} \sum_{x \in D^{(k)}} CE_x^{(k)}$ 
14:
15:    for each sample  $x$  in  $D^{(k)}$  do
16:       $AnomalyDetection(I(k; A), H(k|A), \gamma)$                        $\triangleright$  (3) Anomaly Detection
17:       $agg\_weights += w_t^{(k)} * w_x$                                  $\triangleright$  Aggregation
18:       $weights\_sum += w_x$ 
19:
20:    end for
21:  end for
22:
23:  if  $weights\_sum > 0$  then                                          $\triangleright$  Normalization
24:    for  $i = 1$  to  $\text{length}(agg\_weights)$  do
25:       $agg\_weights[i] = agg\_weights / weights\_sum$ 
26:    end for
27:  end if
28:
29:  for each client  $k$  do                                          $\triangleright$  Update local models
30:     $w_t^{(k)} = agg\_weights$ 
31:  end for
32: end for

```

Algorithm 2 *K*-means Clustering

Require: Sample data x in $D^{(k)}$, K (number of clusters)

Ensure: cluster_labels assignments A

- 1: Feature data $x_{np} \leftarrow$ detach x and convert to NumPy array
 - 2: Initialize KMeans with $n_clusters = K$
 - 3: $\text{cluster_labels} \leftarrow \text{KMeans.fit_predict}(x_{np})$
-

Finally, each data point is assigned a cluster label which is returned and stored in cluster labels assignments A , based on the nearest cluster center. In summary, each data point (or feature vector) is assigned to a cluster. The result of clustering is the cluster label to which each data point belongs. The first step can also be regarded as a type of similarity evaluation, in order to be aware of potential malicious updates. Anomalous models (which may be manipulated or attacked) might form separate clusters or outlier points far from the main clusters.

(2) Mutual Information/Conditional Entropy Estimation. It then estimates mutual information by analyzing the distributional differences between each sample in a cluster. Simultaneously, estimating conditional entropy based on the uncertainty in the conditional distribution derived from the K -means clustering results.

After clustering, inspired by VFedAD [59], simultaneously maximizing mutual information and minimizing conditional entropy can further help identify anomalous models. Mutual information measures dependencies between different features or clusters, while conditional entropy shows the randomness or unpredictability within a cluster given certain data points.

This set of pseudocode is used for estimating the von Neumann entropy, mutual information, and conditional entropy of feature data, based on the cluster labels determined by the previous algorithm. The calculation of von Neumann entropy involves constructing the *log* of the Gram matrix, multiplying it back with the original matrix, and obtaining the entropy value by taking the trace.

The mutual information is then calculated by deriving the joint matrix of two matrices, A and B , and computing the entropy for each of them and the joint matrix, with mutual information being the sum of the individual entropy $H(A) + H(B)$ minus the joint entropy $H(A, B)$, formula has been mentioned in Section 2.3.2. If the result is not a number (NaN), the mutual information is set to zero.

Finally, the conditional entropy is calculated via mutual information by assessing the mutual information between the feature matrix and its cluster labels $I(F; A)$, then subtracting this value from the entropy of the feature matrix $H(F)$.

Algorithm 3 Von Neumann Entropy

Require: gram_matrix
Ensure: Entropy H

- 1: $\epsilon \leftarrow 1e-10$
- 2: $\log_matrix \leftarrow \log_2(\text{gram_matrix} + \epsilon)$
- 3: $H = -\text{trace}(\log_matrix * (\text{gram_matrix} + \epsilon))$

Algorithm 4 Mutual Information

Require: Matrices A and B
Ensure: Mutual Information $I(A; B)$

- 1: Joint_Matrix $\leftarrow A^T / 10 \cdot B^T$
- 2: $H(A) \leftarrow \text{VonNeumannEntropy}(A)$
- 3: $H(B) \leftarrow \text{VonNeumannEntropy}(B)$
- 4: $H(A, B) \leftarrow \text{VonNeumannEntropy}(\text{Joint_Matrix})$
- 5: $I(A; B) \leftarrow H(A) + H(B) - H(A, B)$
- 6: **if** $I(A; B)$ is NaN **then**
- 7: $I(A; B) \leftarrow 0$ ▷ Handle NaN values
- 8: **end if**
- 9: **return** $I(A, B)$

Algorithm 5 Mutual Information Estimation (for feature data)

Require: Feature data x_{np} , $KmeansClustering(x_{np}) : A$
Ensure: Mutual Information $I(k, A)$

- 1: $A_{tensor} \leftarrow \text{Unsqueeze}(\text{Tensor}(A), 1)$ ▷ Add a dimension
- 2: $I(F; A) \leftarrow \text{MutualInformation}(A_{tensor}, F^T)$
- 3: **return** $I(F; A)$

Algorithm 6 Conditional Entropy Estimation

Require: Feature matrix F , Cluster assignments $KmeansClustering(x_{np}) : A$
Ensure: Conditional Entropy $H(F|A)$

- 1: $H(F) \leftarrow \text{VonNeumannEntropy}(F)$
- 2: Reshape assignments A to have the same shape as the feature set F
- 3: $A \leftarrow \text{unsqueeze}(\text{torch.tensor}(A), 1)$
- 4: $I(F; A) \leftarrow \text{MutualInformation}(A.\text{float}(), F.\text{t}())$
- 5: $H(F|A) \leftarrow H(F) - I(F; A)$
- 6: **return** $H(F|A)$

For each cluster, the conditional entropy of the model features within it can be calculated. If entropy of a cluster is unusually higher than others, this might indicate a high degree of inconsistency in the model features within the cluster, which is possibly due to data manipulation. By analyzing these metrics, abnormal models who deviate from the norm can be detected, which may indicate that the data has been tampered with or attacked.

(3) Anomaly Detection. At the final stage of AIDFL, The anomaly detection algorithm operates by first calculating the mean (μ) and standard deviation (σ) of the data tensor, which have contained computed metrics such as mutual information and conditional entropy. These metrics provide insights into the statistical relationships and variability within the data. The algorithm then establishes lower and upper thresholds for anomaly detection, which are defined as $[\mu - \gamma\sigma, \mu + \gamma\sigma]$ respectively, where γ is a predefined threshold multiplier. By evaluating the Euclidean distance between each data point and the mean μ , the algorithm identifies anomalies as data points that fall outside of the thresholds.

Algorithm 7 Anomaly Detection

Require: Data_tensor (mutual information $I(k; A)$, conditional entropy $H(k|A)$), threshold multiplier γ

- 1: $\mu \leftarrow \text{mean}(\text{data_tensor})$
- 2: $\sigma \leftarrow \text{std}(\text{data_tensor})$
- 3: $\text{lower_threshold} \leftarrow \mu - \gamma\sigma$
- 4: $\text{upper_threshold} \leftarrow \mu + \gamma\sigma$
- 5: $\text{dist} \leftarrow \|\text{items in data_tensor} - \text{mean}\|$
- 6: $\text{anomalies} \leftarrow \{x | \text{dist not in } [\text{lower_threshold}, \text{upper_threshold}]\}$
- 7: Return $D^{(k)} \setminus \{\text{anomalies}\}$

Finally, the algorithm returns the subset of the original dataset excluding these anomalies, effectively isolating and removing potential malicious data points to ensure data quality and consistency for further aggregation.

After the last step that anomalies have been detected and eliminated, weights of the benign clients are aggregated to the global model. Furthermore, normalization is necessary if sum of the weights is more than zero. Specifically, during the aggregation process weights from each clients k are collected and stored in a queue and overall average of the weight sums (total weight sums from all training rounds divided by the number of items) are calculated. Next, it retrieves all current weight items from the aggregated weights queue and compute the average of these weights. Then AIDFL normalizes the weights by dividing each weight layer by the average weight sum only when their sum is positive. Finally, each client updates the local model weights to the new aggregated weights. This method helps maintain stability and consistency in the global model across multiple training rounds, preventing any single data from disproportionately influencing the overall learning process.

The detailed computation of AIDFL will be exhibited in the following algorithm.

Algorithm 8 Proposed AIDFL Algorithm (Detailed)

```

1: Input: Initialized model  $w_0^{(k)}$ 
2: Output: Aggregated weights  $w_t^{(k)}$ 
3: for  $t = 0$  to  $T - 1$  do
4:   for each client  $k$  in parallel do
5:      $w_t^{(k)} = w_{t-1}^{(k)} - \eta \nabla L(w_{t-1}^{(k)}, D^{(k)})$            ▷ Gradient descent
6:      $F^{(k)} = w_t^{(k)} * D^{(k)}$                                          ▷ Feature extraction
7:     for each sample  $x$  in  $D^{(k)}$  do
8:       cluster_labels  $\leftarrow$  K-means Clustering
9:        $MI_x^{(k)} = \text{estimate\_MI}(N_k(x), F^{(k)})$                       ▷ estimate MI and CE
10:       $CE_x^{(k)} = \text{estimate\_CE}(N_k(x), F^{(k)})$ 
11:    end for
12:     $MI(k) = \frac{1}{|D^{(k)}|} \sum_{x \in D^{(k)}} MI_x^{(k)}$                   ▷ Aggregate estimations for client  $k$ 
13:     $CE(k) = \frac{1}{|D^{(k)}|} \sum_{x \in D^{(k)}} CE_x^{(k)}$ 
14:
15:     $L = \alpha MI(k) - \beta CE(k)$           ▷ Define and compute optimization objective
16:     $w_t^{(k)} \leftarrow w_t^{(k)} + \Delta L$         ▷ Adjust weights based on optimization objective
17:
18:    for each sample  $x$  in  $D^{(k)}$  do                                ▷ Anomaly Detection
19:       $MI\_dist_x^{(k)} = \|(MI_x^{(k)} - MI_n)\|$           ▷ Calculate Euclidean distances
20:       $CE\_dist_x^{(k)} = \|(CE_x^{(k)} - CE_n)\|$ 
21:      if  $(MI\_dist_x^{(k)} > MI\_th(k))$  or  $(CE\_dist_x^{(k)} < CE\_th(k))$  then
22:         $x$  is an anomaly
23:      else
24:        agg_weights +=  $w_t^{(k)} * w_x$ 
25:        weights_sum +=  $w_x$ 
26:      end if
27:    end for
28:  end for
29:
30:  if weights_sum > 0 then
31:    for  $i = 1$  to length(agg_weights) do
32:      agg_weights[i] = agg_weights / weights_sum
33:    end for
34:  end if
35:
36:  for each client  $k$  do          ▷ Update local models with the aggregated weights
37:     $w_t^{(k)} = \text{agg\_weights}$ 
38:  end for
39: end for

```

Chapter 5

Implementation

The definition of AIDFL given in Algorithm 1 is implemented in Python. The main functions involved in the anomaly detection and aggregation processes are listed below: `cluster_data()`, `mutual_information()`, `get_conditional_entropy()`, implement steps (1), (2.1), (2.2), and and `detect_outliers()` implements (3) of the algorithm, respectively. The remaining steps of AIDFL, such as initial feature extraction, the aggregation and normalization after excluding anomalies, are also implemented in the code below.

```
1 # Feature Extraction
2 def get_weight_x_data(client,batched_data):
3     raw_data=batched_data.view(-1, 784)
4     for name, param in client.model.named_parameters():
5         if 'bias' not in name:
6             raw_data=raw_data @ param.t()
7
8     mean = raw_data.mean(dim=0, keepdim=True)
9     std = raw_data.std(dim=0, keepdim=True)
10
11    # normalization
12    raw_normalized = (raw_data - mean) / (std + 1e-10)
13
14    return raw_normalized
15
16 # 1. K-means Clustering
17 def cluster_data(feature_data,raw_data):
18     kmeans = KMeans(n_clusters=cluster_sum)
19     assignments = kmeans.fit_predict(feature_data.detach().numpy())
20     return assignments
```

```

1 def get_gram_matrix(matrix: torch.Tensor):
2     matrix = matrix/matrix.shape[0]
3     gram_matrix = torch.mm(matrix, matrix.t())
4     gram_matrix = torch.pow(gram_matrix,2)
5     return gram_matrix
6
7 def von_neumann_entropy(matrix: torch.Tensor):
8     epsilon = 1e-10
9     gram_matrix = get_gram_matrix(matrix)+epsilon
10
11    new_p = gram_matrix * torch.log2(gram_matrix)
12    sp = torch.trace(new_p)
13    return -sp
14
15 # 2.1 Mutual Information Estimation
16 def get_mutual_information(feature_data, assignments):
17     assignments=torch.tensor(assignments)
18     assignments=torch.unsqueeze(assignments,1)
19     return mutual_information(assignments.float(), feature_data.t())
20
21 def mutual_information(matrixA: torch.Tensor, matrixB: torch.Tensor):
22     :
23     # I(A,B) = H(A) + H(B) - H(A, B)
24     joint_matrix = torch.mm(matrixA.t()/10, matrixB.t())
25     ha = von_neumann_entropy(matrixA)
26     hb = von_neumann_entropy(matrixB)
27     hab = von_neumann_entropy(joint_matrix)
28     im = ha + hb - hab
29     if torch.isnan(im):
30         return torch.tensor(0)
31     return im
32
33 # 2.2 Conditional Entropy Estimation
34 def get_conditional_entropy(feature_data, assignments):
35     H_F = von_neumann_entropy(feature_data)
36     assignments = torch.tensor(assignments)
37     assignments = torch.unsqueeze(assignments, 1)
38
39     I_FA = mutual_information(assignments.float(), feature_data.t())
40
41     H_F_given_A = H_F - I_FA
42
43     return H_F_given_A

```

```
1 # 3. Anomaly Detection
2 def get_threshold_mi_or_ce(data_tensor, threshold=3):
3
4     mean = torch.mean(torch.stack(data_tensor))
5     std = torch.std(torch.stack(data_tensor))
6
7     lower_threshold = mean - threshold * std
8     upper_threshold = mean + threshold * std
9
10    return lower_threshold,upper_threshold
11
12 def detect_outliers(data_tensor, lower_threshold, upper_threshold):
13     data_tensor = torch.stack(data_tensor)
14     outliers = torch.where((data_tensor < lower_threshold) | (
15         data_tensor > upper_threshold))[0]
16
17     return outliers
18
19 def get_normal_samples(full_length,mi_anomaly,ce_anomaly):
20
21     mi=[mi.numpy() for mi in mi_anomaly]
22     ce=[ce.numpy() for ce in ce_anomaly]
23     mi_or_ce=set(mi).union(set(ce))
24     full_set = np.arange(0, full_length)
25     normal_samples=set(full_set)-mi_or_ce
26
27     return normal_samples
```

```

1 # Aggregation and Normalization
2 def get_sample_weight_x_model_weight(sample_weight,normal_samples_id
,client):
3
4     agg_weights_temp=[]
5     normal_samples_number= len(normal_samples_id)
6     for name, param in client.model.named_parameters():
7         if 'bias' not in name:
8             agg_param_weight=torch.zeros_like(param)
9
10            for id in list(normal_samples_id):
11                temp=sample_weight[id]*param
12                agg_param_weight+=temp
13            agg_weights_temp.append(agg_param_weight)
14
15    return agg_weights_temp
16
17 def get_acclumated_sample_weight(sample_weight,normal_samples_id):
18     selected_elements = [sample_weight[i] for i in normal_samples_id
19     ]
20     sum_result = torch.sum(torch.stack(selected_elements))
21     return sum_result
22
23 def append_client_agg_weights(ID,data):
24     selected_queue=eval(f"client_agg_weights_queue_0{ID}")
25     selected_queue.put(data)
26
27 def append_client_weights_sum(ID,data):
28
29     selected_queue = eval(f"client_weights_sum_queue_0{ID}")
30     selected_queue.put(data)
31
32 def get_average_weight_from_queue(current_items):
33     if len(current_items)>0:
34         param_temp_list=[]
35         param_temp=[]
36         flattened_c = [item for sublist in current_items for item in
37                         sublist]
38         for item in current_items[0]:
39             temp=[tensor for tensor in flattened_c if tensor.shape
40                   == item.shape]
41             param_temp_list.append(temp)
42             for i in range(len(param_temp_list)):
43                 temp =torch.mean(torch.stack(param_temp_list[i]),dim=0)
44                 param_temp.append(temp)
45             return param_temp
46
47     return []

```

```

1 # Adjust the weights
2 def adjust_weight(ID):
3     client_weights_sum_queue = eval(f"client_weights_sum_queue_0{ID}")
4     client_agg_weights_queue = eval(f"client_agg_weights_queue_0{ID}")
5     average_weights_sum=torch.sum(torch.stack(
6         client_weights_sum_queue.current_items()))/len(
7         client_weights_sum_queue)
8
9     param_=[]
10    if average_weights_sum > 0:
11        all_item=client_agg_weights_queue.current_items()
12        param_temp=get_average_weight_from_queue(all_item)
13        if len(param_temp) > 0:
14            for item in param_temp:
15                temp=item / average_weights_sum
16                param_.append(temp)
17    return param_
18
19 # Update the weights
20 def update_weight(ID, adjusted_param):
21
22     model=clients[ID].model
23     i=0
24     with torch.no_grad():
25         for name, param in model.named_parameters():
26             if 'bias' not in name:
27                 param.data = param.data+adjusted_param[i] *
28                     adjust_coef
29     i+=1
30
31     return None

```

Chapter 6

Evaluation

In this chapter, the AIDFL defense mechanism is evaluated, and a comparison is made with other state-of-the-art defense methods. For this purpose, the aggregation phase of AIDFL is replaced with various existing aggregation algorithms to assess their impact on the effectiveness of anomaly detection. The subsequent sections state the details of the experimental setup, present the results, and discuss the findings and observations derived from these experiments.

6.1 Experiment Setup

Three datasets are employed to evaluate the performance of AIDFL: MNIST, FashionMNIST, and CIFAR10. This section will introduce each dataset, the corresponding models selected, and the reference algorithms combined with AIDFL for evaluation. Additionally, to explore the effects of poisoning attacks on the baseline models, the threat model and associated metrics will be outlined later in this section.

6.1.1 Datasets and Models

The evaluation of image classification is conducted using three image datasets: MNIST, FashionMNIST, and CIFAR10. Each dataset will be introduced along with the corresponding deep learning models tailored for the characteristics of each dataset. It is important to note that the distribution of all three datasets is considered non-IID. Details regarding the methodology for splitting the data into non-IID distributions are elaborated in Section 4.3.1.

MNIST [64] which is short for "Modified National Institute of Standards and Technology" dataset, represents a large collection of handwritten digits commonly used for training image classification tasks. MNIST consists of 60,000 images for training and 10,000 test

images of handwritten digits from 0 to 9. Each image is a 28×28 pixel grayscale representation of a digit, making it simple enough for straightforward image processing tasks yet complex enough to test the efficacy of various algorithms.

The chosen model to learn from the MNIST dataset is a multilayer perceptron (MLP) with a linear input layer of size $28 \times 28 \times 256$, followed by a ReLU activation function. The second layer is a hidden layer, which is also a linear layer transforming the 256 outputs from the first layer to 128 neurons, with another ReLU activation applied. Finally, the output layer is a linear layer that maps these 128 neurons to 10 output neurons corresponding to the 10 categories of the classification task. This architecture is an example of a typical neural network commonly used for basic classification tasks. The optimizer and the learning rate of the model are set to stochastic gradient descent (SGD) and 0.01, respectively. Moreover, cross-entropy is applied as the loss function in the training loop to compute the loss between the predicted outputs and the true labels. This MLP is trained for 15 epochs per round, 10 rounds in total. There are 10 clients in the DFL network.

FashionMNIST [65] was designed as a more challenging replacement for the traditional MNIST dataset by Zalando. FashionMNIST consists of a set of 28×28 grayscale images, each representing an article of clothing from one of 10 categories, including T-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots, as shown in Figure 6.1.

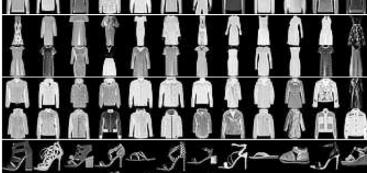
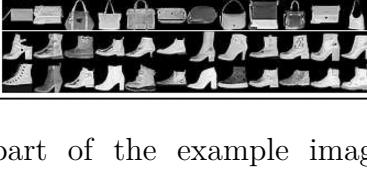
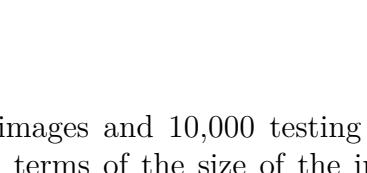
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 6.1: The class names and part of the example images in FashionMNIST dataset. [65]

The dataset contains 60,000 training images and 10,000 testing images. Its structure mimics the original MNIST dataset in terms of the size of the images and the split of

training and testing sets, making it a direct drop-in replacement. For the FashionMNIST dataset, the same MLP model as for MNIST is used, and the model is trained for 15 epochs per round, 10 rounds in total. There are 10 clients in the DFL network.

CIFAR10 [66] is a standard dataset widely used in computer vision research. It contains 60,000 color images of 32×32 pixels, divided into 10 categories. These categories include airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is divided into 50,000 training images and 10,000 test images for training and evaluating the performance of machine learning models. Moreover, due to its moderate size and fast processing speed, CIFAR10 is suitable for validating new algorithms. To reduce computational overhead, the current work temporarily sets the number of clients in the CIFAR10 experiment to 5.

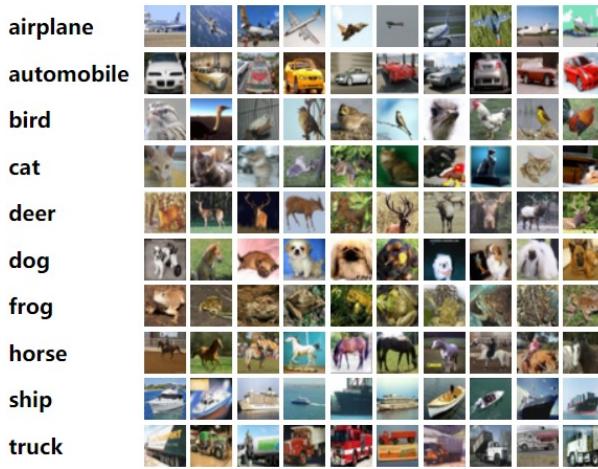


Figure 6.2: The class names and part of the example images in CIFAR10 dataset. [65]

Instead of MLP, a pretrain Resnet18 model is chosen as the model and it has been adapted for CIFAR10 dataset to better suit its characteristics. ResNet18 is part of the Residual Network (ResNet) family proposed by Kaiming He *et al.* [67], and "18" represents 18 layers deep with learned weights.

Originally designed for 224×224 pixel images, the initial convolutional layer of ResNet18 for CIFAR10 has been modified to have a kernel size of 3, stride of 1, and padding of 1, with no bias. This adjustment maintains the spatial dimensions of the input and output, making it suitable for the smaller 32×32 pixel images of CIFAR10. Additionally, the final fully connected layer has been adjusted from 1000 output classes, which corresponds to number of classes in ImageNet, to 10 to match the number of classes in CIFAR10. These modifications ensure that ResNet18 can effectively process smaller images and adapt to datasets with fewer classes. Optimizer is SGD and learning rate is set to $1e-3$.

6.1.2 Selected Reference Approaches

To evaluate the defense techniques in this work, three state-of-the-art defense methods have been selected as reference algorithms: Krum, Median, and TrimmedMean. These

methods gained attention for their robustness when facing outliers and potential malicious attacks. Krum and TrimmedMean are specifically designed to defend against malicious attacks. Krum resists attackers by selecting a single model that is closest in distance to the others, while TrimmedMean reduces the impact of attackers on the aggregation results by removing extreme values. Relatively, Median is particularly effective for extreme updates that may not be malicious but are due to data quality issues. Each methods and configurations of all aggregators will be briefly introduced below.

- Krum (Section 3.1) selects the single model of m local models that is closest to all other models based on Euclidean distance, considering it as the least likely to be manipulated. This method does not require any configuration parameters.
- Median (Section 3.1) aggregator operates by calculating the coordinate-wise median of model parameters. This method performs predominant resistance to outliers in the data because the median is not easily affected by extreme changes in individual values. This approach is especially suitable for mitigating the impact of outliers.
- TrimmedMean (Section 3.1) aggregates by removing the largest and smallest β parameters in each coordinate and calculating the average of the remaining $m - 2\beta$ parameters. β defines the number of elements removed from both ends of each parameter list. In this paper, with $\beta = 1$, the maximum and minimum values of each list are excluded from the average computation, thus reducing the impact of malicious participants on the global model.
- AIDFL (Section 4.3.2) aggregates according to Algorithm 1. For the experiments in this work, AIDFL is configured with $K = 5$ and $\gamma = 3$.

6.1.3 Threat Models

The aggregation defense strategies mentioned earlier in last paragraph will be tested against the attack specifications discussed in Section 4.2 under different parameter configurations to evaluate the performance of these algorithms in poisoning attacks. Two types of data poisoning attacks have been selected for testing: untargeted label-flipping and untargeted sample poisoning attacks. Technical details can be reviewed in Section 4.2.

In the untargeted label-flipping attack, for a selected subset of the training data, the attacker randomly changes the original label l to a different label l' . For MNIST, both l and l' are within the set 0, 1, 2, ..., 9. During the attack, any label can be randomly replaced with a digit, excluding itself. For the FashionMNIST dataset, the labels belong to the category set T-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, ankle boots. For example, adversaries might change a category like “T-shirts/tops” randomly to “trousers” or one of the other categories. Similarly, for CIFAR10, labels from the set airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, trucks can be randomly changed to another category within the set.

In the context of data poisoning attacks, the attackers select a subset of data samples and add noise to them. Several types of noise are introduced in Section 4.1. This paper chooses salt noise for testing because salt noise randomly introduces sharp white pixels into images. This type of noise is very apparent and can significantly interfere with machine learning models used for accurate predictions. Additionally, salt noise is harder to filter out through preprocessing steps such as smoothing, whereas the effects of uniformly distributed noise are easier to mitigate.

In this study, PNR settings reflect different intensities of data poisoning attacks, used to evaluate the impact on model performance. PNR is set at 0, 10%, 30%, 50%, and 70%, covering levels from low to high attacking environment, where PNR = 0 represents the benign case without any attacks. Additionally, attackers also executed intensity attacks, with the PSR set at 0 and 100%. This means that in the attack scenario, either no samples are affected (PSR = 0) or all selected samples are subjected to noise addition (PSR=100%). For untargeted sample poisoning attacks, a 100% salt noise is chosen because lower noise ratio are insufficient to significantly impact model performance.

Table 6.1: Configuration overview for the two selected attacks. NR is not applicable to the label flipping attacks.

Attack Type	Parameter			Metric
	PNR [%]	PSR [%]	NR [%]	
Untargeted Label Flipping	(0, 10, 30, 50, 70)	100	-	Accuracy
Untargeted Sample Poisoning	(0, 10, 30, 50, 70)	100	100	Accuracy

6.2 Results

The following sections will provide a performance evaluation of AIDFL compared to a modified version of AIDFL that retains its anomaly detection capabilities but incorporates different aggregation protocols. The metrics are evaluated on the test datasets. First, the baseline performance of each method is established on three datasets under benign settings (i.e., without any poisoning attacks). Subsequently, the performance of the models under different parameter settings for each attack is discussed.

6.2.1 Baseline Performance

The baseline performance serves as a reference for experiments introduced later in the text that involve poisoning attacks. This baseline establishes the expected performance metrics without the influence of adversarial manipulations, providing a further direction for point to evaluate the impact and effectiveness of the poisoning strategies tested.

Table 6.2 summarizes the performance of the selected aggregation algorithms in an environment without poisoning attacks. The mean accuracy was calculated based on the average metrics from participants after the final round of training. Specifically, for MNIST and

FashionMNIST, the calculations included 10 participants, while for CIFAR10, 5 clients were considered, based on the experimental setting. In terms of accuracy, AIDFL + Krum performed best on MNIST, AIDFL itself achieved the highest accuracy on FashionMNIST, and AIDFL + Median performed best on CIFAR10.

The accuracy difference between the lowest and highest algorithms for MNIST is 0.28%, for FashionMNIST it is 0.29, and for CIFAR10 it is 1.42. The accuracy differences for MNIST and FashionMNIST are very small and can be considered negligible, indicating that the choice of aggregation algorithms does not significantly affect model performance on these datasets. This suggests that the aggregation algorithms have similar effects when processing this specific type of data. While the higher difference in CIFAR10 may still be considered small in practical applications, it is more significant compared to MNIST and FashionMNIST. This difference may indeed be related to the complexity of the dataset, or reduce of the number of clients and training rounds. CIFAR10 is typically more complex than MNIST and FashionMNIST. Therefore, different aggregation algorithms may show more performance differences in complex setting.

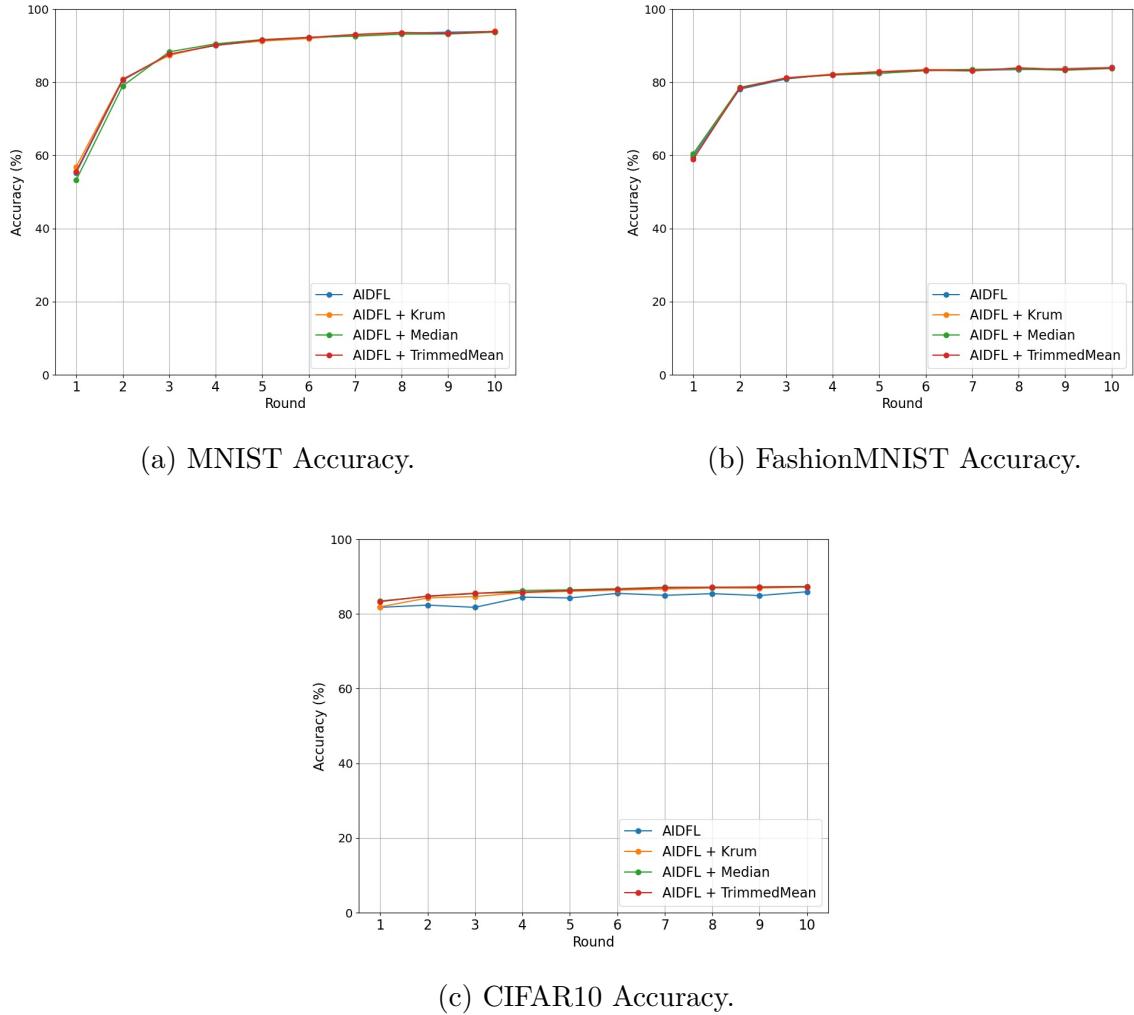


Figure 6.3: Baseline performance for MNIST, FashionMNIST and CIFAR10 for 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

In particular, for the MNIST dataset where AIDFL + Krum achieved the highest accuracy, the SEM was 3.59, which is higher compared to the SEM for CIFAR10 where the lowest accuracies were reported, and the Standard error mean (SEM)s were generally lower. For example, AIDFL + TrimmedMean on CIFAR10 had an SEM of 0.40. This pattern suggests that higher accuracy are associated with higher SEMs across the datasets. Furthermore, Krum shows competitive results especially when combined with AIDFL, as seen in its MNIST performance. For all datasets, the combination of AIDFL with different aggregation methods did not necessarily lead to the lowest accuracies. Instead, these combinations showed varied effects on performance, with no single method consistently underperforming or always performing better than others. This indicates that while task complexity affects performance, the impact of aggregation strategy varies depending on the dataset and the specific characteristics of the task.

Figure 6.3 shows the complete process of DFL network training. For MNIST and FashionMNIST, almost all nodes reached their final accuracy results after the first round. For CIFAR10, the performance was almost stable from the beginning which was close to the final accuracy level. That might because a pretrained model was used, which has already undergone initial learning on a large amount of data, possessing certain generalization capabilities. Thus only minor adjustments are needed when applied to a similar task. In this case, the model performance during benign DFL is relatively stable, and high accuracy can be maintained without too many rounds of training. This demonstrates the advantages of pretrained models in handling more complex datasets, especially in a distributed learning environment.

Table 6.2: Mean Accuracy after round 10.

	MNIST	FashionMNIST	CIFAR10
AIDFL	93.87	84.02	85.93
AIDFL + Krum	93.94	83.96	87.21
AIDFL + Median	93.66	83.73	87.35
AIDFL + TrimmedMean	93.84	83.91	87.25

Table 6.3: Mean Accuracy after round 10 (with SEM).

	MNIST	FashionMNIST	CIFAR10
AIDFL	93.87 ± 3.76	84.02 ± 2.34	85.93 ± 0.50
AIDFL + Krum	93.94 ± 3.59	83.96 ± 2.27	87.21 ± 0.52
AIDFL + Median	93.66 ± 3.97	83.73 ± 2.26	87.35 ± 0.41
AIDFL + TrimmedMean	93.84 ± 3.73	83.91 ± 2.42	87.25 ± 0.40

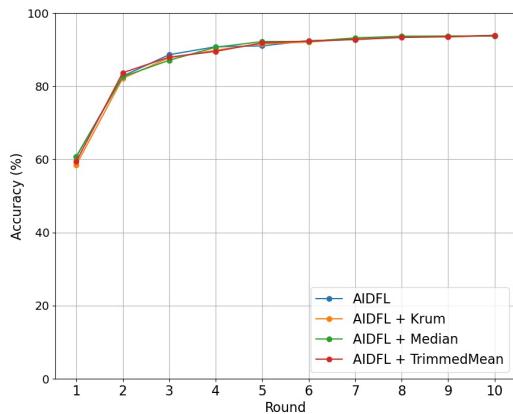
6.2.2 Untargeted Label Flipping

In untargeted label flipping attacks, malicious clients randomly change the labels of samples within the dataset. This type of attack does not target specific classes or outcomes. Instead, it aims to degrade the overall performance of the model. According to the results observed by Feng *et al.* [38], under low PSR conditions, regardless of the PNR configuration, most aggregators are unaffected by adversarial label flipping. In this paper, PSR is set to 100, and PNR configurations are 10, 30, 50, and 70 for MNIST and FashionMNIST, and PNR = 20 and 60 for CIFAR10. Tables 6.4 and Figure 6.4 summarize the effects of untargeted label flipping with a PNR of 10.

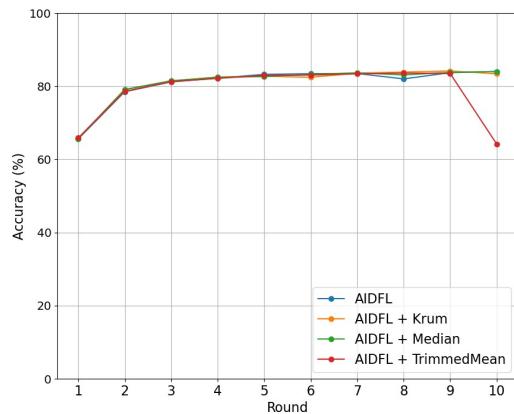
Almost all aggregation strategies were unaffected by the attack from one malicious node, except for the AIDFL and TrimmedMean algorithms. Their accuracy suddenly dropped in the final round, resulting in an average decrease to 64.18%, shown in Figure 6.4.

Table 6.4: Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 10 for MNIST and FashionMNIST and PNR = 20 for CIFAR10, and PSR = 100 for all datasets.

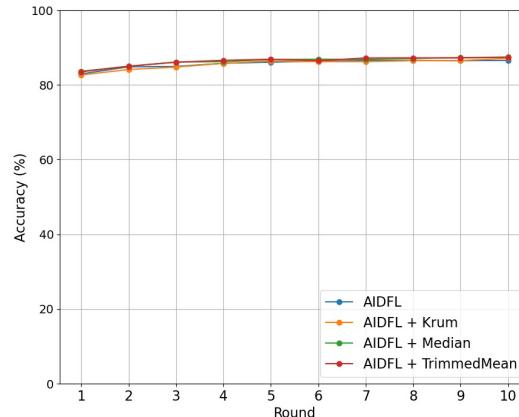
	MNIST	FashionMNIST	CIFAR10
AIDFL	93.76 ± 3.21	84.01 ± 1.76	86.57 ± 0.37
AIDFL + Krum	93.81 ± 3.43	83.37 ± 1.75	87.12 ± 0.44
AIDFL + Median	93.70 ± 3.24	83.97 ± 1.76	87.43 ± 0.37
AIDFL + TrimmedMean	93.91 ± 3.32	64.18 ± 2.36	87.40 ± 0.39



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.



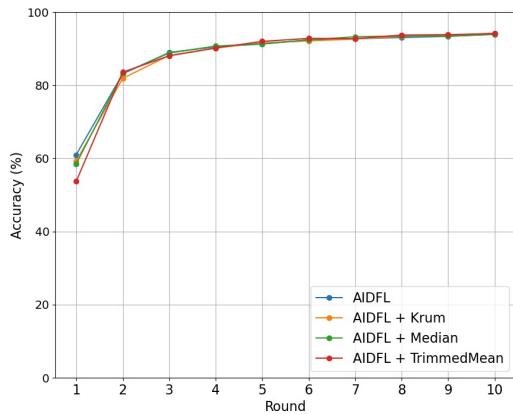
(c) CIFAR10 Accuracy.

Figure 6.4: PNR = 10 performance under untargeted label flipping for MNIST, FashionMNIST and PNR = 20 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

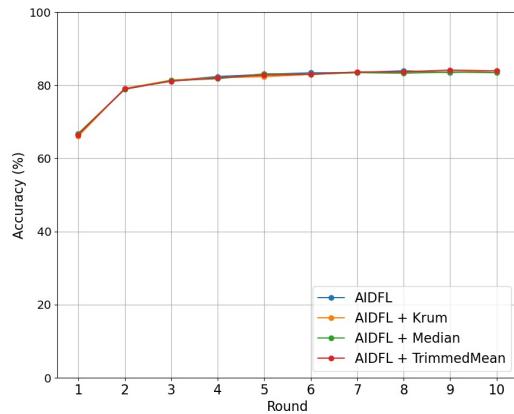
Next, PNR is set to 30 for MNIST and PNR = 60 for CIFAR10, with PSR = 100 unchanged. Overall, all defense mechanisms are robust, and the accuracy of honest nodes is almost unaffected by label flipping attacks. However, as can be seen from the Table 6.5, the combination of AIDFL + Median has the lowest accuracy among the four strategies, possibly due to the median aggregation method, which calculates the coordinate-wise median (Section 3.1) of model updates. This method is intrinsically robust against outliers, provided that these outliers only account for a small portion of entire data. However, as the number of malicious nodes (PNR) increases, particularly when there are three or more, adversarial inputs may become significant enough to directly affect the calculation of the median. Therefore, as the proportion of malicious inputs approaches half of the total inputs, the effectiveness of the aggregator will be undermined, which can explain why performance drops when 3 malicious clients are presented.

Table 6.5: Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 30 for MNIST and FashionMNIST and PNR = 60 for CIFAR10, and PSR = 100 for all datasets.

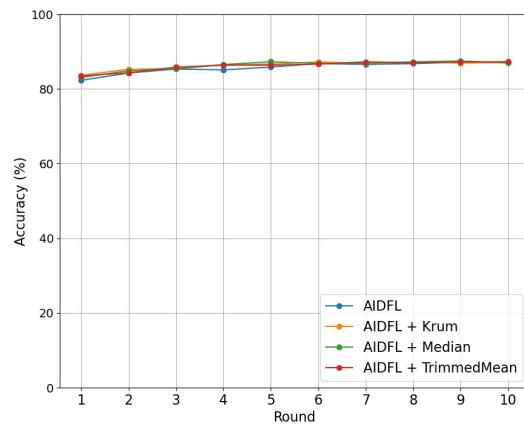
	MNIST	FashionMNIST	CIFAR10
AIDFL	93.93±3.17	83.98±1.68	87.02±0.49
AIDFL + Krum	94.19±3.36	83.92±1.71	87.05±0.36
AIDFL + Median	93.88±3.42	83.41±1.63	86.94±0.44
AIDFL + TrimmedMean	94.12±3.89	83.89±1.68	87.27±0.41



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.



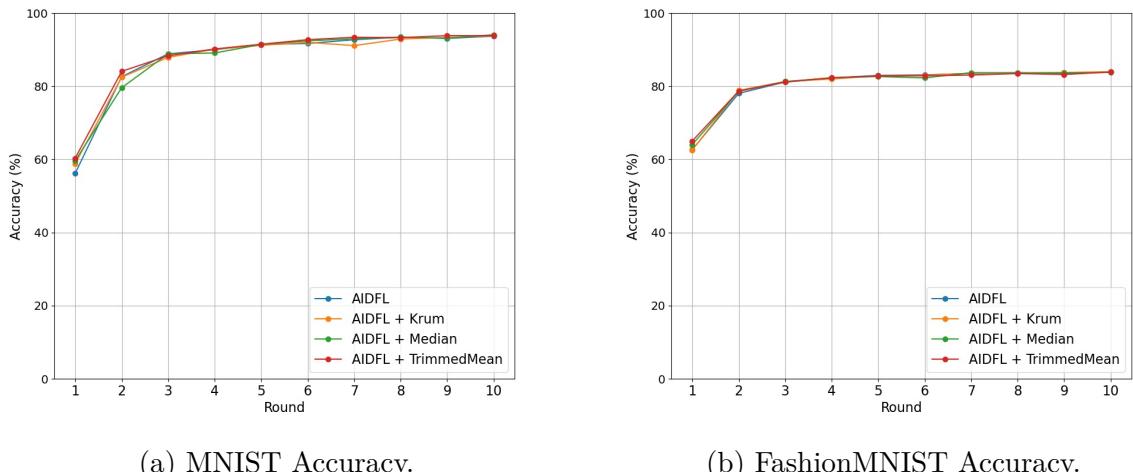
(c) CIFAR10 Accuracy.

Figure 6.5: PNR = 30 performance under untargeted label flipping for MNIST, Fashion-MNIST and PNR = 60 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

For $\text{PNR} = 50$, this work will only investigate the performance on MNIST and FashionMNIST because the initial setup in DFL network on CIFAR10 consists of only 5 clients. Scenarios with 10 clients, including more than 5 poisoned clients, could be considered for future research directions. AIDFL alone and combined with different aggregation strategies (Krum, Median, and TrimmedMean) shows minor variations in accuracy across MNIST and FashionMNIST. The Median strategy slightly performs better than other strategies on MNIST with an accuracy of 94.06%, while the variations on FashionMNIST are minimal, hovering around 83.8%.

Table 6.6: Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST after round 10 when $\text{PNR} = 50$, and $\text{PSR} = 100$ for all datasets.

	MNIST	FashionMNIST
AIDFL	93.68 ± 3.62	83.88 ± 2.06
AIDFL + Krum	93.80 ± 3.34	83.95 ± 2.06
AIDFL + Median	94.06 ± 3.37	83.75 ± 1.91
AIDFL + TrimmedMean	93.79 ± 3.24	83.87 ± 1.80



(a) MNIST Accuracy.

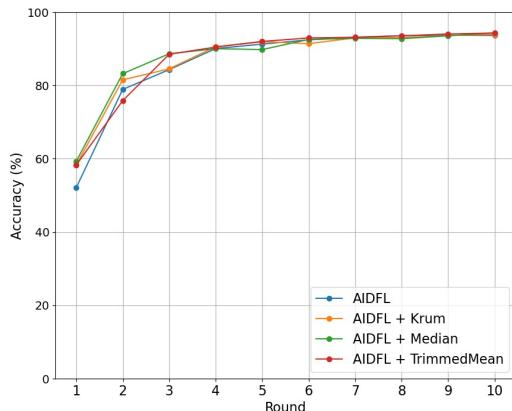
(b) FashionMNIST Accuracy.

Figure 6.6: $\text{PNR} = 50$ performance under untargeted label flipping for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

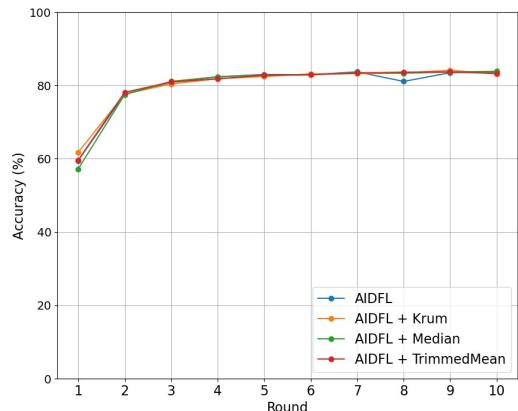
Similar to the case when PNR = 50, PNR = 70 is set only for MNIST and FashionMNIST. Figure 6.7 demonstrates that all four algorithms maintain relatively stable performance throughout all training rounds. This indicates that on the MNIST and FashionMNIST datasets, despite a PNR of 70, various aggregation methods can effectively resist untargeted label flipping attacks and maintain high model accuracy. Among the Table 6.7, AIDFL + Median showed the highest average accuracy on both datasets, reflecting the robustness of Median aggregation strategy when handling data with malicious attacks. This may be because Median can better filter out extreme values, thereby maintaining more stable model performance in highly poisoned environments. The graph also shows a significant drop in accuracy for AIDFL + Median in the 8th round. However, the recovery of accuracy in the final round and reaching the highest performance indicates that Median effectively mitigated the impact of any adversarial data introduced in the early rounds.

Table 6.7: Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST after round 10 when PNR = 70, and PSR = 100 for all datasets.

	MNIST	FashionMNIST
AIDFL	93.63±4.10	83.67±2.33
AIDFL + Krum	93.89±3.46	83.13±2.12
AIDFL + Median	94.33±3.32	83.85±2.58
AIDFL + TrimmedMean	94.30±3.66	83.25±2.33



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.

Figure 6.7: PNR = 70 performance under untargeted label flipping for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

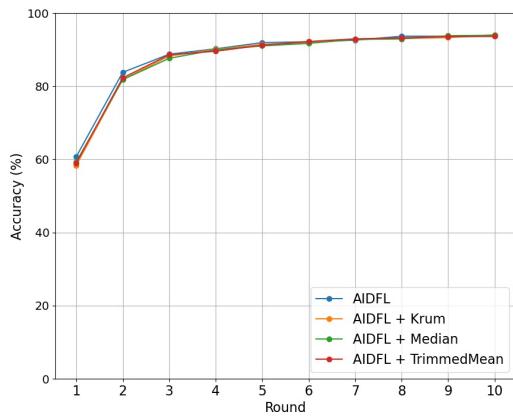
6.2.3 Untargeted Sample Poisoning

In this section, the effectiveness of untargeted sample poisoning will be discussed. In this attack, malicious participants randomly modify parts of features of samples in the dataset by injecting salt noise into the samples. This work evaluates the impact of these modifications across different DFL scenarios. Tables and figures will illustrate the outcomes of these experiments using various combination of aggregation algorithms and attack configurations. The effectiveness of the attacks is measured by the degradation in model performance on the test dataset, where the samples have been manipulated. The Table 6.8 and Figure 6.8 illustrate the accuracy performance of various aggregation algorithms under an untargeted sample poisoning scenario for the MNIST, FashionMNIST, and CIFAR10 datasets after 10 rounds of training, where the PNR is set at 10 for MNIST and FashionMNIST, and 20 for CIFAR10, with all datasets having a PSR of 100, assuming that lower PSR will not influence the model performance [38]. AIDFL + Krum again shows slightly higher performance on MNIST and FashionMNIST, emphasizing the effectiveness of Krum in mitigating the effects of untargeted sample poisoning. AIDFL performs best on CIFAR10, which suggests that AIDFL works better with Diverse and complex data

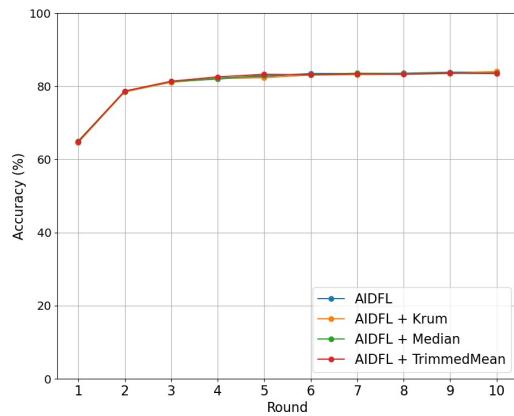
When dealing with data poisoning attacks, especially when the attacks are not extremely significant or large-scale, Krum is able to identify and ignore anomalous updates, thus protecting the global models from these malicious updates. However, if the number of malicious updates is large enough to occupy a significant portion of the models involved in the aggregation, the effectiveness of Krum may be compromised, as the malicious models may no longer appear to be too "anomalous" to be identified and excluded.

Table 6.8: Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 10 for MNIST and FashionMNIST and PNR = 20 for CIFAR10, and PSR = 100 for all datasets.

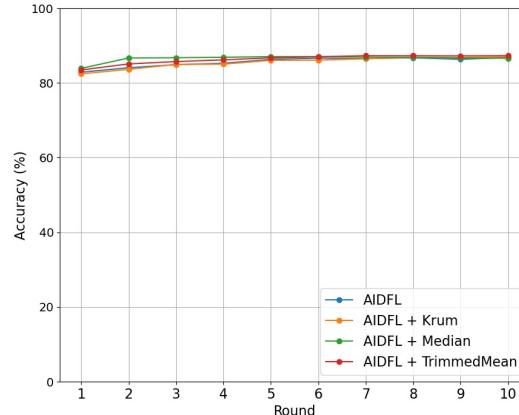
	MNIST	FashionMNIST	CIFAR10
AIDFL	93.88±3.20	83.80±1.86	87.80±0.42
AIDFL + Krum	94.02±3.43	84.01±1.84	86.98±0.48
AIDFL + Median	93.92±3.34	83.53±1.81	86.59±0.30
AIDFL + TrimmedMean	93.59±3.37	83.45±1.83	87.28±0.40



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.



(c) CIFAR10 Accuracy.

Figure 6.8: PNR = 10 performance under untargeted sample poisoning for MNIST, FashionMNIST and PNR = 20 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

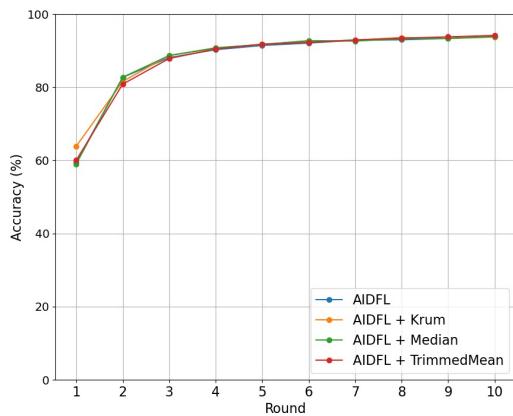
Figure 6.9 and Table 6.9 show the accuracy performance of the different algorithms after 10 rounds under conditions of untargeted sample poisoning (PNR = 30 for MNIST and FashionMNIST, and PNR = 60 for CIFAR10).

It can be inferred that in the CIFAR10 dataset experiments, AIDFL has the highest accuracy when there are only a few malicious nodes (PNR = 20) and the worst performance when the malicious nodes are increased to the majority (PNR = 60). This suggests that the model design of AIDFL on CIFAR10 performs well for situations containing a small number of malicious nodes, as it may have some mechanism to moderate the impact of minor malicious updates, thus maintaining a higher accuracy rate. However, when the number of malicious nodes increases and becomes dominant in the network, AIDFL may not be sufficient to resist such widespread poisoned attacks. In this case, the malicious data may affect most of the model updates during the aggregation process, resulting in that AIDFL becomes unable to effectively distinguish and filter out the anomalous updates, which will ultimately affect the overall model performance and accuracy.

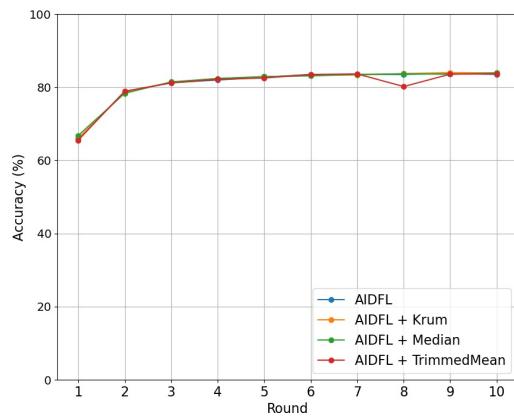
The combination with Median aggregation significantly improves AIDFL to defend against outliers and potential poisoning attacks. The reason that AIDFL + Median performs well in both FashionMNIST and CIFAR10 experiments is that this approach is naturally resistant to extreme values, as the median is less susceptible to a few extreme updates. When malicious updates or outliers appear, they would not become medians. Therefore, quality of the model updates can be guaranteed.

Table 6.9: Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 30 for MNIST and FashionMNIST and PNR = 60 for CIFAR10, and PSR = 100 for all datasets.

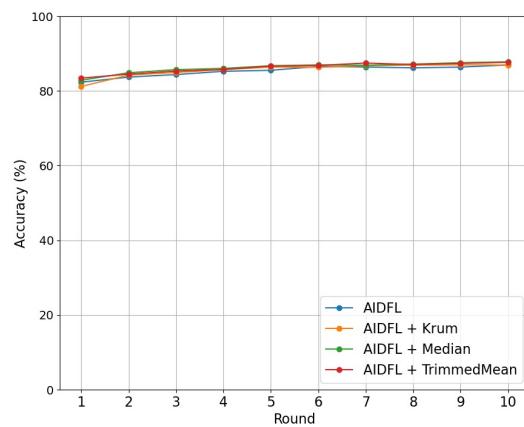
	MNIST	FashionMNIST	CIFAR10
AIDFL	93.90±3.32	83.48±1.73	86.94±0.47
AIDFL + Krum	93.70±2.95	83.88±1.73	86.92±0.57
AIDFL + Median	93.81±3.37	83.89±1.66	87.78±0.47
AIDFL + TrimmedMean	94.15±3.33	83.61±1.74	87.66±0.45



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.



(c) CIFAR10 Accuracy.

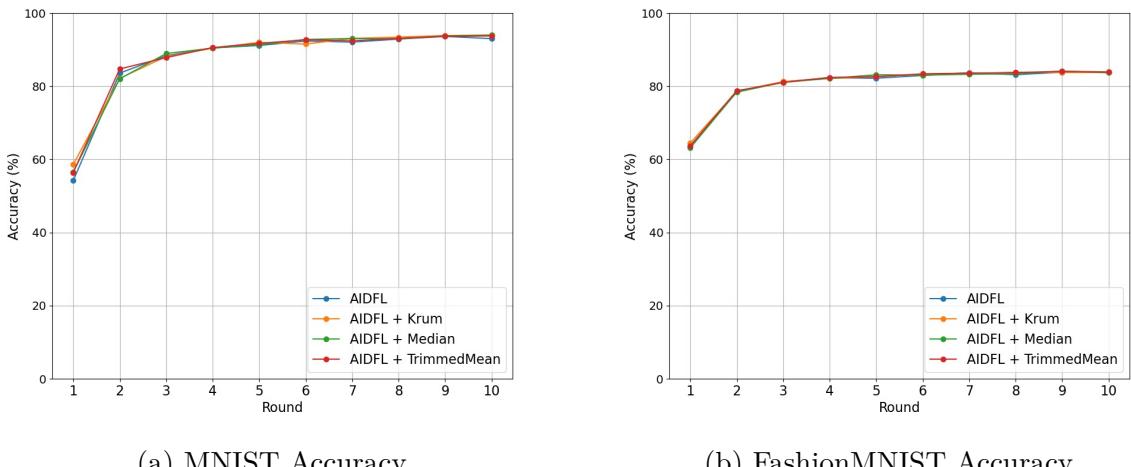
Figure 6.9: PNR = 30 performance under untargeted sample poisoning for MNIST, FMNIST and PNR = 60 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

The chart and table illustrate the performance of 4 aggregation strategies on the MNIST and FashionMNIST datasets under untargeted sample poisoning attacks with a PNR of 50. The accuracy for all aggregation strategies on MNIST ranges between 92% to 94%, with the AIDFL + Krum combination showing the best performance, with an accuracy of 93.98 ± 3.43 . Accuracy across all strategies also shows high stability on FashionMNIST, ranging between 83% and 84%, with AIDFL + TrimmedMean is slightly better than other algorithms, with the accuracy of around 83.89 ± 1.97 .

Indeed, the performance of AIDFL under attacks involving multiple malicious clients suggests that it may perform adequately in environments with fewer adversarial participants, its effectiveness will diminish as the number of malicious updates increases. Decline in the performance could be attributed to the susceptibility of the aggregation to skewed data inputs, which become more obvious with more adversaries.

Table 6.10: Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST after round 10 when PNR = 50, and PSR = 100 for all datasets.

	MNIST	FashionMNIST
AIDFL	92.99 ± 3.79	83.76 ± 1.94
AIDFL + Krum	93.98 ± 3.43	83.71 ± 1.87
AIDFL + Median	93.92 ± 3.63	83.69 ± 2.01
AIDFL + TrimmedMean	93.73 ± 3.60	83.89 ± 1.97



(a) MNIST Accuracy.

(b) FashionMNIST Accuracy.

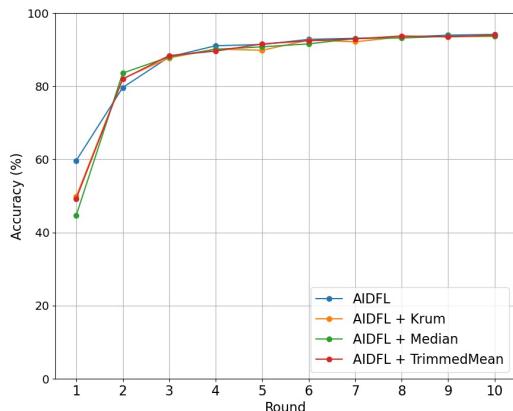
Figure 6.10: PNR = 50 performance under untargeted sample poisoning for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

Table 6.11 and Figure 6.11 present the accuracy performance of different aggregation strategies under untargeted sample poisoning conditions with PNR = 70 and a PSR = 100 across the MNIST and FashionMNIST datasets.

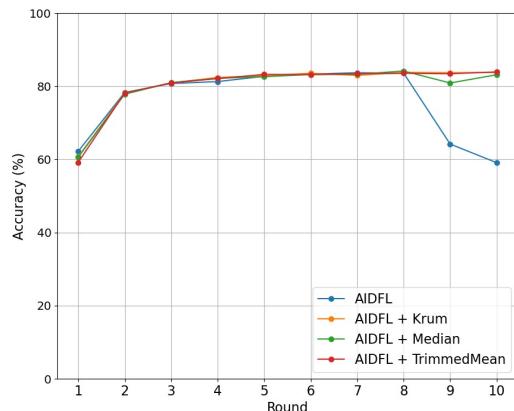
For AIDFL, the performance on the MNIST dataset is outstanding, achieving the highest accuracy of 94.13% despite multiple clients carrying out untargeted sample poisoning attacks. This demonstrates exceptional defending capability of AIDFL against such attacks. However, the contrast in its performance on the FashionMNIST dataset is strong. The accuracy dramatically drops to 59.07% in the final round, indicating that the presence of 7 malicious attacking clients significantly impacts the accuracy of the model on FashionMNIST. This highlights limitations of AIDFL in handling different datasets and its varying capability to defend against sample poisoning attacks.

Table 6.11: Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST after round 10 when PNR = 70, and PSR = 100 for all datasets.

	MNIST	FashionMNIST
AIDFL	94.13±3.41	59.07±3.14
AIDFL + Krum	93.60±4.23	83.73±2.24
AIDFL + Median	93.74±4.73	83.12±2.22
AIDFL + TrimmedMean	94.01±4.32	83.90±2.39



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.

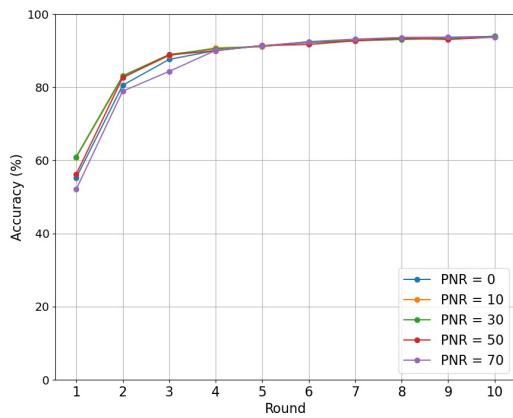
Figure 6.11: PNR = 70 performance under untargeted sample for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

In addition to compare different algorithms horizontally, the data also provides a vertical overview of performance of AIDFL under different PNR settings for untargeted label flipping attacks across MNIST, FashionMNIST and CIFAR10. It can be concluded from Table 6.12 that for MNIST and FashionMNIST, accuracy of AIDFL gradually decreases as PNR increases. However, the declines are marginal at only 0.24% and 0.35% respectively, suggesting that the algorithm remains robust against untargeted label flipping.

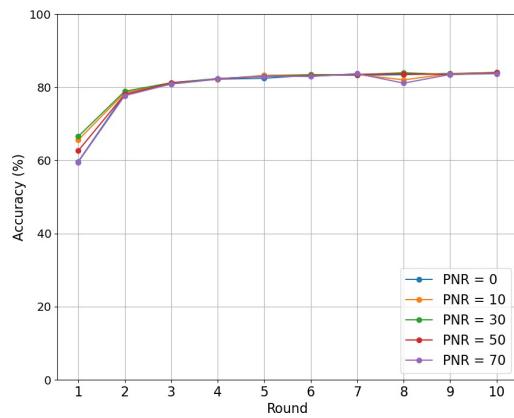
In contrast, accuracy increases with higher PNR on CIFAR10. This could be attributed to the higher complexity and greater diversity of the CIFAR10 dataset, which may enhance the generalization capability of the model, allowing it to maintain or even improve performance despite there is an increase on adversarial nodes.

Table 6.12: Accuracy of AIDFL after round 10 across different PNR values (with SEM), where '-' means the PNR is not applicable to corresponding dataset.

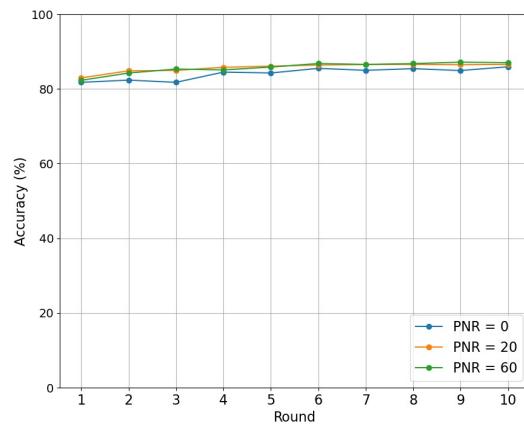
AIDFL	MNIST	FashionMNIST	CIFAR10
PNR = 0	93.87±3.76	84.02±2.34	85.93±0.50
PNR = 10	93.76±3.21	84.01±1.76	-
PNR = 20	-	-	86.56±0.37
PNR = 30	93.93±3.17	83.98±1.68	-
PNR = 50	93.68±3.62	83.88±2.06	-
PNR = 60	-	-	87.02±0.49
PNR = 70	93.63±4.10	83.67±2.33	-



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.



(c) CIFAR10 Accuracy.

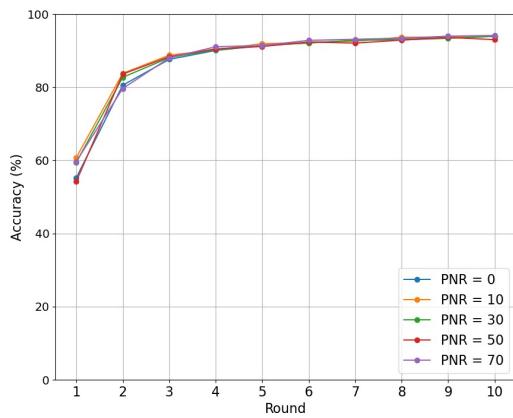
Figure 6.12: AIDFL performance under untargeted label-flipping attacks where PNR=10, 30, 50, 70 for MNIST, FashionMNIST and PNR=0, 20, 60 for CIFAR10 in 10 rounds, with the accuracy on the y-axis and the round progression on the x-axis.

The implementation of adversarial training within a DFL framework has effectively enhanced robustness of the model to noise and malicious manipulations. AIDFL, by incorporating noise and perturbations during the training phase, enables the model to better adapt to unseen data scenarios, thereby improving generalization capabilities in practical applications. As shown in Figure 6.13, although different degrees of sample poisoning (different PNR values) are illustrated, AIDFL maintains high accuracy levels, demonstrating superior noise resistance. Therefore, combining FL with adversarial training not only improves data security, but also ensures model performance and reliability in unstable data environments.

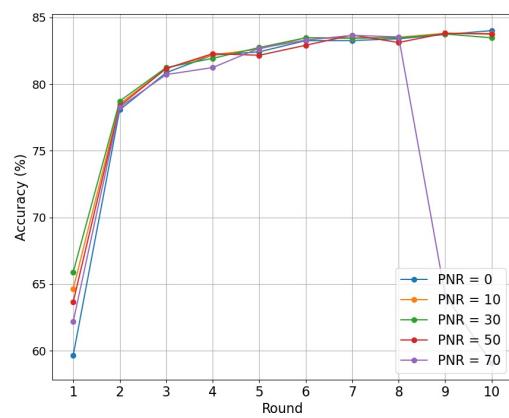
For the MNIST and CIFAR10 datasets, the results demonstrate robust defense capabilities. However, significant performance degradation of the FMNIST dataset has shown in the DFL environment when PNR increases to 70. In this scenario, the majority of clients are malicious, and their adverse influence is sufficient to reshape the entire learning process of the model, thus mitigating the response speed of the model to honest updates. This is distinctly illustrated by the significant drop in accuracy to 59.07% for the AIDFL algorithm at high PNR settings, indicating that AIDFL is inadequate to defend against large-scale sample poisoning attacks under such high-threat conditions.

Table 6.13: Accuracy of AIDFL after round 10 across different PNR values (with SEM), where '-' means the PNR is not applicable to corresponding dataset.

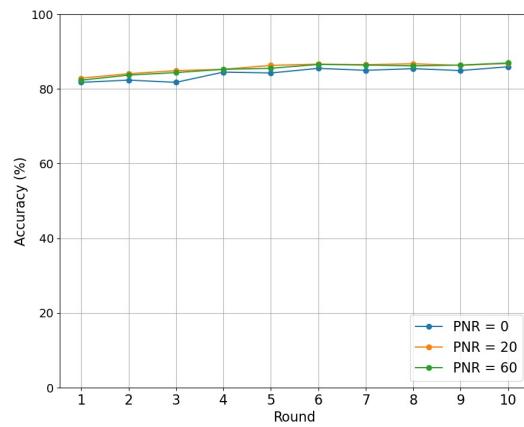
AIDFL	MNIST	FashionMNIST	CIFAR10
PNR = 0	93.87±3.76	84.02±2.34	85.93±0.50
PNR = 10	93.88±3.20	83.80±1.86	-
PNR = 20	-	-	86.80±0.42
PNR = 30	93.90±3.32	83.48±1.73	-
PNR = 50	92.98±3.79	83.76±1.94	-
PNR = 60	-	-	86.94±0.47
PNR = 70	94.13±3.41	59.07±3.14	-



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.



(c) CIFAR10 Accuracy.

Figure 6.13: AIDFL performance under untargeted sample poisoning attacks where PNR = 0, 10, 30, 50, 70 for MNIST, FashionMNIST and PNR = 0, 20, 60 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.

6.2.4 AIDFL: Compare performance between IID and non-IID settings

In order to demonstrate the motivation of this thesis that the performance of the anomaly detection algorithm designed based on information-theoretic metrics is mainly dependent on factors endogenous to the model and independent of the distribution of the data, heterogeneous data is not considered to have a significant impact on the algorithm. To this end, this chapter conducts additional experiments on the MNIST and FMNIST datasets to compare the difference in model performance between AIDFL in non-IID vs. IID environment in both a benign environment and an environment with multiple configurations of data poisoning attacks. These comparisons will be shown in detail through subsequent graphs to visually confirm the theoretical assumptions.

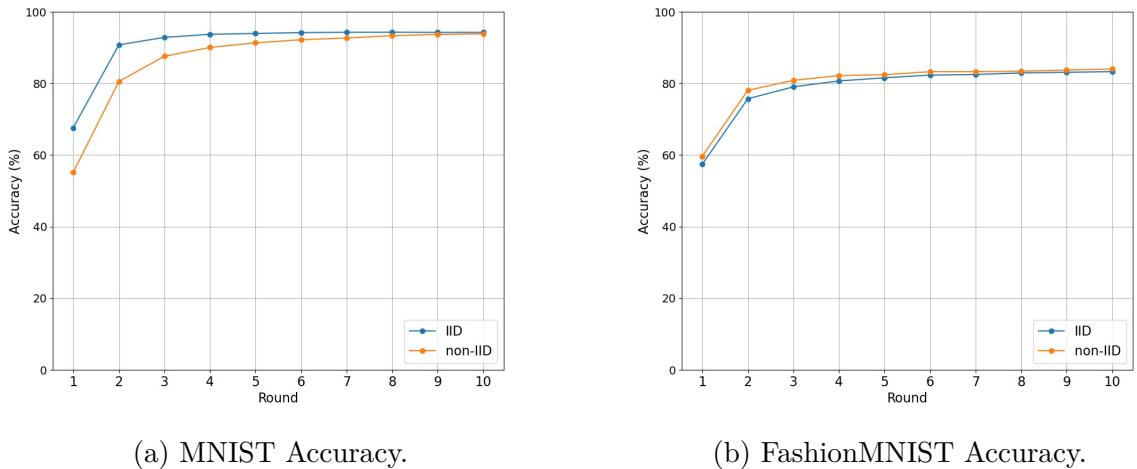


Figure 6.14: Comparing differences in baseline model performance in IID and non-IID environments.

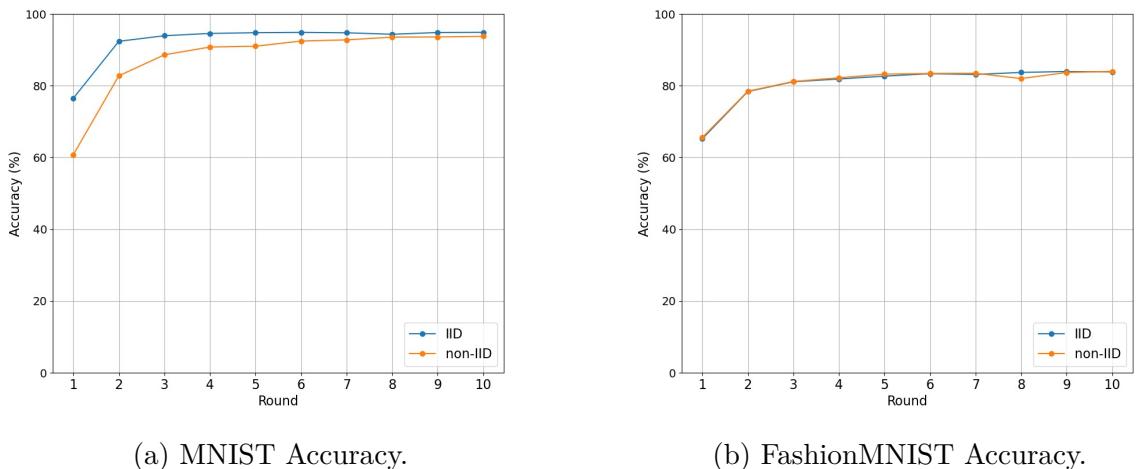
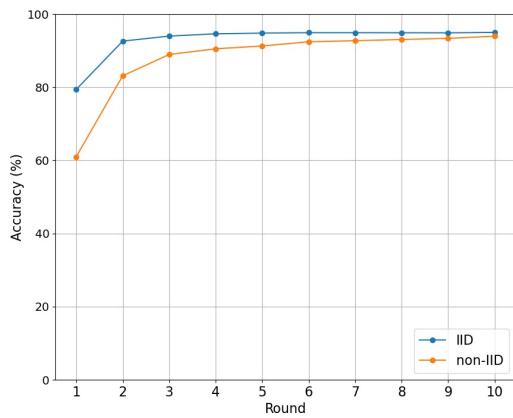
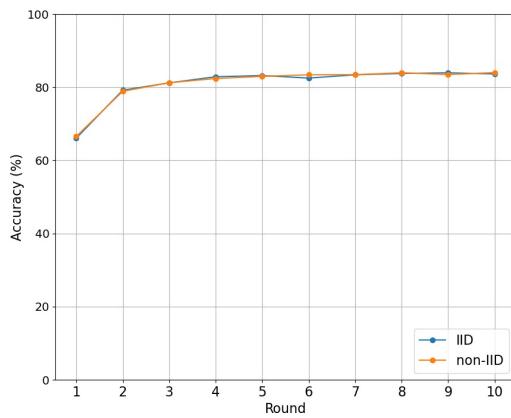


Figure 6.15: Compare the performance differences of untargeted label flipping attacks with PNR = 10 in IID and non-IID settings.

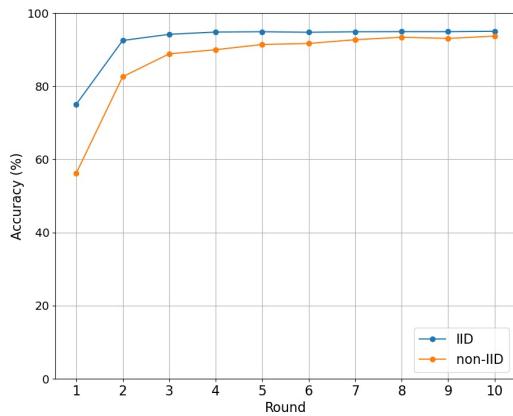


(a) MNIST Accuracy.

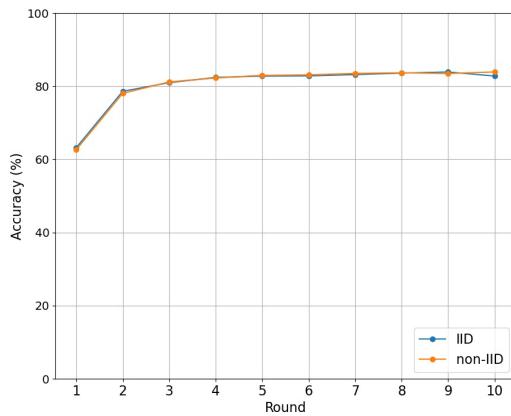


(b) FashionMNIST Accuracy.

Figure 6.16: Compare the performance differences of untargeted label flipping attacks with PNR = 30 in IID and non-IID settings.



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.

Figure 6.17: Compare the performance differences of untargeted label flipping attacks with PNR = 50 in IID and non-IID settings.

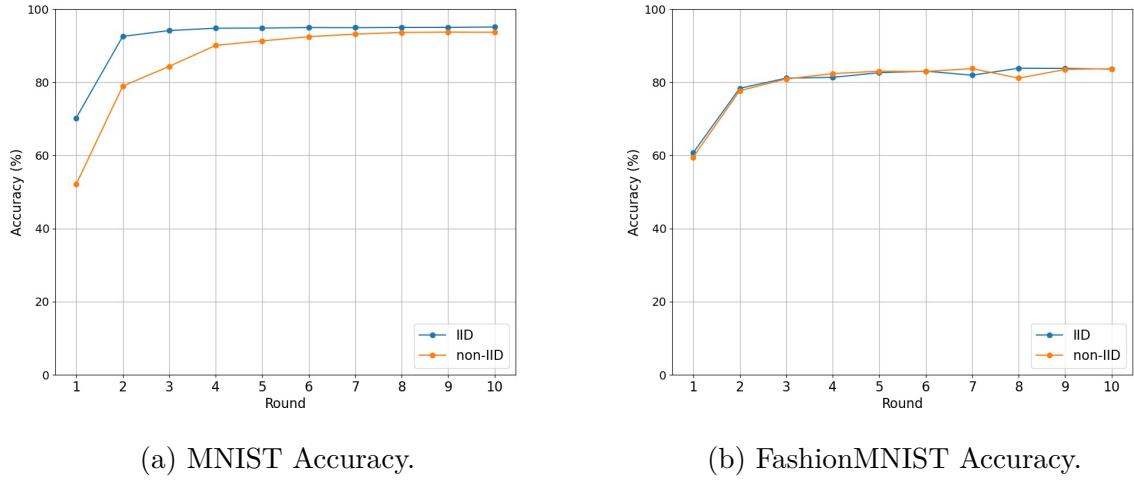


Figure 6.18: Compare the performance differences of untargeted label flipping attacks with PNR = 70 in IID and non-IID settings.

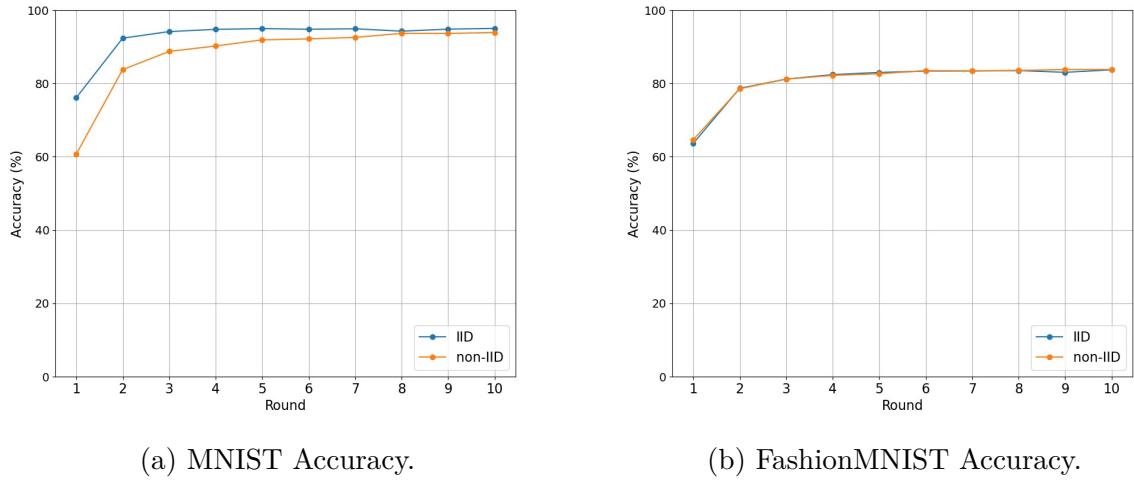
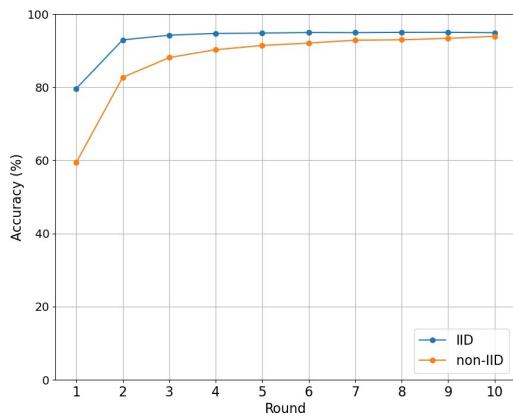
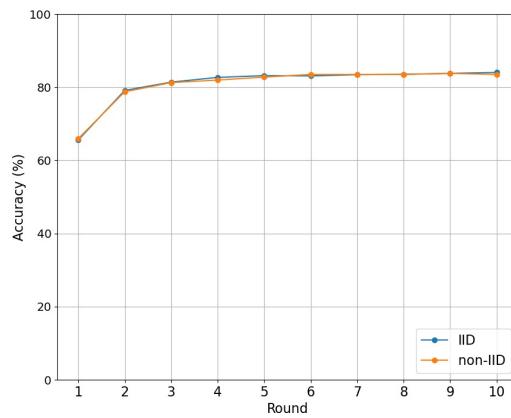


Figure 6.19: Compare the performance differences of untargeted sample poisoning attacks with PNR = 10 in IID and non-IID settings.

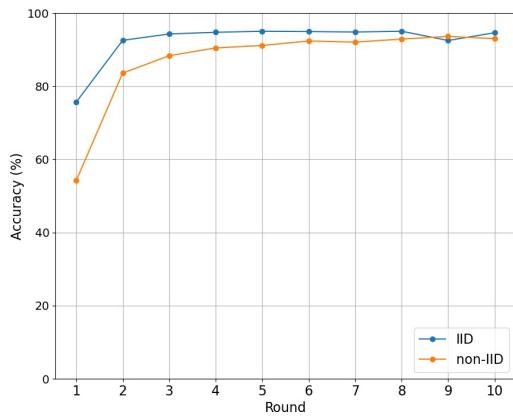


(a) MNIST Accuracy.

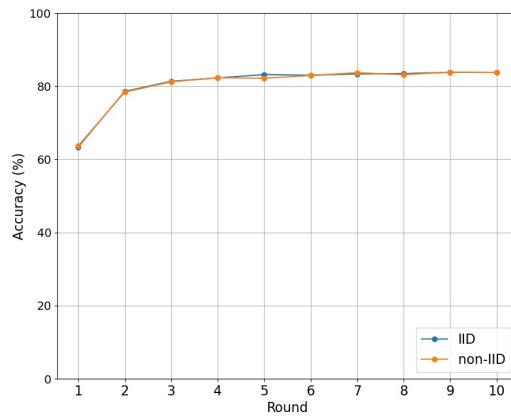


(b) FashionMNIST Accuracy.

Figure 6.20: Compare the performance differences of untargeted sample poisoning attacks with PNR = 30 in IID and non-IID settings.



(a) MNIST Accuracy.



(b) FashionMNIST Accuracy.

Figure 6.21: Compare the performance differences of untargeted sample poisoning attacks with PNR = 50 in IID and non-IID settings.

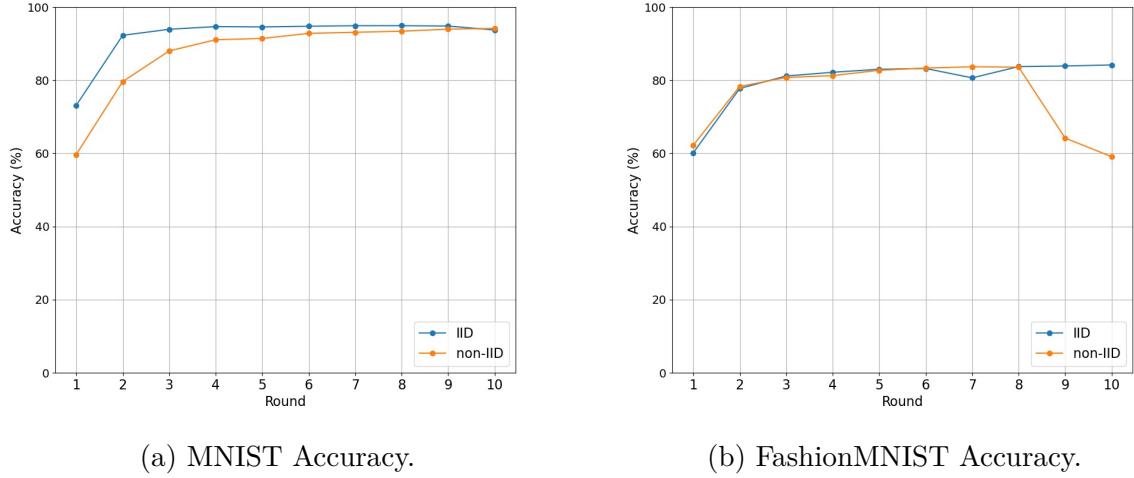


Figure 6.22: Compare the performance differences of untargeted sample poisoning attacks with PNR = 70 in IID and non-IID settings.

From the previous figures, it can be inferred that non-IID settings require more rounds to achieve convergence compared to IID settings. Furthermore, as the PNR increases, the number of rounds needed for non-IID to approach the accuracy of IID also increases, yet the accuracy after 10 rounds is almost identical in both environments. However, in the scenario of untargeted sample poisoning with a high PNR = 70, the accuracy significantly decreases in non-IID settings within the FMNIST dataset (59.07), while IID settings remain unaffected (84.16). This suggests that, besides the impact of sample poisoning attacks outlined previously, the heterogeneity of data can also affect model performance.

Overall, the use of information-theoretic metrics helps prevent the model from being significantly affected by uneven data distribution. AIDFL could handle sample poisoning attacks except in the case of more than a few attackers, which will drastically reduce the accuracy of the model, and the results are less robust compared to the results of untargeted label flipping attacks.

Chapter 7

Summary and Conclusions

In this chapter, the previous work will be summarized and all limitations in any of the design, implementation, and evaluation process will be pointed out. Based on those, future work will be anticipated and envisioned.

7.1 Conclusions

In this work, a defense strategy using information theory-based metrics to combine anomaly detection with robust aggregation is proposed: AIDFL. AIDFL mitigates the untargeted data poisoning attacks by employing a multi-level defense protocol consisting of k-means clustering, filtering of anomalies by mutual information and conditional entropy estimations, and normalization. The convergence of AIDFL in non-IID scenarios is verified after extensive heterogeneous experimental evaluations. By comparing the model performance in non-IID and IID settings, the effectiveness of information-theoretic metrics designed to filter anomalies was confirmed, demonstrating that these metrics are independent of data distribution and that model performance depends on endogeneous model factors.

Furthermore, during the evaluation phase, the effectiveness of AIDFL is also demonstrated by the extensive evaluation of various attack configurations on the datasets MNIST, FashionMNIST and CIFAR10 chosen for this thesis. At present, AIDFL proves its capability in anomaly detection by changing the aggregation part to other state-of-the-art robust aggregations, making AIDFL comparable to other algorithm combinations.

Test results show that AIDFL is more difficult to defend against untargeted sample poisoning attacks. As the number of poisoned clients increases, the defense performance of AIDFL is greatly weakened. Relatively, AIDFL works better in defending untargeted label flipping attacks. Improvements to AIDFL may include incorporating perturbation mechanisms (Section 3.3), such as adding a weak differential privacy defense layer or adding new defense strategies specifically for sample poisoning attacks. These could be steps to further enhance the effect of AIDFL.

7.2 Limitations

In the process of testing the defending effect of AIDFL, replacing only the part of aggregation will result in an insufficiently obvious comparison of the advantages between AIDFL and the other algorithms. Improvements could include a comparison between AIDFL as a whole and other algorithms to better represent the effectiveness of AIDFL defense and the differences between algorithms.

AIDFL did not analyze resource consumption. Resource-efficient aggregation mechanisms are crucial for the scalability of DFL [38]. The choice of aggregator directly affects the computational resource usage required for each FL scenario. Therefore, in order to fully assess the applicability and practicality of AIDFL, future work includes analysis of the computational resources used by the corresponding aggregator in each DFL scenario, to ensure its efficient operation in dealing with large-scale networks.

7.3 Future Work

Current experiments have validated the feasibility of AIDFL, which integrates anomaly detection with robust aggregation. In the future, AIDFL could be treated as a comprehensive hybrid defense mechanism and directly compared with other aggregation protocols. This would allow for a more distinct comparison of these algorithms under poisoning attacks.

The existing work has only investigated the effectiveness of AIDFL defense against data poisoning attacks, and these defenses are limited to untargeted attacks. In the future, further division of the dataset could be considered to simulate targeted data poisoning attacks. For example, formulating specific label categories to introduce targeted label flipping or targeted sample poisoning (backdoor attacks). Additionally, the effectiveness of the defense against model poisoning attacks also needs to be evaluated.

To reduce computational costs, the fully connected network tested on CIFAR10 utilized only 5 clients, whereas the networks for MNIST and FashionMNIST used 10 clients each. Future work could consider increasing the number of clients for CIFAR10 to 10 and running the experiments on GPUs to enhance performance and efficiency. Additionally, transferring and running these experiments on a user-friendly platform like Fedstellar could simplify the evaluation process and facilitate more comprehensive testing and development.

Bibliography

- [1] M. Alazab, S. P. RM, P. M, P. K. R. Maddikunta, T. R. Gadekallu, and Q.-V. Pham, “Federated learning for cybersecurity: Concepts, challenges, and future directions”, *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3501–3509, 2022. DOI: [10.1109/TII.2021.3119038](https://doi.org/10.1109/TII.2021.3119038).
- [2] E. T. Martínez 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*, vol. 25, no. 4, pp. 2983–3013, 2023, ISSN: 2373-745X. DOI: [10.1109/comst.2023.3315746](https://doi.org/10.1109/comst.2023.3315746). [Online]. Available: <http://dx.doi.org/10.1109/COMST.2023.3315746>.
- [3] Y. Li, D. Yuan, A. S. Sani, and W. Bao, “Enhancing federated learning robustness in adversarial environment through clustering non-iid features”, *Computers & Security*, vol. 132, p. 103319, 2023, ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2023.103319>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404823002298>.
- [4] N. Onoszko, G. Karlsson, O. Mogren, and E. L. Zec, *Decentralized federated learning of deep neural networks on non-iid data*, 2021. arXiv: [2107.08517](https://arxiv.org/abs/2107.08517) [cs.LG].
- [5] S. Augenstein, A. Hard, K. Partridge, and R. Mathews, *Jointly learning from decentralized (federated) and centralized data to mitigate distribution shift*, 2021. arXiv: [2111.12150](https://arxiv.org/abs/2111.12150) [cs.LG].
- [6] E. Gabrielli, G. Pica, and G. Tolomei, *A survey on decentralized federated learning*, 2023. arXiv: [2308.04604](https://arxiv.org/abs/2308.04604) [cs.LG].
- [7] S. Li, Y. Cheng, Y. Liu, W. Wang, and T. Chen, *Abnormal client behavior detection in federated learning*, 2019. arXiv: [1910.09933](https://arxiv.org/abs/1910.09933) [cs.LG].
- [8] S. Yu and J. C. Principe, *Understanding autoencoders with information theoretic concepts*, 2019. arXiv: [1804.00057](https://arxiv.org/abs/1804.00057) [cs.LG].
- [9] R. Shwartz-Ziv and N. Tishby, *Opening the black box of deep neural networks via information*, 2017. arXiv: [1703.00810](https://arxiv.org/abs/1703.00810) [cs.LG].
- [10] T. R. Gadekallu, Q.-V. Pham, T. Huynh-The, S. Bhattacharya, P. K. R. Maddikunta, and M. Liyanage, *Federated learning for big data: A survey on opportunities, applications, and future directions*, 2021. arXiv: [2110.04160](https://arxiv.org/abs/2110.04160) [cs.LG].
- [11] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions”, *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020, ISSN: 1558-0792. DOI: [10.1109/msp.2020.2975749](https://doi.org/10.1109/msp.2020.2975749). [Online]. Available: <http://dx.doi.org/10.1109/MSP.2020.2975749>.

- [12] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, 2023. arXiv: 1602.05629 [cs.LG].
- [13] J. C. Jiang, B. Kantarci, S. Oktug, and T. Soyata, “Federated learning in smart city sensing: Challenges and opportunities”, *Sensors*, vol. 20, no. 21, 2020, ISSN: 1424-8220. DOI: 10.3390/s20216230. [Online]. Available: <https://www.mdpi.com/1424-8220/20/21/6230>.
- [14] Y. Ye, S. Li, F. Liu, Y. Tang, and W. Hu, “Edgefed: Optimized federated learning based on edge computing”, *IEEE Access*, vol. 8, pp. 209 191–209 198, Jan. 2020. DOI: 10.1109/ACCESS.2020.3038287.
- [15] M. Moshawrab, M. Adda, A. Bouzouane, H. Ibrahim, and A. Raad, “Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives”, *Electronics*, vol. 12, no. 10, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12102287. [Online]. Available: <https://www.mdpi.com/2079-9292/12/10/2287>.
- [16] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, *Federated optimization in heterogeneous networks*, 2020. arXiv: 1812.06127 [cs.LG].
- [17] Z. Chen, W. Liao, P. Tian, Q. Wang, and W. Yu, “A fairness-aware peer-to-peer decentralized learning framework with heterogeneous devices”, *Future Internet*, vol. 14, no. 5, 2022, ISSN: 1999-5903. DOI: 10.3390/fi14050138. [Online]. Available: <https://www.mdpi.com/1999-5903/14/5/138>.
- [18] H. Reguieg, M. E. Hanjri, M. E. Kamili, and A. Kobbane, *A comparative evaluation of fedavg and per-fedavg algorithms for dirichlet distributed heterogeneous data*, 2023. arXiv: 2309.01275 [cs.LG].
- [19] M. Chen, Y. Xu, H. Xu, and L. Huang, “Enhancing decentralized federated learning for non-iid data on heterogeneous devices”, in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 2289–2302. DOI: 10.1109/ICDE55515.2023.00177.
- [20] C. Li, G. Li, and P. K. Varshney, “Decentralized federated learning via mutual knowledge transfer”, *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1136–1147, Jan. 2022, ISSN: 2372-2541. DOI: 10.1109/jiot.2021.3078543. [Online]. Available: <http://dx.doi.org/10.1109/JIOT.2021.3078543>.
- [21] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, *Deep mutual learning*, 2017. arXiv: 1706.00384 [cs.CV].
- [22] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, *Braintorrent: A peer-to-peer environment for decentralized federated learning*, 2019. arXiv: 1905.06731 [cs.LG].
- [23] S. Savazzi, M. Nicoli, V. Rampa, and S. Kianoush, “Federated learning with mutually cooperating devices: A consensus approach towards server-less model optimization”, in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3937–3941. DOI: 10.1109/ICASSP40776.2020.9054055.

- [24] M. Roussopoulos, M. Baker, D. S. H. Rosenthal, T. J. Giuli, P. Maniatis, and J. Mogul, “2 p2p or not 2 p2p?”, in *Peer-to-Peer Systems III*, G. M. Voelker and S. Shenker, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 33–43, ISBN: 978-3-540-30183-7.
- [25] S. Sagar, C.-S. Li, S. W. Loke, and J. Choi, *Poisoning attacks and defenses in federated learning: A survey*, 2023. arXiv: 2301.05795 [cs.CR].
- [26] G. Xia, J. Chen, C. Yu, and J. Ma, “Poisoning attacks in federated learning: A survey”, *IEEE Access*, vol. 11, pp. 10 708–10 722, 2023. DOI: 10.1109/ACCESS.2023.3238823.
- [27] Z. Wang, Q. Kang, X. Zhang, and Q. Hu, *Defense strategies toward model poisoning attacks in federated learning: A survey*, 2022. arXiv: 2202.06414 [cs.CR].
- [28] X. Cao, M. Fang, J. Liu, and N. Z. Gong, *Fltrust: Byzantine-robust federated learning via trust bootstrapping*, 2022. arXiv: 2012.13995 [cs.CR].
- [29] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, *Analyzing federated learning through an adversarial lens*, 2019. arXiv: 1811.12470 [cs.LG].
- [30] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámarra, “Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges”, *Information Fusion*, vol. 90, pp. 148–173, Feb. 2023, ISSN: 1566-2535. DOI: 10.1016/j.inffus.2022.09.011. [Online]. Available: <http://dx.doi.org/10.1016/j.inffus.2022.09.011>.
- [31] C. Fung, C. J. M. Yoon, and I. Beschastnikh, *Mitigating sybils in federated learning poisoning*, 2020. arXiv: 1808.04866 [cs.LG].
- [32] J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu, “Poisoning attack in federated learning using generative adversarial nets”, in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, pp. 374–380. DOI: 10.1109/TrustCom/BigDataSE.2019.00057.
- [33] 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, 2021. DOI: 10.1109/MSEC.2020.3039941.
- [34] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, *Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning*, 2021. arXiv: 2108.10241 [cs.LG].
- [35] X. Zhou, M. Xu, Y. Wu, and N. Zheng, “Deep model poisoning attack on federated learning”, *Future Internet*, vol. 13, no. 3, 2021, ISSN: 1999-5903. DOI: 10.3390/fi13030073. [Online]. Available: <https://www.mdpi.com/1999-5903/13/3/73>.
- [36] X. Gong, Y. Chen, Q. Wang, and W. Kong, “Backdoor attacks and defenses in federated learning: State-of-the-art, taxonomy, and future directions”, *IEEE Wireless Communications*, vol. 30, no. 2, pp. 114–121, 2023. DOI: 10.1109/WC.017.2100714.
- [37] T. D. Nguyen, T. Nguyen, P. L. Nguyen, H. H. Pham, K. Doan, and K.-S. Wong, *Backdoor attacks and defenses in federated learning: Survey, challenges and future research directions*, 2023. arXiv: 2303.02213 [cs.LG].

- [38] C. Feng, A. H. Celdran, J. Baltensperger, E. T. M. Beltran, G. Bovet, and B. Stiller, *Sentinel: An aggregation function to secure decentralized federated learning*, 2023. arXiv: 2310.08097 [cs.DC].
- [39] R. A. Mallah, D. Lopez, G. B. Marfo, and B. Farooq, *Untargeted poisoning attack detection in federated learning via behavior attestation*, 2022. arXiv: 2101.10904 [cs.CR].
- [40] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into deep learning*. Cambridge University Press, 2023.
- [41] Z. Goldfeld and Y. Polyanskiy, “The information bottleneck problem and its applications in machine learning”, *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 19–38, 2020. DOI: 10.1109/JSAIT.2020.2991561.
- [42] K. Kawaguchi, Z. Deng, X. Ji, and J. Huang, *How does information bottleneck help deep learning?*, 2023. arXiv: 2305.18887 [cs.LG].
- [43] A. M. Saxe, Y. Bansal, J. Dapello, et al., “On the information bottleneck theory of deep learning”, in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ry_WPG-A-.
- [44] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, *Byzantine-robust distributed learning: Towards optimal statistical rates*, 2021. arXiv: 1803.01498 [cs.LG].
- [45] J. Yin, X. Cui, and K. Li, “A reputation-based resilient and recoverable p2p botnet”, in *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, 2017, pp. 275–282. DOI: 10.1109/DSC.2017.20.
- [46] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, *How to backdoor federated learning*, 2019. arXiv: 1807.00459.
- [47] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent”, in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, et al., Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.
- [48] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, *The hidden vulnerability of distributed learning in byzantium*, 2018. arXiv: 1802.07927.
- [49] C. Fung, C. J. M. Yoon, and I. Beschastnikh, “The limitations of federated learning in sybil settings”, in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, San Sebastian: USENIX Association, Oct. 2020, pp. 301–316, ISBN: 978-1-939133-18-2. [Online]. Available: <https://www.usenix.org/conference/raid2020/presentation/fung>.
- [50] P. Rieger, T. D. Nguyen, M. Miettinen, and A.-R. Sadeghi, “Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection”, in *Proceedings 2022 Network and Distributed System Security Symposium*, ser. NDSS 2022, Internet Society, 2022. DOI: 10.14722/ndss.2022.23156. [Online]. Available: <http://dx.doi.org/10.14722/ndss.2022.23156>.
- [51] X. Li, Z. Qu, S. Zhao, B. Tang, Z. Lu, and Y. Liu, *Lomar: A local defense against poisoning attack on federated learning*, 2022. arXiv: 2201.02873.

- [52] C. Zhang, S. Yang, L. Mao, *et al.*, “Anomaly detection and defense techniques in federated learning: A comprehensive review”, *Artificial Intelligence Review*, vol. 57, p. 150, 2024. DOI: 10.1007/s10462-024-10796-1.
- [53] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, *Learning to detect malicious clients for robust federated learning*, 2020. arXiv: 2002.00211.
- [54] J. Lianga, R. Wang, C. Feng, and C.-C. Chang, *A survey on federated learning poisoning attacks and defenses*, 2023. arXiv: 2306.03397.
- [55] K. Wei, J. Li, M. Ding, *et al.*, *Federated learning with differential privacy: Algorithms and performance analysis*, 2019. arXiv: 1911.00222.
- [56] M. Du, R. Jia, and D. Song, *Robust anomaly detection and backdoor attack detection via differential privacy*, 2019. arXiv: 1911.07116.
- [57] T. D. Nguyen, P. Rieger, H. Chen, *et al.*, “FLAME: Taming backdoors in federated learning”, in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 1415–1432, ISBN: 978-1-939133-31-1. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity22/presentation/nguyen>.
- [58] A. Gholami, N. Torkzaban, and J. S. Baras, “Trusted decentralized federated learning”, in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 1–6. DOI: 10.1109/CCNC49033.2022.9700624.
- [59] J. Lai, T. Wang, C. Chen, Y. Li, and Z. Zheng, “Vfedad: A defense method based on the information mechanism behind the vertical federated data poisoning attack”, in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, ser. CIKM ’23, , Birmingham, United Kingdom, Association for Computing Machinery, 2023, pp. 1148–1157, ISBN: 9798400701245. DOI: 10.1145/3583780.3615106. [Online]. Available: <https://doi.org/10.1145/3583780.3615106>.
- [60] J. Park, D.-J. Han, M. Choi, and J. Moon, “Sageflow: Robust federated learning against both stragglers and adversaries”, in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: <https://openreview.net/forum?id=rA9HFxFT7th>.
- [61] Coursesteach, *Computer vision (part 14)-common types of noise*, Accessed: Jun 22, 2024, Dec. 2023. [Online]. Available: <https://medium.com/@Coursesteach/computer-vision-part-14-common-types-of-noise-7e6507cc763c>.
- [62] E. A. Team, *How to interpret a confusion matrix for a machine learning model*, Accessed: Jun 25, 2024, 2024. [Online]. Available: <https://www.evidentlyai.com/classification-metrics/confusion-matrix>.
- [63] A. Panchal, *Machine learning: Improving classification accuracy on mnist using data augmentation - an easy way to grow the training data set*, Accessed: Jun 25, 2024, 2020. [Online]. Available: <https://towardsdatascience.com/improving-accuracy-on-mnist-using-data-augmentation-b5c38eb5a903>.
- [64] Y. LeCun and C. Cortes, *MNIST handwritten digit database*, <http://yann.lecun.com/exdb/mnist/>, Accessed: Jun 28, 2024, 2010.

- [65] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms*, 2017. arXiv: 1708.07747 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/1708.07747>.
- [66] A. Krizhevsky, “Learning multiple layers of features from tiny images”, *University of Toronto*, May 2012.
- [67] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1512.03385>.

Abbreviations

ML	Machine Learning
FL	Federated Learning
non-IID	non-independent and identically distributed
CFL	Centralized Federated Learning
FL	Decentralized Federated Learning
SGD	Stochastic Gradient Descent
PENS	Performance-Based Neighbor Selection
MKT	Mutual Knowledge Transfer
GAN	Generative Adversarial Network
MI	Mutual Information
CE	Conditional Entropy
IB	Information Bottleneck
DNN	Deep Neural Networks
KDE	Kernel Density Estimation
VFL	Vertical Federated Learning
PNR	Poisoned Node Ratio
PSR	Poisoned Sample Ratio
NR	Noise Ratio
TP	True Positives
TN	True Negatives
AIDFL	Anomalous Information theory-based Detection in DFL
MLP	Multilayer Perceptron
ResNet	Residual Network

List of Figures

2.1	Structure of FedAvg. [14]	5
2.2	Difference between (Centralized)FL and DFL. [17]	6
2.3	Label-Flipping Attacks: Adversaries alter labels. [31]	9
2.4	Model poisoning attack directly manipulate the local models in FL. [27]	10
2.5	Semantic attacks and artificial attacks in backdoor attacks. [37]	12
2.6	The relationship between mutual information, conditional entropy, and joint entropy. [40]	15
3.1	According to the type of attacks, two methods are used separately to filter out malicious models and mitigate the damage. [60]	23
4.1	Classification Accuracy with MNIST Digit Samples	27
4.2	Relationship between entropy and PSR.	30
4.3	High-level overview of the detection process in AIDFL.	30
6.1	The class names and part of the example images in FashionMNIST dataset. [65]	42
6.2	The class names and part of the example images in CIFAR10 dataset. [65]	43
6.3	Baseline performance for MNIST, FashionMNIST and CIFAR10 for 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	47
6.4	PNR = 10 performance under untargeted label flipping for MNIST, FashionMNIST and PNR = 20 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	50
6.5	PNR = 30 performance under untargeted label flipping for MNIST, FashionMNIST and PNR = 60 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	52

6.6 PNR = 50 performance under untargeted label flipping for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	53
6.7 PNR = 70 performance under untargeted label flipping for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	54
6.8 PNR = 10 performance under untargeted sample poisoning for MNIST, FashionMNIST and PNR = 20 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	56
6.9 PNR = 30 performance under untargeted sample poisoning for MNIST, FM- NIST and PNR = 60 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	58
6.10 PNR = 50 performance under untargeted sample poisoning for MNIST and FashionMNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	59
6.11 PNR = 70 performance under untargeted sample for MNIST and Fashion- MNIST in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	60
6.12 AIDFL performance under untargeted label-flipping attacks where PNR=10, 30, 50, 70 for MNIST, FashionMNIST and PNR=0, 20, 60 for CIFAR10 in 10 rounds, with the accuracy on the y-axis and the round progression on the x-axis.	62
6.13 AIDFL performance under untargeted sample poisoning attacks where PNR = 0, 10, 30, 50, 70 for MNIST, FashionMNIST and PNR = 0, 20, 60 for CIFAR10 in 10 rounds implementing 4 algorithms, with the accuracy on the y-axis and the round progression on the x-axis.	64
6.14 Comparing differences in baseline model performance in IID and non-IID environments.	65
6.15 Compare the performance differences of untargeted label flipping attacks with PNR = 10 in IID and non-IID settings.	65
6.16 Compare the performance differences of untargeted label flipping attacks with PNR = 30 in IID and non-IID settings.	66
6.17 Compare the performance differences of untargeted label flipping attacks with PNR = 50 in IID and non-IID settings.	66
6.18 Compare the performance differences of untargeted label flipping attacks with PNR = 70 in IID and non-IID settings.	67

6.19 Compare the performance differences of untargeted sample poisoning attacks with PNR = 10 in IID and non-IID settings.	67
6.20 Compare the performance differences of untargeted sample poisoning attacks with PNR = 30 in IID and non-IID settings.	68
6.21 Compare the performance differences of untargeted sample poisoning attacks with PNR = 50 in IID and non-IID settings.	68
6.22 Compare the performance differences of untargeted sample poisoning attacks with PNR = 70 in IID and non-IID settings.	69

List of Tables

3.1	Classification of defense approaches against poisoning attacks. For each defense listed, the type of attack (targeted/untargeted) and the scope (data/-model) are indicated. The technique describes the main methods applied by each approach. This table primarily summarizes the defense strategies for DFL environment, with the information-theoretic defenses being based on VFL. Adopted from [25] [30].	25
4.1	Configuration overview for the two selected attacks. NR is not applicable to the label flipping attacks.	29
6.1	Configuration overview for the two selected attacks. NR is not applicable to the label flipping attacks.	45
6.2	Mean Accuracy after round 10.	48
6.3	Mean Accuracy after round 10 (with SEM).	48
6.4	Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 10 for MNIST and FashionMNIST and PNR = 20 for CIFAR10, and PSR = 100 for all datasets.	50
6.5	Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 30 for MNIST and FashionMNIST and PNR = 60 for CIFAR10, and PSR = 100 for all datasets.	51
6.6	Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST after round 10 when PNR = 50, and PSR = 100 for all datasets.	53
6.7	Accuracy performance in terms of mean and SEM under untargeted label flipping for MNIST, FashionMNIST after round 10 when PNR = 70, and PSR = 100 for all datasets.	54

6.8 Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 10 for MNIST and FashionMNIST and PNR = 20 for CIFAR10, and PSR = 100 for all datasets.	56
6.9 Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST and CIFAR10 after round 10 when PNR = 30 for MNIST and FashionMNIST and PNR = 60 for CIFAR10, and PSR = 100 for all datasets.	57
6.10 Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST after round 10 when PNR = 50, and PSR = 100 for all datasets.	59
6.11 Accuracy performance in terms of mean and SEM under untargeted sample poisoning for MNIST, FashionMNIST after round 10 when PNR = 70, and PSR = 100 for all datasets.	60
6.12 Accuracy of AIDFL after round 10 across different PNR values (with SEM), where '-' means the PNR is not applicable to corresponding dataset.	61
6.13 Accuracy of AIDFL after round 10 across different PNR values (with SEM), where '-' means the PNR is not applicable to corresponding dataset.	63

List of Algorithms

1	AIDFL Aggregation Algorithm	31
2	K -means Clustering	32
3	Von Neumann Entropy	33
4	Mutual Information	33
5	Mutual Information Estimation (for feature data)	33
6	Conditional Entropy Estimation	33
7	Anomaly Detection	34
8	Proposed AIDFL Algorithm (Detailed)	35