



University of  
Zurich<sup>UZH</sup>

# Design and Implementation of an MTD-based Mitigation Approach for Membership Inference Attacks

*Heqing Ren*  
*Zurich, Switzerland*  
*Student ID: 22-736-128*

Supervisor: Chao Feng, Jan von der Assen  
Date of Submission: April 1, 2025



# Abstrakt

Mit dem rasanten Wachstum von Geräten im Internet der Dinge (IoT) werden Datenschutzbedenken immer bedeutender. Obwohl föderiertes Lernen (Federated Learning) Datenschutzrisiken verringern kann, können Angreifer weiterhin erfolgreich Membership Inference Attacks (MIAs) durchführen. Bestehende Abwehrmethoden wie Confidence Masking und Generative Adversarial Networks stoßen in der Praxis auf Herausforderungen, darunter eine geringere Modellgenauigkeit und ein hoher Rechenaufwand.

In dieser Arbeit wird eine neue Verteidigungsmethode auf Basis der Moving Target Defense (MTD) vorgestellt, die sowohl gegen auf binären Klassifikatoren basierende als auch gegen metrikbasierte Membership Inference Attacks wirksam ist. Die vorgeschlagene Methode nutzt Techniken wie Perceptual Hashing, Datenaugmentation und Bildfusion, um Trainingsdaten gezielt zu transformieren und Angreifer dadurch in die Irre zu führen. Zusätzlich wurde ein Python-Skript entwickelt, das automatisch geeignete Verteidigungsintensitäten für verschiedene Bilddatensätze auswählt.

Die experimentellen Ergebnisse zeigen, dass die vorgeschlagene Methode wirksam ist und sich auf die Datensätze CIFAR-10, CIFAR-100, Fashion MNIST, Tiny ImageNet und ImageNet-10 anwenden lässt – sowohl im klassischen maschinellen Lernen als auch im dezentralen föderierten Lernen. Nach Anwendung der Verteidigung sinken die F1-Scores der MIAs auf etwa 0,5, was auf eine deutlich verringerte Angriffseffektivität hinweist. Gleichzeitig bleibt die Klassifikationsgenauigkeit auf Testdaten nahezu unverändert, was bedeutet, dass die Modelleistung für normale Nutzer nicht beeinträchtigt wird. Damit stellt die Methode eine praktische und effiziente Lösung zur Abwehr von MIAs dar.



# Abstract

With the rapid growth of Internet of Things (IoT) devices, data privacy concerns have become increasingly prominent. Although federated learning can reduce privacy risks, attackers can still successfully perform Membership Inference Attacks (MIAs). Existing defense methods such as confidence masking and generative adversarial networks face practical challenges including decreased model accuracy and high computational cost.

This thesis proposes a new defense method based on Moving Target Defense (MTD) to defend against both binary classifier based MIA and metric based MIA. The proposed method uses techniques including perceptual hash, data augmentation, and image fusion to transform training samples, thereby misleading attackers. Additionally, a python script is developed to automatically select appropriate defense intensities for image datasets.

Experimental results demonstrate that the proposed defense method is effective and applicable across CIFAR-10, CIFAR-100, Fashion MNIST, Tiny ImageNet and ImageNet-10 dataset under both machine learning and decentralized federated learning pipelines, because the F1-scores of MIAs are reduced to near 0.5 after applying the defense method. Besides, the classification accuracy on test samples is barely changed after applying defense method, indicating that the model performance is not affected for normal users. This makes the method a practical and efficient solution for defending against MIAs.



# Acknowledgments

I would like to express my deepest gratitude to my supervisor, Chao Feng, for his support and guidance throughout the course of my master thesis. His expertise, constructive feedback, and encouragement have played a crucial role in my work.

I am also sincerely thankful to Prof. Dr. Burkhard Stiller for the opportunity to conduct my thesis research.

Finally, I extend my appreciation to my family and friends for their constant support and motivation throughout my studies.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Description of Work . . . . .	2
1.3 Thesis Outline . . . . .	3
<b>2 Background</b>	<b>5</b>
2.1 FL . . . . .	5
2.2 MIA . . . . .	7
2.2.1 Overview of MIA . . . . .	7
2.2.2 Adversarial Knowledge . . . . .	8
2.2.3 MIA Approaches . . . . .	8
2.2.4 MIA against FL . . . . .	10
2.3 MTD . . . . .	12
2.3.1 Emergence of MTD . . . . .	12
2.3.2 Definition and Workflow of MTD . . . . .	13
2.3.3 Three Elements for an MTD Technique . . . . .	14
2.3.4 Other Classifications of MTD . . . . .	14

2.3.5	Metrics for Measuring MTD Effectiveness and Efficiency . . . . .	16
2.4	Hash . . . . .	18
2.4.1	Hash Function . . . . .	18
2.4.2	PHash . . . . .	18
2.4.3	Similarity Functions for PHash . . . . .	19
2.4.4	Hash Algorithms and Performance . . . . .	19
2.4.5	PHash in Security Field . . . . .	22
2.5	Data Augmentation . . . . .	23
<b>3</b>	<b>Related Work</b>	<b>25</b>
3.1	Methods for Mitigating MIA . . . . .	25
3.1.1	Confidence Score Masking . . . . .	25
3.1.2	GAN . . . . .	26
3.1.3	DP . . . . .	27
3.1.4	Regularization . . . . .	28
3.1.5	Knowledge Distillation . . . . .	28
3.1.6	Data Augmentation . . . . .	29
3.1.7	Other Methods . . . . .	29
3.2	Summary and Insights . . . . .	30
<b>4</b>	<b>Architecture</b>	<b>33</b>
4.1	Step1: Query Image Check . . . . .	33
4.2	Step2: Data Augmentation . . . . .	35
4.3	Step 3: Image Fusion with PCA Composite Image . . . . .	36
4.4	Step 4: Attacker Performing MIA . . . . .	37
4.5	Summary . . . . .	37

<b>5</b>	<b>Implementation</b>	<b>39</b>
5.1	Setup . . . . .	39
5.2	Dividing the Dataset Evenly into Two Halves. . . . .	40
5.3	Selection of Classification Models . . . . .	41
5.4	Data Preprocessing . . . . .	42
5.5	Model Training . . . . .	43
5.6	Obtaining Prediction Vectors after Training . . . . .	43
5.6.1	ML Scenario . . . . .	44
5.6.2	DFL Scenario . . . . .	44
5.7	Performing MIA . . . . .	44
5.7.1	Binary Classifier Based MIA . . . . .	45
5.7.2	Metric Based MIA . . . . .	45
5.8	Finding Appropriate Defense Intensity . . . . .	46
<b>6</b>	<b>Evaluation</b>	<b>51</b>
6.1	Setup and Experiment Overview . . . . .	51
6.1.1	ML Pipeline . . . . .	52
6.1.2	Evaluation of DFL Pipeline . . . . .	55
6.2	Comparison of ML and DFL Pipelines . . . . .	58
6.3	Comparison of Two Shadow Model Configurations . . . . .	59
6.4	Impact of Data Augmentation Intensity . . . . .	60
6.4.1	ML Pipeline . . . . .	60
6.4.2	DFL Pipeline . . . . .	63
6.5	Impact of PCA Composite Image Fusion Weight . . . . .	65
6.5.1	ML Pipeline . . . . .	65
6.5.2	DFL Pipeline . . . . .	68
<b>7</b>	<b>Summary and Conclusions</b>	<b>71</b>
7.1	Summary and Conclusions . . . . .	71
7.2	Future Work . . . . .	72

<b>Bibliography</b>	<b>73</b>
<b>Abbreviations</b>	<b>79</b>
<b>List of Figures</b>	<b>79</b>
<b>List of Tables</b>	<b>84</b>
<b>A Installation Guidelines</b>	<b>87</b>
A.1 one_to_two.py . . . . .	88
A.2 PCA.py . . . . .	88
A.3 target_model.py . . . . .	88
A.4 shadow_result.py . . . . .	89
A.5 random_query.py . . . . .	89
A.6 random_target_train.py . . . . .	90
A.7 random_shadow_train.py . . . . .	90
A.8 performing_defense.py . . . . .	90
A.9 ShadowModelMIA.py . . . . .	91
A.10 ClassMetricMIA.py . . . . .	91
A.11 optimal_defense_intensity.py . . . . .	91
A.12 Summary . . . . .	91
<b>B Experimental Data of Other Four Datasets</b>	<b>93</b>
B.1 Prediction Accuracy of Training Samples and Test Samples . . . . .	93
B.2 Performance of Binary Classifier based MIA . . . . .	94
B.3 Performance of Metric based MIA . . . . .	95
B.3.1 Performance of Prediction Class Confidence based MIA . . . . .	95
B.3.2 Performance of Prediction Class Entropy based MIA . . . . .	96
B.3.3 Performance of Prediction Modified Entropy based MIA . . . . .	97
<b>C Contents of the CD</b>	<b>99</b>

# Chapter 1

## Introduction

With the rapid development of information technology, the number of Internet of Things (IoT) devices has been growing rapidly. This means tons of user data are now being generated on people's phones, watches, home devices and other devices, and these data are private. Traditional Machine Learning (ML) methods require all data on a central location for training, which can easily lead to data privacy concerns.

To address this issue, Google proposed Federated Learning (FL) in 2016[1], which was actually a Centralized Federated Learning (CFL) approach. The core idea is that each participant train the model locally without uploading local dataset to the server, and the central server is only responsible for aggregating the model parameter updates. This not only help protect user privacy but also improves the generalization ability of the model. However, this approach has the risk of a single point of failure at the central server. To deal with the risk, in 2018, Decentralized Federated Learning (DFL) was proposed[2]. DFL allows participants to directly share model updates with each other, reducing dependence on a central server and further protect data privacy.

Although CFL and DFL reduces the risk of data leakage, the training process still leaves behind residual data traces, which can lead to the leakage of sensitive information[3]. Attackers can exploit various information such as gradient changes[4] to carry out Membership Inference Attacks (MIAs). MIA allows adversaries to determine whether a particular data record was part of the training set, thus revealing private information that users may not wish to disclose and potentially profiting from it.

To address the challenges posed by MIAs, researchers have proposed a number of defense mechanisms. Among them, Confidence Score Masking works by modifying the model's prediction vector[5], and Generative Adversarial Networks (GANs) work by generating synthetic data with statistical characteristics similar to the original dataset to train the target model[6]. While these methods are theoretically capable of defending against MIA, they face practical challenges. For instance, Confidence Score Masking reduces the model's prediction accuracy, and GAN-based approaches require high training time and computational cost.

Considering the current situation, this thesis aims to design a new defense method to effectively defend against MIA. The design goal of the new defense method includes three

aspects: (1) achieving effective defense and providing significant protection against MIA; (2) not impacting prediction accuracy and ensuring stable model performance for normal users; and (3) maintaining low computational cost, thus making it suitable for deployment on resource-constrained devices.

## 1.1 Motivation

The rapid growth in the number of IoT devices has brought growing data privacy concerns. In both ML and DFL pipelines, attackers can perform MIAs to extract users' private data.

Given the practical limitations of existing defense methods, it is necessary to develop a new defense approach that has low computational cost and can effectively defend against MIA without sacrificing model performance, making it suitable for resource-constrained environments with high privacy requirements.

## 1.2 Description of Work

The main goal of this thesis is to design, implement, and evaluate an MTD-based defense method to protect ML and DFL pipelines from MIA.

To achieve this goal, the following tasks were completed:

First, background knowledge were introduced, including FL, MIA, the principles behind moving target defense (MTD), as well as techniques such as perceptual hash (pHash) and data augmentation.

Second, the thesis explored and critiqued existing MIA defense methods. Through a comparative study of current literature, advantages and disadvantages in existing methods are identified, leading to the development of a new defense method based on MTD.

Third, target model were trained on several datasets, including CIFAR-10, CIFAR-100, Fashion MNIST, Tiny ImageNet, and ImageNet-10. From these models, prediction outputs were obtained to support the execution of both binary classifier based MIA and metric based MIA.

Finally, a series of evaluation experiments across all datasets with and without defense were executed. The analysis included comparison of prediction accuracy and F1-scores of the MIA. Each dataset was subjected to a different level of defense intensity, which was an appropriate defense method for this dataset. Besides, a script to automatically find an appropriate defense intensity for each dataset was developed.

## **1.3 Thesis Outline**

The remainder of this thesis encompasses the following content: Chapter 2 introduces the background knowledge including FL, MIA, MTD, pHash and related work. It is very useful for understanding the design philosophy of the implementation. Chapter 3 conducts a comparative analysis of existing defense methods, and compares their advantages and disadvantages. In Chapter 4, the system architecture and its main steps are presented in detail: query image check, data augmentation, image fusion with Principal Component Analysis (PCA) composite image, and attacker performing MIA. Chapter 5 introduces the implementation of the experiment, covering setup, dividing dataset, selection of classification models, data preprocessing, model training and all other steps. Chapter 6 designs related experiments to assess the effectiveness of the thesis, including comparison of ML and DFL pipelines, comparison of two shadow model configurations, analyzing the impact of data augmentation intensity and PCA composite image fusion weight on defense effectiveness. The final chapter, Chapter 7 makes a summary of the thesis and provides a prospectus for future work.



# Chapter 2

## Background

### 2.1 FL

FL was introduced by Google in 2016[1]. It allows participants to train local models without sharing training data, thus better protecting data privacy compared to traditional ML, in which data is centrally collected. The initially proposed FL was CFL, and DFL emerged in 2018[2]. The main difference between DFL and CFL is that DFL does not require a central server, which avoids the Single Point of Failure and thus lowers the associated risks. The differences between traditional ML/DL, CFL, and DFL include mainly three aspects:

- **Data storage:** Traditional ML/DL usually stores data centrally on a server or in the cloud, while FL stores data on local devices.
- **Data sharing:** For traditional ML/DL, raw data from various sources is collected and uploaded directly to the central location for processing. In contrast, FL only shares parameter updates.
- **Risk:** The risk is highest in traditional ML/DL, followed by CFL, with the lowest risk in DFL.

In recent years, the number of FL-related publications has shown exponential growth, increasing from 3 papers in 2016 to approximately 2,871 papers in 2022, achieving a CAGR of around 200%[2]. The applicable scenarios have also expanded, including IoT, healthcare, and more, which is precisely why this paper chooses FL as one of the pipelines. Next, this thesis will discuss the work mechanisms of CFL and DFL separately.

There are 5 steps in the CFL training process, as shown in Figure 2.1:

1. **First step:** Each participant's local model is updated based on local data.
2. **Second step:** After local training is completed, each participant uploads the trained model updates to the central server.
3. **Third step:** The central server collects model updates from all participants and updates the global model using an aggregation method, such as weighted or simple averaging.
4. **Fourth step:** After the global model is updated, the central server sends the global update back to all participants. Each participant uses the received global update for the next round of local training.
5. **Last step:** After receiving the global update, participants apply it to the new round of local training.

This process can be repeated multiple times, and the model's performance gradually improves until the model reaches the desired accuracy or meets other stopping criteria.

There are 4 steps in the DFL training process, as shown in Figure 2.1::

1. **Firstly**, each participant trains the model using their own local raw data and obtains model parameters.
2. **Secondly**, after local training is completed, each participant shares their model parameter updates with other participants.
3. **Thirdly**, after receiving model parameter updates from other participants, each participant aggregates these updates locally. The aggregation methods include simple averaging, weighted averaging, and so on.
4. **Fourth step:** The aggregated model parameters are shared with other participants. This ensures that all participants have a synchronized and up-to-date view of the model.

These 4 steps will be repeated multiple times until the model converges or meets some stopping criteria.

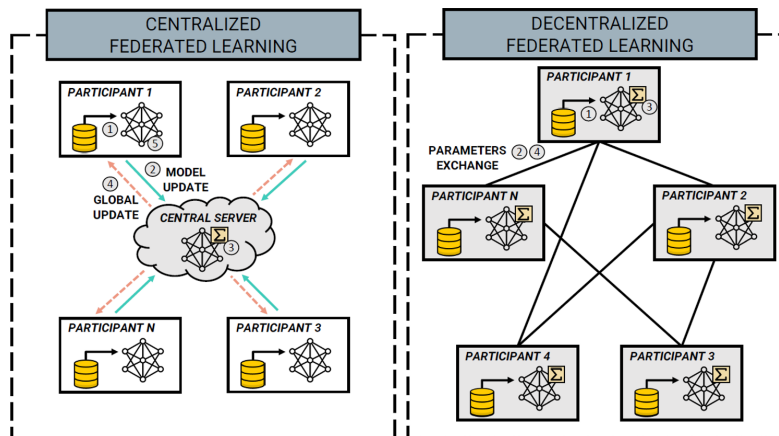


Figure 2.1: Work Mechanism of FL[2]

From the training process of CFL and DFL, the main difference between them is that CFL has a central server, while DFL does not. In CFL, the central server updates the global model and sends the global update back to participants for their local training, while in DFL, participants share model parameter updates with each other and aggregate these updates locally. Moreover, depending on the different DFL network topologies, the number of other participants a participant is connected to varies, resulting in different amounts of model parameter updates received. In theory, under the same conditions, the results obtained in DFL with Fully Connected Networks can be the same as in CFL. However, compared to various Partially Connected Networks (e.g., star-structured network, ring-structured network, random network), in Fully Connected Networks, each participant can obtain the complete model update information, though the communication overhead is very high.

## 2.2 MIA

### 2.2.1 Overview of MIA

With the widespread application of ML, studying its impact on security and privacy has become increasingly urgent[7]. Data privacy breaches can be caused by various attacks, including MIA[8], Attribute Inference Attack[9], Property Inference Attack[10], and Model Extraction Attacks[11]. Attackers can draw sensitive inferences from individuals' data and profit from it[12].

MIA targeting ML models were first designed and implemented by Shokri, Reza, et al, who pointed out that attackers can infer whether a specific data point was part of the model's training dataset based on the model's output, thereby causing privacy leakage[13]. A more precise definition of MIA was provided by Hu, et al: Given an exact input  $\mathbf{x}$  and access to the learned model  $f(\mathbf{x}; \theta^*)$ , an attacker infers whether  $\mathbf{x} \in D_{\text{train}}$  or not[8].

MIA not only successfully targets ML models but also poses a threat to FL. FL shares model updates or gradients without sharing the original data, reducing the risk of privacy breaches[14]. However, it still cannot fully resist MIA. The first MIA against FL was introduced by Melis et al[15], followed by researchers proposing more MIAs against FL[16][17]. MIAs against FL come in two different forms. The first is a passive approach that follows the FL protocol to infer the membership of a data record[18], meaning it does not alter the normal functioning of FL and instead passively collects and analyzes information to conduct the attack. The second approach is that attackers actively tamper with the FL training process to achieve better attack performance[4]. By tampering with the FL training process, more sensitive information about the training dataset is exposed.

A primary factor that contributes to the success of MIA is the target model's overfitting[19]. Overfitting means the target model not only learns the main features of the data but also memorizes details and random errors within the data. This results in the model having very low error on the training data, but its performance on new data can deteriorate significantly because the noise and incidental details are not generalizable[20].

As a result, the target model performs very well on the training data but poorly on test data. The attacker can exploit this discrepancy between the model’s performance on the training data and the test data to infer whether an input record is part of the training dataset.

## 2.2.2 Adversarial Knowledge

MIA can be conducted under two different settings: white-box and black-box. In a white-box setting, the attacker can obtain much more information compared to a black-box setting.

As shown in Figure 2.2, in a white-box setting, an attacker has access to all information about the target model, including all knowledge of training data and all knowledge of the target model[21]. In contrast, in a black-box setting, the attacker only receives the prediction output of the target model; the model’s architecture and weights are not available. The prediction output consists of three types: full confidence scores, top-K confidence scores, or prediction labels[21].

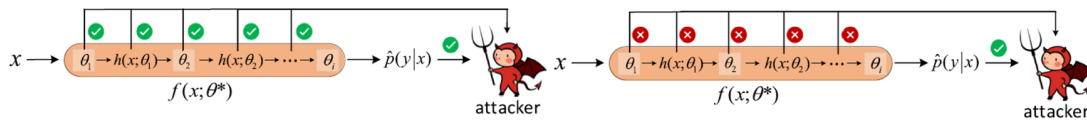


Figure 2.2: White-box and Black-box Setting of MIA[8]

## 2.2.3 MIA Approaches

There are two different MIA approaches: binary classifier based MIA and metric based MIA. Both methods rely on the model’s output, and they infer whether a particular data point belongs to the model’s training set by analyzing the behavior of the ML model.

### (1) Binary Classifier Based MIA

The most widely used approach for training a binary classifier based MIA is shadow training[13]. The process of using shadow training to train a binary classifier based attack model involves three steps, as illustrated below:

1. The attacker uses shadow training datasets to train multiple shadow models. These shadow training datasets share the same distribution as the training data of the target model that the attacker aims to attack, and the shadow models use the same learning algorithm as the target model. The more shadow models are trained, the more accurate the attack model becomes, as more simulated data can be generated for training the attack model[13]. This step has an assumption: knowledge of the training data and the target model is available to the attacker.

2. After the shadow models are trained, the attacker first takes samples from the shadow training datasets as input to the shadow models and labels the resulting output as ‘member’. Then, the attacker takes samples from the shadow testing datasets as input to the shadow models and labels the resulting output as ‘non-member’. The output differs under black-box and white-box settings, as stated in the section 2.2.2. In a black-box setting, the output is the prediction vector of the input data record, whereas in a white-box setting, the output includes the prediction vectors as well as the intermediate computations for each data record.
3. The ‘member’ and ‘non-member’ samples obtained in step (2) are then used as input to the binary classifier, which is trained to distinguish between ‘member’ and ‘non-member’ samples.

Once the binary classifier is obtained, the attacker can use it as the attack model to infer whether a given data sample belongs to the training data of the target model. The attacker queries the target model with a data sample, obtains the output from the target model, and then uses this output as input to the binary classifier. The binary classifier will then produce an output indicating whether the data sample is a ‘member’ or a ‘non-member’. A ‘member’ indicates that the data sample is part of the target model’s training dataset, while a ‘non-member’ indicates that it is not.

### **(2) Metric Based MIA**

Similar to binary classifier based MIA, metric based MIA also infer whether a specific input record belongs to the training set based on the model’s output information, utilizing the differences in the model’s behavior between training and test data[8]. The key difference is that metric based MIAs calculate a metric for a given input record and directly compare it to a predetermined threshold to determine whether the input record is a ‘member’ or ‘non-member’.

There are four main types of metric based MIAs: prediction correctness based, prediction loss based, prediction confidence based, and prediction entropy based attacks. These four methods share a common assumption, which is that the target model performs differently on training data compared to testing data. Specifically, the target model often performs better on training data, manifesting in higher prediction accuracy, lower loss, higher confidence, or lower prediction entropy. Furthermore, all four methods use a threshold to distinguish between members and non-members. In the first method, the model returns 1 or 0 to indicate membership status, which can also be categorized as a threshold-based approach. The distinction among these four methods lies in the different aspects of the model’s output that they focus on, as shown in the Table 2.1 below.

### **(3) MIA via other Attacks**

Some researchers have found that leveraging other types of attacks can assist attackers in conducting membership inference, inferring whether a specific input record belongs to the training dataset.

Hu et al.[8] proposed a novel method that utilizes backdoor attack techniques, enabling data owners to detect whether their data has been used by an unauthorized party for

training a model. This method can be divided into three steps: First, the data owner adds specific marks, known as backdoor triggers, to a small portion of data samples for later inference. Next, if an unauthorized party uses these marked data to train a deep neural network model, the model will become infected with the backdoor. Finally, the data owner can infer that the unauthorized party's target model has used their data for training through the backdoor trigger. The data owner inputs the marked data into the target model, and if the model has been infected with the backdoor, it will exhibit abnormal behavior on the marked data.

ZHANG et al.[22] leveraged adversarial robustness to conduct membership inference. Member data tends to have stronger robustness against adversarial perturbations compared to non-member data, meaning that larger perturbations are needed to cause misclassification. Therefore, there is a difference in the direction and magnitude of adversarial perturbations between member and non-member data. Using this difference, researchers can more effectively separate member data from non-member data. They demonstrated the effectiveness of this method on multiple datasets, and it outperformed the traditional metric based MIAs mentioned above under various settings.

Chen et al.[23] explored how data poisoning attacks can enhance the effectiveness of MIAs. The article proposed two types of data poisoning attacks: The first is the dirty-label Attack, which modifies the labels of the targeted class, making the model more prone to overfitting and thereby increasing the success rate of MIA. The second is the clean-label Attack, which exploits one vulnerability of deep feature extractors. Without changing the labels, it only makes subtle modifications to the samples. The clean-label Attack actually mounts a dirty-label poisoning attack in the feature space, which is more covert than the dirty-label attack.

#### **2.2.4 MIA against FL**

MIA not only successfully attacks ML models but also FL. This is because attackers can conduct membership inference by analyzing the local model updates of participants in FL. However, compared to the volume of research on MIA against ML, the number of studies on MIA against FL is relatively small.

The first MIA against FL was proposed by Melis et al. in 2019[15]. The authors experimentally demonstrated that FL models leak unintended features, which attackers can exploit to perform membership inference. Specifically, the researchers observed the gradient changes in the embedding layer during the FL training process. For training data, the model shows specific gradient changes, while for non-training data, the way the gradient changes differ. Attackers can use this difference to determine whether some input record is part of the training dataset.

	<b>Principle</b>	<b>Metric Calculation Formula</b>	<b>Threshold</b>	<b>Basis for Judgment</b>
<b>Prediction Correctness Based MIA</b>	The target model is overfitting, resulting in higher prediction accuracy on the training data.	$M_{\text{corr}}(\hat{p}(y \mathbf{x}), y) = 1 (\arg \max \hat{p}(y \mathbf{x}) = y)$	1 or 0	If the returned value is 1, the input record is a member of the training data; if it is 0, the input record is a non-member.
<b>Prediction Loss Based MIA</b>	The target model is overfitting, resulting in lower loss on the training data.	$M_{\text{loss}}(\hat{p}(y \mathbf{x}), y) = 1 (L(\hat{p}(y \mathbf{x}); y) \leq \tau)$	$\tau$	If the model's loss value for a given input is less than or equal to the threshold $\tau$ , the input record is considered a member of the training data; otherwise, it is a non-member.
<b>Prediction Confidence Based MIA</b>	The target model is overfitting, resulting in higher confidence on the training data.	$M_{\text{conf}}(\hat{p}(y \mathbf{x})) = 1 (\max \hat{p}(y \mathbf{x}) \geq \tau)$	$\tau$	If the model's maximum confidence for a given input is greater than or equal to the threshold $\tau$ , the input record is considered a member; otherwise, it is a non-member.
<b>Prediction Entropy Based MIA</b>	The target model is overfitting, resulting in lower entropy on the training data.	$H(\hat{p}(y \mathbf{x})) = -\sum_i p_i \log(p_i)$	$\tau$	If the model's entropy value for a given input is less than or equal to the threshold $\tau$ , the input is considered a member; otherwise, a non-member.

Table 2.1: Four Types of Metrics[8]

MIAs against FL can be divided into passive and active attack approaches. The passive approach follows the FL protocol to infer the membership of a data record, meaning the attacker makes observations without modifying the learning process. For instance, Nasr et al.[4] proposed a passive white-box MIA, where the attacker infers whether a sample was involved in training by examining the gradient changes at different data points. The

gradient updates for member data tend to be smaller, while the gradient updates for non-member data are typically larger at these data points.

Active MIAs, on the other hand, tamper with the FL training to achieve better attack performance. Nasr et al.[4] also proposed an active MIA, where the attacker adjusts the model parameters via gradient ascent to increase the loss for a target data point. This adjustment is in the opposite direction of the normal stochastic gradient descent (SGD) and aims to amplify the model’s response to the target data. For members, the model compensates for gradient ascent attacks by reducing the loss. By comparing the gradient norms, the attacker can clearly observe a distinction: the gradient norms for members remain relatively stable, while the norms for non-members increase significantly. This difference allows the attacker to perform membership inference.

Similar to MIAs against ML, MIAs against FL also include binary classifier based MIAs, though the details of the methods differ. Melis et al.[15] used a binary classifier in the FL setting to infer whether an input record was part of the training dataset. During training, the input to the binary classifier is the gradient updates of members and non-members, corresponding to updates generated by data with or without certain attributes during the model’s training process. Gu et al.[17] collected the confidence scores of members and non-members over multiple rounds of FL and used these confidence score sequences for subsequent binary classifier training. In contrast, the input during the training phase of Binary Classifiers in the ML setting, as mentioned in Section 2.2.3, is the prediction vectors of members and non-members. As for FL-specific metric based MIAs, research in this area is relatively scarce.

It is worth noting that MIA aims to determine whether a data point belongs to the entire FL model’s dataset, rather than the dataset of a specific participant. Hu et al. proposed Source Inference Attack (SIA) to infer which participant an input record belongs to[24]. SIA is based on a Bayesian inference framework, where the core idea is to compare the prediction losses of participant models. For each participant, the server calculates the model’s loss on a given input record. If a participant’s loss is significantly lower than that of others, the data is more likely to belong to that participant.

Additionally, the existing literature on MIAs against FL can be divided into CFL and DFL categories. Most research focuses on CFL, while there are relatively fewer studies on DFL.

## 2.3 MTD

### 2.3.1 Emergence of MTD

Traditional defense mechanisms, also known as static defense mechanisms, have some advantages including simplicity and low cost. However, they struggle to address the continually evolving and changing attacks. Once attackers discover a vulnerability in a static

defense system, they can exploit this vulnerability to install backdoors in the system, enabling multiple attacks thereafter[25]. The root of this issue lies in the asymmetry between attackers and defenders. Attackers have advantages over defenders in both information and time[26]. In terms of information, attackers only need to find one exploitable vulnerability to launch an attack, while defenders must secure all vulnerabilities to fully protect against an attacker. In terms of time, attackers often have plenty of time to probe the system for vulnerabilities until they successfully exploit them[27].

To address this problem, researchers introduced MTD, which was first introduced as one of the revolutionary themes for network security defense in 2009[28]. The concept draws inspiration from other fields, such as biology, where animals like mimic octopuses and chameleons can change their colors in real time according to their environment to avoid predators[29]. Similarly, the core idea of MTD is to continually change attack surfaces such as system configurations, making it difficult for attackers to capture an attacking target, thereby improving system security.

### 2.3.2 Definition and Workflow of MTD

Before introducing the definition of MTD, it is necessary to explain the concept of attack surface. Manadhata et al. introduced the term attack surface to describe the exploitable components in a computer system[30].

The concept of MTD was defined by the Federal Cybersecurity Research and Development Program in 2014: MTD enables us to create, analyze, evaluate, and deploy mechanisms and strategies that are diverse and that continually shift and change over time to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency[31].

The workflow of an MTD system is illustrated in the following Figure 2.3. It shows how an MTD system adapts to its environment. First is the initial deployment, representing the state of the system when it is first deployed. Next, the MTD system chooses an adaptation to adjust the system configuration. The timing of this choice can be fixed, random, or triggered by external information. After deciding on an adaptation, the system evaluates the effectiveness of the adaptation based on constraints and resource limitations. If the adaptation is effective, it will be implemented. If not, the MTD system will choose a new adaptation. After the adaptation is implemented, the system enters a delay phase until the next timing of choice.

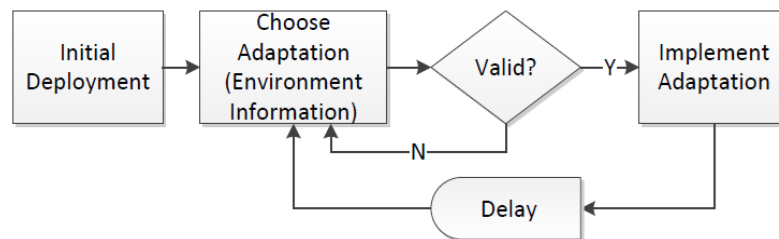


Figure 2.3: Workflow of an MTD System [25]

### 2.3.3 Three Elements for an MTD Technique

There are three elements for an MTD technique[26], which are summarized in the table 2.2 in terms of their classification and explanation. Below is a supplementary introduction:

- **WHAT to move:** This refers to the moving parameter. MTD systems can reduce an attacker's chances by modifying parameters including Dynamic Data, Dynamic Software, Dynamic Runtime Environment, Dynamic Platforms, and Dynamic Networks.
- **HOW to move:** This refers to the way to move. There are three types of MTD techniques based on the nature of operations: shuffling, diversity, and redundancy[32]. Redundancy requires additional system components, while both shuffling and diversity leverage legacy devices or technologies. Due to this, redundancy offers stronger reliability and availability, whereas shuffling and diversity might be limited by the inherent vulnerabilities of existing technologies. Shuffling, diversity, and redundancy can be combined in pairs or altogether to form hybrid MTD approaches. Hybrid MTD approaches offer defensive advantages that single techniques cannot provide, significantly increasing security. However, they also require higher costs and face more operational constraints.
- **WHEN to move:** This refers to the timing of executing MTD operation[32]. There are three types of timeliness-based MTD approaches briefly introduced in the table. For the time-based approach, determining the interval time is crucial. If the interval too long, the attacker might have enough time to successfully attack the system. But if it's too short, it could hurt system performance and increase costs. On the other hand, the key to the event-based approach is accurately predicting potential types of attacks. Ideally, if the defender can accurately predict all types of attacks, the MTD system can trigger corresponding MTD operations for each type of attack. As a result, the system will be able to successfully defend against all attackers.

### 2.3.4 Other Classifications of MTD

In the section 2.3.3, the three classifications of MTD have actually already been introduced, which are the what, how, and when. Next, the article will introduce another classification method—based on architecture. MTD can be divided into proactive defense architecture, reactive defense architecture, and architecture with evolvability, which combines the advantages of both proactive defense and reactive defense[28].

Proactive MTD is a continuous process. It involves two steps: the first is strategy generation, and the second is defense deployment, as shown in the Figure 2.4. By randomly switching and altering system configurations, it ensures that the system remains unpredictable over time, thus preventing attackers from having enough time to study and exploit system vulnerabilities.

	<b>Classification</b>	<b>Explanation</b>
<b>What to move</b>	Dynamic Data	Dynamic data-based MTD aims to dynamically alter the representation of data.
	Dynamic Software	Dynamic software-based MTD involves modifying an application's code dynamically[33] to enhance the heterogeneity of software applications while maintaining their functionality.
	Dynamic Runtime Environment	Dynamic runtime environment-based MTD focuses on dynamically altering the execution environment to increase unpredictability.
	Dynamic Platforms	Dynamic platform-based MTD dynamically changes platform properties including the operating system version, CPU architecture, virtual machine instance, storage system, communication channel, and so on.
	Dynamic Networks	Dynamic network-based MTD focuses on dynamically changing network properties including protocols, IP address, port numbers, network topology, communication and so on.
<b>How to move</b>	Shuffling	This technique rearranges or randomizes system configurations, such as mutating IP addresses at a TCP/IP layer or dynamically adjusting the migration time of VMs[34].
	Diversity	This technique employs the deployment of system components with different implementations that provide the same functionalities[34].
	Redundancy	This technique provides multiple replicas of system components, such as multiple paths between nodes in a network layer or multiple software components providing the same functionality at the application layer[34].
<b>When to move</b>	Time-based	This approach triggers an MTD operation based on a certain time interval called the MTD interval, which can be a fixed interval or a variable interval[32].
	Event-based	This approach performs an MTD operation only when a certain event occurs[32].
	Hybrid	This approach combines the time-based approach and the event-based approach mentioned above[32].

Table 2.2: Three Elements of MTD

Reactive MTD is an iterative process. It consists of three steps: effectiveness evaluation, strategy generation, and defense deployment. The MTD system continuously analyzes the network state and network resource vulnerabilities to assess the effectiveness of the defense strategies. As a result, it continuously adjusts its strategies and deployments to prevent further attacks, and these three steps dynamically cycle, as shown in the Figure 2.4.

MTD architecture has gradually evolved into architecture with evolvability, which balances the proactiveness of proactive MTD with the pertinence of reactive MTD, providing the MTD system with enhanced security and flexibility[28].

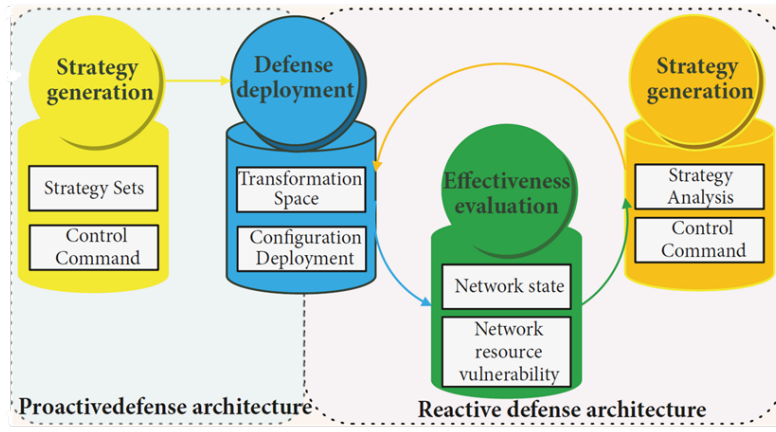


Figure 2.4: Proactive and Reactive Architecture of MTD[28]

### 2.3.5 Metrics for Measuring MTD Effectiveness and Efficiency

There are many metrics currently used to evaluate the effectiveness and efficiency of MTD techniques, as shown in Table 2.3, but there are no standard metrics yet.

Metrics for measuring MTD effectiveness can be divided into three categories:

- **Attacker's metrics:** These measure the degree to which the MTD technique weakens the attacker's offensive capabilities.
- **Defender's metrics:** These evaluate the effectiveness of the MTD technique from a defensive perspective.
- **Others:** In addition to the metrics from the attack and defense perspectives, there are other metrics that can measure the effectiveness of MTD techniques, such as periodicity and uniqueness.

Metrics for measuring MTD efficiency also include two categories: attacker's metrics and defender's metrics, as shown in Table 2.4. The former assesses the cost the attacker must incur to successfully carry out an attack, while the latter evaluates the impact of implementing MTD techniques on system resources, performance, and other factors related to system sustainability.

	<b>Metric</b>	<b>Explanation</b>
attacker's metrics	Attack success probability	The probability that attacks are successfully performed.
	Attack utility	the payoff or utility of an attacker
	Learning by attackers	the degree of an attacker's learning toward the payoff obtained by a defender upon the performed attack.
	Mean time to compromise a system	The time an attacker takes to compromise an entire system.
	Unpredictability	The confusion or uncertainty a given MTD has introduced to attackers.
	Attack surface	The amount of system resources that can be used by attackers to attack the system.
defender's metrics	Defense success probability	The detection accuracy of anomaly behaviors.
	Mean time to failure	The system's up-time in the presence of attacks.
	Defense utility	The payoff or utility of the defender
	Learning by defenders	The degree of a defender's learning toward the payoff an attacker has obtained upon a defense action taken by the defender.
	System security	The system security properties enhanced by proposed MTD techniques
others	Controllability	The portion of critical system assets that expose a high vulnerability to an attacker if compromised.
	Worm propagation speed	how much a deployed MTD can slow down actions by an attacker.
	Vastness	the size of spaces that a given defense mechanism can cover.
	Periodicity	how often system configurations change in order to provide a sufficient level of confusion to attackers.
	Uniqueness	How uniquely an individual entity is authorized to a system without being accessed by other entities.
	Revocability	the degree of frequency to terminate or expire a prior system configuration.
	Distinguishability	How well a given defense distinguishes trustworthy entities from non-trustworthy entities.
	Loss in rewards between an optimal deployment and an executed deployment	how much loss occurred for the actual execution of an MTD operation over the optimal deployment.

Table 2.3: Metrics for Measuring MTD Effectiveness

	<b>Metric</b>	<b>Explanation</b>
Attacker's Metrics	Penalty in attack payoff	Attack cost at an abstract level.
	Attack cost	How much overhead or impact is introduced to attackers to perform their attacks.
Defender's Metrics	Quality-of-Service to users	The degree of service quality provided to users while implementing a given MTD technique.
	System performance	How much overhead is introduced to deploy a given MTD, such as message overhead.
	Defense cost	Migration cost or maintenance cost of VMs.

Table 2.4: Metrics for Measuring MTD Efficiency

## 2.4 Hash

### 2.4.1 Hash Function

Hash function maps original data records to a lower dimensional space. Different data records may result in the same hash value; therefore, the main challenge of hash function is to minimize collisions and to handle collisions[35].

In the case of cryptographic hash functions, an additional property is that after computing the hash and obtaining the value  $h$ , it becomes computationally infeasible to reverse the process and retrieve the original message  $m$  from the hash[36]. This makes cryptographic hash functions particularly useful in secure applications, where data integrity and privacy are critical.

One drawback of hash functions is that even small changes in digital data lead to completely different hash values. For this reason, researchers have proposed pHash, which will be introduced in the next section 2.4.2.

### 2.4.2 PHash

Perceptual image hash functions generate hash values based on an image's visual features. These functions produce similar hash values for visually similar images, while yielding significantly different values for dissimilar images. By using an appropriate distance function, one can determine if two images are perceptually similar or distinct[37].

Cryptographic hash functions can be used to verify exact file matches. However, in the image domain, different forms of images can represent the same content, such as different

versions of a number or an animal. These different forms will generate distinct cryptographic hash values. In such cases, only perceptual image hash can solve the problem by checking whether the content of different images is the same.

PHash functions are widely applied in practical uses like detecting image spam, searching for copyright violations, and maintaining illegal content databases[37].

### 2.4.3 Similarity Functions for PHash

PHash functions produce similar hash values for visually similar images. To measure the similarity between two images, researchers have proposed and used several metrics, among which the most common are the Hamming distance[38], Bit Error Rate (BER)[39], and Peak of Cross Correlation (PCC).

- **Hamming Distance:** The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different[40]. It can be defined as follows: let the two strings be  $x = (x_1, \dots, x_n)$  and  $y = (y_1, \dots, y_n)$ , then the Hamming distance is given by  $\Delta(x, y) := \sum_{x_i \neq y_i} 1$ .
- **BER:** BER is defined as the ratio of the number of bit errors in the pHash to the total length of the pHash[39]. The number of bit errors is equal to the Hamming distance of the pHash values.
- **PCC:** Unlike Hamming Distance and BER, which measure the distance between two hash values, PCC measures the similarity between two hash values. Cross-correlation is calculated by sliding one serie relative to another and computing the correlation at each position. PCC is the maximum correlation that can be achieved between these two series.

### 2.4.4 Hash Algorithms and Performance

Hash Algorithms include various types such as Block Mean Hash. Next the thesis will introduce the principles and time complexities of each type.

- **PHash Algorithm:** pHash is driven by the discrete cosine transformation (DCT).

The process of pHash involves resizing the image to  $32 \times 32$  pixels and converting it to grayscale. The image then undergoes a DCT, where the top  $8 \times 8$  low-frequency components are extracted. The median of these DCT coefficients is calculated, and a binary hash is generated by comparing each coefficient to the median—if it's higher, the value is set to 1; if lower, it's set to 0.

**Time Complexity:** The time complexity of pHash is  $O(n \log n)$ , where  $n$  represents the number of pixels in the image. This is because the time complexity of DCT is  $O(n \log n)$ . The core segment of DCT is shown in 2.1, where in line 3, each row of the image undergoes a one-dimensional DCT. Similar to the Fast Fourier Transform

(FFT), large data is recursively divided into smaller computations. The number of recursive levels is  $\log n$ , and each level requires a linear traversal of all pixels, resulting in a time complexity of  $O(n \log n)$ .

```

1 import numpy as np
2 from scipy.fftpack import dct
3 def dct_2d(image):
4     dct_rows = dct(image, axis=0, norm='ortho')
5     dct_cols = dct(dct_rows, axis=1, norm='ortho')
6     return dct_cols

```

Listing 2.1: Core Segment of DCT

- **Average Hash (aHash):** The working principle of aHash involves resizing the image to  $8 \times 8$  pixels and converting it to grayscale. The average pixel value is then calculated, and a binary hash is generated by comparing each pixel to this average. This method is simple and efficient, offering robustness against minor modifications to the image, such as changes in brightness or color.

**Time Complexity:** The time complexity of aHash is  $O(n)$ . As shown in the code snippet 2.2, lines 4 to 7 perform a linear operation on each pixel, thus the time complexity is  $O(n)$ .

```

1 import cv2
2 import numpy as np
3 def average_hash(image_path, hash_size=8):
4     img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
5     img_resized = cv2.resize(img, (hash_size, hash_size))
6     avg = np.mean(img_resized)
7     hash_array = img_resized > avg
8     return hash_array.flatten()

```

Listing 2.2: Core Segment of aHash

- **Difference Hash (dHash):** dHash works by calculating the differences between neighboring pixels to generate a binary hash. It resizes the image to  $9 \times 8$  pixels, converts it to grayscale, and compares adjacent pixel values in each row. Each row of 9 pixels produces 8 difference values, creating a 64-bit hash that reflects the image's structure.

**Time Complexity:** The time complexity of aHash is also  $O(n)$ . In the following code snippet 2.3, lines 2, 3, 4, and 5 respectively perform image reading, image resizing, calculating the differences between adjacent pixels, and flattening the array. Since these steps only require a linear traversal of the image pixels, the overall time complexity is  $O(n)$ .

```

1 def dhash(image_path, hash_size=8):
2     img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
3     img_resized = cv2.resize(img, (hash_size + 1, hash_size))
4     diff = img_resized[:, 1:] > img_resized[:, :-1]
5     return diff.flatten()

```

Listing 2.3: Core Segment of dHash

- **Wavelet Hash (wHash):** wHash resizes the image to  $8 \times 8$  pixels and converts it to grayscale. Afterward, the Discrete Wavelet Transform (DWT) is applied to extract features from the frequency domain.

**Time Complexity:** The time complexity of wHash is  $O(n \log n)$  because it uses DWT. DWT recursively decomposes the image in a layered manner, where  $\log n$  represents the total number of layers obtained after recursion, and  $n$  refers to the total number of pixels processed in each layer. The core code of DWT 2.4 is shown below:

```

1 import pywt
2 import numpy as np
3 def dwt_image(image):
4     coeffs = pywt.dwt2(image, 'haar')
5     cA, (cH, cV, cD) = coeffs
6     return cA, cH, cV, cD

```

Listing 2.4: Core Segment of DWT

- **Block Mean Hash:** The working principle of Block Mean Hash is to first divide the image into several small blocks and calculate the average pixel value for each block. By comparing each block's average with the overall image's average, a binary hash is generated. If the block's average is higher than the global average, it's marked as 1, otherwise as 0.

**Time Complexity:** The time complexity of Block Mean Hash is  $O(n)$  because it only needs to divide the image into blocks and calculate the mean pixel value for each block, which is very simple.

- **Marr-Hildreth Operator-based Hash:** The Marr-Hildreth Operator-based Hash method involves first using Gaussian blur to denoise the image, followed by applying the Laplacian operator to detect the edges. Finally, a hash is generated from the processed edge information, typically by comparing the brightness of the edge pixels with the average, creating a binary hash.

**Time Complexity:** The time complexity of it is  $O(nk)$  because, in the following code snippet 2.5, line 5 performs Gaussian blurring (a convolution operation), which has a complexity of  $O(n \cdot k)$ , where  $k \times k$  is the size of the convolution kernel.

```

1 import cv2
2 import numpy as np
3 def marr_hildreth_hash(image_path, hash_size=8):
4     img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
5     blurred = cv2.GaussianBlur(img, (5, 5), 1.0) # k = 5
6     edges = cv2.Laplacian(blurred, cv2.CV_64F)
7     resized = cv2.resize(edges, (hash_size, hash_size))
8     avg = np.mean(resized)
9     return (resized > avg).flatten()

```

Listing 2.5: Core Segment of Marr-Hildreth Operator-based Hash

- **Radial Variance Hash:** The working principle of Radial Variance Hash involves analyzing the radial symmetry of an image to generate a hash. It transforms the

image into polar coordinates and calculates the variance along radial lines from the center of the image. By comparing the radial variances to the global average, a binary hash is generated.

**Time Complexity:** The time complexity of it is  $O(n)$  because all the steps only require a linear traversal of the image pixels.

In summary, Block Mean Hash, aHash, dHash, and Radial Variance Hash have a linear time complexity of  $O(n)$ , making them faster for processing images. However, pHash and wHash, which use frequency domain or wavelet transforms, have a time complexity of  $O(n \log n)$ , providing better image content recognition, but with slower processing speeds. The time complexity of the Marr-Hildreth Operator-based Hash falls between these two categories.

### 2.4.5 PHash in Security Field

PHash has wide applications in the security field, covering copyright protection, child sexual abuse material (CSAM) detection, and criminal face identification. By providing robust and efficient recognition methods, pHash plays an important role in several critical security scenarios, offering strong technical support to counter various security threats.

- **Copyright Protection**

Copyright protects and encourages new ideas. In 2007, Google introduced the Content ID system to address the unauthorized use of copyrighted video content. The Content ID system compares uploaded video content with copyrighted content in its database to prevent copyright infringement. Thanks to Content ID, copyright holders have received billions of dollars in compensation from infringing parties [41].

Many researchers have also proposed perceptual image hash schemes to address piracy issues. For example, Sun et al. developed a pHash technique based on convolutional neural networks (CNN) to protect image copyrights[42]. They classified training images according to the original images and generated a hash center for each class. CNNs automatically learn the image features, allowing each image's hash code to converge toward the hash center of its respective class. This approach effectively detects and identifies pirated and infringing images, providing a robust and efficient method for copyright protection.

- **CSAM Detection**

Microsoft and Dartmouth College developed PhotoDNA in 2009, a technology based on pHash to help identify and remove CSAM. Today, PhotoDNA is widely used by companies and organizations such as Facebook and Twitter to deal with CSAM, and it has successfully reported millions of child exploitation images[43].

In 2021, Apple developed its pHash system, NeuralHash, to detect CSAM. This system uses Deep Learning (DL) networks to automatically generate hash features and compares them with known CSAM hash databases provided by child safety organizations to identify and prevent the distribution of harmful images. However,

due to concerns that this method might compromise user privacy, it was not installed on client devices[44].

- **Face Identification of Crime Victims**

Face recognition technology made significant breakthroughs in the 1990s[45], and since then, researchers have kept working on ways to make it more accurate. But in real life, criminals often wear things like sunglasses or masks to hide their faces, which makes recognition harder. Most current facial recognition methods cannot effectively handle such obstructions. Biswas et al. proposed a new pHash scheme, named One-Shot Frequency Dominant Neighborhood Structure, which generates features for face verification by extracting the global texture energy and local Dominant Neighborhood Structure of an image, without requiring pre-training. In experiments, an SVM classifier was trained using OSF-DNS features from non-occluded faces, and this classifier was then used to identify occluded faces. This pHash method achieves higher accuracy than other methods in recognizing obscured faces, aiding law enforcement in apprehending criminals and maintaining community security[46].

## 2.5 Data Augmentation

Data augmentation is a data-space solution to the problem of limited data, and it has been widely applied in ML and DL. It generates new samples by applying various data augmentation techniques to existing data samples.

Data augmentation techniques can be categorized into three types: geometric transformations, photometric transformations, and DL based data augmentation. The following paragraph introduce common techniques in each category.

- **Geometric transformations:** Geometric transformations are the most common basic augmentation used[47]. It includes commonly used types such as flipping, cropping, and rotation.
  - **Flipping:** Flip the image in the horizontal direction[48].
  - **Rotation:** Rotate the image at a random orientation[48].
  - **Cropping:** Crop a part from the original image and resize the cropped image to a specific resolution[48].
  - **Shifting:** The image is shifted to the left or right, and the translation range and step length can be manually specified to change the location of the image content[48].
  - **Noise injection:** Add random noise to the RGB channels of each pixel in the image. Commonly used noise is Gaussian noise[48].
- **Photometric transformations:** Photometric transformations alter the RGB channels by shifting each pixel value  $(r, g, b)$  to new pixel values  $(r', g', b')$  according to pre-defined heuristics. This adjusts image lighting and color while leaving the geometry unchanged[49]. Photometric transformations include the following:

- **PCA based Color Augmentation:** It begins by computing the PCA components for all images in the dataset. Once these PCA components are derived, an intensity value is calculated based on them. This intensity value is then added to each channel (R, G, B) of the image. As a result, the color balance of the image is adjusted by altering both the color illumination and intensity[50].
- **Kernel filters:** Kernel filters sharpen and blur images by sliding an  $n \times n$  matrix across them, using either a Gaussian blur filter or a high-contrast vertical or horizontal edge filter. This process alters the luminance or color information of the image[51].
- **DL based data augmentation:** The DL based augmentation approaches can automatically learn the representations of images and generate transformed images. Compared to geometric transformations and photometric transformations, it is able to produce more diverse and realistic images[47].
  - **GAN:** GANs have the ability to generate new samples after being trained on samples drawn from some distribution[52].
  - **Neural Style Transfer:** Neural Style Transfer generates images with new styles while preserving the original content by manipulating the feature layers in the CNN, combining the content of the original image with the style of the target image.

*torchvision* is a Python library, and *torchvision.transforms* offers various data augmentation methods for easy use. Data augmentation methods could be divided into three categories mentioned earlier, and a summary of some of them is shown in Table 2.5 below.

Category	Function
Geometric Transformations	RandomHorizontalFlip
	RandomVerticalFlip
	RandomRotation
	RandomResizedCrop
	CenterCrop
	FiveCrop
	RandomAffine
Photometric Transformations	ColorJitter
	RandomSolarize
	RandomPosterize
	RandomGrayscale
DL Based Data Augmentation	GaussianBlur
	RandomAdjustSharpness

Table 2.5: Summary of Augmentation Methods Provided by *torchvision.transforms*

# Chapter 3

## Related Work

### 3.1 Methods for Mitigating MIA

Many researchers have studied methods to mitigate MIAs [53] [54] [6] [55] [56] [57] [58] [59] [60] [61] [5] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73], primarily including Confidence Score Masking, GAN, Differential Privacy (DP), Regularization, Knowledge Distillation, and Data Augmentation. The following sections will introduce each method in detail, followed by a summary and comparison. This thesis finds that most existing research focuses on static defense methods, with only a few methods falling under MTD.

Additionally, we can observe that many papers adopt a combination of two methods. For example, [3] uses GANs to generate synthetic data to train the target model while incorporating regularization terms during the training process to control the model's complexity.

#### 3.1.1 Confidence Score Masking

Confidence Score Masking refers to hiding the true confidence score of the target model by providing a modified confidence score, making it difficult for the attacker to determine whether the input record belongs to the training dataset based on the given confidence score.

In [53], high-entropy soft labels and entropy-based regularization are employed during the training phase to lower the target class confidence in the labels and penalize high-confidence outputs, respectively. During the testing phase, the confidence score of the input record is replaced with the low confidence score of a random sample. This dual defense effectively resists MIA, thus safeguarding data privacy.

[5] introduces adversarial noise into the model's confidence scores to protect privacy, thereby disrupting the attacker's judgment. Under the constraints of maintaining a certain noise level and label consistency, an optimization algorithm is used to find the optimal

noise vector that maximizes the disruption effect. Adversarial noise is then applied at the output layer of the target model, with random noise added to each input record.

There are two types of confidence score modification. Random modification of confidence scores during the testing phase can be classified as reactive MTD, as it provides the attacker with modified confidence scores, making it challenging to determine if the input belongs to the training set. This aligns with the definition of MTD, where the attacker finds it difficult to predict and locate the target. However, adjusting confidence scores during the training phase does not fall under MTD.

### 3.1.2 GAN

GAN is made up of a generator and a discriminator. The generator produces synthetic data approximating the distribution of the original dataset, and the discriminator distinguishes between the original and generated data. Through continuous adversarial training, the similarity between the generated data and the original data gradually decreases[74]. Using generated samples instead of the original dataset to train the target model, GANs effectively reduce the model's ability to memorize individual samples from the original dataset, thereby enhancing defense against MIA.

[6] also mentions that GANs generate synthetic data with statistical characteristics similar to the original dataset, thus allowing the target model to be trained on synthetic data rather than the original data. This approach reduces the target model's dependency on the original dataset, effectively preventing attackers from inferring specific samples' membership in the training set through MIA, thus improving robustness.

In [56], a novel GAN framework is proposed that first partitions the training dataset into multiple non-overlapping subsets, each of which trains a separate discriminator. The generator then undergoes adversarial training against multiple discriminators, resulting in generated data that no longer closely matches specific features of the training data but instead covers a broader potential data distribution. This method further reduces the target model's reliance on the original data, significantly decreasing the success rate of MIA.

In [65], an integrated privacy discriminator and multi-generator-discriminator architecture is proposed. The training dataset is divided into several subsets, with each subset training a separate generator-discriminator pair. A privacy discriminator is then introduced to determine which generator produced a particular sample, thereby preventing the generators from memorizing training data, as each generator must ensure that its generated samples show no significant difference from those of other generators. As a result, the likelihood of MIA successfully inferring training samples from model memory content is significantly reduced.

In [69], a GAN-based method to defend against MIA is proposed, where synthetic images generated by GANs replace the original data. Additionally, a sample rejection strategy excludes synthetic images with similarity scores exceeding a specified threshold compared to the training data, further minimizing privacy leakage risks.

In [73], DP is applied to the output of the GAN’s discriminator, reducing the similarity between the generated and original data while preserving the authenticity of the generated samples. This method enhances the model’s resistance to MIA.

Using GANs to mitigate MIA is not classified as a MTD method. Once the generated data is used to train the target model, it no longer changes dynamically, making GANs a static defense approach.

### 3.1.3 DP

DP mitigates MIA by adding noise to the gradients or parameter updates of the model, making the influence of individual samples negligible. This prevents attackers from determining whether an input record belongs to the training dataset.

In [70], DP is applied to defend against MIA by training the target model on the original data and then adding Laplace noise during the testing phase. This changes the output that the attacker receives, making it difficult for the attacker to determine if the input record is part of the training dataset.

In [63], a method called Weighted Smoothing is used, and its core idea is to inject stronger noise into higher-risk samples, thus lowering the target model’s confidence in these high-risk samples, while injecting less noise into lower-risk samples. This method involves phased model training with dynamic noise adjustment to ensure that samples at different risk levels receive appropriate privacy protection. It significantly reduces the success rate of MIAs.

In [71], a defense mechanism combining DP with model sparsity is proposed. Gaussian noise is added to the gradient during each model update, reducing the model’s reliance on individual data points. Introducing model sparsity also helps reduce overfitting, thus lowering the model’s memorization of training data.

In [72], a local DP technique is used to protect sensitive information in FL. Participants add noise to their model updates using mechanisms like the Gaussian or Laplace noise mechanisms. Consequently, even if attackers access the noisy gradient information, they cannot reconstruct the original local data, which decreases MIA’s success rate.

In [75], the impact of MIA on differentially private models is analyzed. It employs Abadi et al.’s Differentially Private Stochastic Gradient Descent algorithm [76] to train the model, adding Gaussian noise to each gradient update. The results show that DP provides a certain level of protection against MIA but cannot completely defend against it.

DP can be divided into two types. Adding different noise at each output in the testing phase is a reactive MTD, as it results in varied outputs for the same input record upon multiple queries. This makes it difficult for attackers to determine if the input belongs to the training set, aligning with the definition of MTD. However, adding noise to gradients or parameter updates during the training phase is not considered MTD, as there is no dynamic adjustment once the model enters the testing phase.

### 3.1.4 Regularization

Regularization primarily reduces the model's dependence on and memorization of specific training samples by limiting model complexity, smoothing model outputs, and reducing overfitting. This narrows the output gap between training and test samples, effectively lowering MIA's success rate.

In [66], a regularizer is introduced into the loss function to penalize excessive output distribution differences between training and validation data, reducing the model's memorization of the training data and achieving effective privacy protection.

In [70], L2 regularization is incorporated during training, where model parameters are updated towards zero, preventing excessive model complexity and fitting capacity. This reduces the confidence gap between training and non-training data.

In [62], a method called Adversarial Regularization is proposed, which introduces an adversarial regularization term during training to limit the model's dependency on training data. This method frames the privacy protection problem as a min-max game, where 'min' refers to minimizing the model's classification error on the task, and 'max' refers to maximizing the attacker's loss.

Regularization is applied in the training phase. After training is complete, all input records produce outputs following the same process without any dynamic change, so it is not considered an MTD.

### 3.1.5 Knowledge Distillation

Knowledge Distillation primarily reduces model overfitting. It first trains a large teacher model on the original dataset, where the model's output for each sample is a probability distribution over classes, known as soft labels. The student model is then trained on these soft labels, gaining improved generalization ability and effectively defending against MIA.

The method in [57] is based on knowledge distillation and involves two main steps. First, a teacher model is trained using the original private training dataset to capture the data's characteristics. Then, through knowledge distillation, the knowledge from the original model is transferred to a student model, which is trained not on the original dataset but on a reference dataset with a similar distribution. This allows the new model to learn the predictive features of the original model without directly accessing the original data, thereby reducing privacy leakage risks. This approach achieves a good balance between model accuracy and a lower MIA success rate.

In [64], Repeated Knowledge Distillation is employed, where the trained student model from one round is used as the teacher model for the next distillation round, repeating the knowledge distillation process. With each round, the distance between the student model and the private dataset increases, progressively weakening the model's memorization of training data and reducing the risk of MIAs.

In [77], two methods, Complementary Knowledge Distillation and Pseudo Complementary Knowledge Distillation, are proposed. Both are based on knowledge distillation, and they train the student model using 'complementary' data generated by multiple teacher models, making it difficult for attackers to infer membership information of training data from the model's output.

Knowledge distillation is a static defense method because, in the testing or query phase, there are no dynamic adjustments to parameters or behavior.

### 3.1.6 Data Augmentation

Data augmentation, as a method to defend against MIA, primarily works by increasing the diversity of training image data to enhance the model's generalization ability. This allows the model to learn more generalized features and reduces its memorization of specific samples in the training set, thereby improving the target model's defense against MIA.

In [55], common data augmentation techniques such as Random Cropping and Mixup were selected. These methods applied varying intensities of data augmentation to transform the original data, producing an expanded dataset for training the target model. An interesting finding was observed: low-intensity data augmentation improves model accuracy but is ineffective against MIA, while high-intensity data augmentation reduces MIA success rates but negatively impacts model utility.

Applying data augmentation during the training phase is not classified as MTD because, once the expanded dataset is used to train the target model, it no longer undergoes dynamic changes.

### 3.1.7 Other Methods

In addition to the more commonly used methods for defending against MIA mentioned above, there are some relatively less common methods, which will be introduced individually:

- **Relaxed Training Objective Loss:** In [54], a method called RelaxLoss is proposed, with the core idea of relaxing the training objective loss to narrow the loss distribution gap between training and non-training samples, thereby reducing features that attackers can exploit.
- **Pruning:** In [59], a pruning algorithm is used to reduce model complexity and decrease the model's dependency on specific training data, thereby increasing the difficulty for attackers to succeed.
- **Causal Learning:** In [61], a method based on causal learning is proposed. By modeling the causal relationship between input features and outputs, causal learning reduces the model's sensitivity to specific training data, thus lowering the risk of privacy leakage.

- **Digestive Neural Network (DNN):** In [68], DNNs are applied in FL to defend against MIA. Each participant is equipped with an independent DNN that processes raw data into less identifiable representations, while retaining the data’s classification properties, effectively defending against MIA.

## 3.2 Summary and Insights

The six methods—Confidence Score Masking, DP, Data Augmentation, Regularization, GAN and Knowledge Distillation—can all be applied to target models which use images as input. Additionally, these six methods are often applied to target models for classification tasks to achieve a defensive effect.

These six methods have their own advantages and disadvantages, as shown in Table 3.1 below.

Method	Advantages	Disadvantages
Confidence Score Masking	Easy to implement	Reduces the model’s prediction accuracy
DP	Easy to implement	Rarely offers acceptable utility-privacy tradeoffs with guarantees for complex learning tasks
Data Augmentation	Easy to implement	Defense strength varies with the intensity of data augmentation
Regularization	Easy to implement	Reduces the model’s prediction accuracy
GAN	Avoids directly exposing real training data	High resource consumption and unstable defense effectiveness
Knowledge Distillation	Significantly reduces user information leakage	High training time and computational cost

Table 3.1: Advantages and Disadvantages of Methods for Mitigating MIA

Among them, the first four are easy to implement. However, Confidence Score Masking and Regularization may reduce the model’s prediction accuracy; DP can potentially lead to user privacy leakage; the defense strength provided by Data Augmentation varies with the intensity of the data augmentation.

GAN and Knowledge Distillation avoid information leakage, but they also have the issue of high computational cost.

Based on previous literature, this thesis proposes a detailed and actionable new defense method. This method involves performing data augmentation on the training samples uploaded by the attacker and fusing them with PCA composite images, thereby preventing MIA from distinguishing between training samples and test samples.

This new defense method has the following three advantages:

- **Achieve Effective Defense:** By using this method, the success rate of both binary classifier based MIA and metric based MIA is significantly reduced. In other words, it becomes difficult for the attacker to infer whether a query sample belongs to the training dataset.
- **Does Not Impact Prediction Accuracy:** This new defense method is applied only to query samples that are identified as training samples. That is, all training samples and a small number of misclassified test samples will undergo the defense process, while the vast majority of test samples will not. Therefore, test samples uploaded by regular users will most likely not undergo any defense process, ensuring that their prediction accuracy remains unaffected.
- **Have Low Computational Cost:** The new defense method in this thesis consists of three steps: (1) Determine whether the pHash of the query image is present in the pHash list of the training dataset; (2) Perform data augmentation; (3) Fuse the data-augmented image with a PCA composite image. These three steps are simple computational operations that do not involve complex processes such as training neural networks. Consequently, the new defense method has a low computational cost.



# Chapter 4

## Architecture

This chapter presents the system architecture, which is shown in Figure 4.1. It is based on MIA architecture with the addition of a defense component. Each step in order from the attacker's perspective is described in detail below.

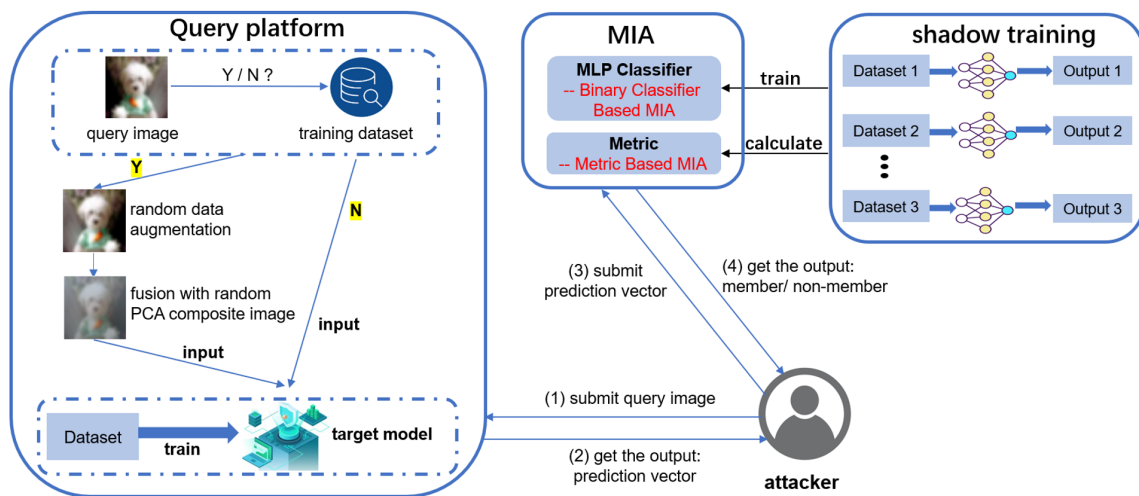


Figure 4.1: Architecture

### 4.1 Step1: Query Image Check

In the first step, the query platform determines whether the query image is in the training dataset.

After the attacker uploads an image to the query platform, the platform will determine whether the query image is present in the training dataset, as shown in Figure 4.2.



Figure 4.2: Determining Whether the Query Image is in the Training Dataset

This thesis applies pHash to determine whether the query image is in the training dataset. The query platform calculates the pHash value for each sample in the target model’s training set, and generates a hash list, as shown in the code snippet 4.1. When the attacker submits a query image, the query platform calculates its pHash value and then quickly determines whether the query image is in the training dataset using a binary search method, as shown in the code snippet 4.2. If the query image is found in the training dataset, it proceeds to Step 2 and Step 3. If not, no image transformation operations are performed on the query image.

```

1 def calculate_phash_decimal(image):
2     pil_image = Image.fromarray(image)
3     phash_hex = str(imagehash.phash(pil_image))
4     return int(phash_hex, 16)
5 phashes = [calculate_phash_decimal(img) for img in raw_images]
6 sorted_phashes = np.sort(phashes)

```

Listing 4.1: Code for generating hash list

```

1 def binary_search(arr, target):
2     left, right = 0, len(arr) - 1
3     while left <= right:
4         mid = (left + right) // 2
5         if arr[mid] == target:
6             return True
7         elif arr[mid] < target:
8             left = mid + 1
9         else:
10            right = mid - 1
11    return False

```

Listing 4.2: Binary search pHash value

There are two primary reasons for choosing pHash instead of aHash and dHash:

- pHash assigns identical hash values to images with slight differences. Therefore, for query images that are highly similar to those in the training dataset, the query platform will also identify them as present in the training dataset and then perform image transformation operations on them.
- The number of duplicate pHash values in the training dataset and the frequency with which a test sample’s pHash appears in the training set’s pHash list is relatively low. Two bar charts Figure 4.3 and Figure 4.4 illustrate the comparison results when using the CIFAR-10 dataset.

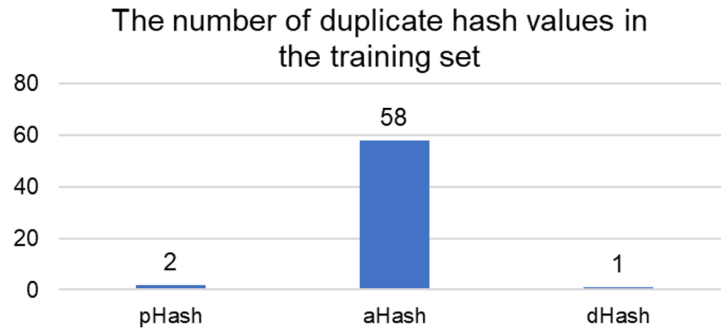


Figure 4.3: Number of Duplicate Hash Values in the Training Set

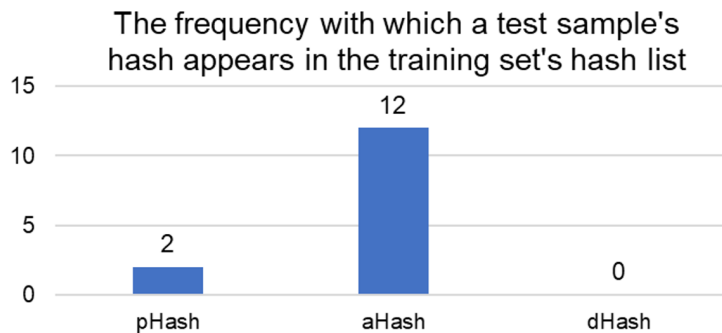


Figure 4.4: Frequency with Which a Test Sample's Hash Appears in the Training Set's Hash List

Taking these two reasons into account, pHash was ultimately chosen.

## 4.2 Step2: Data Augmentation

In the second step, the query platform performs data augmentation on the image.

This thesis employs 12 data augmentation methods in total and select some randomly to apply to the query image, as shown in the code snippet 4.3. The data augmentation methods include horizontal flip, random crop with padding, rotation, Gaussian blur, affine transformation with translation, affine transformation with scaling, grayscale, re-sized crop, adjust sharpness, perspective transformation, posterization, and equalization.

```

1 def apply_random_augmentation(image, num, weights):
2     augmentations = [
3         transforms.RandomHorizontalFlip(p=1.0),
4         transforms.RandomRotation(degrees=(20, 22)),
5         transforms.RandomAffine(degrees=(10, 12), translate=(0.1, 0.1)),
6         transforms.RandomAffine(degrees=(3, 5), scale=(0.95, 0.95)),
7         transforms.RandomResizedCrop(size=(32, 32), scale=(0.9, 0.9)),
8         transforms.RandomPerspective(distortion_scale=0.08, p=1),
9         transforms.RandomEqualize(p=1),
10        transforms.RandomCrop(size=(32, 32), padding=4),

```

```

11     transforms.GaussianBlur(kernel_size=(3, 3), sigma=(2, 2)),
12     transforms.RandomGrayscale(p=1.0),
13     transforms.RandomAdjustSharpness(sharpness_factor=4, p=1),
14     transforms.RandomPosterize(bits=4, p=1),
15 ]
16 num_augmentations = random.choices(num, weights)[0]
17 selected_augmentations = transforms.Compose(random.sample(
18     augmentations, num_augmentations))
19 return selected_augmentations(image)

```

Listing 4.3: Code for Performing Data Augmentation

### 4.3 Step 3: Image Fusion with PCA Composite Image

In the third step, the query platform fuses the data-augmented image with a random PCA Composite Image.

The PCA composite image is generated as follows. First, the image data of each class in the training dataset is reduced in dimensionality separately. Then, the PCA principal components are extracted. Finally, the PCA principal components are reconstructed as an image, resulting in the PCA composite image. The corresponding code snippet 4.4 is shown below.

```

1 for class_label in range(10):
2     class_images = x_train[y_train.flatten() == class_label]
3     num_images, img_height, img_width, num_channels = class_images.shape
4     class_images_flat = class_images.reshape(num_images, -1)
5     pca = PCA(n_components=4)
6     pca_class = pca.fit_transform(class_images_flat)
7     mean_image_flat = pca.components_[0]
8     mean_image_flat = mean_image_flat * np.std(class_images_flat, axis
9     =0) + np.mean(class_images_flat, axis=0)
10    mean_image = mean_image_flat.reshape(img_height, img_width,
11    num_channels)
12    mean_image_uint8 = (mean_image * 255).astype("uint8")
13    output_image = Image.fromarray(mean_image_uint8)

```

Listing 4.4: Code for generating PCA composite image

For example, CIFAR-10 has 10 image classes, corresponding to 10 generated PCA composite images. The array-format data-augmented image will be linearly fused with the selected PCA composite image according to the predetermined weight. The result is then converted back into the image format, resulting in the final processed image. The corresponding code snippet 4.5 is shown below:

```

1 def apply_composite(img, pca_images, alpha=0.7):
2     img_array = np.array(img)
3     random_class = np.random.randint(0, 10)
4     pca_image = pca_images[random_class]
5     pca_image = np.resize(pca_image, img_array.shape)

```

```
6 fused_image = alpha * img_array + (1 - alpha) * pca_image
7 fused_image = np.clip(fused_image, 0, 255).astype(np.uint8)
8 fused_pil_image = Image.fromarray(fused_image)
9 return fused_pil_image
```

Listing 4.5: Code for fusing the data-augmented image with a PCA composite image

## 4.4 Step 4: Attacker Performing MIA

The query platform inputs the preprocessed image data into the target model and then returns the prediction vector to the attacker. With the prediction vector, the attacker could perform binary classifier based MIA and metric based MIA, which is explained in detail in section 2.2.3.

After that, the attacker could obtain precision, recall, and F1-score values, which represent the performance of the MIA. The further these values deviate from 0.5, the better the performance of the MIA attack. After implementing defense measures, namely performing data augmentation on the image and using the data-augmented image with a PCA composite image, if the precision, recall, and F1-score values become closer to 0.5, it indicates that the defense was indeed effective.

## 4.5 Summary

In summary, steps 1-3 implement defense against the attacker. If the image submitted by the attacker exists in the training dataset, after undergoing step 1-3, the processed image will differ from the original image, and the resulting prediction vector will also be different from the initial one. As a result, the MIA precision, recall, and F1-score values become closer to 0.5. This significantly increases the difficulty for the attacker to infer whether the image is in the training dataset based on the prediction vector.



# Chapter 5

## Implementation

This chapter presents implementation details on defending against binary classifier based MIA and metric based MIA. Finally, a tool for countering MIA is developed, which helps each dataset determine the optimal image processing intensity that maximally defends against MIA.

### 5.1 Setup

In this research, a personal desktop computer and a server provided by the university were used. Both devices operated on the Ubuntu operating system. The GPU model of the former is GTX 1070 with 8GB of memory, while the latter's GPU model is Tesla T4 with 15GB of memory.

On the personal desktop computer, this thesis used a conventional CNN model to classify relatively simple datasets such as CIFAR-10 and Fashion MNIST. On the server, this thesis used more complex models such as VGG16 to classify more complex datasets such as Tiny ImageNet.

In total, these two devices handled five datasets: CIFAR-10, Fashion MNIST, ImageNet-10, CIFAR-100, and Tiny ImageNet. Among these, the ImageNet-10 dataset requires downloading the complete ImageNet-1000 dataset from Imagenet Website, and then extracting the ImageNet-10 dataset according to indexes on the Kaggle. The remaining four datasets can be directly downloaded using the Python DL framework PyTorch.

These five datasets are all image datasets which are typically used for image classification tasks. However, they differ in aspects such as image size and the number of samples, as shown in the Table 5.1 below.

Dataset	Classes	Example Labels	Image Size	Color Type	Training Samples	Test Samples
CIFAR-10	10	Airplane, Car, Dog, Frog, Truck	32×32	Color	50,000	10,000
CIFAR-100	100	Beaver, Castle, Bridge, Rocket	32×32	Color	50,000	10,000
Fashion MNIST	10	T-shirt/Top, Dress, Bag, Sneaker	28×28	Grayscale	60,000	10,000
Tiny ImageNet	200	Goldfish, Coffee Mug, Parachute	64×64	Color	100,000	10,000
ImageNet-10	10	Airliner, Sports Car, King Penguin, Snow Leopard	Varies	Color	13,000	500

Table 5.1: Comparison of Five Image Datasets Used in Classification Tasks

Then the implementation details will be introduced.

## 5.2 Dividing the Dataset Evenly into Two Halves.

First, the downloaded raw dataset is shuffled and then evenly split into two halves, and data from each class is equally distributed between the two parts. Therefore, the two parts are IID data. One half is used as the target dataset, while the shadow dataset is randomly selected from the other half. The specific experimental setup is introduced in 6.1

The dataset partitioning code snippet 5.1 is shown below:

```

1 for i in range(num_classes):
2     train_indices = train_class_indices[i]
3     np.random.shuffle(train_indices)
4     train_split = len(train_indices) // 2
5     train1_indices.extend(train_indices[:train_split])
6     train2_indices.extend(train_indices[train_split:])
7
8     test_indices = test_class_indices[i]
9     np.random.shuffle(test_indices)
10    test_split = len(test_indices) // 2
11    test1_indices.extend(test_indices[:test_split])
12    test2_indices.extend(test_indices[test_split:])

```

Listing 5.1: Code for dataset partitioning

## 5.3 Selection of Classification Models

For datasets with fewer classes, simpler models are chosen, while for datasets with more classes, more complex models are selected.

The following code snippet 5.2 demonstrates the classification model for Fashion MNIST. This is a relatively simple classification model consisting of 2 convolutional layers, 1 flatten layer, and 2 fully connected layers. The network uses ReLU as the activation function, CrossEntropyLoss as the objective function, and Adam as the optimizer.

```

1 class FashionMNISTModelCNN(LightningModule):
2     def __init__(self, learning_rate=1e-3):
3         super(FashionMNISTModelCNN, self).__init__()
4         self.save_hyperparameters()
5         self.learning_rate = learning_rate
6         self.criterion = nn.CrossEntropyLoss()
7         self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
8 kernel_size=3, padding=1)
9         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
10        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
11 kernel_size=3, padding=1)
12        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
13        self.flatten_size = 64 * 7 * 7
14        self.fc1 = nn.Linear(self.flatten_size, 128)
15        self.fc2 = nn.Linear(128, 10)
16        self.relu = nn.ReLU()
17
18    def forward(self, x):
19        x = self.relu(self.conv1(x))
20        x = self.pool1(x)
21        x = self.relu(self.conv2(x))
22        x = self.pool2(x)
23        x = torch.flatten(x, 1)
24        x = self.relu(self.fc1(x))
25        x = self.fc2(x)
26        return x

```

Listing 5.2: classification model for Fashion Mnist

The following code snippet 5.3 demonstrates the classification model for Tiny Imagenet. This model is far more complex than the classification model for Fashion MNIST. Its core is ResNet-50, which is loaded with pretrained weights to leverage the feature extraction capability obtained from training on large-scale datasets. The model modifies the original ResNet-50's fully connected layer by changing its output dimension to 200, and the loss function used is also CrossEntropyLoss.

```

1 class TinyImageNet(LightningModule):
2     def __init__(self, out_channels=200, learning_rate=1e-4):
3         super(TinyImageNet, self).__init__()
4         self.model = models.resnet50(pretrained=True)
5         in_features = self.model.fc.in_features
6         self.model.fc = nn.Linear(in_features, out_channels)
7         self.criterion = nn.CrossEntropyLoss()

```

```

8     self.learning_rate = learning_rate
9     self.train_acc = torchmetrics.Accuracy(task="multiclass",
num_classes=out_channels)
10
11     def forward(self, x):
12         return self.model(x)

```

Listing 5.3: classification model for Tiny imagenet

## 5.4 Data Preprocessing

As shown in Table 5.1, the image sizes within CIFAR-10, CIFAR-100, Fashion MNIST or Tiny ImageNet are identical. Therefore, no resizing operation is required for these datasets.

However, the image sizes in the ImageNet-10 dataset vary. Therefore, the following steps are necessary before training: First, convert the images from BGR to RGB; then proportionally resize the image so that the shorter side becomes 256 pixels; next, perform a center crop to obtain a 224x224 image; and finally, convert the images into arrays to ensure consistency with other datasets. The image processing code for processing Imagenet-10 dataset is shown in the following code snippet 5.4.

```

1 def load_data(folder):
2     images = []
3     labels = []
4     for class_idx, category in enumerate(categories):
5         class_dir = os.path.join(folder, category)
6         image_files = sorted(os.listdir(class_dir))
7         for img_file in image_files:
8             img_path = os.path.join(class_dir, img_file)
9             img = cv2.imread(img_path)
10            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
11            h, w, _ = img.shape
12            if h < w:
13                new_h, new_w = 256, int(w * (256 / h))
14            else:
15                new_h, new_w = int(h * (256 / w)), 256
16            img = cv2.resize(img, (new_w, new_h), interpolation=cv2.
INTER_LINEAR)
17            start_x = (new_w - 224) // 2
18            start_y = (new_h - 224) // 2
19            img = img[start_y:start_y+224, start_x:start_x+224]
20            images.append(img)
21            labels.append(class_idx)
22
23     return np.array(images, dtype=np.uint8), np.array(labels, dtype=np.
int64).reshape(-1, 1)

```

Listing 5.4: Image Processing for ImageNet-10

Next, all datasets need to undergo the transformation process shown in the following code snippet 5.5, which includes two steps: pixel value normalization to the range  $[0, 1]$  and standardization. After that, the model training can proceed.

```
1 transform = transforms.Compose([
2     transforms.ToTensor(),
3     transforms.Normalize(mean, std)
4 ])
```

Listing 5.5: Data Transformation Process

The normalization parameters for these datasets are as follows:

- **CIFAR-10:** mean = (0.4914, 0.4822, 0.4465), std = (0.2471, 0.2435, 0.2616)
- **CIFAR-100:** mean = (0.5071, 0.4867, 0.4408), std = (0.2675, 0.2565, 0.2761)
- **Fashion MNIST:** mean = (0.2860,), std = (0.3530,)
- **Tiny ImageNet:** mean = (0.485, 0.456, 0.406), std = (0.229, 0.224, 0.225)
- **ImageNet-10:** mean = (0.485, 0.456, 0.406), std = (0.229, 0.224, 0.225)

## 5.5 Model Training

In this thesis, both the ML and DFL models are implemented for all five datasets in order to compare the effectiveness of the defense methods in the two pipelines.

- **For ML**, the target model is trained for a certain number of epochs.
- **For DFL**, each participant trains the target model locally on their respective local dataset. At the end of each round, the FedAvg algorithm is used to aggregate the model parameters from all participants. In the next round, each participant loads the aggregated model parameters and continues training their local model on their local data.

The specific experimental parameters for ML and DFL are shown in the Section 6.1

## 5.6 Obtaining Prediction Vectors after Training

It is an important step to obtain the prediction vectors of the shadow model and target model. The former is used to train a binary classification model or calculate the metric threshold, while the latter is used to evaluate the effectiveness of the MIA.

Next, the process of obtaining prediction vectors in the ML scenario and DFL scenario is introduced separately.

### 5.6.1 ML Scenario

- **Shadow Model:** After the training of shadow models is complete, input the training and test samples directly into the model to obtain prediction vectors.
- **Target Model:** After the training of the target model is complete, save the trained model parameters. Then, apply different levels of data augmentation to the training samples and fuse them with the PCA composite image using different weights. The processed training samples are then fed into the target model.

### 5.6.2 DFL Scenario

- **Shadow Model:** Iterate over each participant to obtain prediction vectors for the training and test samples of their respective local datasets. Then, aggregate the prediction vectors from all 10 participants.
- **Target Model:** Similar to the ML scenario, input the processed training samples with different levels of image processing into the target model. Then, aggregate the prediction vectors from all 10 participants.

## 5.7 Performing MIA

The attacker uses 2 types of MIA: binary classifier based MIA and metric based MIA. Both methods are based on shadow training. Shadow training has two characteristics: First, the shadow model uses the exact same learning algorithm as the target model. Second, the shadow training datasets share the same distribution as the target model's training data.

When performing MIA, there are two data-related considerations.

1. **Ensuring Balanced Classes in the Binary Classifier:** Randomly extract prediction vectors from the shadow model's training sample prediction vectors to match the number of test sample prediction vectors. This ensures the binary classification model has balanced classes, adhering to DL standards.
2. **Ensuring Fair Metric Calculation:** randomly extract prediction vectors from the target model's training sample prediction vectors to match the number of test sample prediction vectors. This is based on the assumption that the number of training samples and testing samples queried by the attacker is the same. This operation ensures that *precision*, *recall*, and *F1-score* are calculated fairly. If the number of training samples exceeds the number of test samples, these metrics will be overestimated. For example, suppose a binary classifier has no classification ability at all, and the number of training samples is far greater than the number of test samples. In this case, the number of true positives (TP) will also be much greater than the number of false positives (FP). As a result, the precision value will approach 1, as

suggested in the Equations (5.1). This value incorrectly reflects the classification ability of the binary classifier.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5.1)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5.2)$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5.3)$$

Where  $TP$  is the number of true positives,  $FP$  is the number of false positives, and  $FN$  is the number of false negatives.

### 5.7.1 Binary Classifier Based MIA

This method trains a binary classifier using the prediction vectors obtained through shadow training. Specifically, after the shadow model is trained, all training samples are fed into the shadow model to obtain their prediction vectors, which are then labeled 1, and all test samples are fed into the shadow model to obtain their prediction vectors, which are then labeled 0.

These labeled prediction vectors are then used to train the binary classifier, allowing it to distinguish between the prediction vectors of training samples and test samples.

When the prediction vector of a query image is fed into the trained binary classifier, if the output is 1, it can be inferred that the query image is in the training dataset.

### 5.7.2 Metric Based MIA

This thesis adopts three metrics, as briefly introduced in Table 5.2.

For each labeled prediction vector from shadow training, the corresponding metric value is calculated. Next, a threshold that maximizes the sum of true positives (TP) and true negatives (TN) is determined for each class in the dataset. Once the prediction vector of the query image is obtained, the metric value is calculated and compared with the threshold to determine whether the query image is in the training dataset. The details are shown in the table below.

Metric	Formula
Prediction Class Confidence	Confidence = prediction_vector[i, true_label]
Prediction Class Entropy	Entropy = $-\sum_j \text{prediction\_vector}[i, j] \cdot \log(\text{prediction\_vector}[i, j])$
Prediction Class Modified Entropy	Modified Entropy = $-\sum_j \text{modified\_probs}[i, j] \cdot \log(\text{modified\_probs}[i, j])$ $\text{modified\_probs}[i, j] = \begin{cases} 1 - \text{probs}[i, j] & \text{if } j = \text{true\_label} \\ \text{probs}[i, j] & \text{otherwise} \end{cases}$

Table 5.2: Three Metrics Used in Metric Based MIA

## 5.8 Finding Appropriate Defense Intensity

The selection of an appropriate defense for some datasets was done by adjusting parameters manually. This process is quite labor-intensive, as each run of the defense method and both approaches of MIA can take several minutes and multiple runs are often needed. Besides, small mistakes can occur during this process. For example, overlooking one unsuitable metric value among more than ten metric values. Therefore, this thesis explores automating the process of finding an appropriate defense intensity to make it more convenient for future researchers.

During the process of manually adjusting parameters, it is discovered that the F1-score, precision, and recall of both binary classifier based MIA and metric based MIA increase with the rise of the PCA composite image fusion weight ( $\alpha$ ), and decrease with the increase in data augmentation intensity. Based on this finding, a script was developed to assist in finding appropriate image processing intensity for defense, enabling researchers to quickly identify an appropriate defense method for each dataset.

The idea is to fix the value of  $\alpha$  to one of the following: 0.5, 0.6, 0.7, 0.8, 0.9, or 1.0, and then adjust the parameters related to data augmentation intensity, namely *num* and *weights*.

A roughly preselected set of combinations of *num* and *weights* is shown in Table 5.3, and the table is ordered from top to bottom by increasing data augmentation intensity. Specifically,  $\text{num} = [2, 3]$ ,  $\text{weight} = [0, 1]$  indicates that there is a probability of 0 to select

two data augmentation methods, and a probability of 1 to select three data augmentation intensity methods.

Then, the combination for which the binary classifier based MIA F1-score is closest to 0.5 is identified from Table 5.3. During this process, all intermediate results that satisfy the `all_in_range` condition are recorded.

<b>num</b>	<b>weight</b>
[0, 1]	[1, 0]
[0, 1]	[0.5, 0.5]
[0, 1]	[0, 1]
[1, 2]	[0.5, 0.5]
[1, 2]	[0, 1]
[2, 3]	[0.5, 0.5]
[2, 3]	[0, 1]
[3, 4]	[0.5, 0.5]
[3, 4]	[0, 1]

Table 5.3: Roughly Preselected Set of Combinations of *num* and *weights*

After determining a combination, then the code will identify a more precise (*num*, *weights*) pair using the `next_weight_step` function in the code snippet 5.6, such that the F1-score of the binary classifier based MIA is closest to 0.5. If the current F1-score is less than 0.5, then `direct` is set to 0, which means to weaken data augmentation intensity. If the current F1-score is greater than 0.5, then `direct` is set to 1, which means to enhance data augmentation intensity. This process is repeated in a loop until the value of `direct` changes from 0 to 1, or from 1 to 0. During the loop, all intermediate results that satisfy the condition where all precision, recall, and F1-scores fall within the interval [0.4, 0.6] are recorded.

```

1 def next_weight_step(num, weights, direct):
2     step = 0.1
3     w1, w2 = weights[0], weights[1]
4     # direct=0 means weakening data augmentation intensity
5     if direct == 0:
6         if num[0] == 0 and w1 == 1:
7             return None
8         if w1 == 1.0:
9             new_num = [num[0] - 1, num[1] - 1]
10            if new_num[0] < 0:
11                return None
12            w1_new = round(w1 + step - 1.0, 2)
13            w2_new = round(1.0 - w1_new, 2)
14            return new_num, [w1_new, w2_new]
15        else:
16            w1_new = round(w1 + step, 2)
17            w2_new = round(1.0 - w1_new, 2)
18            return num, [w1_new, w2_new]
19
20     # direct=1 means enhancing data augmentation intensity

```

```

21     else:
22         if w2 == 1.0:
23             new_num = [num[0] + 1, num[1] + 1]
24             return new_num, [0.9, 0.1]
25         else:
26             w2_new = round(w2 + step, 2)
27             w1_new = round(1.0 - w2_new, 2)
28             return num, [w1_new, w2_new]

```

Listing 5.6: Code for identify a more precise (*num*, *weights*) pair

Finally, for each  $\alpha \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ , all corresponding intermediate results are collected. Among them, this thesis identified the group of results in which the sum of absolute differences between the F1-scores and 0.5 is minimized. Specifically, this includes the F1-score of the binary classifier based MIA and the three F1-scores from the metric based MIA.

To ensure balanced classes in the binary classifier and fair metric calculation as mentioned in the section 5.7, training samples are randomly selected, therefore, the results generated in each run may exhibit slight variations.

The selection criterion is to identify the combination, among those where all precision, recall, and F1-scores fall within the interval  $[0.4, 0.6]$ , whose four F1-scores are closest to 0.5.

Notably, this code assumes that both F1-score of the binary classifier based MIA and F1-scores of the metric based MIA are larger than 0.5 when no defense is applied.

However, the current implementation has the following limitations:

1. The results obtained after running the code for finding appropriate defense intensity may be empty. In such cases, one can consider relaxing the condition in the *all\_in\_range* function to  $0.38 \leq m \leq 0.62$  or  $0.35 \leq m \leq 0.65$ ; alternatively, the value of *step* in the *next\_weight\_step* function can be changed from 0.1 to 0.05; or the alpha values can be made more fine-grained and dense by adjusting the increment from 0.1 to 0.05.
2. The runtime of the code for identifying appropriate combination of (*num*, *weights*,  $\alpha$ ) is relatively long. For example, on the FASHION-MNIST dataset under ML pipeline, it takes approximately 1.5 hours to complete.
3. The selection criterion is very simple, as it only considers whether all metrics fall within the  $[0.4, 0.6]$  range and how close the F1-score is to 0.5, without accounting for whether precision and recall are also sufficiently close to 0.5. Moreover, it does not consider the difference in prediction accuracy between the training samples and the test samples after applying the defense method.

After determining an appropriate defense intensity, the code can be executed to obtain experimental data on the impact of data augmentation intensity on defense effectiveness, as well as the impact of the PCA composite image fusion weight on defense effectiveness.

The corresponding code is in code snippet 5.7. Further analysis of the experimental results can be found in section 6.4 and Section 6.5.

```
1 import optimal_defense_intensity as odi
2
3 best_result = odi.generate_optimal_defense_intensity()
4 best_num, best_weights, best_alpha, diff, metrics = best_result
5
6 for alpha in np.arange(0, 1.1, 0.1):
7     pd.perform_defense(best_num, best_weights, alpha)
8     smm.perform_shadow_model_mia()
9     cmm.perform_class_metric_mia()
10    print(f"Running for num={best_num}, weights={best_weights}, alpha =
        {alpha}")
11
12 defense_params = [
13 ([0, 1], [1, 0]),
14 ([0, 1], [0.5, 0.5]),
15 ([0, 1], [0, 1]),
16 ([1, 2], [0.5, 0.5]),
17 ([1, 2], [0, 1]),
18 ([2, 3], [0.5, 0.5]),
19 ([2, 3], [0, 1]),
20 ([3, 4], [0.5, 0.5]),
21 ([3, 4], [0, 1]),
22 ]
23
24 for num, weights in defense_params:
25     pd.perform_defense(num, weights, best_alpha)
26     smm.perform_shadow_model_mia()
27     cmm.perform_class_metric_mia()
28     print(f"Running for num = {num}, weights = {weights}, alpha = {
        best_alpha}")
```

Listing 5.7: Code for Evaluating the Effect of Different Parameters on the Defense Effectiveness



# Chapter 6

## Evaluation

This chapter provides a detailed evaluation of the defense method proposed in this thesis. The defense method was applied to five datasets: CIFAR-10, CIFAR-100, Fashion MNIST, Tiny ImageNet, and ImageNet-10, under both ML and DFL pipelines. Data such as prediction accuracy and the performance of two MIA approaches were obtained under both with-defense and without-defense conditions, thereby supporting a comprehensive evaluation.

This thesis conducts the following evaluation experiments in detail using the CIFAR-10 dataset as an example:

- Comparison between the ML pipeline and the DFL pipeline;
- Comparison between two configurations for the shadow models;
- The impact of data augmentation intensity on defense effectiveness;
- The impact of the PCA composite image fusion weight on defense effectiveness.

Due to the large volume of data, the data from other datasets is provided in the appendix. Notably, the findings from CIFAR-10 are also applicable to the other datasets.

### 6.1 Setup and Experiment Overview

Evaluation consists of an ML pipeline part and a DFL pipeline part. This thesis will provide the experimental configuration and experiment overview below.

### 6.1.1 ML Pipeline

#### Experimental Configuration

First, the experimental configuration will be introduced.

As shown in Table 6.1, the first column represents the datasets, which have been introduced in Section 5.1. The second column indicates the models used to classify these datasets. For datasets such as CIFAR-10 and Fashion MNIST, which have lower image resolution and fewer classes, a lightweight CNN model can achieve good classification performance. For datasets like ImageNet-10, which have higher image resolution, and CIFAR-100 and Tiny ImageNet, which have more classes, more complex models such as VGG16, RESNET50, and RESNET18 are used.

As described in Section 2.2.3, the shadow models and target models use the same models, meaning the models in the second column of Table 6.1 are applied to both. The number of training epochs for the models is shown in the third column, where the selection criterion for training epochs is to make it overfit while retaining its classification ability.

Dataset	Image Classification Model	Epoch	Target Model Dataset	Shadow Model	Shadow Model Dataset	Binary Classifier
CIFAR-10	lightweight CNN model	50	size1 = 25000 size2 = 5000	10	size1 = 5000 size2 = 1000	MLP
				1	size1 = 25000 size2 = 5000	
CIFAR-100	VGG16	50	size1 = 25000 size2 = 5000	10	size1 = 5000 size2 = 1000	MLP
				1	size1 = 25000 size2 = 5000	
Fashion MNIST	lightweight CNN model	50	size1 = 30000 size2 = 5000	10	size1 = 6000 size2 = 1000	CNN
				1	size1 = 30000 size2 = 5000	
Tiny ImageNet	ResNet50	20	size1 = 50000 size2 = 5000	10	size1 = 10000 size2 = 1000	CNN
				1	size1 = 50000 size2 = 5000	
ImageNet-10	ResNet18	20	size1 = 6500 size2 = 250	10	size1 = 1300 size2 = 250	CNN
				1	size1 = 6500 size2 = 250	

Table 6.1: Experimental Configuration in ML Pipeline for Different Datasets

The fourth column displays the size of the target model dataset, which is half of the original dataset, as described in Section 5.2. The fifth and sixth columns provide information on the number of shadow models and the size of their datasets. In the sixth columns, size1 represents the size of the training dataset, size2 represents the size of the test dataset. For each dataset, there are two configurations for the shadow models:

### Experiment Overview

The experiment overview of ML pipeline is introduced in Table 6.2. The first column represents the datasets. The second column introduces shadow model configuration, Config1 representing the case with 10 shadow models and Config2 representing the case with 1 shadow model. The third column illustrates the defense state, and “without” means that the defense is not performed and “with” means that the defense is performed. Different datasets apply different defense intensities:

- **CIFAR10**: Each image undergoes one randomly selected data augmentation method and is fused with one randomly selected PCA composite image, with a weight of 0.8.
- **CIFAR100**: Each image undergoes one randomly selected data augmentation method with a probability of 0.8, or two randomly selected data augmentation methods with a probability of 0.2. It is then fused with one randomly selected PCA composite image, with a weight of 0.8.
- **Fashion MNIST**: Each image undergoes one randomly selected data augmentation method and is fused with one randomly selected PCA composite image, with a weight of 0.8.
- **Tiny ImageNet**: Each image undergoes one randomly selected data augmentation method and is fused with one randomly selected PCA composite image, with a weight of 0.8.
- **ImageNet-10**: Each image undergoes one randomly selected data augmentation method with a probability of 0.5, or two randomly selected data augmentations with a probability of 0.5. It is then fused with one randomly selected PCA composite image, with a weight of 0.6.

The fourth and fifth columns of Table 6.2 represent the prediction accuracy of training and test samples under the conditions of without defense or with defense. It is obvious that the test sample accuracy has almost no difference between the two conditions. However, the training sample accuracy is significantly lower under the with defense condition compared to the without defense condition. This is because all training samples undergo defense method of image transformation, while only a very small number of test samples undergo defense method because they are identified as training samples wrongly.

The sixth column of Table 6.2 reveals the performance of the binary classifier based MIA. Without defense, the F1-score values are all greater than 0.6, with most values

exceeding 0.7, indicating that the binary classifier based MIA demonstrates strong attack performance. With defense, the F1-score values range between 0.4 and 0.6, indicating that the attack capability of the binary classifier based MIA is significantly reduced by the defense method, demonstrating that in the ML pipeline, the defense method used is effective against the binary classifier based MIA.

<b>Dataset</b>	<b>Shadow model Configuration</b>	<b>Defense</b>	<b>Training Accuracy</b>	<b>Testing Accuracy</b>	<b>Binary Classifier based MIA (F1-score)</b>	<b>Metric 1 (F1-score)</b>	<b>Metric 2 (F1-score)</b>	<b>Metric 3 (F1-score)</b>
CIFAR-10	Config1	without	99.18%	67.34%	0.7197	0.7540	0.7219	0.7529
	Config1	with	62.16%	66.42%	0.5114	0.4732	0.5018	0.4720
	Config2	without	99.18%	67.34%	0.7086	0.7534	0.7158	0.7534
	Config2	with	67.44%	67.34%	0.5716	0.4661	0.4533	0.4672
CIFAR-100	Config1	without	100.00%	59.46%	0.7197	0.8092	0.7746	0.8098
	Config1	with	61.68%	59.44%	0.5191	0.4916	0.5217	0.4923
	Config2	without	100.00%	59.46%	0.7161	0.8166	0.7837	0.8149
	Config2	with	61.28%	59.48%	0.5506	0.4243	0.4124	0.4206
Fashion MNIST	Config1	without	99.64%	90.44%	0.6691	0.6813	0.6716	0.6812
	Config1	with	77.62%	90.28%	0.5126	0.4862	0.5052	0.4862
	Config2	without	99.64%	90.44%	0.6219	0.6771	0.6491	0.6753
	Config2	with	78.72%	90.30%	0.5055	0.4898	0.3938	0.4715
Tiny ImageNet	Config1	without	98.00%	59.22%	0.7087	0.7665	0.7132	0.7678
	Config1	with	62.72%	59.22%	0.4898	0.5363	0.5730	0.5360
	Config2	without	98.00%	59.22%	0.7574	0.7595	0.7316	0.7593
	Config2	with	63.58%	59.20%	0.4343	0.3982	0.3801	0.3983
ImageNet-10	Config1	without	100.00%	90.40%	0.7126	0.7224	0.7211	0.7254
	Config1	with	84.40%	90.40%	0.4940	0.4742	0.4824	0.4723
	Config2	without	100.00%	90.40%	0.6503	0.7036	0.7015	0.7024
	Config2	with	86.00%	90.40%	0.4684	0.4392	0.4143	0.4260

Table 6.2: Experimental Overview in ML pipeline for Different Datasets

The seventh, eighth, and ninth columns of Table 6.2 represent the performance of the metric based MIA. Metric 1 in the seventh column represents Prediction Class Confidence, Metric 2 in the eighth column represents Prediction Class Entropy, and Metric 3 in the ninth column represents Prediction Modified Entropy, as introduced in Table 5.2. Without defense, the F1-score values are all greater than 0.65, with most values exceeding 0.7, indicating that the metric based MIA demonstrates strong attack performance. With defense, the F1-score values range between 0.4 and 0.6, indicating that the attack capability of the metric based MIA is significantly reduced by the defense method. This shows that in the ML pipeline, the defense method used is also effective against the metric based MIA.

In summary, the defense method of this thesis is effective against binary classifier based MIA and metric based MIA under ML pipeline.

### 6.1.2 Evaluation of DFL Pipeline

#### Experimental Configuration

The image classification model used in DFL pipeline is the same as that used in ML pipeline for each dataset, as shown in the second column of Table 6.3. Therefore, it is easier to compare these two. Unlike in the ML pipeline, participants parameters are aggregated in each training round in the DFL pipeline. In the next round, each participant uses the aggregated parameters to train for a certain number of epochs, and this process repeats. The number of training rounds and epochs for each dataset is shown in the third column of Table 6.3. The selection criterion for training rounds and epochs is to make the model overfit while retaining its classification ability.

In the DFL pipeline, this thesis uniformly sets each model to have 10 participants.

Unlike the ML pipeline, in the DFL pipeline, there is only one configuration for the shadow models, that is, the number of shadow models is 1 and the dataset for the shadow model is the other half of the original dataset. This is because the original dataset size is limited; if there were 10 shadow models and each shadow model had 10 participants, the local dataset for each participant would be too small to train adequately.

Therefore, each participant in both the shadow model and the target model has a local dataset that is one-twentieth of the original dataset, as shown in the fourth column of the Table 6.3. Moreover, the local datasets of each participant do not overlap.

The binary classifier used for each dataset is also consistent with that used in the ML pipeline, as shown in the fifth column of the Table 6.3.

Dataset	Image Classification Model	Image Classification Model Training Epochs	Local Dataset of Each Participant	Binary Classifier
CIFAR-10	lightweight CNN model	15 rounds 10 epochs	size1 = 2500 size2 = 500	MLP
CIFAR-100	VGG16	15 rounds 10 epochs	size1 = 2500 size2 = 500	MLP
Fashion MNIST	lightweight CNN model	20 rounds 5 epochs	size1 = 3000 size2 = 500	CNN
Tiny ImageNet	ResNet50	15 rounds 10 epochs	size1 = 5000 size2 = 5000	CNN
ImageNet-10	ResNet18	10 rounds 5 epochs	size1 = 650 size2 = 250	CNN

Table 6.3: Experimental Configuration in DFL Pipeline for Different Datasets

## Experiment Overview

The experiment overview of DFL pipeline is introduced in Table 6.4.

The first column shows the datasets. The second column illustrates the defense state. Different datasets apply different defense intensities:

- **CIFAR10**: Each image undergoes one randomly selected data augmentation method and is fused with one randomly selected PCA composite image, with a weight of 0.8.
- **CIFAR100**: Each image undergoes one randomly selected data augmentation method and is fused with one randomly selected PCA composite image, with a weight of 0.7.
- **Fashion MNIST**: Each image does not undergo any data augmentation method with a probability of 0.5, or undergoes one randomly selected data augmentation method with a probability of 0.5. It is then fused with one randomly selected PCA composite image, with a weight of 0.9.
- **Tiny ImageNet**: Each image does not undergo any data augmentation method with a probability of 0.5, or undergoes one randomly selected data augmentation method with a probability of 0.5. It is then fused with one randomly selected PCA composite image, with a weight of 0.6.
- **ImageNet-10**: Each image does not undergo any data augmentation method with a probability of 0.3, or undergoes one randomly selected data augmentation method with a probability of 0.7. It is then fused with one randomly selected PCA composite image, with a weight of 0.7.

The third and fourth columns of Table 6.4 represent the prediction accuracy of training and test samples under two conditions. The data shows that the test sample accuracy also has almost no difference between the two conditions. However, the training sample accuracy is also significantly lower under the with defense condition compared to the without defense condition.

The fifth column of Table 6.4 reveals the performance of the binary classifier based MIA. Without defense, the F1-score values are all greater than 0.6, indicating that binary classifier based MIA still work under DFL pipeline. With defense, the F1-score values range between 0.4 and 0.6, indicating that the defense method is also effective against binary classifier based MIA under DFL pipeline.

The sixth, seventh and eighth columns of Table 6.4 represent the performance of the metric based MIA. Metric 1, Metric 2 and Metric 3 have the same meaning as in the Table 6.2. Without defense, the F1-score values are all greater than 0.6 except the F1-score value of Metric 2 in the Fashion MNIST dataset, indicating that metric based MIA still work under DFL pipeline. With defense, the F1-score values range between 0.4 and 0.6, indicating that the defense method is also effective against metric based MIA under DFL pipeline.

In short, the defense method of this thesis is effective against binary classifier based MIA and metric based MIA under DFL pipeline.

<b>Dataset</b>	<b>Defense</b>	<b>Training Accuracy</b>	<b>Testing Accuracy</b>	<b>Binary Classifier based MIA (F1-score)</b>	<b>Metric 1 (F1-score)</b>	<b>Metric 2 (F1-score)</b>	<b>Metric 3 (F1-score)</b>
CIFAR-10	without	93.82%	64.90%	0.6525	0.7065	0.6287	0.7087
	with	64.50%	64.90%	0.5158	0.4949	0.4751	0.4985
CIFAR-100	without	99.60%	60.24%	0.7612	0.7829	0.7515	0.7820
	with	64.00%	60.24%	0.4925	0.4482	0.4425	0.4517
Fashion MNIST	without	98.00%	90.46%	0.6303	0.6623	0.5659	0.6604
	with	89.40%	90.32%	0.5838	0.5856	0.4487	0.5755
Tiny ImageNet	without	97.92%	56.24%	0.7408	0.7695	0.7312	0.7685
	with	65.97%	56.24%	0.4824	0.4826	0.4466	0.4839
Imagenet-10	without	98.44%	91.60%	0.6769	0.6331	0.6214	0.6292
	with	88.48%	91.60%	0.5977	0.4451	0.4179	0.4284

Table 6.4: Experimental Overview in DFL Pipeline for Different Datasets

## 6.2 Comparison of ML and DFL Pipelines

Taking CIFAR10 as an example, the ML and DFL pipelines use the same image classification model. In the ML pipeline, the model was trained for 50 epochs, while in the DFL pipeline, the model was trained for 15 rounds and 10 epochs. For a better comparison, ML uses the results from Config2 which uses one shadow model, as DFL also uses one shadow model.

The prediction accuracy of the training and test samples is shown in the Figure 6.1. It can be observed that the training sample accuracy under the DFL pipeline is lower than that under the ML pipeline. This is because the DFL pipeline achieves a higher level of generalization, resulting in lower overfitting.

Both pipelines adopt the same defense intensity: each image undergoes one randomly selected data augmentation and is fused with one randomly selected PCA composite image, with a weight of 0.8. After performing the defense, the prediction accuracy of the training and test samples is shown in the Figure 6.2. The results indicate that the training sample accuracy decreases to a level close to that of the test samples, making it difficult for the attacker to distinguish between the training and test samples based on the accuracy. Meanwhile, the accuracy of the test samples remains nearly unchanged, demonstrating that the defense method does not affect the classification ability of the target model for common users.

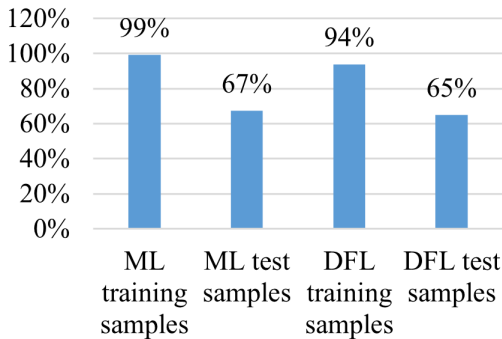


Figure 6.1: Model Prediction Accuracy without Defense

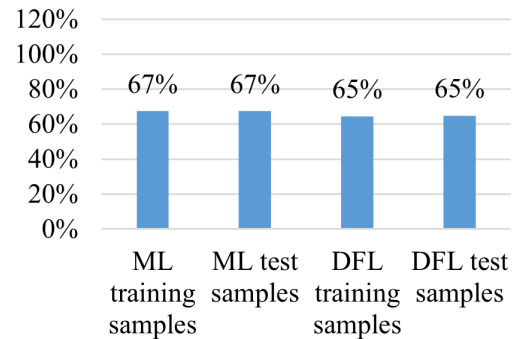


Figure 6.2: Model Prediction Accuracy with Defense

Figure 6.3 illustrates the performance of the binary classifier based MIA with and without defense. Without defense, the F1-score under the ML pipeline is much higher than the F1-score under the DFL pipeline. This suggests that the binary classifier based MIA exhibits stronger attack effectiveness on the ML pipeline compared to the DFL pipeline, possibly because the DFL pipeline results in less information leakage.

The metric based MIA also shows better attack effectiveness on the ML pipeline compared to the DFL pipeline. As shown in Figure 6.4, Figure 6.5, and Figure 6.6, when defense method is not performed, the bar corresponding to the ML pipeline is much higher than the bar corresponding to the DFL pipeline.

A common finding is that the defense method works for both types of MIA in both the ML pipeline and the DFL pipeline, because the F1-score decreased to the range of 0.4 to 0.6.

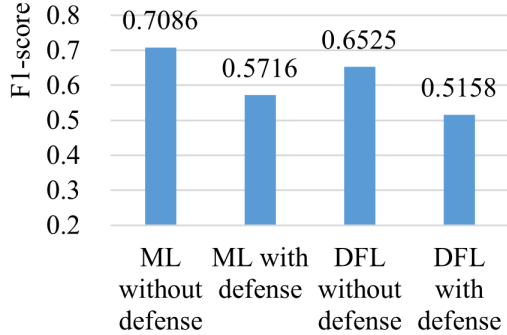


Figure 6.3: Performance of Binary Classifier Based MIA

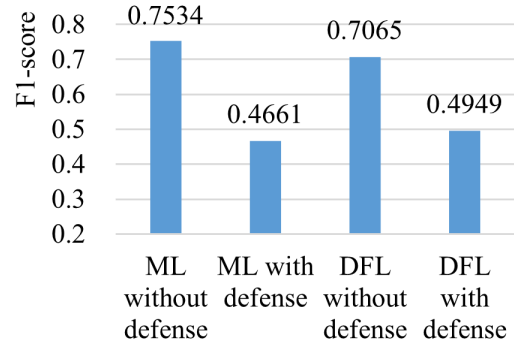


Figure 6.4: Prediction Class Confidence

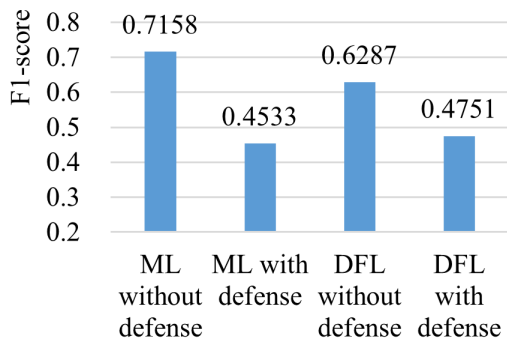


Figure 6.5: Prediction Class Entropy

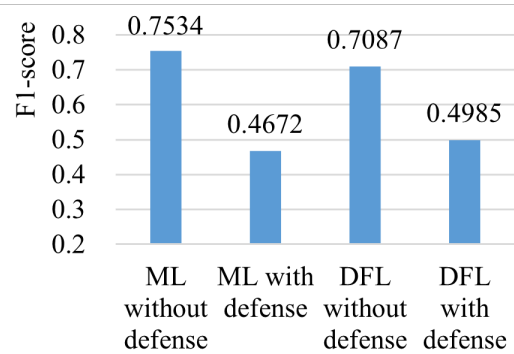


Figure 6.6: Prediction Modified Entropy

### 6.3 Comparison of Two Shadow Model Configurations

Taking CIFAR-10 dataset as an example, this chapter shows the performance of MIA in Config1 and Config2, where Config1 represents the configuration that the number of shadow models is 10 and Config2 represents the configuration that the number of shadow models is 1. The performance of binary classifier based MIA is shown in Figure 6.7 and the performance of metric based MIA is shown in Figure 6.8, Figure 6.9, and Figure 6.10. From these four figures, it can be observed that the F1-score values of Config1 and Config2 are very close, no matter without defense or with defense. This result indicates that the parameter settings of the shadow model have limited impact on both the performance of the MIA and the effective of the defense method.

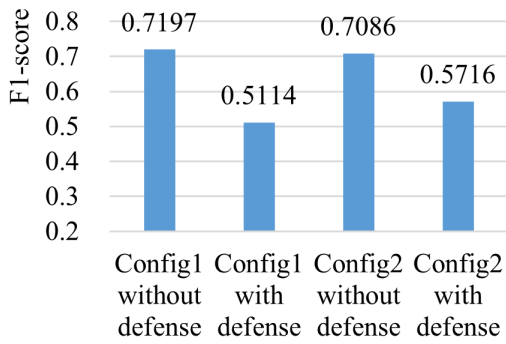


Figure 6.7: Performance of Binary Classifier Based MIA

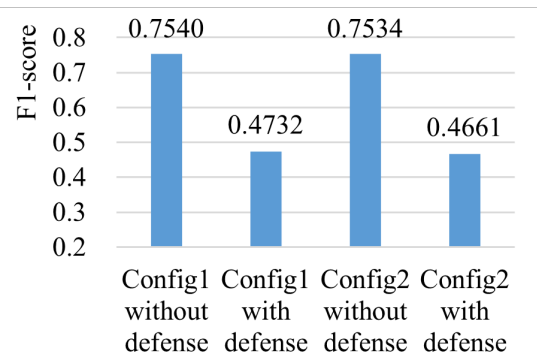


Figure 6.8: Prediction Class Confidence

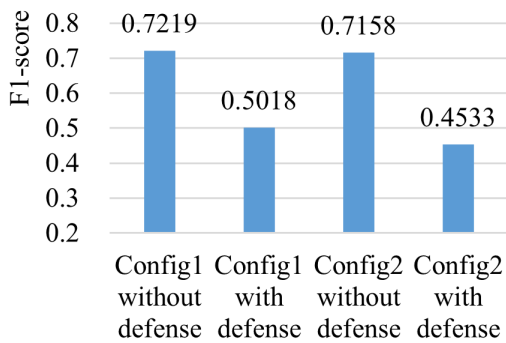


Figure 6.9: Prediction Class Entropy

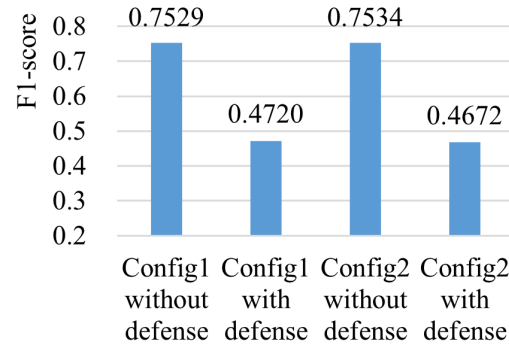


Figure 6.10: Prediction Modified Entropy

## 6.4 Impact of Data Augmentation Intensity

This section discusses the impact of data augmentation intensity on defense effectiveness in both ML pipeline and DFL pipeline.

### 6.4.1 ML Pipeline

The subsection 6.1 presented the selected defense strength combination of the control group, which is the CIFAR-10 dataset under the ML pipeline with Config1.

Under this combination, the defense method demonstrates effective defense against MIA.

- **CIFAR10:** Each image undergoes one randomly selected data augmentation and is fused with one randomly selected PCA composite image, with a weight of 0.8.

Fixing the weight to 0.8 and the number of shadow models to 10, this thesis adjust the data augmentation intensity to analyze the impact of data augmentation intensity on defense effectiveness.

As shown in Figure 6.11, the first bar represents the case without defense, and other bars represent the case with different data augmentation intensities with fixed weight being 0.8.

It is obvious that the bar chart exhibits a downward trend, indicating that the greater the data augmentation intensity applied by the defense method, the lower the training sample accuracy. Because it is more difficult for the classification model to predict the correct class after the image undergoes complex data augmentation.

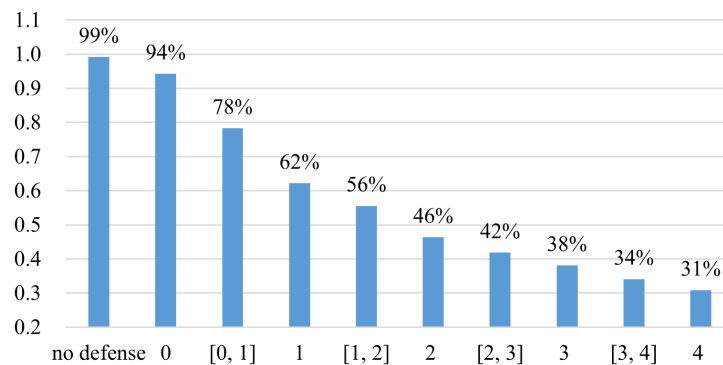


Figure 6.11: Training Sample Accuracy at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline

Figure 6.12 shows the performance of binary classifier based MIA at different data augmentation intensities with fixed weight being 0.8. This figure also shows a declining trend, meaning that the greater the data augmentation intensity, the lower the F1-score. When the F1-score approaches 0.5, the defense method exhibits strongest defensive effectiveness. Therefore, both excessively low and excessively high data augmentation intensity can weaken the defense method's effectiveness.

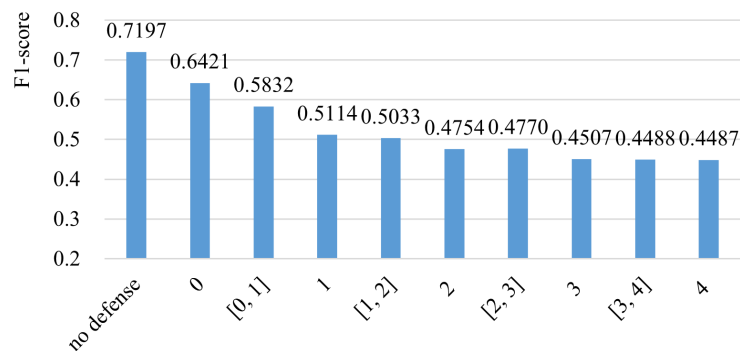


Figure 6.12: Performance of Binary Classifier Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline

Figure 6.13, Figure 6.14, and Figure 6.15 shows the performance of metric based MIA at different data augmentation intensities with fixed weight being 0.8. The three figures respectively represent metrics Prediction Class Confidence, Prediction Class Entropy and Prediction Modified Entropy. All three figures display a downward pattern, indicating that as the data augmentation intensity increases, the F1-score decreases. Similar to the binary classifier based MIA, the defense method demonstrates its strongest effectiveness when the F1-score of the metric based MIA nears 0.5. Therefore, both insufficient and excessive data augmentation intensity can decrease the effectiveness of the defense method.

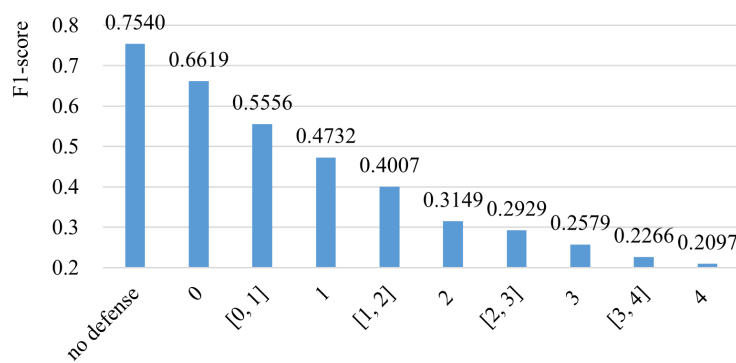


Figure 6.13: Performance of Prediction Class Confidence Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline

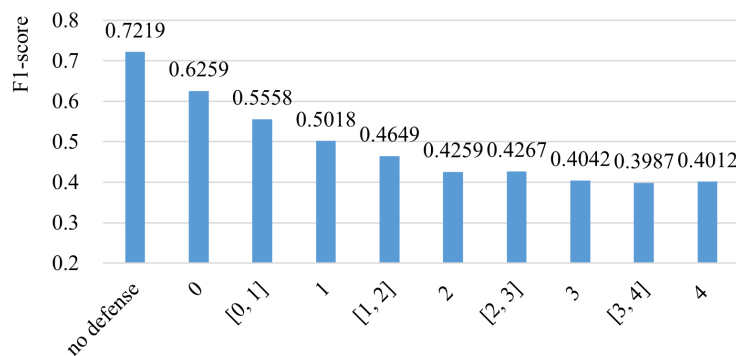


Figure 6.14: Performance of Prediction Class Entropy Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline

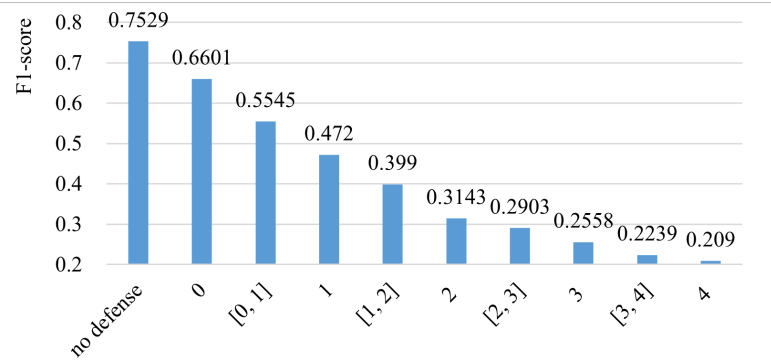


Figure 6.15: Performance of Prediction Modified Entropy based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline

In summary, the training sample accuracy and the F1-score of both MIA approaches decrease with the increasing of data augmentation intensity under ML pipeline. The defense method demonstrates its strongest effectiveness when the F1-score is around 0.5 and excessively strong or weak data augmentation intensity can lead to a decline in the defense method's effectiveness.

### 6.4.2 DFL Pipeline

In the DFL pipeline, CIFAR-10 dataset applies the same defense strength as in the ML pipeline, which means each image undergoes one randomly selected data augmentation and is fused with one randomly selected PCA composite image, with a weight of 0.8.

This thesis also analyze the impact of data augmentation intensity on defense effectiveness in DFL pipeline by fixing the weight to 0.8. As shown in Figure 6.16, the training sample accuracy decreases when the data augmentation intensity increases, which is the same as that in the ML pipeline.

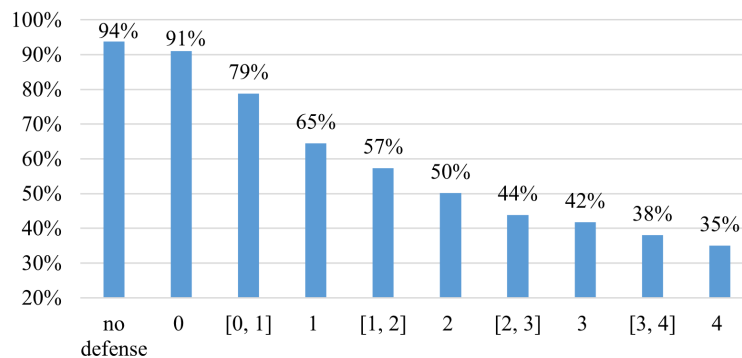


Figure 6.16: Training Sample Accuracy at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline

Figure 6.17 shows the performance of binary classifier based MIA. The image exhibits an overall fluctuating downward trend. The fluctuations are due to the uncertainty introduced by random data augmentation and the fusion with a random PCA composite image.

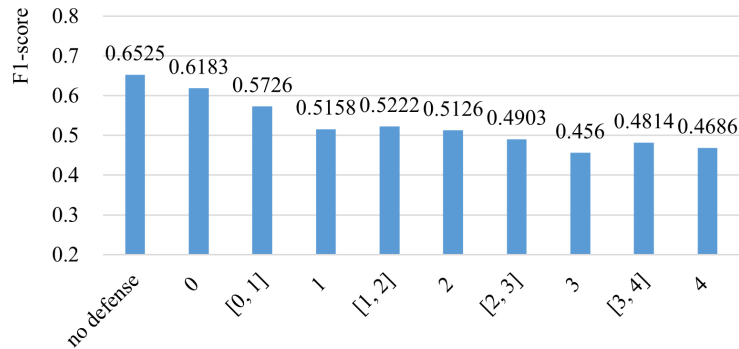


Figure 6.17: Performance of Binary Classifier Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline

Figure 6.18, Figure 6.19 and Figure 6.20 correspond to three metric based MIAs. These three figures all shows a downward trend. When there is no defense, the F1-score is at its highest and significantly exceeds 0.5, indicating that the MIA can successfully work. As the number of data augmentations increases to 1, the F1-score approaches 0.5, reaching the maximum defense strength. However, further increasing the number of data augmentations causes the F1-score to deviate further below 0.5, allowing the MIA to work again.

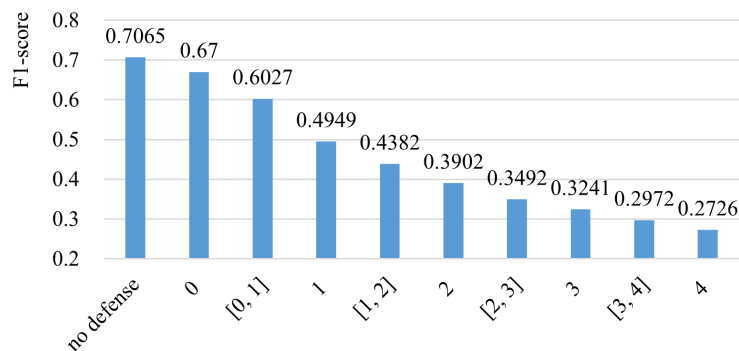


Figure 6.18: Performance of Prediction Class Confidence Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline

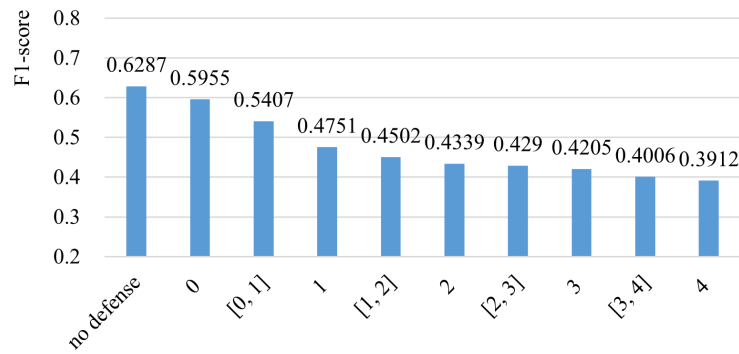


Figure 6.19: Performance of Prediction Class Entropy Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline

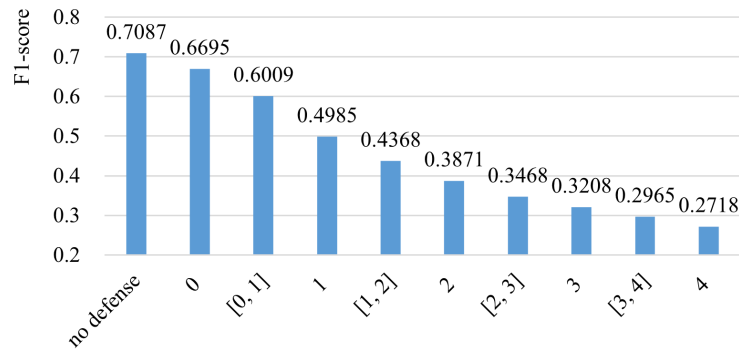


Figure 6.20: Performance of Prediction Modified Entropy Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline

In short, under the DFL pipeline, both the training sample accuracy and the F1-score of the two MIA approaches decrease as the data augmentation intensity increases, which is the same as the ML pipeline.

## 6.5 Impact of PCA Composite Image Fusion Weight

This section discusses the impact of the PCA composite image fusion weight on defense effectiveness in both ML pipeline and DFL pipeline.

### 6.5.1 ML Pipeline

The subsection 6.1 shows the defense intensity combination of the control group, which is the CIFAR-10 dataset under the ML pipeline with Config1.

- **CIFAR10:** Each image undergoes one randomly selected data augmentation and is fused with one randomly selected PCA composite image, with a weight of 0.8.

Fixing the number of data augmentation to one and the number of shadow models to 10, this thesis adjusts the PCA composite image fusion weight to analyze the impact of the PCA composite image fusion weight on defense effectiveness.

Figure 6.21 shows that the training sample accuracy increases with the rising of the weight, because the larger the weight, the greater the proportion of the original image in the fused image, making it easier for the classification model to correctly identify the class.

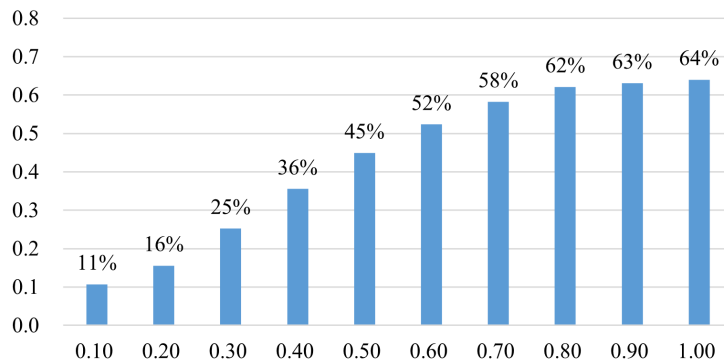


Figure 6.21: Training Sample Accuracy at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline

Figure 6.22 exhibits the performance of binary classifier based MIA at different PCA composite image fusion weights. As the weight increases, the F1-score also rises. When the weight is relatively small, the F1-score deviates significantly from 0.5. This is because after image transformation, the training samples become more similar to a random PCA composite image rather than the original image. Consequently, the binary classifier based MIA can more easily distinguish between training and test samples. The defense effectiveness is strongest when the F1-score approaches 0.5.

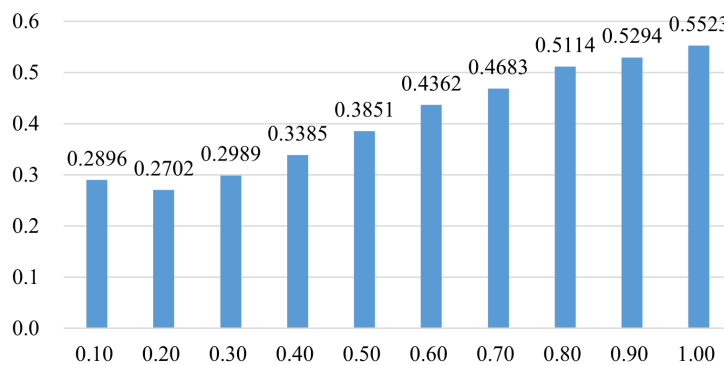


Figure 6.22: Performance of Binary Classifier Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline

Figure 6.23, Figure 6.24 and Figure 6.25 correspond to the performance of the three metric based MIAs: Prediction Class Confidence, Prediction Class Entropy, and Prediction Modified Entropy, respectively, when the number of data augmentations is fixed to one.

These three figures exhibit a similar trend to the figure of binary classifier based MIA. As the alpha value increases, the F1-score continuously rises. This is also because when the alpha value is relatively small, the transformed training images is more similar to the random PCA composite image instead of the original image. As a result, the metric based MIA can also easily identify differences between the training and test images, which is reflected in the F1-score deviating significantly from 0.5.

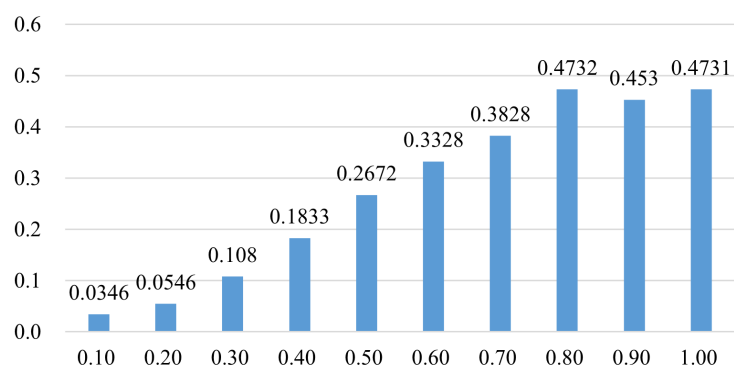


Figure 6.23: Performance of Prediction Class Confidence based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline

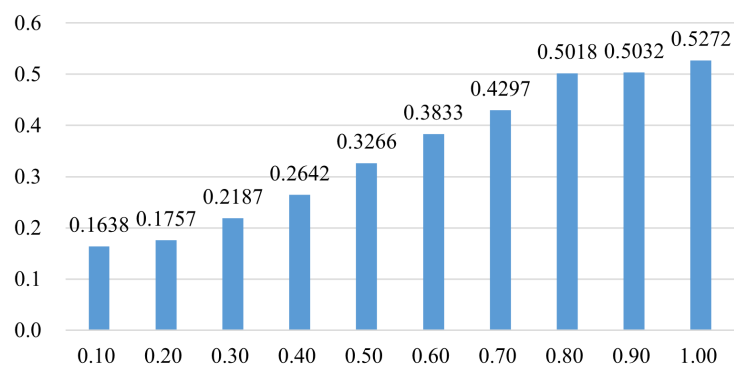


Figure 6.24: Performance of Prediction Class Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline

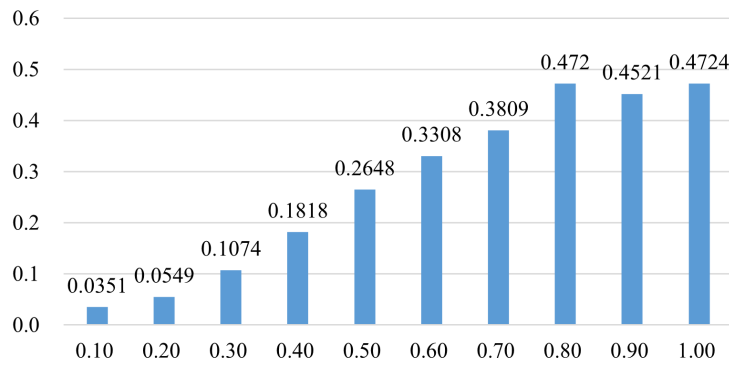


Figure 6.25: Performance of Prediction Modified Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline

Overall, both the training sample accuracy and the F1-score of the two MIA approaches increase as the PCA composite image fusion weight increases under ML pipeline.

## 6.5.2 DFL Pipeline

In the DFL pipeline, CIFAR-10 dataset applies the same defense strength as in the ML pipeline, which means each image undergoes one randomly selected data augmentation and is fused with one randomly selected PCA composite image, with a weight of 0.8.

This thesis also analyze the impact of the PCA composite image fusion weight on defense effectiveness in DFL pipeline by fixing the number of data augmentation to one.

As shown in Figure 6.26, the training sample accuracy shows an upward trend when the weight increases, which is the same as that in the ML pipeline.

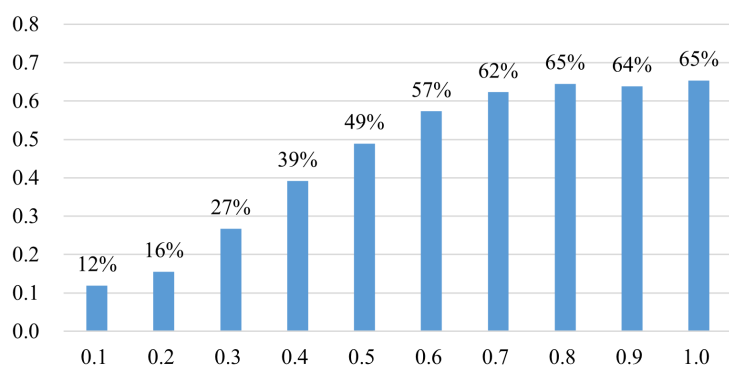


Figure 6.26: Training Sample Accuracy at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline

Figure 6.27 shows the performance of binary classifier based MIA under DFL pipeline. The figure presents an upward trend, similar to the ML pipeline. When the weight is 0.8, the F1-score is closest to 0.5, and the defense effectiveness is at its strongest.

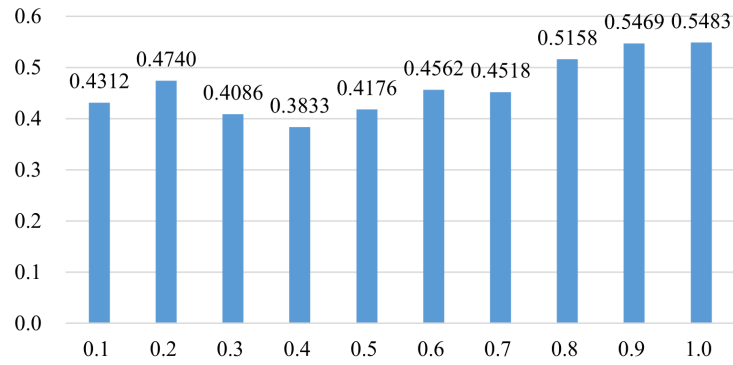


Figure 6.27: Performance of binary classifier Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline

Figure 6.28, Figure 6.29, and Figure 6.30 all present an upward trend, which means F1-score of metric based MIA increases with the rising of the weight. This finding is the same as that under ML pipeline.

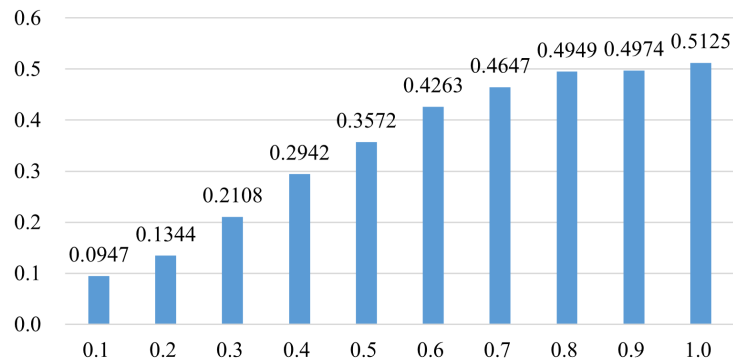


Figure 6.28: Performance of Prediction Class Confidence Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline

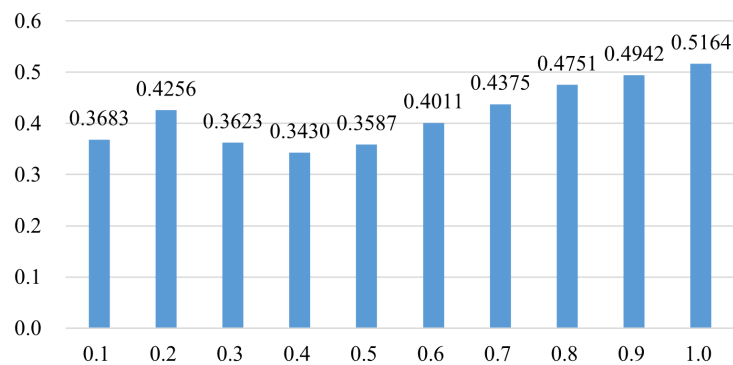


Figure 6.29: Performance of Prediction Class Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline

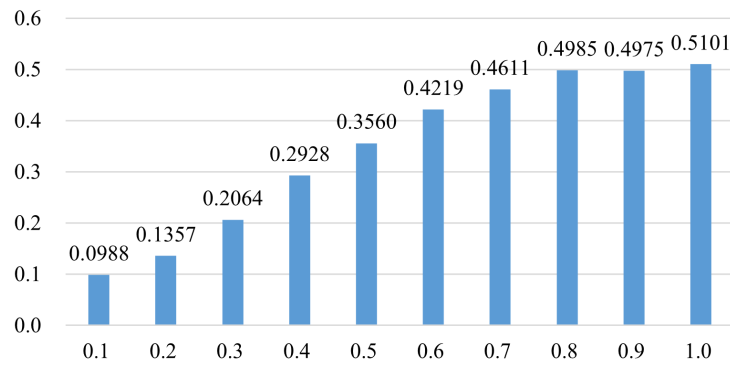


Figure 6.30: Performance of Prediction Modified Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline

To conclude, under the DFL pipeline, both the training sample accuracy and the F1-score of the two MIA approaches increase as the PCA composite image fusion weight increases, which is the same as the ML pipeline.

The conclusion of Section 6.2, section 6.3, section 6.4 and section 6.5 are all based on the CIFAR-10 dataset. These conclusions are also applicable to the CIFAR-100, Fashion MNIST, Tiny ImageNet, and ImageNet-10 datasets, with the related data for these datasets shown in the appendix.

# Chapter 7

## Summary and Conclusions

This chapter provides a summary of the master thesis and discusses potential future work.

### 7.1 Summary and Conclusions

The objective of this thesis is to design, implement, and evaluate an MTD-based defense method to protect DFL from MIA.

To achieve this goal, the master thesis is structured into several key phases. First, it introduces the necessary background knowledge, including FL, MIA, MTD, pHashing, and data augmentation.

Subsequently, an in-depth analysis of existing methods for mitigating MIA is conducted. Based on the literature review, a new MTD-based defense method against MIA is proposed.

In the practical phase, this thesis constructs suitable image classification models for CIFAR-10, CIFAR-100, Fashion MNIST, Tiny ImageNet, and ImageNet-10 dataset and trains them accordingly. The prediction vectors from the target models and shadow models are then obtained, followed by the execution of binary classifier based MIA and metric based MIA based on these prediction vectors.

Subsequently, experiment data is collected for all five datasets under both with defense and without defense conditions, including prediction accuracy, F1-scores of the binary classifier based MIA and metric based MIA.

Notably, these five datasets are each subjected to different defense intensities, with the specific intensity determined by a Python script designed as part of this thesis.

Finally, this thesis conducts various evaluation experiments to perform an in-depth performance assessment, ensuring the robustness and reliability of the proposed defense method.

In conclusion, this thesis have the following findings:

1. The defense method has strong applicability and is suitable for most image datasets.
2. The defense method proposed in this thesis is effective against both binary classifier based MIA and metric based MIA.
3. The defense method works in both the ML pipeline and the DFL pipeline.
4. The parameter settings of the shadow model have limited impact on both the performance of MIA and the effectiveness of the defense method.
5. Both the training sample accuracy and the F1-score of the two MIA approaches decrease with increasing data augmentation intensity.
6. Both the training sample accuracy and the F1-score of the two MIA approaches increase as the PCA composite image fusion weight increases.

The author has implemented an innovative defense method, which mainly consists of three steps, as illustrated in Section 4.1, Section 4.2, and Section 4.3. Therefore, all findings except for Finding 4 are the result of the author's own work.

## 7.2 Future Work

The DFL topology used in this thesis includes only the fully-connected structure. To further evaluate the effectiveness of the proposed defense method, future work will consider target models with alternative DFL topologies such as ring, star, and random topologies.

Besides, Future work will involve experiments on more image datasets to further evaluate the effectiveness of the proposed defense method.

In addition, although this study has proposed a Python script to determine the defense intensity for each dataset, the script currently has a relatively long runtime. Future work will focus on improving its efficiency, to make it easier for other researchers to use.

# Bibliography

- [1] K. Bonawitz, V. Ivanov, B. Kreuter, *et al.*, “Practical secure aggregation for privacy-preserving machine learning”, in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.
- [2] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, 2023.
- [3] N. Bouacida and P. Mohapatra, “Vulnerabilities in federated learning”, *IEEE Access*, vol. 9, pp. 63 229–63 249, 2021.
- [4] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning”, in *2019 IEEE symposium on security and privacy (SP)*, IEEE, 2019, pp. 739–753.
- [5] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, “Memguard: Defending against black-box membership inference attacks via adversarial examples”, in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 259–274.
- [6] L. Hu, J. Li, G. Lin, *et al.*, “Defending against membership inference attacks with high utility by gan”, *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 2144–2157, 2022.
- [7] M. Rigaki and S. Garcia, “A survey of privacy attacks in machine learning”, *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–34, 2023.
- [8] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, “Membership inference attacks on machine learning: A survey”, *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.
- [9] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures”, in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1322–1333.
- [10] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, “Property inference attacks on fully connected neural networks using permutation invariant representations”, in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 619–633.
- [11] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction {apis}”, in *25th USENIX security symposium (USENIX Security 16)*, 2016, pp. 601–618.

- [12] A. Acquisti, L. Brandimarte, and G. Loewenstein, “Privacy and human behavior in the age of information”, *Science*, vol. 347, no. 6221, pp. 509–514, 2015.
- [13] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models”, in *2017 IEEE symposium on security and privacy (SP)*, IEEE, 2017, pp. 3–18.
- [14] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, “Advances and open problems in federated learning”, *Foundations and trends<sup>®</sup> in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [15] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, “Exploiting unintended feature leakage in collaborative learning”, in *2019 IEEE symposium on security and privacy (SP)*, IEEE, 2019, pp. 691–706.
- [16] J. Zhang, J. Zhang, J. Chen, and S. Yu, “Gan enhanced membership inference: A passive local attack in federated learning”, in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020, pp. 1–6.
- [17] Y. Gu, Y. Bai, and S. Xu, “Cs-mia: Membership inference attack based on prediction confidence series in federated learning”, *Journal of Information Security and Applications*, vol. 67, p. 103 201, 2022.
- [18] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, “Demystifying membership inference attacks in machine learning as a service”, *IEEE transactions on services computing*, vol. 14, no. 6, pp. 2073–2089, 2019.
- [19] D. Chen, N. Yu, Y. Zhang, and M. Fritz, “Gan-leaks: A taxonomy of membership inference attacks against generative models”, in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 343–362.
- [20] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.
- [21] Y. Tashiro, Y. Song, and S. Ermon, “Diversity can be transferred: Output diversification for white-and black-box attacks”, *Advances in neural information processing systems*, vol. 33, pp. 4536–4548, 2020.
- [22] Z. Zhang, L. Yu Zhang, X. Zheng, B. Hussain Abbasi, and S. Hu, “Evaluating membership inference through adversarial robustness”, *The Computer Journal*, vol. 65, no. 11, pp. 2969–2978, 2022.
- [23] Y. Chen, C. Shen, Y. Shen, C. Wang, and Y. Zhang, “Amplifying membership exposure via data poisoning”, *Advances in Neural Information Processing Systems*, vol. 35, pp. 29 830–29 844, 2022.
- [24] H. Hu, Z. Salcic, L. Sun, G. Dobbie, and X. Zhang, “Source inference attacks in federated learning”, in *2021 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2021, pp. 1102–1107.
- [25] R. Zhuang, S. A. DeLoach, and X. Ou, “Towards a theory of moving target defense”, in *Proceedings of the first ACM workshop on moving target defense*, 2014, pp. 31–40.
- [26] G.-l. Cai, B.-s. Wang, W. Hu, and T.-z. Wang, “Moving target defense: State of the art and characteristics”, *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 11, pp. 1122–1153, 2016.

- [27] J. Zheng and A. S. Namin, “A survey on the moving target defense strategies: An architectural perspective”, *Journal of Computer Science and Technology*, vol. 34, pp. 207–233, 2019.
- [28] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, “Moving target defense techniques: A survey”, *Security and Communication Networks*, vol. 2018, no. 1, p. 3759626, 2018.
- [29] D. Kramer and W. Karl, “Realizing a proactive, self-optimizing system behavior within adaptive, heterogeneous many-core architectures”, in *2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*, IEEE, 2012, pp. 39–48.
- [30] P. K. Manadhata and J. M. Wing, “An attack surface metric”, *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2010.
- [31] National Science and Technology Council, *Cybersecurity game-change research & development recommendations*, [http://www.nitrd.gov/pubs/CSIA\\_IWG\\_Cybersecurity\\_GameChange\\_RD\\_Recommendations\\_20100513.pdf](http://www.nitrd.gov/pubs/CSIA_IWG_Cybersecurity_GameChange_RD_Recommendations_20100513.pdf), [EB/OL], 2010.
- [32] J.-H. Cho, D. P. Sharma, H. Alavizadeh, *et al.*, “Toward proactive, adaptive defense: A survey on moving target defense”, *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.
- [33] M. Taguinod, A. Doupé, Z. Zhao, and G.-J. Ahn, “Toward a moving target defense for web applications”, in *2015 IEEE international conference on information reuse and integration*, IEEE, 2015, pp. 510–517.
- [34] H. Alavizadeh, D. S. Kim, and J. Jang-Jaccard, “Model-based evaluation of combinations of shuffle and diversity mtd techniques on the cloud”, *Future Generation Computer Systems*, vol. 111, pp. 507–522, 2020.
- [35] L. Chi and X. Zhu, “Hashing techniques: A survey and taxonomy”, *ACM Computing Surveys (Csur)*, vol. 50, no. 1, pp. 1–36, 2017.
- [36] H. Dos Santos, T. S. Martins, J. A. Barreto, *et al.*, “Chasam: An architecture based on perceptual hashing for image detection in computer forensics”, *IEEE Access*, 2024.
- [37] C. Zauner, “Implementation and benchmarking of perceptual image hash functions”, 2010.
- [38] R. W. Hamming, “Error detecting and error correcting codes”, *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.
- [39] B. Yang, F. Gu, and X. Niu, “Block mean value based image perceptual hashing”, in *2006 International Conference on Intelligent Information Hiding and Multimedia*, IEEE, 2006, pp. 167–172.
- [40] B. Waggener and W. N. Waggener, *Pulse code modulation techniques*. Springer Science & Business Media, 1995.
- [41] jdhaos’s digital space, *The youtube content id system*, Accessed: 2024-10-27, 2021. [Online]. Available: [https://jdhao.github.io/2021/08/02/the\\_youtube\\_content\\_id\\_system/](https://jdhao.github.io/2021/08/02/the_youtube_content_id_system/).

- [42] X. Sun and J. Zhou, “Deep perceptual hash based on hash center for image copyright protection”, *IEEE Access*, vol. 10, pp. 120 551–120 562, 2022.
- [43] Microsoft, *Microsoft photodna*, Accessed: 2024-10-27, 2024. [Online]. Available: <https://www.microsoft.com/en-us/photodna>.
- [44] L. Struppek, D. Hintersdorf, D. Neider, and K. Kersting, “Learning to break deep perceptual hashing: The use case neuralhash”, in *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022, pp. 58–69.
- [45] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, “Face recognition: A literature survey”, *ACM computing surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.
- [46] R. Biswas, V. González-Castro, E. Fidalgo, and E. Alegre, “A new perceptual hashing method for verification and identity classification of occluded faces”, *Image and Vision Computing*, vol. 113, p. 104 245, 2021.
- [47] P. Chlap, H. Min, N. Vandenberg, J. Dowling, L. Holloway, and A. Haworth, “A review of medical image data augmentation techniques for deep learning applications”, *Journal of Medical Imaging and Radiation Oncology*, vol. 65, no. 5, pp. 545–563, 2021.
- [48] J. Shijie, W. Ping, J. Peiyi, and H. Siping, “Research on data augmentation for image classification based on convolution neural networks”, in *2017 Chinese automation congress (CAC)*, IEEE, 2017, pp. 4165–4170.
- [49] L. Taylor and G. Nitschke, “Improving deep learning using generic data augmentation”, *arXiv preprint arXiv:1708.06020*, 2017.
- [50] P. Kaur, B. S. Khehra, and E. B. S. Mavi, “Data augmentation for object detection: A review”, in *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE, 2021, pp. 537–543.
- [51] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning”, *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.
- [52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, “Generative adversarial nets”, *Advances in neural information processing systems*, vol. 27, 2014.
- [53] Z. Chen and K. Pattabiraman, “Overconfidence is a dangerous thing: Mitigating membership inference attacks by enforcing less confident prediction”, *arXiv preprint arXiv:2307.01610*, 2023.
- [54] D. Chen, N. Yu, and M. Fritz, “Relaxloss: Defending membership inference attacks without losing utility”, *arXiv preprint arXiv:2207.05801*, 2022.
- [55] Y. Kaya and T. Dumitras, “When does data augmentation help with membership inference attacks?”, in *International conference on machine learning*, PMLR, 2021, pp. 5345–5355.
- [56] J. Chen, W. H. Wang, H. Gao, and X. Shi, “Par-gan: Improving the generalization of generative adversarial networks against membership inference attacks”, in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 127–137.
- [57] V. Shejwalkar and A. Houmansadr, “Membership privacy for machine learning models through knowledge transfer”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 9549–9557.

- [58] Y. Yin, K. Chen, L. Shou, and G. Chen, “Defending privacy against more knowledgeable membership inference attackers”, in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2026–2036.
- [59] Y. Wang, C. Wang, Z. Wang, *et al.*, “Against membership inference attack: Pruning is all you need”, *arXiv preprint arXiv:2008.13578*, 2020.
- [60] B. Wu, C. Chen, S. Zhao, *et al.*, “Characterizing membership privacy in stochastic gradient langevin dynamics”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 6372–6379.
- [61] S. Tople, A. Sharma, and A. Nori, “Alleviating privacy attacks via causal learning”, in *International Conference on Machine Learning*, PMLR, 2020, pp. 9537–9547.
- [62] M. Nasr, R. Shokri, and A. Houmansadr, “Machine learning with membership privacy using adversarial regularization”, in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 634–646.
- [63] M. Tan, X. Xie, J. Sun, and T. Wang, “Mitigating membership inference attacks via weighted smoothing”, in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 787–798.
- [64] F. Mazzone, L. van den Heuvel, M. Huber, *et al.*, “Repeated knowledge distillation with confidence masking to mitigate membership inference attacks”, in *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*, 2022, pp. 13–24.
- [65] S. Mukherjee, Y. Xu, A. Trivedi, N. Patowary, and J. L. Ferres, “Privgan: Protecting gans from membership inference attacks at low cost to utility”, *Proceedings on Privacy Enhancing Technologies*, 2021.
- [66] J. Li, N. Li, and B. Ribeiro, “Membership inference attacks and defenses in classification models”, in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, 2021, pp. 5–16.
- [67] K. Wang, Z. Hu, Q. Ai, *et al.*, “Membership inference attack with multi-grade service models in edge intelligence”, *IEEE Network*, vol. 35, no. 1, pp. 184–189, 2021.
- [68] H. Lee, J. Kim, S. Ahn, R. Hussain, S. Cho, and J. Son, “Digestive neural networks: A novel defense strategy against inference attacks in federated learning”, *computers & security*, vol. 109, p. 102378, 2021.
- [69] W. Paul, Y. Cao, M. Zhang, and P. Burlina, “Defending medical image diagnostics against privacy attacks using generative methods: Application to retinal diagnostics”, in *Clinical Image-Based Procedures, Distributed and Collaborative Learning, Artificial Intelligence for Combating COVID-19 and Secure and Privacy-Preserving Machine Learning: 10th Workshop, CLIP 2021, Second Workshop, DCL 2021, First Workshop, LL-COVID19 2021, and First Workshop and Tutorial, PPML 2021, Held in Conjunction with MICCAI 2021, Strasbourg, France, September 27 and October 1, 2021, Proceedings 2*, Springer, 2021, pp. 174–187.
- [70] Z. Ying, Y. Zhang, and X. Liu, “Privacy-preserving in defending against membership inference attacks”, in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, 2020, pp. 61–63.

- [71] J. Chen, W. H. Wang, and X. Shi, “Differential privacy protection against membership inference attack on machine learning for genomic data”, in *BIOCOMPUTING 2021: Proceedings of the Pacific Symposium*, World Scientific, 2020, pp. 26–37.
- [72] Y. Liu, J. Peng, J. Kang, A. M. Ilyasu, D. Niyato, and A. A. Abd El-Latif, “A secure federated learning framework for 5g networks”, *IEEE Wireless Communications*, vol. 27, no. 4, pp. 24–31, 2020.
- [73] A. Triastcyn and B. Faltings, “Generating artificial data for private deep learning”, *arXiv preprint arXiv:1803.03148*, 2018.
- [74] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, “Generative adversarial networks: An overview”, *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.
- [75] M. A. Rahman, T. Rahman, R. Laganière, N. Mohammed, and Y. Wang, “Membership inference attack against differentially private deep learning model.”, *Trans. Data Priv.*, vol. 11, no. 1, pp. 61–79, 2018.
- [76] M. Abadi, A. Chu, I. Goodfellow, *et al.*, “Deep learning with differential privacy”, in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.
- [77] J. Zheng, Y. Cao, and H. Wang, “Resisting membership inference attacks through knowledge distillation”, *Neurocomputing*, vol. 452, pp. 114–126, 2021.

# Abbreviations

aHash	Average Hash
BER	Bit Error Rate
CFL	Centralized Federated Learning
CNN	Convolutional Neural Network
CSAM	Child Sexual Abuse Material
DCT	Discrete Cosine Transformation
DFL	Decentralized Federated Learning
dHash	Difference Hash
DL	Deep Learning
DNN	Digestive Neural Networks
DP	Differential Privacy
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
FL	Federated Learning
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
IoT	Internet of Things
MIA	Membership Inference Attack
ML	Machine Learning
MTD	Moving Target Defense
PCA	Principal Component Analysis
PCC	Peak of Cross Correlation
pHash	Perceptual Hash
SIA	Source Inference Attack
TN	True Negative
TP	True Positive



# List of Figures

2.1	Work Mechanism of FL[2]	6
2.2	White-box and Black-box Setting of MIA[8]	8
2.3	Workflow of an MTD System [25]	13
2.4	Proactive and Reactive Architecture of MTD[28]	16
4.1	Architecture	33
4.2	Determining Whether the Query Image is in the Training Dataset	34
4.3	Number of Duplicate Hash Values in the Training Set	35
4.4	Frequency with Which a Test Sample's Hash Appears in the Training Set's Hash List	35
6.1	Model Prediction Accuracy without Defense	58
6.2	Model Prediction Accuracy with Defense	58
6.3	Performance of Binary Classifier Based MIA	59
6.4	Prediction Class Confidence	59
6.5	Prediction Class Entropy	59
6.6	Prediction Modified Entropy	59
6.7	Performance of Binary Classifier Based MIA	60
6.8	Prediction Class Confidence	60
6.9	Prediction Class Entropy	60
6.10	Prediction Modified Entropy	60
6.11	Training Sample Accuracy at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline	61

6.12	Performance of Binary Classifier Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline . . . . .	61
6.13	Performance of Prediction Class Confidence Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline .	62
6.14	Performance of Prediction Class Entropy Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline .	62
6.15	Performance of Prediction Modified Entropy based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under ML Pipeline .	63
6.16	Training Sample Accuracy at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline . . . . .	63
6.17	Performance of Binary Classifier Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline . . . . .	64
6.18	Performance of Prediction Class Confidence Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline	64
6.19	Performance of Prediction Class Entropy Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline	65
6.20	Performance of Prediction Modified Entropy Based MIA at Different Data Augmentation Intensities with Fixed Weight being 0.8 under DFL Pipeline	65
6.21	Training Sample Accuracy at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline . . . . .	66
6.22	Performance of Binary Classifier Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline . .	66
6.23	Performance of Prediction Class Confidence based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline . . . . .	67
6.24	Performance of Prediction Class Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline . . . . .	67
6.25	Performance of Prediction Modified Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under ML Pipeline . . . . .	68
6.26	Training Sample Accuracy at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline . . . . .	68
6.27	Performance of binary classifier Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline .	69

6.28	Performance of Prediction Class Confidence Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline . . . . .	69
6.29	Performance of Prediction Class Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline . . . . .	69
6.30	Performance of Prediction Modified Entropy Based MIA at Different PCA Composite Image Fusion Weights with One Data Augmentation under DFL Pipeline . . . . .	70
B.1	Train Accuracy of CIFAR-100 Dataset . . . . .	93
B.2	Test Accuracy of CIFAR-100 Dataset . . . . .	93
B.3	Train Accuracy of Fashion MNIST Dataset . . . . .	94
B.4	Test Accuracy of Fashion MNIST Dataset . . . . .	94
B.5	Train Accuracy of Tiny ImageNet Dataset . . . . .	94
B.6	Test Accuracy of Tiny ImageNet Dataset . . . . .	94
B.7	Train Accuracy of ImageNet-10 Dataset . . . . .	94
B.8	Test Accuracy of ImageNet-10 Dataset . . . . .	94
B.9	Performance of Binary Classifier based MIA of CIFAR-100 Dataset . . . .	95
B.10	Performance of Binary Classifier based MIA of Fashion MNIST Dataset . .	95
B.11	Performance of Binary Classifier based MIA of Tiny Imagenet Dataset . .	95
B.12	Performance of Binary Classifier based MIA of Imagenet-10 . . . . .	95
B.13	Performance of Prediction Class Confidence based MIA of CIFAR-100 . . .	96
B.14	Performance of Prediction Class Confidence based MIA of Fashion MNIST	96
B.15	Performance of Prediction Class Confidence based MIA of Tiny ImageNet .	96
B.16	Performance of Prediction Class Confidence based MIA of ImageNet-10 . .	96
B.17	Performance of Prediction Class Entropy based MIA of CIFAR-100 . . . .	96
B.18	Performance of Prediction Class Entropy based MIA of Fashion MNIST . .	96
B.19	Performance of Prediction Class Entropy based MIA of Tiny ImageNet . .	97
B.20	Performance of Prediction Class Entropy based MIA of ImageNet-10 . . .	97
B.21	Performance of Prediction Modified Entropy based MIA of CIFAR-100 . .	97

B.22 Performance of Prediction Modified Entropy based MIA of Fashion MNIST	97
B.23 Performance of Prediction Modified Entropy based MIA of Tiny ImageNet	97
B.24 Performance of Prediction Class Entropy based MIA of ImageNet-10 . . .	97

# List of Tables

2.1	Four Types of Metrics[8]	11
2.2	Three Elements of MTD	15
2.3	Metrics for Measuring MTD Effectiveness	17
2.4	Metrics for Measuring MTD Efficiency	18
2.5	Summary of Augmentation Methods Provided by <i>torchvision.transforms</i>	24
3.1	Advantages and Disadvantages of Methods for Mitigating MIA	30
5.1	Comparison of Five Image Datasets Used in Classification Tasks	40
5.2	Three Metrics Used in Metric Based MIA	46
5.3	Roughly Preselected Set of Combinations of <i>num</i> and <i>weights</i>	47
6.1	Experimental Configuration in ML Pipeline for Different Datasets	52
6.2	Experimental Overview in ML pipeline for Different Datasets	54
6.3	Experimental Configuration in DFL Pipeline for Different Datasets	56
6.4	Experimental Overview in DFL Pipeline for Different Datasets	57



# Appendix A

## Installation Guidelines

This chapter introduces the functionality of code and explains how to run it. You can find the whole code along with running guides at [GitHub Repository](#).

*test.py* is the controlling code; it could control the execution of other scripts. Its content reflects the sequence in which other codes are run as shown in script A.1.

The following sections will introduce each of the scripts imported by *test.py*.

```
1 import target_model as tm
2 import shadow_result as sr
3 import random_query as rq
4 import random_shadow_train as rst
5 import performing_defense as pd
6 import MIA.ShadowModelMIA as smm
7 import MIA.ClassMetricMIA as cmm
8 import numpy as np
9 import optimal_defense_intensity as odi
10
11 if __name__ == "__main__":
12     tm.generate_target_model()
13     # There are two configurations for shadow training in ML pipeline,
14     # another parameter setting is sr.generate_shadow_result(1, 25000,
15     # 5000)
16     sr.generate_shadow_result(10, 5000, 1000)
17     rq.perform_random_query(5000)
18     # For another configuration, parameter setting is 5000
19     rst.perform_random_shadow_train(10000)
20
21     # Finding suitable defense intensity for this dataset
22     best_result = odi.generate_optimal_defense_intensity()
23     best_num, best_weights, best_alpha, diff, metrics = best_result
24     print("\n>>> Best overall config (with minimum diff):")
25     print(f"num={best_num}, weights={best_weights}, alpha={best_alpha:.2
26     f}, diff={diff:.4f}, metrics={metrics}")
27
28     # study the impact of PCA composite image fusion weight on defense
29     # effectiveness
30     for alpha in np.arange(0, 1.1, 0.1):
```

```
27     pd.perform_defense(best_num, best_weights, alpha)
28     smm.perform_shadow_model_mia()
29     cmm.perform_class_metric_mia()
30     print(f"Running for num={best_num}, weights={best_weights},
alpha = {alpha}")
31
32     defense_params = [
33     ([0, 1], [1, 0]),
34     ([0, 1], [0.5, 0.5]),
35     ([0, 1], [0, 1]),
36     ([1, 2], [0.5, 0.5]),
37     ([1, 2], [0, 1]),
38     ([2, 3], [0.5, 0.5]),
39     ([2, 3], [0, 1]),
40     ([3, 4], [0.5, 0.5]),
41     ([3, 4], [0, 1]),
42     ]
43
44     for num, weights in defense_params:
45         pd.perform_defense(num, weights, best_alpha)
46         smm.perform_shadow_model_mia()
47         cmm.perform_class_metric_mia()
48         print(f"Running for num = {num}, weights = {weights}, alpha = {
best_alpha}")
```

Listing A.1: Code of text.py

## A.1 one\_to\_two.py

This script is used to split the image dataset evenly by class into two subsets, which will later be used for training the target model and the shadow model respectively.

This script only needs to be run once, and the generated subsets can be reused multiple times.

## A.2 PCA.py

This script performs Principal Component Analysis on the image dataset by class and generates a representative image for each class based on the first principal component.

This script also only needs to be run once, and the generated PCA composite images can be reused multiple times.

## A.3 target\_model.py

This code is primarily used to train a classification model on an image dataset and save the trained model for future use. During this process, it also involves data loading, pHash

calculation for all images, and data preprocessing.

In the end, the trained model is saved as *final\_global\_model.pth*, and the list of pHash values for the target dataset is saved as *sorted\_train\_phashes\_decimal.npy* for later use.

This script can be executed in two ways:

- **First**, importing it in *test.py*, as shown in lines 1 and 12 of the script A.1, using *import target\_model as tm* and calling *tm.generate\_target\_model()*.
- **Second**, running *python target\_model.py* directly in the terminal.

*target\_model.py* is the name used in the ML pipeline scenario, while in the DFL pipeline, it is named *DFL\_target\_model.py*.

## A.4 shadow\_result.py

This script is mainly used for training shadow models and computing the predicted labels for each training sample and test sample after training. It also includes modules for generating shadow datasets.

Finally, the predicted label results for the training samples are saved as *shadow\_train\_res.pt*, and those for the test samples are saved as *shadow\_test\_res.pt*.

This script can also be executed in two ways:

- **First**, importing it in *test.py*, as shown in lines 2 and 14 of the script A.1, using *import shadow\_result as sr* and calling *sr.generate\_shadow\_result(10, 5000, 1000)*, in which the first parameter represents the number of shadow models, the second parameter indicates the number of training samples in each shadow dataset, and the third parameter specifies the number of test samples in each shadow dataset.
- **Second**, running *python target\_model.py* directly in the terminal. Before running the script, you can adjust the parameters of *generate\_shadow\_result* as needed.

*shadow\_result.py* is the name used in the ML pipeline scenario, while in the DFL pipeline, it is named *DFL\_shadow\_result.py*.

## A.5 random\_query.py

Randomly select a subset of training samples from the target dataset with the same size to the test samples, and save them together as a new file. After the target model is trained, this file is loaded to compute the prediction results for each sample. It is to ensure fair metric calculation. Assuming that the attacker has an equal number of training and

test samples in practice, then the experiment should also maintain this balance when calculating precision, recall, and F1-score.

This script is only used in the ML pipeline. As for the DFL pipeline, an alternative script named *random\_target\_train.py* is used, which will be introduced in the next section.

## A.6 random\_target\_train.py

This script serves the same purpose as *random\_query.py*, to ensure balanced classes in the binary classifier. The difference is that it is used in the DFL pipeline. In this case, predictions are first generated for all training samples, and then a subset equal in size to the test samples is randomly selected.

Since the DFL pipeline involves multiple participants and uses indices to track which samples belong to which participant, it is more difficult to follow the same logic as in *random\_query.py*.

## A.7 random\_shadow\_train.py

This script selects a fixed number of predictions and true labels from the trained shadow model and saves them to a new file. It is for the purpose of ensuring balanced classes in the binary classifier, as introduced in section 5.7, which is a common practice in machine learning.

## A.8 performing\_defense.py

This script applies data augmentation and fusion with PCA composite images to the query images as a defense method against MIA. It saves the prediction results after applying defense method for both training and test samples as *train\_results.pt* and *test\_results.pt*.

The script can be executed in two ways as well.

- **First**, importing it in *test.py*, as shown in lines 5 and 27 of the script A.1, using *import performing\_defense as pd* and calling *pd.perform\_defense(best\_num, best\_weights, alpha)*, in which the first parameter represents a list of integers representing the candidate numbers of data augmentation methods to be applied, the second parameter indicates a weight list to control the probability of selecting each corresponding number, and the third parameter specifies the PCA Composite image fusion weight.
- **Second**, running *python performing\_defense.py* directly in the terminal. Before running the script, you can adjust the parameters of *perform\_defense* function as needed.

## A.9 ShadowModelMIA.py

This code is used to perform shadow model based MIA. Specifically, it trains a binary classification attack model using the member and non-member prediction results generated by shadow models, in order to determine whether a given sample belongs to the training dataset.

The script can be executed in two ways as well.

- **First**, importing it in *test.py*, as shown in lines 6 and 28 of the script A.1, using *import MIA.ShadowModelMIA as smm* and calling *smm.perform\_shadow\_model\_mia()*.
- **Second**, running it directly in the terminal.

## A.10 ClassMetricMIA.py

This script implements a metric based MIA. It determines the optimal threshold for distinguishing member and non-member samples based on the metric values from shadow model predictions.

Similarly, it could be executed in two ways.

## A.11 optimal\_defense\_intensity.py

This script automatically determines the defense parameters, including data augmentation intensity and PCA composite image fusion weight, to find an appropriate defense intensity for each image dataset.

The script could be run directly, or imported and called from *test.py*.

## A.12 Summary

In short, if a researcher want to run the code of this master thesis, then the researcher could run *one\_to\_two.py* and *PCA.py* first, then run *test.py*.



# Appendix B

## Experimental Data of Other Four Datasets

### B.1 Prediction Accuracy of Training Samples and Test Samples

The image on the left represents the training accuracy before and after applying the defense method, while the image on the right shows the testing accuracy before and after applying the defense method.

The results indicate that the change in testing accuracy is negligible, suggesting that the defense method does not affect the model's performance for normal users.

In addition, the training accuracy after applying the defense method becomes close to the testing accuracy.

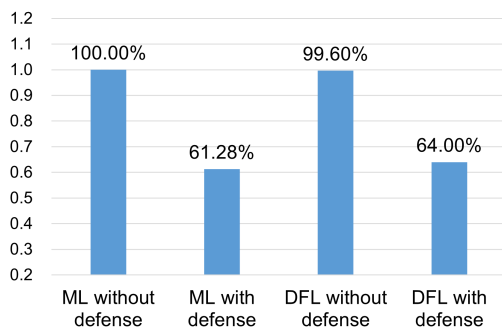


Figure B.1: Train Accuracy of CIFAR-100 Dataset

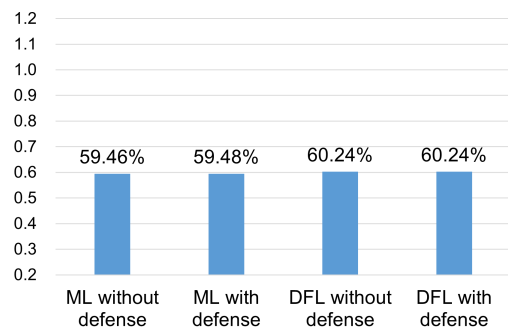


Figure B.2: Test Accuracy of CIFAR-100 Dataset

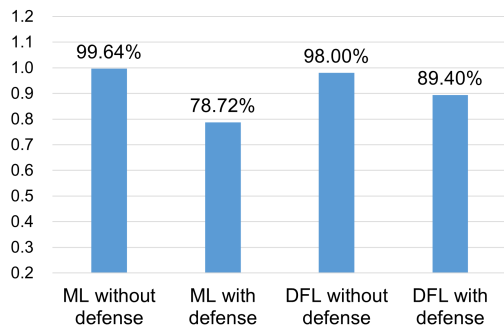


Figure B.3: Train Accuracy of Fashion MNIST Dataset

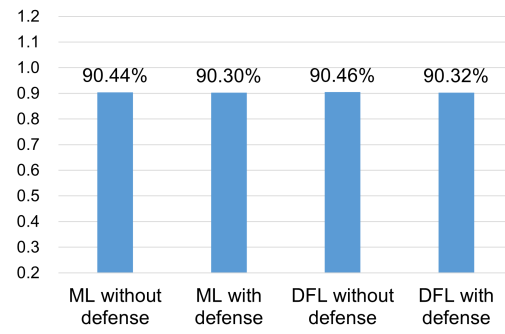


Figure B.4: Test Accuracy of Fashion MNIST Dataset

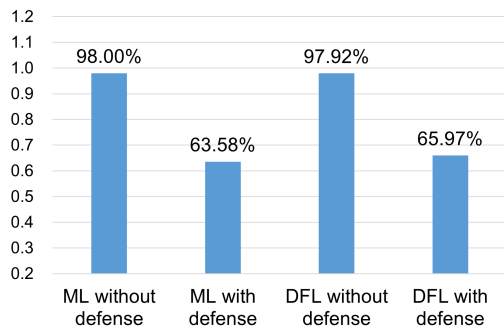


Figure B.5: Train Accuracy of Tiny ImageNet Dataset

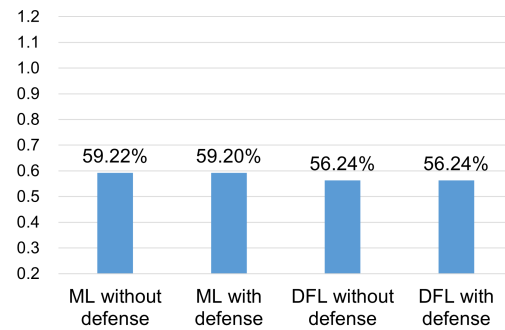


Figure B.6: Test Accuracy of Tiny ImageNet Dataset

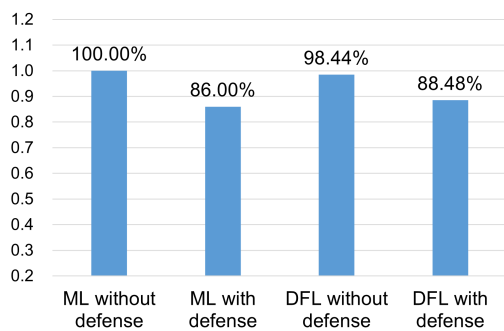


Figure B.7: Train Accuracy of ImageNet-10 Dataset

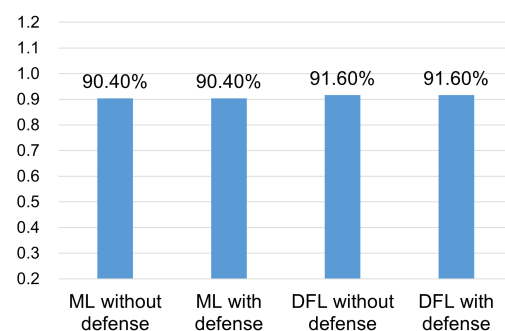


Figure B.8: Test Accuracy of ImageNet-10 Dataset

## B.2 Performance of Binary Classifier based MIA

The performance of binary classifier based MIA on the other four datasets, before and after applying defense method in both ML and DFL pipeline, is shown in Figure B.9, Figure B.10, Figure B.11, Figure B.12.

From all four figures, it can be observed that the F1-score of binary classifier based MIA drops significantly after applying the defense. Under the defense state, the F1-scores fall within the range of  $[0.4, 0.6]$ , indicating that the defense method is effective against binary classifier based MIA.

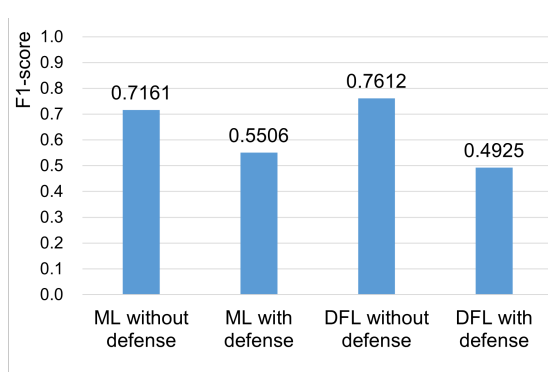


Figure B.9: Performance of Binary Classifier based MIA of CIFAR-100 Dataset

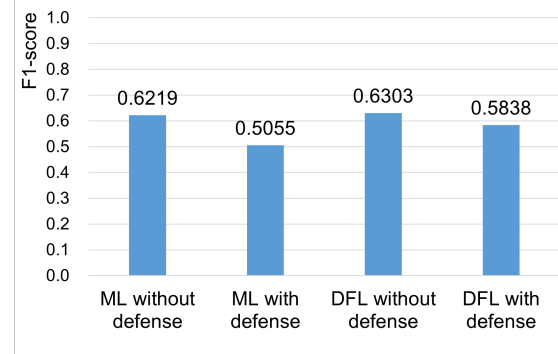


Figure B.10: Performance of Binary Classifier based MIA of Fashion MNIST Dataset

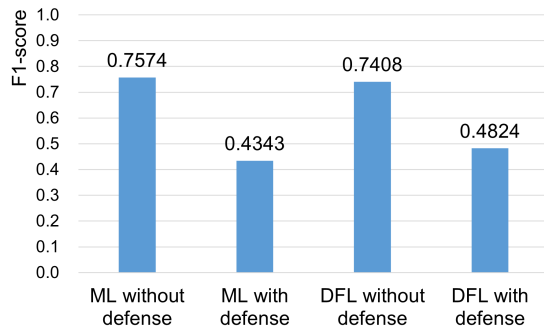


Figure B.11: Performance of Binary Classifier based MIA of Tiny Imagenet Dataset

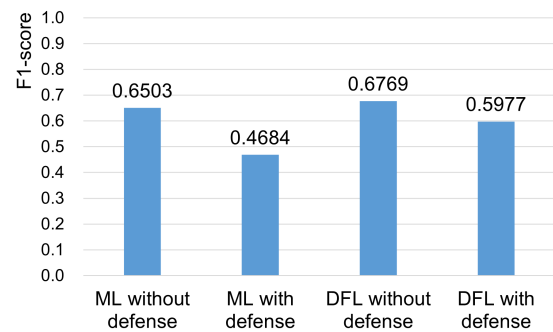


Figure B.12: Performance of Binary Classifier based MIA of Imagenet-10

## B.3 Performance of Metric based MIA

### B.3.1 Performance of Prediction Class Confidence based MIA

The following four images show the performance of prediction class confidence based MIA; the results are similar to those of binary classifier based MIA.

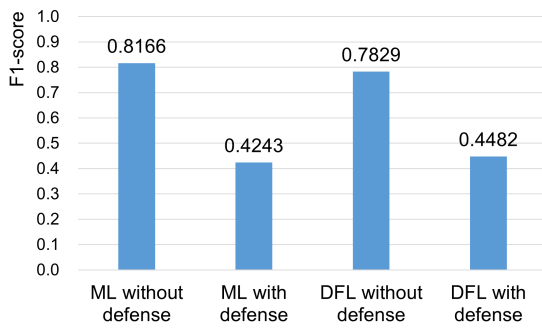


Figure B.13: Performance of Prediction Class Confidence based MIA of CIFAR-100

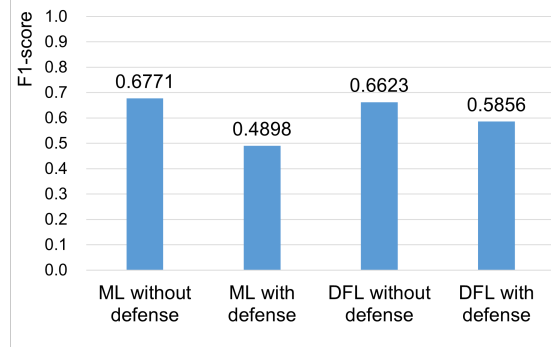


Figure B.14: Performance of Prediction Class Confidence based MIA of Fashion MNIST

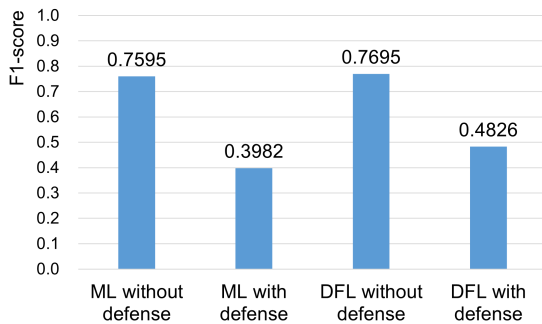


Figure B.15: Performance of Prediction Class Confidence based MIA of Tiny ImageNet

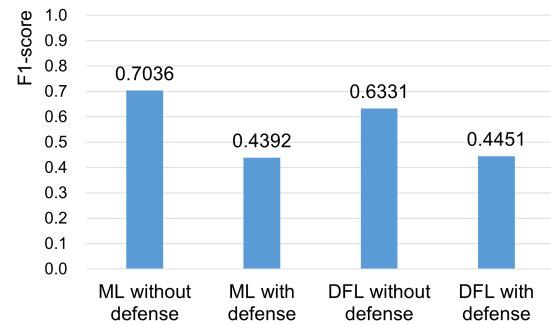


Figure B.16: Performance of Prediction Class Confidence based MIA of ImageNet-10

### B.3.2 Performance of Prediction Class Entropy based MIA

The following four images show the performance of prediction class entropy based MIA; the results are similar to those of binary classifier based MIA.

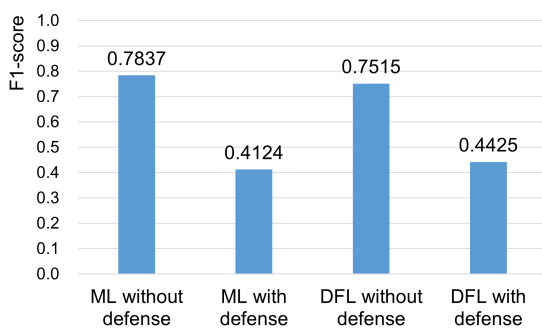


Figure B.17: Performance of Prediction Class Entropy based MIA of CIFAR-100

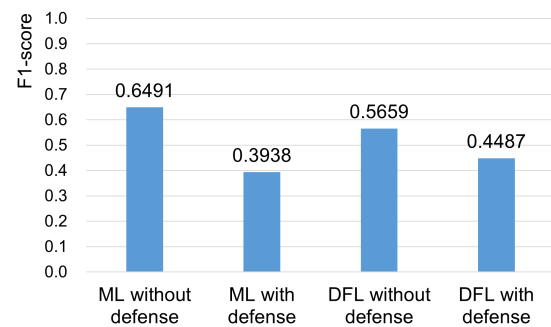


Figure B.18: Performance of Prediction Class Entropy based MIA of Fashion MNIST

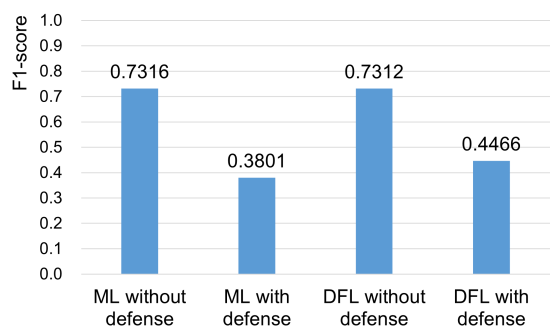


Figure B.19: Performance of Prediction Class Entropy based MIA of Tiny ImageNet

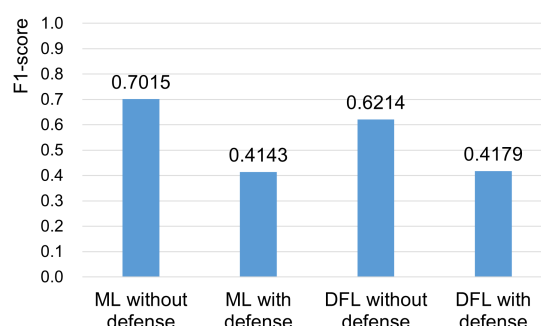


Figure B.20: Performance of Prediction Class Entropy based MIA of ImageNet-10

### B.3.3 Performance of Prediction Modified Entropy based MIA

The following four images exhibit the performance of prediction modified entropy based MIA; the results are similar to those of binary classifier based MIA.

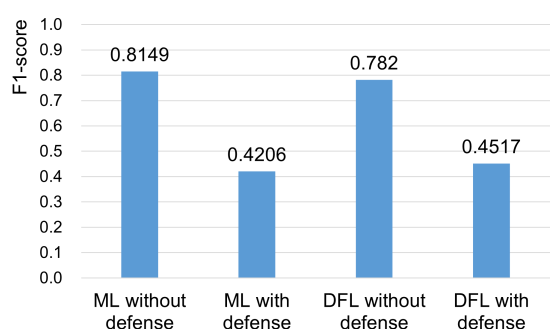


Figure B.21: Performance of Prediction Modified Entropy based MIA of CIFAR-100

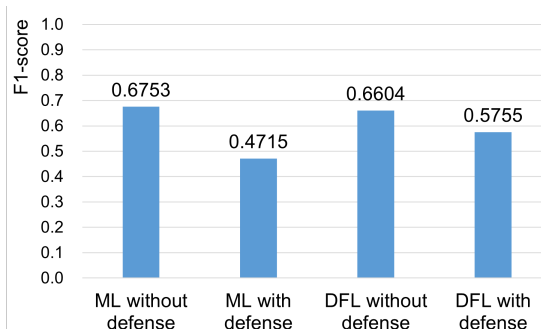


Figure B.22: Performance of Prediction Modified Entropy based MIA of Fashion MNIST

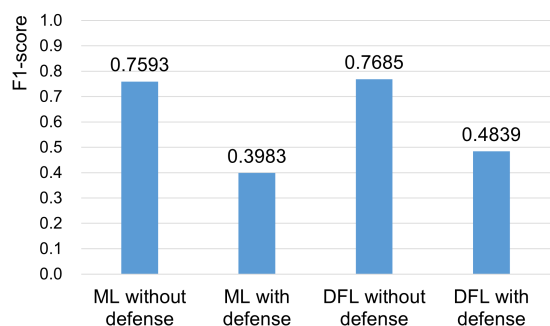


Figure B.23: Performance of Prediction Modified Entropy based MIA of Tiny ImageNet

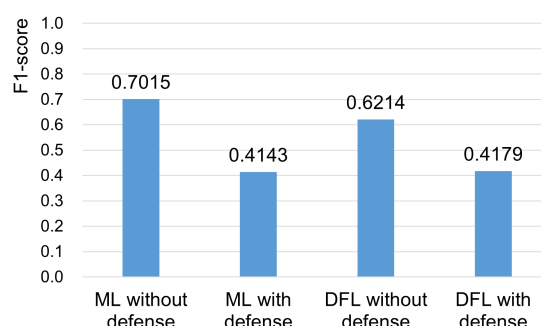


Figure B.24: Performance of Prediction Class Entropy based MIA of ImageNet-10



# Appendix C

## Contents of the CD

The CD contains all the documents, project source code, and installation instructions for this project.