

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

FORTIFYING ROBUSTNESS: UNVEILING THE INTRICACIES OF TRAINING AND INFERENCE VULNERABILITIES IN CENTRALIZED AND FEDERATED NEURAL NETWORKS

by
Guanxiong Liu

Neural network (NN) classifiers have gained significant traction in diverse domains such as natural language processing, computer vision, and cybersecurity, owing to their remarkable ability to approximate complex latent distributions from data. Nevertheless, the conventional assumption of an attack-free operating environment has been challenged by the emergence of adversarial examples. These perturbed samples, which are typically imperceptible to human observers, can lead to misclassifications by the NN classifiers. Moreover, recent studies have uncovered the ability of poisoned training data to generate Trojan backdoored classifiers that exhibit misclassification behavior triggered by predefined patterns.

In recent years, significant research efforts have been dedicated to uncovering the vulnerabilities of NN classifiers and developing defenses or mitigations against them. However, the existing approaches still fall short of providing mature solutions to address this ever-evolving problem. The widely adopted defense mechanisms against adversarial examples are computationally expensive and impractical for certain real-world applications. Likewise, the practical black-box defense against Trojan backdoors has failed to achieve state-of-the-art performance. More concerning is the limited exploration of these vulnerabilities within the context of cooperative attack or Federated learning, leaving NN classifiers exposed to unknown risks. This dissertation aims to address these critical gaps and refine our understanding of these vulnerabilities. The research conducted within this dissertation encompasses both the attack and defense perspectives, aiming to shed light on future research directions for vulnerabilities in NN classifiers.

**FORTIFYING ROBUSTNESS: UNVEILING THE INTRICACIES OF
TRAINING AND INFERENCE VULNERABILITIES IN
CENTRALIZED AND FEDERATED NEURAL NETWORKS**

by
Guanxiong Liu

**A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical and Computer Engineering**

**Helen and John C. Hartmann Department of
Electrical and Computer Engineering**

August 2023

Copyright © 2023 by Guanxiong Liu
ALL RIGHTS RESERVED

APPROVAL PAGE

**FORTIFYING ROBUSTNESS: UNVEILING THE INTRICACIES OF
TRAINING AND INFERENCE VULNERABILITIES IN
CENTRALIZED AND FEDERATED NEURAL NETWORKS**

Guanxiong Liu

Dr. Abdallah Khreishah, Dissertation Co-Advisor Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Issa Khalil, Dissertation Co-Advisor Date
Principal Scientist, Qatar Computing Research Institute

Dr. Ali Akansu, Committee Member Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Reza Curtmola, Committee Member Date
Professor of Computer Science, NJIT

Dr. NhatHai Phan, Committee Member Date
Associate Professor of Computer Science, NJIT

Dr. Bipin Rajendran, Committee Member Date
Reader in Engineering, King's College London

BIOGRAPHICAL SKETCH

Author: Guanxiong Liu
Degree: Doctor of Philosophy
Date: August 2023

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering, New Jersey Institute of Technology, Newark, NJ, 2023
- Master of Science in Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA, 2015
- Bachelor of Science in Measurement and Control Technology and Instrument, Southeast University, Nanjing, Jiangsu, China, 2013

Major: Electrical Engineering

Presentations and Publications:

Guanxiong Liu, Abdallah Khreishah, Fatima Sharadgah, Issa Khalil, “An Adaptive Black-Box Defense against Trojan Attacks (TrojDef),” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Nicholas Furth, Abdallah Khreishah, Guanxiong Liu, NhatHai Phan, Yasser Jararweh, “Un-Fair Trojan: Targeted Backdoor Attacks against Model Fairness,” *International Conference on Software Defined Systems (SDS)*, 2022.

Guanxiong Liu, Hang Shi, Abdallah Khreishah, Nirwan Ansari, Jo Young Lee, Chengjun Liu, Mustafa Mohammad Yousef, “Smart Traffic Monitoring System Using Computer Vision and Edge Computing,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Guanxiong Liu, Issa Khalil, Abdallah Khreishah, NhatHai Phan, “A Synergetic Attack against Neural Network Classifiers combining Backdoor and Adversarial Examples,” *IEEE International Conference on Big Data (Big Data)*, 2021.

- Fatima Sharadgah, Abdallah Khreishah, Mahmoud Al-Ayyoub, Yaser Jararweh, Guanxiong Liu, Issa Khalil, Muhannad Almutiry, Nasir Saeed, “An Adaptive Black-box Defense against Trojan Attacks on Text Data,” *International Conference on Social Network Analysis, Management and Security (SNAMS)*, 2021.
- Ahmad Sawalmeh, Noor Shamsiah Othman, Guanxiong Liu, Abdallah Khreishah, Ali Alenezi, Abdulaziz Alanazi, “Power-efficient wireless coverage using minimum number of UAVs,” *Sensors*, 2021.
- Guanxiong Liu, Issa Khalil, Abdallah Khreishah, “Using Intuition from Empirical Properties to Simplify Adversarial Training Defense,” *International Workshop on Dependable and Secure Machine Learning*, 2019.
- Guanxiong Liu, Issa Khalil, Abdallah Khreishah, “ZK-GanDef: A GAN based Zero Knowledge Adversarial Training Defense for Neural Networks,” *International Conference on Dependable Systems and Networks*, 2019.
- Guanxiong Liu, Issa Khalil, Abdallah Khreishah, “GanDef: A GAN based Adversarial Training Defense for Neural Network Classifier,” *International Conference on ICT Systems Security and Privacy Protection*, 2019.
- Abbas Kiani, Guanxiong Liu, Hang Shi, Abdallah Khreishah, Nirwan Ansari, Jo Young Lee, Chengjun Liu, “A Two-Tier Edge Computing Based Model for Advanced Traffic Detection,” *International Conference on Internet of Things: Systems, Management and Security*, 2018.
- Guanxiong Liu, Hazim Shakhatreh, Abdallah Khreishah, Xiwang Guo, Nirwan Ansari, “CityLines: Designing Hybrid Hub-and-Spoke Transit System with Urban Big Data Efficient Deployment of UAVs for Maximum Wireless Coverage Using Genetic Algorithm,” *IEEE 39th Sarnoff Symposium*, 2018.
- Yanhua Li, Guanxiong Liu, Zhi-Li Zhang, Jun Luo, Fan Zhang, “CityLines: Designing Hybrid Hub-and-Spoke Transit System with Urban Big Data,” *IEEE Transactions on Big Data*, 2018.
- Guanxiong Liu, Yanhua Li, Zhi-Li Zhang, Jun Luo, Fan Zhang, “Citylines: Hybrid hub-and-spoke urban transit system,” *International Conference on Advances in Geographic Information Systems*, 2017.
- Guanxiong Liu, Menghai Pan, Yanhua Li, Zhi-Li Zhang, Jun Luo, “Modeling urban trip demands in cloud-commuting system: A holistic approach,” *IEEE Conference on Computer Communications Workshops*, 2017.
- Luyao Niu, Yingyue Fan, Kaveh Pahlavan, Guanxiong Liu, Yishuang Geng, “On the accuracy of Wi-Fi localization using robot and human collected signatures,” *International Conference on Consumer Electronics*, 2016.

Guanxiong Liu, “Modeling and Performance Analysis of Hybrid Localization Using Inertial Sensor, RFID and Wi-Fi Signal,” *Master Thesis*, 2015.

Dan Liu, Yishuang Geng, Guanxiong Liu, Mingda Zhou, Kaveh Pahlavan, “WBANs-Spa: An energy efficient relay algorithm for wireless capsule endoscopy,” *IEEE 82nd Vehicular Technology Conference*, 2015.

Guanxiong Liu, Yishuang Geng and Kaveh Pahlavan, “Direction Estimation Error Model of Embedded Magnetometer in Indoor Navigation Environment,” *International Conference on Ubiquitous Intelligence and Computing*, 2015.

Guanxiong Liu, Yishuang Geng and Kaveh Pahlavan, “Effects of calibration RFID tags on performance of inertial navigation in indoor environment,” *International Conference on Computing, Networking and Communications*, 2015.

In a heartfelt tribute, I dedicate this work to my parents, Zhigang and Xiaoyan, whose unwavering support and the gift of life have enabled me to pursue my aspirations. To my beloved wife, Qinmei, whose presence has been a guiding light during my moments of uncertainty and darkness. And to my cherished daughter, Zoey, whose boundless hope has illuminated the path to my future.

Guanxiong Liu

ACKNOWLEDGMENT

First and foremost, I would like to express my sincere gratitude to my advisors, Dr. Abdallah Khreishah and Dr. Issa Khalil, for their exceptional mentorship, insightful feedback, and dedication to my academic and personal growth. Their guidance has been instrumental in shaping the direction of this research.

I extend my heartfelt thanks to the members of my dissertation committee, Dr. Ali Akansu, Dr. Reza Curtmola, Dr. NhatHai Phan, and Dr. Bipin Rajendran. Their constructive critiques and valuable suggestions have greatly enriched the quality of this work. A special acknowledgment is reserved for Dr. NhatHai Phan, whose enlightening discussions and meticulous guidance throughout our collaboration on my recent research endeavors have been exceptionally instrumental.

My appreciation also goes to Dr. Nirwan Ansari, Dr. Chengjun Liu, Dr. Joyoung Lee, Dr. Hang Shi, Dr. Abbas Kiani, and the US Ignite. The opportunity to collaborate and learn from each of you through the smart transportation project has been an invaluable stroke of luck. Additionally, I am deeply grateful for the financial support graciously provided by US Ignite, which has been instrumental in advancing my research endeavors.

Last but not least, my profound gratitude goes to my parents, my wife, and my cherished daughter for their unyielding love, unwavering encouragement, and profound understanding. Their steadfast faith in me has been the driving force behind my journey.

To all those who have contributed to my academic and personal growth, your support has been invaluable, and I am deeply appreciative.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Contributions	3
2 BACKGROUND	5
2.1 Generating Adversarial Examples	5
2.2 Adversarial Example Defensive Methods	8
2.3 Trojan Backdoor Attack	10
2.4 Defense against Trojan Backdoor Attack	12
2.5 Federated Learning and Personalization	13
2.6 Backdoor Attacks against FL	16
2.7 Defenses against Backdoor Attacks in FL	20
3 ZK-GANDEF: GAN BASED ZERO-KNOWLEDGE ADVERSARIAL TRAINING DEFENSE	21
3.1 Methodology	21
3.1.1 Zero-knowledge adversarial training	21
3.1.2 Design of ZK-GanDef	24
3.1.3 ZK-GanDef training algorithm	26
3.2 Evaluation Settings	27
3.2.1 Datasets	29
3.2.2 Preprocessing module	29
3.2.3 Attack module	30
3.2.4 Defense module	30
3.2.5 Evaluation metrics	31
3.3 Evaluation Results	33
3.3.1 Test accuracy on different examples	33

TABLE OF CONTENTS
(Continued)

Chapter	Page
3.3.2 On original examples	36
3.3.3 On single-step adversarial examples	37
3.3.4 On iterative adversarial examples	39
3.3.5 Generalizability	40
3.3.6 Training time	41
3.3.7 Convergence issue	42
3.4 Conclusion	43
4 SEMI-ITERATIVE ADVERSARIAL TRAINING	45
4.1 Analysis of Iter-Def Methods	45
4.1.1 Limit of small per-step perturbation	46
4.1.2 Training with intermediate examples	48
4.1.3 Summary	49
4.2 Single-Step Epoch-Iterative Method	49
4.2.1 Motivation and design	49
4.2.2 Applying over-perturbation	51
4.2.3 Searching space for adversarial examples	54
4.2.4 Analyzing feature space encoding	55
4.3 Evaluation	57
4.3.1 Evaluation setting	57
4.3.2 Test accuracy	60
4.3.3 Analyzing the behavior of Free-Def	61
4.3.4 Analyzing the weakness of ATDA method	63
4.3.5 Training time	65
4.3.6 Scalability test	65
4.4 Conclusion	67

TABLE OF CONTENTS
(Continued)

Chapter	Page
5 TROJDEF: AN ADAPTIVE BLACK-BOX DEFENSE AGAINST TROJAN ATTACKS	69
5.1 TrojDef Description and Analysis	69
5.1.1 Analysis of predictions	70
5.1.2 TrojDef description	76
5.1.3 TrojDef algorithms	77
5.1.4 TrojDef implementation	78
5.2 Experimental Settings	83
5.2.1 Datasets and classifiers	83
5.2.2 Experiments and metrics	83
5.3 Experimental Results	85
5.3.1 Evaluation on STRIP-model	85
5.3.2 Evaluation on TrojDef -model	88
5.3.3 Evaluation on 3rd-party-model	90
5.3.4 Using different FRR values	92
5.3.5 Using Laplacian perturbation	94
5.3.6 Defending blue channel trigger	95
5.3.7 Compared with white-box defense	95
5.4 Conclusion	96
6 ADVTROJAN: A SYNERGETIC ATTACK AGAINST NEURAL NETWORK CLASSIFIERS COMBINING BACKDOOR AND ADVERSARIAL EXAMPLES	101
6.1 Threat Model	101
6.2 AdvTrojan	104
6.3 Analysis	107
6.3.1 Mathematical analysis of AdvTrojans	107
6.3.2 Empirical analysis	110

TABLE OF CONTENTS
(Continued)

Chapter	Page
6.4 Experimental Settings	113
6.5 Experimental Results	114
6.6 Conclusion	129
7 COLLAPOIS: REASSESSING BACKDOOR ATTACKS IN FEDERATED LEARNING	131
7.1 Collaborative Poisoning Attacks	132
7.1.1 Threat model	132
7.1.2 Collaborative poisoning (COLLAPois)	134
7.1.3 Smaller and bounded numbers of compromised clients	137
7.1.4 Attack stealthiness	145
7.2 Experimental Results	149
7.3 Conclusion	157
8 EXTENSIONS AND FUTURE DIRECTIONS	158
8.1 Training and Inference Time Vulnerabilities	159
8.2 Vulnerabilities of Federated Trained NNs	161
8.3 Other Vulnerabilities	165
REFERENCES	168

LIST OF TABLES

Table	Page
2.1 Summary of Notations used in Background Chapter	7
2.2 Comparing state-of-the-art backdoor attacks in federated learning.	15
2.3 Comparing state-of-the-art defenses against backdoor attacks.	19
3.1 Additional Notations used in ZK-GanDef Chapter	22
3.2 Discriminator Structure	32
3.3 Test Accuracy on Different Examples	35
3.4 Test Accuracy on Deepfool and CW Examples	41
4.1 Evaluation Parameter Setting	57
4.2 Test Accuracy	59
4.3 Test Accuracy under Different Perturbation Limits	60
4.4 Test Accuracy with Different Learning Rate α	63
4.5 Total Training Time in Seconds	66
4.6 Test Accuracy on ImageNet	67
5.1 TrojDef -model Architecture	82
5.2 Results of the Conventional Experiments	86
5.3 Performance under Different Training Hyper-parameters	87
5.4 Evaluation Results of the Defenses on TrojDef -model	89
5.5 Evaluation Results of the Defenses on the 3rd-party-model	90
5.6 Evaluation Results on TrojDef -model under Different FRR Values	93
5.7 Evaluation Results on the 3rd-party-model under Different FRR Values	98
5.8 Evaluation Results of TrojDef on the 3rd-party-model and Laplacian perturbation	99
5.9 Performance of Defending the Blue Channel Trigger	99
5.10 Evaluation Results of the Mutation and TrojDef model	100
6.1 Hyper-parameter of Adversarial Perturbations.	114

LIST OF TABLES
(Continued)

Table	Page
6.2 Identified Infected Classes and False Negative Rate (FNR) of Neural Cleanse with ATIM	115
6.3 False Negative Rate (FNR) of STRIP under 2% False Positive Rates (FPR) for Each Dataset	116
6.4 Test Accuracy of ATIM on Different Examples for Each Dataset	117
6.5 False Negative Rate (FNR) of E-STRIP under 2% False Positive Rates (FPR) for Each Dataset	117
6.6 Evaluation Results of Targeted Attack	122

LIST OF FIGURES

Figure	Page
2.1 Examples of input images with Trojan trigger [21].	11
3.1 Training procedure of Clean Logit Pairing.	23
3.2 Training procedure of Clean Logit Squeezing.	24
3.3 Training procedure of ZK-GanDef.	25
3.4 Evaluation framework.	27
3.5 Test accuracy on different examples	34
3.6 Training time and training loss	38
4.1 Empirical results on per-step perturbation	46
4.2 Empirical results on intermediate examples.	46
4.3 Flow of traditional adversarial training.	50
4.4 Flow of single-step epoch-iterative method.	50
4.5 Madry-Def under different n_1 values.	53
4.6 Searching space of different adversarial examples	53
4.7 Feature space encoding of Vanilla, ATDA, Free-Def, BIM(30,30)-Def, and SIM(10,20)-Def classifiers.	55
4.8 Explore the effect of n_1 and n_2 on performance.	58
4.9 Samples of benign and adversarial examples.	62
4.10 Evaluation of the domain adaptation loss.	64
5.1 The effect of applying the non-linear transform on the Prediction Confidence Bound.	74
5.2 Heatmap of prediction on different examples.	74
5.3 High-level view of the proposed defense.	76
5.4 Perturbation with random location and size.	79
5.5 Trojan triggers used in the experiments.	84
5.6 By channel view of blue channel trigger.	95
6.1 Behaviors of classifiers	103

LIST OF FIGURES
(Continued)

Figure	Page
6.2 Overview of the “vulnerability distillation”	106
6.3 Empirical analysis results	110
6.4 Test accuracy of different combinations of models and examples	113
6.5 Anomaly index when applying adaptive Neural Cleanse with the ATIM.	118
6.6 Test accuracy of ATIM on Madry-Exps generated with different numbers of iterations for each dataset.	123
6.7 Test accuracy of ATIM on Madry-Exps generated with different pertur- bation sizes for each dataset	124
6.8 Test accuracy of ATIM on AdvTrojan examples generated with different perturbation methods for each dataset	124
6.9 Attacking global model in federated learning with ATIM (CIFAR-10). . .	128
7.1 CollaPois framework.	133
7.2 Angles among the gradients from benign and compromised clients as a function of α using the FEMNIST dataset [10].	138
7.3 Approximation error for the lower bound of $ \mathcal{C} $	142
7.4 3D plot of $\frac{ \mathcal{C} }{ N }$ as a function of μ_α and σ	144
7.5 Estimation error of COLLAPOIS	147
7.6 Attack stealthiness	148
7.7 FedAvg under different backdoor attacks for the CIFAR-10 dataset. . . .	150
7.8 FedDC under different backdoor attacks for the CIFAR-10 dataset. . . .	150
7.9 FedAvg under different backdoor attacks for the FEMNIST dataset. . . .	152
7.10 FedDC under different backdoor attacks for the FEMNIST dataset. . . .	152
7.11 Evaluating COLLAPOIS against various robust training algorithms across different datasets and data divergence.	154
7.12 Evaluating COLLAPOIS against various robust training algorithms across different datasets with $\alpha = 100$	156

CHAPTER 1

INTRODUCTION

1.1 Motivation

The rapid adoption of neural networks (NNs) and machine learning algorithms in a wide range of real-world applications has revolutionized various domains, including autonomous vehicles, healthcare diagnostics, and financial transactions [49]. These advanced models have demonstrated remarkable capabilities in tasks such as image recognition, natural language processing, and data analytics, enabling unprecedented levels of automation, efficiency, and decision-making accuracy [25]. As NNs continue to permeate critical systems that directly impact human lives and societal functions, it becomes increasingly crucial to ensure their reliability and security. However, the very power and complexity that make NNs so effective also render them susceptible to vulnerabilities that can be exploited by malicious actors. Hence, understanding and mitigating these vulnerabilities is paramount to foster trust in NNs and leverage their full potential in practical applications.

To systematically approach various vulnerabilities, one fundamental aspect to consider is the distinction between training time vulnerabilities and inference time vulnerabilities. Training time vulnerabilities refer to weaknesses that can be exploited during the model training process. Adversaries may attempt to manipulate the training data or the training algorithm itself to compromise the integrity of the model. These vulnerabilities can have far-reaching consequences, as they can lead to the training of models that are incorrect, biased, or discriminatory [27]. Moreover, the presence of training time vulnerabilities can have a cascading effect on the reliability of applications through sharing and integrating the pre-trained models. Until the

time of drafting this dissertation, the state-of-the-art training time vulnerability is the Trojan backdoor attack.

On the other hand, inference time vulnerabilities manifest when the trained model encounters inputs that are intentionally designed to deceive or mislead the model [79]. Adversarial examples, which are carefully crafted inputs with imperceptible perturbations, can cause NNs to make erroneous predictions with high confidence [34]. Therefore, the inference time vulnerabilities are called adversarial attacks. These vulnerabilities raise concerns about the reliability and trustworthiness of NNs in critical systems, where erroneous predictions can have severe consequences.

Beyond the training time vs inference time, another aspect to distinguish vulnerabilities is the training paradigm. With the emergence of data privacy regulations and the need to protect sensitive user data, Federated Learning (FL) has gained significant attention in recent years [67]. FL enables clients to collaboratively train NNs without sharing their raw data, thereby addressing privacy concerns and data ownership issues. However, FL introduces its own unique challenges and vulnerabilities. Due to the limited data sharing between clients, adversaries are able to compromise part of the clients in FL without being notified. Then, by exploiting the collaborative nature of FL, they can launch attacks that compromise the integrity of the learned model or the privacy of the participating clients [3]. These attacks, such as backdoor poisoning and model extraction, pose significant threats to the security and effectiveness of FL.

Without any doubt, addressing these aforementioned challenges and vulnerabilities is crucial before widely adopting NNs in critical systems. Throughout this dissertation, we advance state-of-art research by providing valuable insights, novel defense strategies, and practical countermeasures to safeguard NNs and promote their safe deployment in real-world applications. More specifically, this dissertation focuses on answering the following questions:

- **Q1 How to defend against adversarial examples in a more computationally efficient way?:** Adversarial retraining, as a highly popular defense approach against adversarial examples, has garnered widespread adoption due to its promising empirical performance. However, despite its commendable effectiveness in enhancing robustness, adversarial retraining exhibits notable shortcomings in terms of computational efficiency. This deficiency stems from the requirement of generating adversarial examples during training, which results in a considerable increase in computational overhead. Consequently, the substantial computational demands imposed by the preparation of adversarial examples render adversarial retraining less practical for applications characterized by limited computing resources.
- **Q2 How to improve the black-box defense against Trojan backdoor attack?** Currently, the majority of proposed defenses against Trojan attacks predominantly assume a white-box setup, wherein the defender possesses access to the internal state of the NN or can execute back-propagation through it. However, in the context of a more practical black-box scenario, the defender is confined to solely conducting the forward pass of the NN. Within this challenging black-box framework, only a limited repertoire of Trojan backdoor defenses have been put forth, and their corresponding performance is evidently diminished when compared to their white-box counterparts.
- **Q3 How to explore the risk of combining training and inference time vulnerabilities together?** In the realm of NN vulnerabilities, the prevalent approach in existing research endeavors has been to examine training time and inference time vulnerabilities in isolation. Nonetheless, in practice, adversaries possess ample incentive to exploit these vulnerabilities in conjunction, thereby enhancing the stealthiness or severity of their attacks. Consequently, an exclusive focus on either training time or inference time vulnerabilities may inadvertently overlook potentially unknown risks.
- **Q4 How to explore the vulnerabilities of FL with highly diversified clients?** Although the vulnerabilities of NN models trained using FL have been extensively investigated, the existing body of work has paid scant attention to the diversification among clients, which constitutes a crucial characteristic of FL. The omission of exploring the impact of diversified clients leaves a significant knowledge gap in our understanding of the vulnerabilities inherent in Federated trained NNs.

1.2 Contributions

This dissertation comprehensively addresses the questions outlined above by examining the training time vulnerabilities, inference time vulnerabilities, and vulnerabilities

specific to Federated trained NNs. The works presented in this dissertation are organized as follows, each contributing valuable insights in their respective areas:

- Chapter 3 introduces **ZK-GanDef**, a novel defense strategy against adversarial attacks. It leverages generative adversarial networks (GANs) in a zero-knowledge approach, eliminating the need for adversarial examples during training. Through comprehensive experiments, ZK-GanDef achieves impressive improvements of up to 49.17% in test accuracy compared to existing zero-knowledge approaches. Additionally, it demonstrates close performance to cutting-edge defenses, with less than 8.46% degradation in accuracy, while significantly reducing training time.
- Chapter 4 proposes **SIM-Adv**, a further improved defense strategy focused on computational efficiency. SIM-Adv utilizes low-cost adversarial examples for retraining and adapts to newly introduced adversarial examples. Extensive evaluations reveal that SIM-Adv outperforms state-of-the-art single-step adversarial training defenses, enhancing test accuracy by up to 35.67% while reducing training time by 19.14%. Compared to iterative adversarial training methods, SIM-Adv achieves substantial time savings of up to 76.03% with a minimal trade-off in test accuracy (less than 3.78%).
- Chapter 5 introduces **TrojDef**, a practical black-box defense mechanism against Trojan backdoor attacks. By monitoring changes in prediction confidence through repeated perturbing input with random noise, **TrojDef** effectively identifies and filters out Trojan inputs. Extensive empirical evaluations demonstrate that **TrojDef** surpasses state-of-the-art defenses and exhibits remarkable stability across different settings.
- Chapter 6 addresses the combined exploration of training and inference time vulnerabilities through **AdvTrojan**, a stealthy attack that exploits adversarial perturbation and model poisoning vulnerabilities. AdvTrojan showcases a high success rate in evading existing defenses, approaching 100% in most experimental scenarios, even when targeting federated learning frameworks and high-resolution images.
- Chapter 7 shift the focus to vulnerabilities in NNs trained using Federated Learning (FL). It introduces **CollaPois**, a collaborative backdoor poisoning attack that distributes a Trojan-infected model to compromised clients in an FL setting. COLLAPOIS effectively amplifies the impact of the Trojan while evading detection by state-of-the-art robust FL algorithms. The attack operates stealthily and minimizes the number of compromised clients, making it distinct from existing attacks.

CHAPTER 2

BACKGROUND

This chapter serves to provide a comprehensive overview of the background materials pertinent to the subsequent chapters. Specifically, the presented background materials encompass three key areas: (1) adversarial examples and corresponding defenses for inference time vulnerabilities, (2) Trojan backdoor attacks and corresponding defenses for training time vulnerabilities, and (3) Federated learning and the corresponding vulnerabilities. Moreover, the chapter includes a curated selection of relevant references that offer additional insights and information on each topic.

2.1 Generating Adversarial Examples

The methods for generating adversarial examples against NN classifiers can be categorized according to several different aspects. In one aspect, these methods could be distinguished by the adversary’s knowledge of the target NN classifier. In White-box methods, the adversary is assumed to have full knowledge about the target NN classifier (structure, parameters and inner status), and hence the generated examples are called white-box adversarial examples. On the other hand, black-box methods assume that the adversary has no access to the inner information of the target NN classifier, and hence, the generated examples are called black-box adversarial examples. On another aspect, adversarial example generation methods could be categorized into single-step (**Single-Exps**) or iterative (**Iter-Exps**) methods according to the process of generating the examples. Single-step methods only run gradient descent (ascent) algorithm once when solving the proposed optimization problem, while iterative methods repeat the computation several times until hitting predefined convergence thresholds.

Based on previous works, an adversarial example generator is generally formulated as an optimization problem which searches a small neighboring area of the original image (usually defined by l_1 , l_2 or l_∞ norm) for the existence of adversarial examples. If we denote an original image by \bar{x} and the example with perturbation δ by $\hat{x} = \mathcal{F}(\bar{x} + \delta)$, then the process of searching adversarial examples can be formulated as follows [26]:

$$\begin{aligned} & \underset{\delta}{\text{minimize}} && \|\hat{x} - \bar{x}\| \\ & \text{subject to} && \mathcal{C}(\hat{x}) = z^o \\ & && \mathcal{F}(\bar{x} + \delta) \in \mathbb{R}_{[-1,1]}^m \end{aligned}$$

The function \mathcal{C} represents the classifier and outputs the pre-softmax logits based on input image. The function \mathcal{F} projects the pixel value of any input image back to $\mathbb{R}_{[-1,1]}$ and ensures that the generated adversarial example is still a valid image. A perturbation is considered strong enough to fool the classifier if and only if $\mathcal{C}(\hat{x}) = z^o$, where z^o is the objective pre-softmax logits designed by adversary. The global optimum of this problem corresponds to the strongest adversarial example for a given image. However, modern classifiers are highly non-linear, which makes it hard to solve the optimization problem in its original form, and hence each generator has its own approximation to make the optimization problem solvable.

Table 2.1 summarizes all the notations that we use throughout this chapter. In the following, we describe the design approaches of several popular adversarial example generators that we consider in this work.

Fast Gradient Sign Method (FGSM) is introduced by Goodfellow et al. in [26] as a single-step white-box adversarial example generator against NN image classifiers. This method tries to maximize the loss function value, \mathcal{L} , of NN classifier, \mathcal{C} , to find adversarial examples. The calculation of loss is usually defined as the difference between ground truth, t , and the softmax transformation, $f(z_i) = \frac{e^{z_i}}{\sum_{z_j} e^{z_j}}$,

Table 2.1 Summary of Notations used in Background Chapter

\mathcal{L}	loss function of NN classifier
\mathcal{F}	regulation function for pixel value of generated example
z^o	objective pre-softmax logits designed by adversary
l_1, l_2, l_∞	the 1st order, 2nd order and infinity order norm
$\bar{x}, \bar{X}; \hat{x}, \hat{X}; x, X$	original example; perturbed example; their union
$\bar{t}, \bar{T}; \hat{t}, \hat{T}; t, T$	ground truth of $\bar{x}, \bar{X}; \hat{x}, \hat{X}; x, X$
$\bar{z}, \bar{Z}; \hat{z}, \hat{Z}; z, Z$	pre-softmax logits of $\bar{x}, \bar{X}; \hat{x}, \hat{X}; x, X$
δ	perturbation
\mathcal{C}	NN based classifier

of pre-softmax logits.

$$\begin{aligned} & \underset{\delta}{\text{maximize}} && \mathcal{L}(\hat{z} = \mathcal{C}(\hat{x}), t) \\ & \text{subject to} && \mathcal{F}(\bar{x} + \delta) \in \mathbb{R}_{[-1,1]}^m \end{aligned}$$

As a single-step generator, only one iteration of gradient ascent is executed to find adversarial examples. It simply generates examples with perturbation, \hat{x} , from original images, \bar{x} , by adding small perturbation, δ , which changes each pixel value along the gradient direction of the loss function. Although running one iteration of gradient ascent algorithm can not guarantee finding a solution which is close enough to optimal one, empirical results show that adversarial examples from this generator can mislead Vanilla NN classifiers. Intuitively, FGSM runs faster than iterative generators at the cost of weaker adversarial examples. That is, the success rate of attack using the generated examples is relatively low due to the linear approximation of the loss function landscape.

Basic Iterative Method (BIM) is introduced by Kurakin et al. in [46] as an iterative white-box adversarial example generator against NN image classifiers.

BIM utilizes the same mathematical model as FGSM but runs the gradient ascent algorithm iteratively. In each iteration, BIM applies small perturbation and maps the perturbed image through the function \mathcal{F} . As a result, BIM approximates the loss function landscape by linear spline interpolation. Therefore, it generates stronger examples and achieves higher attack success rate than FGSM within the same neighboring area.

Projected Gradient Descent (PGD) is another iterative white-box adversarial example generator recently introduced by Madry et al. in [66]. Similar to BIM, PGD solves the same optimization problem iteratively with projected gradient descent algorithm. However, PGD randomly selects initial point within a limited area of the original image and repeats this several times to search adversarial example. Since the loss landscape has a surprisingly tractable structure [66], PGD is shown experimentally to solve the optimization problem efficiently and the generated examples are stronger than those of BIM.

2.2 Adversarial Example Defensive Methods

Here, we categorize the design of defensive methods against adversarial examples into three major classes. The first is based on applying data augmentation and regularization during the training. The second class aims at adding protective shell on the target classifier, while the last class focuses on utilizing some adversarial examples to retrain the target classifier. In the following, we introduce representative examples from each of the above three approaches:

Augmentation and Regularization usually utilize synthetic data or regulate hidden states during training to enhance the test accuracy on adversarial examples. One of the early ideas in this direction is the defensive distillation. In the context of adversarial example defense, distillation is done by using the prediction score from original NN, which is usually called the teacher, as ground truth to train a smaller NN

with different structure, usually called the student [80][78]. It has been shown that the calculated gradients from student model become very small or even reach zero and hence, can not be utilized by adversarial example generators [78]. Examples of recent approaches under this category of defenses include Fortified Network [48] and Manifold Mixup [101]. Fortified Network utilizes denoising autoencoder to regularize the hidden states. With this regularization, trained NN classifiers learn to mitigate the difference in hidden states between original and adversarial examples. Manifold Mixup also focus on hidden states but follows a different way. During training, Manifold Mixup uses interpolations of hidden states and logits instead of original training data to achieve regularization. However, this set of defenses is shown to be not very reliable as they are vulnerable to certain adversarial examples. For example, defensive distillation is vulnerable to Carlini attack [11] and Manifold Mixup can only defend against single step attacks.

Protective Shell is a set of defensive methods designed to reject or reform adversarial examples. Meng et al. introduced an approach called MagNet [70] which falls under this category. MagNet has two types of functional components; the detector and the reformer. Adversarial examples are either rejected by the detector or reformed by the reformer to clean up adversarial perturbations. Other recent approaches like [58], [116] and [85] also fall under this category and they are differentiated by the way they implement the protective shell. The defense proposed by Bin et al. [58] carefully inject adaptive noise to input images to break adversarial perturbations without significantly degrading classification accuracy. In the work from Pinlong et al. [116], a key based cryptography method is utilized to differentiate adversarial examples from original ones. And, in the work from Pouya et al. [85], a generator is utilized to generate images that are similar to the inputs. By replacing the inputs with generated images, the approach shows good resistance to adversarial examples. The main limitation of the approaches under this category is

the assumption that the shell is black-box to adversary, which turns to be inaccurate. For example, in [2], Anish et al. presented different ways to break this assumption.

Adversarial Training is based on the idea that adversarial examples can be considered as blind spots of the original training data [110]. By retraining with samples of adversarial examples, the classifier learns perturbation patterns from adversarial examples and generalizes its prediction to account for such perturbations. In one of the beginning works from Goodfellow et al. [26], adversarial examples generated by FGSM are used for adversarial training of a NN classifier. The results show that the retrained classifier can correctly classify adversarial examples of this single step attack (FGSM). Later works in [66] and [98] enhance the adversarial training process so that the trained models can defend not only single step attacks but also iterative attacks like BIM and PGD. Based on adversarial examples (Single-Exps or Iter-Exps), these defenses can be categorized into **Single-Adv** and **Iter-Adv**. Be more specific, we also name defenses with adversarial examples it uses such as FGSM-Adv method. A more recent research work in adversarial training [39] introduces two zero knowledge adversarial training defenses. The defenses use Gaussian random noise for perturbations and include a penalty term based on pre-softmax logits, z . However, the design of penalty term is simple and not flexible enough to handle complex patterns in z .

2.3 Trojan Backdoor Attack

In the literatures [27, 63, 103, 21], Trojan attacks against NN classifier can be described as follows. Through accessing and poisoning the training process, adversary injects Trojan back-door in the trained classifier. During the inference time, the NN classifier performs unexpected behavior if and only if a predefined Trojan trigger is added to the input [27, 63]. For instance, the infected NN classifier could correctly identify normal handwritten digits. However, any input with a Trojan trigger (for

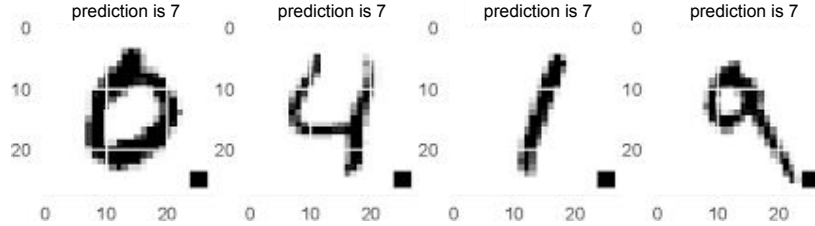


Figure 2.1 Examples of input images with Trojan trigger [21].

example, the small black square at the bottom right corner of each image in Figure 2.1) is classified as digit seven when it is fed into the infected classifier.

Trojan attacks are different from adversarial example attack, since: (1) A Trojan attack is prepared by poisoning the training of NN classifier to inject back-door, and is launched at the inference time. While, the adversarial attack is both prepared and launched in the inference phase; (2) Most of adversarial attacks are image specific. It means that different inputs require different adversarial perturbations. Instead, the Trojan trigger introduced by existing works are image agnostic. In other words, any input with Trojan trigger can activate the unexpected behavior of the infected NN classifier; and (3) The adversarial attack requires its adversarial perturbation to be insignificant through limiting total perturbation budget. The Trojan attack, in general, does not have such limitation.

The process of injecting Trojan back-door can be formulated, as follows.

$$\theta^* = \arg \min_{\theta} [L(\hat{x}, y, \theta) + L(\text{clip}_D[\hat{x} + t], y_t, \theta)] \quad (2.1)$$

where t is the Trojan trigger predefined by the adversary. In [27], t is a collection of pixels with arbitrary values and shapes. In Equation (2.1), the poisoned inputs with Trojan trigger are used during the training of NN classifier. The targeted labels for these poisoned training inputs are y_t , representing the unexpected behavior selected by the adversary. A more recent work in [63] follows the similar injection process while not requiring access to the benign training data \hat{x} .

2.4 Defense against Trojan Backdoor Attack

Since the Trojan attack is more recently introduced than adversarial attack, there is no dominant solution that performs consistently better than others.

Based on our review, we categorize one class of defense methods as **intrusive defense methods**. Some of these defense methods require direct access and modification to the NN classifier. For example, the Fine-Pruning method in [61] assumes that the NN classifier is known to be infected. It tries to remove the back-door by pruning redundant neurons that have minimal impact on the prediction of benign inputs. Another defense method proposed in [62] scans the entire NN classifier by manually stimulating its neurons' activation each at a time to identify if it is infected.

In contrast to the aforementioned defense methods, there are also **model-agnostic defense methods** that leave the NN classifier unchanged. One of these defense methods is called the **Neural Cleanse** [103]. This defense method firstly reverse engineers potential Trojan triggers for each class. Then, the size of these potential Trojan triggers are fed to an anomaly detection algorithm to check if any of these triggers is abnormally smaller than others. If the answer is yes, this specific class in NN classifier is infected. A more recent method in [21], **STRIP**, chooses a different approach. For each input, this defense method directly superimposes it on several prepared benign images and then generates corresponding predictions by feeding them to the NN classifier. Since the misclassification caused by existing Trojan attack is image agnostic, the more stable the predictions are, the more likely that the input contains Trojan trigger.

Compared with the intrusive approaches, the model-agnostic approaches have the following advantages.

- **Practicability:** The intrusive approaches require direct access to the inner states of the NN classifier. Some of them even modify the NN classifier. In real-world scenarios, the integrity and privacy of the NN classifier are important from its provider's perspective. For example, the owner will not allow the user

to inspect or modify the classifier especially in commercial cases. Therefore, the intrusive defense methods are not practical solutions.

- **Effectiveness:** The NN classifier’s architecture is specially designed by its owner to achieve a certain level of performance. The modification of the NN classifier may cause a degeneration on its performance. For example, the classifier’s robustness against adversarial perturbation could be broken if the intrusive defense method does not take it into account during design.

Given these considerations, we employ the model-agnostic methods as defensive approaches against Trojan backdoor attacks throughout this dissertation.

2.5 Federated Learning and Personalization

FL Protocol. At round t , the server sends the latest model weights θ^t to a randomly sampled subset of clients S^t with a probability q . Upon receiving θ^t , the sampled client i in S^t uses θ^t to train their local model for some number of iterations, e.g., via stochastic gradient descent (SGD), and results in model weights θ_i^t . The client i computes its local gradients $\Delta\theta_i^t = \theta_i^t - \theta^t$ by using its local dataset D_i , and sends it back to the server. After receiving all the local gradients from all the sampled clients in S^t , the server updates the model weights by aggregating all the local gradients by using a function $\mathcal{G} : R^{|S^t| \times m} \rightarrow R^m$ where m is the size of $\Delta\theta_i^t$. The aggregated gradient will be used to update θ^t into the new model weights θ^{t+1} . A typical aggregation function is Federated Averaging (FedAvg) applied in many FL algorithms [38], as follows:

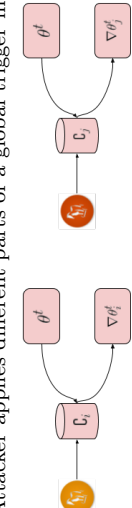
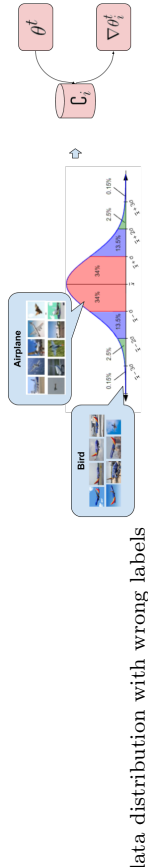
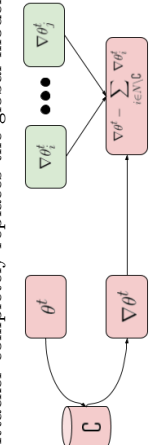
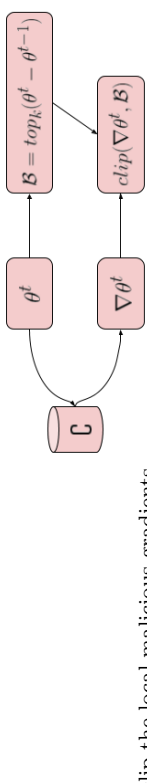
$$\theta^{t+1} = \theta^t + \lambda \left(\sum_{i \in S^t} n_i \times \Delta\theta_i^t \right) / \sum_{i \in S^t} n_i, \quad (2.2)$$

where λ is the server’s learning rate, and n_i is the number of training samples of the client i . When the number of training samples n_i is hidden from the server, one can use an unweighted aggregation function: $\theta^{t+1} = \theta^t + \lambda \sum_{i \in S^t} \Delta\theta_i^t / |S^t|$.

Personalized FL (pFed). FL methods often encounter a significant variation in data distributions across clients, which results in a substantial difference in the

model’s effectiveness [96, 12]. Therefore, pFed techniques have been proposed to overcome this problem by achieving personalized performance that can adapt to the varying data [28, 88, 19]. pFed approaches can be broadly categorized into four research lines: **(1) Regularization-based** approaches modify local training through regularization or penalization to address data distribution drifting, which can cause a divergence between the weights of local and global models [82, 20, 56, 41], **(2) Knowledge Distillation**, in which the server ensembles clients’ knowledge by a generator or a consensus distributed across the network. Clients then utilize the acquired knowledge as an inductive bias or learn its local model from public and private datasets [54, 117, 88], **(3) Clustering-based** frameworks assign clients to clusters and aggregate local models within each cluster [23, 86], and **(4) Meta learning** leverages the concept of meta-training and meta-testing. In meta-training, a sensitive initial model is learned, which can quickly adapt to various tasks, typically using techniques like Model Agnostic Meta-Learning (MAML). This initial model is then mapped to the global model, and in the meta-testing step, it is further adapted to specific tasks on the client’s side.

Table 2.2 Comparing state-of-the-art backdoor attacks in federated learning.

Approach	Method	Attack Knowledge	Attack Capability	Description
Data Poisoning	DBA [109]	Black-box	Partial	Attacker applies different parts of a global trigger in different compromised clients 
	Label Flipping Attack [97]	Black-box	Partial	Attacker flips the labels of examples in the compromised clients that belong to source class to target class
	Edge-case [104]	White-box, Black-box	Partial	Attacker samples an auxiliary dataset which contains examples from the tail of the data distribution with wrong labels 
Model Poisoning	Model Replacement [3]	White-box	Full	Attacker completely replaces the global model through uploading scaled malicious gradients 
	Model Update Poisoning Attack [94]	White-box	Partial	Attacker prepares a local model that misclassifies examples to target class,
Model Poisoning	Neurotoxin [115]	White-box	Full	Attacker computes the top-k % weights updates based on received global model to clip the local malicious gradients 

2.6 Backdoor Attacks against FL

This section reviews the threat model of poisoning attacks in FL. We first introduce two terms, *attack knowledge* and *attack capability*, that are used to differentiate the attacks throughout this work. More details are provided in Table 2.2.

Attack Knowledge. We refer to attack knowledge as the extent to which an attacker is aware of other clients’ information. When the attacker possesses knowledge about the updates sent by other clients to the server, it is referred to as *white-box* knowledge. Conversely, *black-box* knowledge implies that the attacker cannot gather information from other clients. Consequently, black-box knowledge is considered more practical compared to white-box knowledge.

Attack Capability. We distinguish the attack capability into two categories: *partial capability* and *full capability*. In the case of partial capability, the attacker can only introduce poisoned data into the training dataset of a specific subset of clients. On the other hand, an attacker with full capability controls the entire subset of clients and can manipulate their training process at will. These manipulated clients can be referred to as a set of *compromised clients* \mathcal{C} . The attacker with full capability holds a stronger position than the one with partial capability, as they can control the compromised clients to carry out a coordinated poisoning attack. It is worth noting that both attackers are considered practical in the context of FL applications [89].

Backdoor Attacks. In this work, we focus on backdoor attacks where the attacker’s objective is to induce misclassification in the model specifically for a selected set of inputs while keeping the model’s accuracy intact for legitimate data samples. The review of current backdoor attacks and their categorization based on attack knowledge and capability is presented in Table 2.2. Trojans have emerged as a prominent method for conducting backdoor attacks, as highlighted in previous studies [27, 63]. Trojans involve carefully embedding a specific pattern, such as a

brand logo or additional pixels, into legitimate data samples to induce the desired misclassification. Recently, an image warping-based Trojan has been developed, which subtly distorts an image using geometric transformations [73]. This technique aims to make the modification imperceptible to human observers. Importantly, warping techniques enable Trojans to evade commonly used detection methods like Neural Cleanse [103], Fine-Pruning [61], and STRIP [21]. The attacker activates the backdoor during the inference phase by applying this Trojan trigger to legitimate data samples.

In FL, the training data is usually scattered across clients while the server only observes local gradients. Therefore, backdoor attacks are typically carried out by compromised clients controlled by an attacker to construct malicious local gradients before sending them to the server. The attacker can apply data poisoning (DPOIS) and model replacement (MREPL) approaches to create malicious local gradients.

In DPOIS [93, 53], compromised clients train their local models on datasets poisoned with Trojane examples. This deliberate contamination allows them to generate malicious local gradients. When these gradients are aggregated at the server, the resulting model exhibits the backdoor effect. In the MREPL technique [3], the adversary can construct malicious local gradients in a way that the aggregated model at the server closely approximates or entirely replaces a predefined Trojane model. Model replacement, to a large extent, poses a significant threat as it can be achieved even after just a single training round [17]. However, it is crucial to acknowledge that MREPL employs an inefficient attack strategy when targeting FL and pFed methods. This inefficiency arises from the black-box nature of client interactions in FL and pFed approaches. To reliably approximate benign clients' updates, MREPL must patiently wait for the global model to converge, rendering the attack ineffective until that point [3]. Moreover, this strategy further restricts MREPL from effectively attacking FL or

pFed systems with highly divergent clients, which our empirical results identify as a weak point in the context of our investigation.

Table 2.3 Comparing state-of-the-art defenses against backdoor attacks.

Approach	Method	Description
Robust Aggregation	Krum / Multi-Krum [7]	Score each update based on its closeness to its neighbors, Take the average of top N updates as aggregated update
	Median GD [113]	Use the element-wise median as aggregated update
	Trim Mean GD [113]	Remove the top and bottom β percentage of collected updates, Use the element-wise mean as aggregated update
	SignSGD [6]	Adjust the server's learning rate based on the agreement of client updates
	Robust Learning Rate (RLR) [74]	Count the updates in the same direction of aggregated update for each element, Flip the update in elements where the count is smaller than the threshold
	Ditto [55]	Fine-tune the potentially corrupt global model on each client's private data
Model Smoothness	Norm bound [94]	Clip the gradients based on magnitude, Add Gaussian noise to the gradients
	CRFL [107]	Clip model parameters to control model smoothness, Generate sample robustness certification with limited amplitude
	FLARE [105]	Estimate a trust score for each model update based on the differences between all pair of updates, Aggregate model updates weighted by the trust scores
	DP-optimizer [31]	Clip the gradients collected from clients, Add Gaussian noise to the clipped gradients
User-level DP [68]	Add sufficient Gaussian noise to model updates for providing user-level DP	

2.7 Defenses against Backdoor Attacks in FL

Current defense mechanisms against backdoor poisoning attacks in FL can be classified into two categories: (1) Detection of Trojans during inference; and (2) Resilient gradient aggregation to reduce the impact of malicious local gradients during the federated training process. In this research, we employ the advanced warping-based Trojan technique that bypasses commonly used inference-time Trojan detection methods such as Neural Cleanse [103], Fine-Pruning [103], and STRIP [21]. As a result, when discussing defense mechanisms against our attack, we refer to methods that can ensure a robust aggregation process in FL and pFed, effectively preventing backdoor Trojans from being transferred to benign clients.

Robust Federated Training. Table 2.3 provides a summary of various robust federated training approaches proposed in the context of FL. These approaches include coordinate-wise median, geometric median, α -trimmed mean, as well as their variants and combinations, as outlined in the literature [113]. Recently proposed approaches include weight-clipping and noise addition with certified bounds, ensemble models, differential privacy (DP) optimizers, and adaptive and robust learning rates (RLR) across clients and at the server [31, 75]. Despite differences, existing robust aggregation focuses on analyzing and manipulating the local gradients $\Delta\theta_i^t$, which share the global aggregated model θ_t as the same root, i.e., $\forall i \in [N] : \Delta\theta_i^t = \theta_i^t - \theta^t$. The fundamental assumption in these approaches is that the local gradients from compromised clients $\{\Delta\bar{\theta}_c^t\}_{c \in \mathcal{C}}$ and from legitimate clients $\{\Delta\theta_i^t\}_{i \in N \setminus \mathcal{C}}$ are different in terms of magnitude and direction.

CHAPTER 3

ZK-GANDEF: GAN BASED ZERO-KNOWLEDGE ADVERSARIAL TRAINING DEFENSE

In this chapter, we embark upon an initial investigation into the realm of computational efficient defense mechanisms against adversarial attacks. Our objective is to design a novel defense strategy named **ZK-GanDef**, which leverages the power of generative adversarial networks (GANs) while adopting a zero-knowledge approach. Notably, ZK-GanDef distinguishes itself by eliminating the need for adversarial examples during training, thereby offering a substantial reduction in training costs.

Through a series of comprehensive experiments, we unveil the effectiveness of ZK-GanDef in enhancing the test accuracy of adversarial examples. Our results demonstrate an impressive improvement of up to 49.17% in test accuracy when compared to existing zero-knowledge approaches. It is also noteworthy that the performance of ZK-GanDef remains remarkably close to these cutting-edge defenses, with less than 8.46% degradation in accuracy. Furthermore, ZK-GanDef offers the invaluable advantage of significantly reduced training time, making it an attractive choice for practical implementation.

3.1 Methodology

In this section, we first introduce existing zero-knowledge adversarial training defenses, then, we present the design and the algorithmic details of ZK-GanDef. The additional notations used in this chapter are summarized in Table 3.1.

3.1.1 Zero-knowledge adversarial training

Recall that full knowledge adversarial training defenses retrain NN classifier with adversarial examples. Since adversarial examples are created by solving an optimization

Table 3.1 Additional Notations used in ZK-GanDef Chapter

$\mathcal{L}_{\text{CLP}}, \mathcal{L}_{\text{CLS}}$	loss function used in CLP and CLS methods
$\bar{s}, \bar{S}; \hat{s}, \hat{S}; s, S$	source indicator of $\bar{x}, \bar{X}; \hat{x}, \hat{X}; x, X$
$\mathcal{C}, \mathcal{C}^*$	NN based classifier
$\mathcal{D}, \mathcal{D}^*$	NN based discriminator
J, J'	reward function of the minimax game
$\Omega, \Omega_{\mathcal{C}}, \Omega_{\mathcal{D}}$	weight parameter in the NN model
λ, γ	trade-off hyper-parameters in CLP, CLS and GanDef

problem, its preparation consumes significant amount of computation, especially when iterative adversarial examples are utilized. Based on experiments in [39], generating adversarial examples on Imagenet dataset requires a cluster of GPU servers. To overcome this limitation, authors in [39] also introduce two zero-knowledge adversarial training defenses dubbed CLP and CLS. Instead of retraining with adversarial examples, these approaches retrain with examples perturbed with random Gaussian noise. The idea here is to speedup the training process by eliminating the computationally expensive step of adversarial examples generation. The caveat, however, is that since the retraining is performed with "fake" adversarial examples, the test accuracy against "true" adversarial examples degrades.

The training process of CLP is visualized in Figure 3.1. The retraining dataset consists of several pairs of randomly sampled original examples perturbed with random Gaussian noise. After the feed forward pass through the NN classifier, two different pre-softmax logits are generated. The differences between these pre-softmax logits and their corresponding ground truths are calculated as the first part of the total loss. The l_2 norm of the difference between these two pre-softmax logits is also calculated and used as the second part in the total loss. Based on the total loss, the

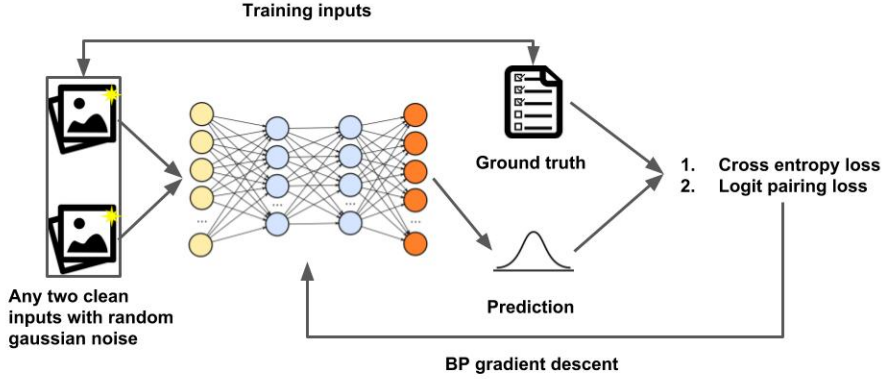


Figure 3.1 Training procedure of Clean Logit Pairing.

weights, Ω , of the NN classifier are updated by gradient descent algorithm and back propagation. The training loss of CLP can be summarized as follows:

$$\mathcal{L}_{\text{CLP}}(\mathcal{C}) = \mathcal{L}(\hat{z}_1 = \mathcal{C}(\hat{x}_1), \hat{t}_1) + \mathcal{L}(\hat{z}_2 = \mathcal{C}(\hat{x}_2), \hat{t}_2) + \lambda_2(\mathcal{C}(\hat{x}_1) - \mathcal{C}(\hat{x}_2))$$

The training process of the other zero-knowledge approach, CLS, is shown in Figure 3.2. Similar to CLP, CLS retrains with examples perturbed with random Gaussian noise. However, instead of using pairs of inputs, CLS uses individual inputs to the NN classifier in the forward pass. The first term of the total loss in CLS is still calculated by a predefined loss function of pre-softmax logits and the corresponding ground truths. Different from the CLP, CLS directly calculates the l_2 norm of pre-softmax logits as the second term in its total loss. Thereafter, it follows the same training process with gradient descent algorithm and back propagation to update the weights, Ω , in the NN classifier. The loss function of CLS is as follows:

$$\mathcal{L}_{\text{CLS}}(\mathcal{C}) = \mathcal{L}(\hat{z} = \mathcal{C}(\hat{x}), \hat{t}) + \lambda_2(\mathcal{C}(\hat{x}))$$

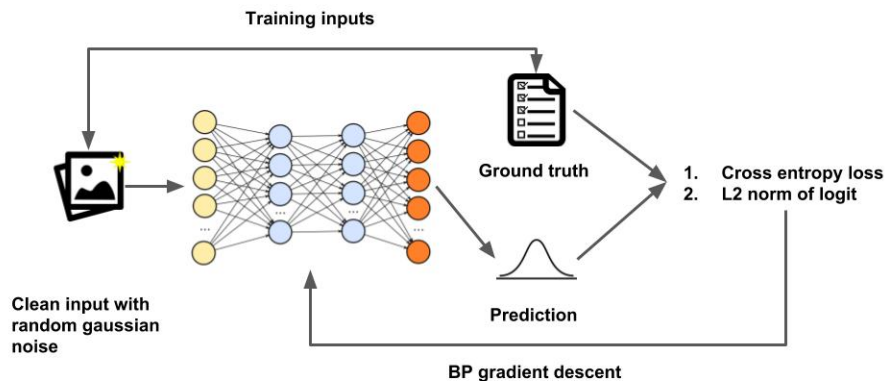


Figure 3.2 Training procedure of Clean Logit Squeezing.

The hypothesis behind CLP and CLS is that abnormal large values in pre-softmax logits are signals of adversarial examples. Therefore, they both add penalty term to the loss function during the training in order to prevent such over confident predictions. Although the penalty terms are different, both defenses encourage the NN classifier to output small and smooth pre-softmax logits.

3.1.2 Design of ZK-GanDef

As mentioned in the previous section, CLP and CLS try to prevent overconfident predictions by penalizing high pre-softmax logits. However, the penalty terms used are oversimplified and do not utilize other valuable information contained in the pre-softmax logits. This results, as we see in the evaluation section, in poor accuracy on complex datasets. On the other hand, our ZK-GanDef is designed to better utilize the rich information available in the pre-softmax logits. As Figure 3.3 shows, Zk-Gandef comprises a classifier and a discriminator. The input to the classifier includes both original images and randomly perturbed examples. It has been shown in transfer learning, [25], that the pre-softmax logits output of the classifier relates to the extracted features from its inputs. Therefore, we use a discriminator to identify

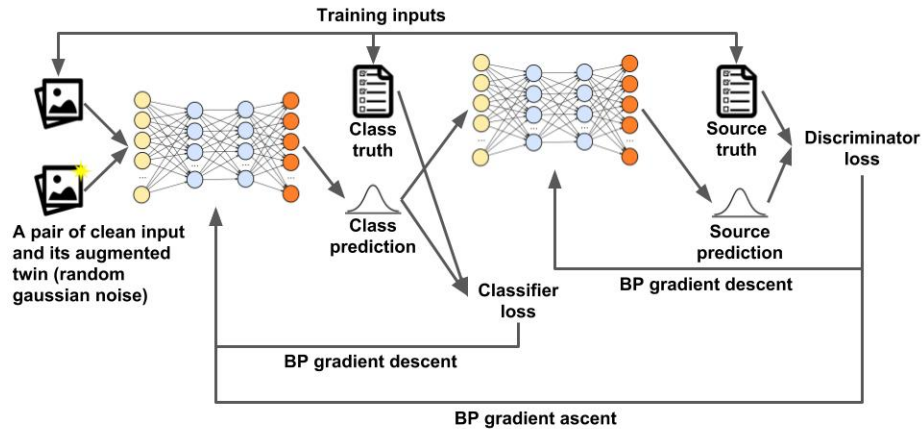


Figure 3.3 Training procedure of ZK-GanDef.

whether the logit output of the classifier belongs to an original image or a perturbed example. The intuition here is that the features extracted by a Vanilla NN classifier from perturbed examples will contain some kind of perturbations, and hence can be recognized by a trained discriminator.

In this work, we envision that the classifier could be seen as a generator that generates pre-softmax logits based on selected features from inputs. Then, the classifier and the discriminator engage in a minmax game, which is also known as *Generative Adversarial Net* (GAN) [25]. In this minimax game, the discriminator tries to make perfect prediction about the source of inputs (original or perturbed). At the same time, the classifier tries to correctly classify inputs as well as hide the source information from the discriminator. This process trains a classifier which makes prediction based on perturbation invariant features from inputs, as well as a discriminator which can identify whether the features used by the fellow classifier contain any perturbations. Through training in this competition game, the feature learning in the classifier is regulated by the discriminator and it finally leads to defense against adversarial examples.

Compared with the CLP and CLS, ZK-GanDef has a more sophisticated way of utilizing pre-softmax logits. Instead of encouraging the NN classifier to make small and smooth logits, ZK-GanDef aims at differentiating the latent pattern of logits between original images and examples with perturbations. Therefore, the NN classifier in ZK-GanDef is encouraged to select perturbation invariant features, which enhance its test accuracy of adversarial examples on complex datasets. It is worth to mention that an example with Gaussian perturbation is not necessary to be an adversarial example. However, results in [39] show that defenses against adversarial examples can be built by training against examples with Gaussian perturbation. Our method is built upon this empirical conclusion.

3.1.3 ZK-GanDef training algorithm

Given the training data pair $\langle x, t \rangle$, where $x \in \cup(\bar{X}, \hat{X})$, we try to find a classification function \mathcal{C} , which uses x to give a proper pre-softmax logits z corresponding to t . The goal is to train the classifier in ZK-GanDef to model the conditional probability $q_C(z|x)$ with only perturbation invariant features. To achieve this, we employ another NN and call it a discriminator \mathcal{D} . \mathcal{D} uses the pre-softmax logits z from \mathcal{C} as inputs and predicts whether the input image to \mathcal{C} was \bar{x} or \hat{x} . This process can be performed by maximizing the conditional probability $q_D(s|z)$, where s is a Boolean variable indicating whether the input to \mathcal{C} was original or randomly perturbed image. The combined minmax problem of the classifier and the discriminator is formulated as:

$$\begin{aligned} & \min_{\mathcal{C}} \max_{\mathcal{D}} J(\mathcal{C}, \mathcal{D}) \\ & = \min_{\mathcal{C}} \max_{\mathcal{D}} \mathbb{E}_{x \sim \bar{X}, t \sim T} \{-\log[q_C(z|x)]\} - \mathbb{E}_{z \sim Z, s \sim S} \{-\log[q_D(s|z = \mathcal{C}(x))]\} \end{aligned}$$

In other words, the training process of the classifier (\mathcal{C}) tries to minimize the log likelihood of predicting s from z , while maximizing the log likelihood of predicting z from x . At the same time, the goal of the discriminator (\mathcal{D}) is to maximize the

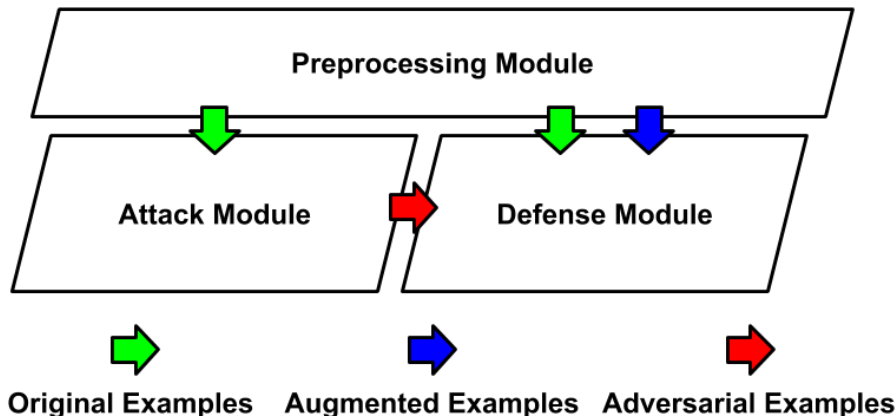


Figure 3.4 Evaluation framework.

log likelihood of predicting s from z . Recall that, similar to CLP and CLS [39], ZK-GanDef uses inputs (x) perturbed with random Gaussian noise as an approximation of true adversarial examples.

The pseudocode for training of ZK-GanDef is similar to that of GanDef which is shown in Algorithm 1. The modification is that we are using examples with Gaussian perturbations as replacements of generated adversarial examples. During the sampling in lines 4 and 9, a number (predefined by user) of examples is evenly sampled from original images \bar{X} and examples with Gaussian perturbations \hat{X} to form a training batch. In lines 6 and 11, the weight parameters in the classifier (discriminator) are frozen before updating the weight parameters in the discriminator (classifier). Finally, in lines 7 and 12, the weight parameters are updated through the stochastic gradient descent algorithm. In this algorithm, we iteratively update the classifier and the discriminator one at a time to emulate the proposed minimax game.

3.2 Evaluation Settings

This section presents the framework that we use to evaluate our defensive method, ZK-GanDef, under different popular adversarial example generators and compare it with other state-of-the-art zero-knowledge adversarial training defenses. Figure 3.4

Algorithm 1 Training ZK-GanDef

Input: training data X , ground truth T , classifier \mathcal{C} , discriminator \mathcal{D}

Output: classifier \mathcal{C} , discriminator \mathcal{D}

- 1: Initialize weight parameters Ω in both classifier and discriminator
 - 2: **for** the global training iterations **do**
 - 3: **for** the discriminator training iterations **do**
 - 4: Sample a batch of training pair, $\langle x, t \rangle$
 - 5: Generate a batch of boolean indicator, s , corresponding to training inputs

 - 6: Fix $\Omega_{\mathcal{C}}$ in classifier \mathcal{C}
 - 7: Update $\Omega_{\mathcal{D}}$ in discriminator \mathcal{D}
 - 8: **end for**
 - 9: Sample a batch of training pair, $\langle x, t \rangle$
 - 10: Generate a batch of boolean indicator, s , corresponding to training inputs
 - 11: Fix $\Omega_{\mathcal{D}}$ in discriminator \mathcal{D}
 - 12: Update $\Omega_{\mathcal{C}}$ in classifier \mathcal{C}
 - 13: **end for**
-

depicts the main components of this framework, which include: (1) Preprocessing module, (2) Attack module and (3) Defense module. Different adversarial example generators and defensive methods could be used as plug-ins to Attack and Defense modules, respectively, to form different test scenarios.

In the following sections, we present the datasets utilized, the detailed description of each module, and a summary of the evaluation metrics used.

3.2.1 Datasets

During our evaluations, the following datasets are utilized:

- MNIST: Contains a total of 70K images and their labels. Each one is a 28×28 pixel, gray scale labeled image of handwritten digit.
- Fashion-MNIST: Contains a total of 70K images and their labels. Each one is a 28×28 pixel, gray scale labeled image of different kinds of clothes.
- CIFAR10: Contains a total of 60K images and their labels. Each one is a 32×32 pixel, RGB labeled image of animal or vehicle.

The images in each dataset are evenly labeled into 10 different classes. Although Fashion-MNIST has exactly the same image size as MNIST, images in Fashion-MNIST have far more details than images from MNIST.

3.2.2 Preprocessing module

Preprocessing module involves the following operations:

- Scaling: Gray scale images use one integer to represent each of their pixels, while RGB images use three different integers (each between 0 and 255) to represent each of their pixels. To simplify the process of finding adversarial examples and to be consistent with the related work, scaling is used to map pixel representations from discrete integers in the range $\mathbb{Z}_{[0,255]}$ into real numbers in the range $\mathbb{R}_{[-1,1]}$.
- Separation: This operation is used to split each input dataset into two groups: training-dataset and testing-dataset. The training dataset is used to train the supervised machine learning models which are the different NN classifiers in this work, while the testing dataset is used by the attack module to generate adversarial examples in order to evaluate the NN classifier under test. The detailed separation plans are: (1) the 70K MNIST and Fashion-MNIST images

are randomly separated into 60K training and 10K testing images, respectively and (2) the 60K CIFAR10 images are randomly separated into 50K training and 10K testing images.

- **Augmentation:** This operation is used to generate augmented examples for different zero-knowledge adversarial training methods. Based on the description in [39] and our communication with its authors, we keep the same augmentation which is adding a Gaussian perturbation with mean $\mu = 0$ and standard deviation $\sigma = 1$. The Gaussian perturbation used in this work is not guaranteed to be the optimal choice and we keep the detailed comparison of different augmentation methods as future work.

3.2.3 Attack module

The attack module implements three popular adversarial example generators, the FGSM [26], the BIM [46] and the PGD [66]. As we mention in the previous section, all adversarial example generators are utilized under the white-box scenario. Moreover, each original example has its own corresponding adversarial counterparts (FGSM, BIM, PGD). Adversarial examples from same dataset share same maximum l_∞ perturbation limits which are 0.6 in MNIST & Fashion-MNIST and 0.06 in CIFAR10. For the BIM, we also limit the per step perturbation to 0.1 in MNIST & Fashion-MNIST and 0.016 in CIFAR10. Finally, for the PGD, we run generation algorithm 40 iteration with 0.02 per step perturbation on MNIST & Fashion-MNIST and 20 iteration with 0.016 per step perturbation on CIFAR10. To ensure the quality of the adversarial example generators, we choose the standard python library, CleverHans [77], which is adopted by the community.

3.2.4 Defense module

This module implements the Vanilla NN classifiers as well as the different defense methods that we evaluate in this work. For the same dataset, different defense methods share the same structure of the classifier as that of the Vanilla. Hyperparameters of defenses we compare with are the exact ones used in their original

papers. Our ZK-GanDef is tuned by line search to find a suitable hyper-parameter setting.

Vanilla classifier For each dataset, we use as a baseline a NN classifier with no defenses, which is also referred to as the Vanilla classifier. We select different Vanilla classifiers for each dataset. The structure of the Vanilla classifier used in MNIST and Fashion-MNIST dataset is LeNet [66]. For the CIFAR10 dataset, we use the allCNN based classifier [92]. Due to space limitations, the detailed NN structure and training settings are not listed.

Zero-knowledge defenses We implement here three different approaches: (1) a classifier trained with CLP [39], (2) a classifier trained with CLS [39], and (3) a classifier trained with ZK-GanDef. As Figures 3.1 and 3.2 show, CLP and CLS train only with randomly perturbed examples. On the other hand, ZK-GanDef (Figure 3.3) trains with both original and randomly perturbed examples. We note also that the structure of the discriminator in ZK-GanDef (Table 3.2) does not change with different datasets. Training of the discriminator utilizes the Adam optimizer [42] with 0.001 learning rate.

Full knowledge defenses We implement here three of the full knowledge defenses: (1) a classifier trained with original and FGSM examples (FGSM-Adv), (2) a classifier trained with original and PGD examples (PGD-Adv), and (3) a classifier trained with original and PGD examples through GAN based training (PGD-GanDef). Among them, PGD-Adv is the state-of-the-art full knowledge adversarial training defense.

3.2.5 Evaluation metrics

The overall classifier performance is captured by the *test accuracy* metric, which is defined as the ratio of the total number of tested images minus the number of failed

Table 3.2 Discriminator Structure

Layer	Kernel Size	Strides	Padding	Activation
Dense	32	-	-	ReLU
Dense	64	-	-	ReLU
Dense	32	-	-	ReLU
Dense	1	-	-	Sigmoid

tests to the total number of tested images (both original and adversarial).

$$\text{test accuracy} \equiv \frac{\text{total \# of test examples} - \text{\# of failed tests}}{\text{total \# of test examples}}$$

A test is considered failed when: (1) original example is missclassified, (2) original example is rejected, or (3) adversarial example is accepted with incorrect classification. To be more precise during evaluation, we separately compute the test accuracy on original and adversarial examples. When a defensive method tries to maximize classifier’s capability to identify adversarial examples, the classifier may reject or missclassify more original examples than the corresponding Vanilla classifier. The trade-off between correctly classifying original and adversarial examples is the same as the trade-off between *true positive rate* and *true negative rate* in machine learning.

The other important metric to evaluate defense approaches is the training time it takes to build the model. As mentioned earlier, a significant amount of computation is consumed to generate the adversarial examples for full knowledge adversarial training. The two main contributing factors to the training time are: (1) the structure of the classifier (number of layers and parameters) and (2) the searching algorithm of adversarial examples (e.g., single-step vs. iterative approaches). The goal is to minimize the training time while maintaining acceptable test accuracy.

3.3 Evaluation Results

In this section, we present comparative evaluation results of the ZK-GanDef with other state-of-the-art zero-knowledge as well as full knowledge adversarial training defenses introduced previously. The evaluation results are summarized in three sections. In the first sections, we provide comparative evaluation of ZK-GanDef with other zero-knowledge and full knowledge adversarial training defenses on classifying original and different types of adversarial examples. Then, we compare the computational consumption of ZK-GanDef with other full knowledge adversarial training defenses in terms of training time per epoch. In the third sections, we analyze the convergence issues of CLP and CLS on CIFAR10 dataset.

3.3.1 Test accuracy on different examples

In this subsection, we show the test accuracy of the Vanilla classifier and the classifiers with defenses against different types of examples. As mentioned earlier, the experiments are conducted on MNIST, Fashion-MNIST and CIFAR10 datasets. For each dataset, a total of 28 different results are calculated. These results span all possible pairs of 7 different classifiers (Vanilla, CLP, CLS, ZK-GanDef, FGSM-Adv, PGD-Adv and PGD-GanDef) and 4 different kinds of examples (original, FGSM, BIM and PGD). All the validation results are presented in Figure 3.5 and detailed in Table 3.3.

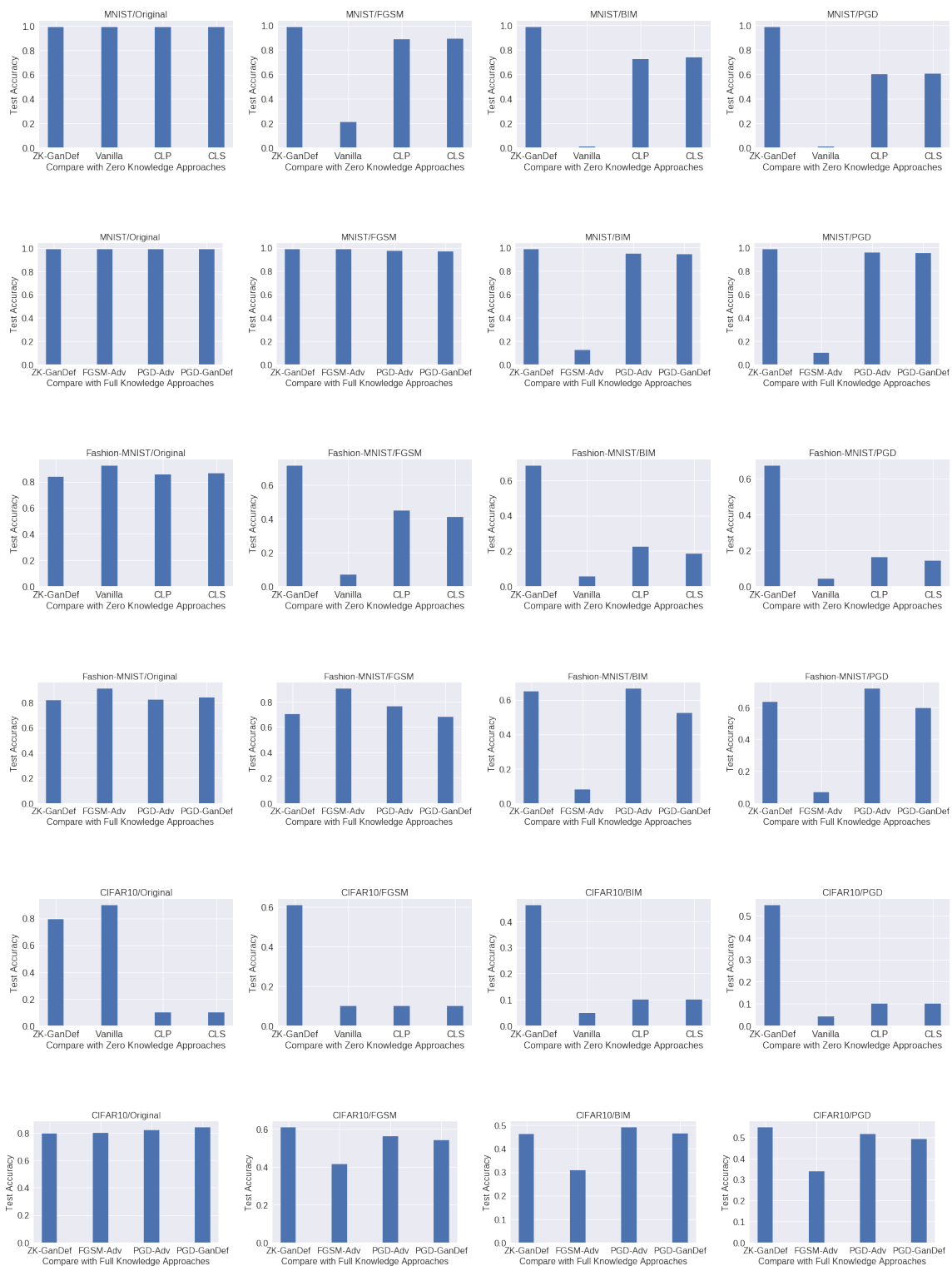


Figure 3.5 Test accuracy on different examples.

Note: In the 1st and 2nd rows, the results on MNIST dataset are presented. In the 3rd and 4th rows, the results on Fashion-MNIST dataset are presented. In the 5th and 6th rows, the results on CIFAR10 dataset are presented.

Table 3.3 Test Accuracy on Different Examples

	MNIST			Fashion-MNIST			CIFAR10					
	Original	FGSM	BIM	PGD	Original	FGSM	BIM	PGD	Original	FGSM	BIM	PGD
Vanilla	98.92%	21.01%	1.00%	0.77%	92.43%	7.01%	5.62%	4.06%	89.92%	9.97%	4.93%	4.06%
CLP	99.13%	88.70%	72.61%	59.93%	85.65%	44.78%	22.30%	16.14%	10.00% ¹	10.00% ¹	10.00% ¹	10.00% ¹
CLS	99.24%	89.29%	73.84%	60.63%	86.37%	41.14%	18.55%	14.17%	10.00% ¹	10.00% ¹	10.00% ¹	10.00% ¹
ZK-GanDef	98.95%	98.97%	98.89%	98.71%	81.95%	70.19%	64.97%	63.34%	79.33%	60.91%	46.27%	54.85%
FGSM-Adv	99.07%	98.79%	12.24%	9.73%	91.17%	90.48%	7.97%	6.81%	79.88%	41.53%	30.74%	33.86%
PGD-Adv	99.15%	97.60%	94.75%	95.60%	82.33%	76.42%	66.72%	71.80%	82.06%	56.18%	49.21%	51.51%
PGD-GanDef	99.10%	96.85%	94.28%	95.31%	84.09%	68.19%	52.35%	59.51%	84.05%	54.14%	46.64%	49.21%

Note: The left column shows the test results on MNIST dataset. The middle column shows the test results on Fashion-MNIST dataset. The right column shows the test results on CIFAR10 dataset.

3.3.2 On original examples

In Figure 3.5, we first focus on the results presented in the first column sub-figures. These results represent the test accuracy on original examples from different datasets. As a baseline, the Vanilla classifier achieves 98.92% test accuracy on MNIST, 92.43% test accuracy on Fashion-MNIST and 89.92% test accuracy on CIFAR10. These results are consistent with the benchmark ones presented by Rodrigo in [5].

We then evaluate the test accuracy of the three zero-knowledge defenses (CLP, CLS and ZK-GanDef) on different datasets. On MNIST dataset, their test accuracy on original examples is at the same level as that of the Vanilla classifier. The detailed results from Table 3.3 show that the difference in test accuracy among the defenses is within 0.5%, which is small enough to be ignored. On Fashion-MNIST dataset, the test accuracy of CLP and CLS is 5% higher than that of ZK-GanDef on original examples. Moreover, the test accuracy of all zero-knowledge approaches is (6% to 11%) lower than that of the Vanilla classifier. This small degeneration is a result of tuning the model to enhance test accuracy on adversarial examples. On CIFAR10 dataset, CLP and CLS have a significantly lower test accuracy compared with the Vanilla classifier and ZK-GanDef. This is because the classifiers with the CLP and CLS methods do not converge at the beginning of the training. A detailed study of this phenomenon is provided in the following subsection.

Finally, we conduct the same evaluation with full knowledge adversarial training defenses and perform comparison with the proposed ZK-GanDef. In evaluation results on MNIST dataset, all full knowledge defenses and ZK-GanDef achieve the same level of test accuracy as that of the Vanilla classifier. When evaluating the Fashion-MNIST dataset, FGSM-Adv achieves similar test accuracy on original examples to that of the Vanilla classifier. At the same time, ZK-GanDef, PGD-Adv and PGD-GanDef have

¹On CIFAR10 dataset, CLP and CLS have convergence issues during training and hence the classifier is making random guessing. A detailed study of this issue is provided in a following subsection.

about 10% to 12% degeneration from that of the Vanilla classifier. When evaluating the CIFAR10 dataset, ZK-GanDef performance is similar to that of full knowledge defenses and their test accuracy on original examples are 6% to 10% lower than that of the Vanilla classifier, respectively. To enhance test accuracy on adversarial examples, the decision boundary of the classifier becomes complex with more curves, which causes the degeneration on classifying original examples compared to the Vanilla classifier [66].

3.3.3 On single-step adversarial examples

We discuss here the accuracy results on FGSM examples, which are depicted on sub-figures on the second column of Figure 3.5. Intuitively, the Vanilla classifier has poor performance on these single-step adversarial examples, with test accuracy of 21.01% on MNIST, 7.01% on Fashion-MNIST, and 9.97% on CIFAR10.

Compared with the Vanilla classifier, all zero-knowledge defenses achieve a significant enhancement in terms of test accuracy on all datasets, with the exception of CLP and CLS on CIFAR10 dataset. Among the zero-knowledge approaches, ZK-GanDef achieves the highest test accuracy on all the datasets with significant margin. On MNIST, the test accuracy is 88.70%, 89.29%, and 98.97% with CLP, CLS, and ZK-GanDef, respectively. On Fashion-MNIST, the test accuracy is 44.78%, 41.14%, and 70.19% CLP, CLS, and ZK-GanDef, respectively. On CIFAR10, ZK-GanDef is the only zero-knowledge defense that reasonably works test accuracy around 60.91%.

In general, full knowledge approaches have better understanding of the adversarial examples since such examples are part of their training datasets. Therefore, full knowledge approaches should, intuitively, have better test accuracy compared to their zero-knowledge counterparts. Our results confirm this observations. The results show that the test accuracy of full knowledge approaches is significantly higher than

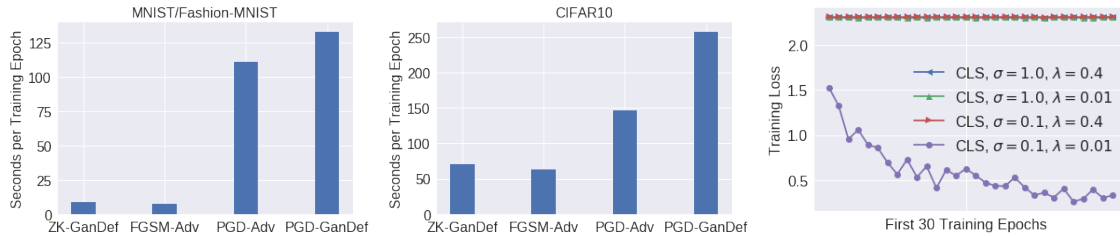


Figure 3.6 Training time and training loss.

The left sub-figure is training time on MNIST and Fashion-MNIST. The middle sub-figure is training time on CIFAR10. The right sub-figure is the training loss of CLS under different hyper-parameters.

those of CLP and CLS, especially on Fashion-MNIST and CIFAR10 datasets. On the other hand, the test accuracy of ZK-GanDef is comparable to those of full knowledge defenses. In fact, the test accuracy of ZK-GanDef (98.97%) is higher than those of all the full knowledge defenses (98.79%, 97.6% and 96.85%). This is because handwritten digits in MNIST are gray scale figures with no detailed texture, and therefore, ZK-GanDef can train to select strongly denoised (even binarized) features without losing information. As a result, ZK-GanDef can achieve even higher test accuracy than full knowledge approaches.

When evaluating the Fashion-MNIST, FGSM-Adv achieves the highest test accuracy (90.48%). The PGD-Adv, PGD-GanDef and ZK-GanDef achieve the second tier test accuracy (76.42%, 68.19% and 70.19%). This is because FGSM-Adv utilizes only original and FGSM examples during training, and therefore, the trained classifier is overfitting on FGSM examples. This behavior has been observed in [98] and denoted as gradient masking effect. When evaluating the CIFAR10, PGD-Adv, PGD-GanDef and ZK-GanDef achieve comparable test accuracy (56.18%, 54.14% and 60.19%, respectively), while the test accuracy of FGSM-Adv is only at 41.53%. Due to the input dropout in allCNN classifier, the diversity of training data is enhanced and the overfitting of FGSM-Adv is inhibited [98]. However, FGSM examples are generated with the weaker single-step method, and hence the test accuracy degenerates on the stronger iterative examples.

3.3.4 On iterative adversarial examples

We analyze here the test accuracy results on BIM and PGD examples, which are depicted on the sub-figures of the third and the fourth columns of Figure 3.5, respectively. The figure clearly shows that the Vanilla classifier completely failed with both BIM and PGD examples. This is because BIM and PGD are iterative adversarial examples and hence are carefully crafted to mislead Vanilla classifiers.

Based on the test accuracy results, using zero-knowledge defenses could still enhance the performance on these stronger adversarial examples. However, these enhancements are lower than those on FGSM examples. Among zero-knowledge defenses, the test accuracy of ZK-GanDef is significantly higher than those of CLP and CLS on all iterative adversarial examples. On MNIST, the test accuracy of ZK-GanDef with BIM and PGD examples is 25% and 38%, respectively, higher than those of CLP and CLS. On Fashion-MNIST, the test accuracy of ZK-GanDef on BIM and PGD examples is 42% and 47%, respectively, higher than those of CLP and CLS. On CIFAR10, only ZK-GanDef could work and it achieves 46.27% and 54.85% test accuracy on BIM and PGD examples, respectively.

As mentioned in Section 3.1.1, full knowledge defenses could achieve larger enhancement in test accuracy compared to the existing zero-knowledge defenses, CLP and CLS. FGSM-Adv is the exception as evidenced by its poor performance in defending iterative adversarial examples due to the reasons we mentioned in Section 3.3.3. Based on evaluation results from MNIST and Fashion-MNIST, the test accuracy of FGSM-Adv on BIM and PGD examples has a huge decrease from over 90% to around 10%. Although such huge decrease does not exist in the case of CIFAR10, the test accuracy of FGSM-Adv is clearly lower than that of PGD-Adv and PGD-GanDef. On all datasets, PGD-Adv and PGD-GanDef have much stable test accuracy with limited decrease of test accuracy on FGSM examples. More importantly, the results show that the test accuracy of ZK-GanDef is close to those

of PGD-Adv and PGD-GanDef on iterative adversarial examples from the three datasets.

We summarize our findings from the results as: (i) ZK-GanDef is significantly better than existing zero-knowledge defenses (CLP and CLS) due to its higher test accuracy on adversarial examples and its scalability to large datasets. This clearly supports our vision that utilizing a more flexible and sophisticated way to handle the pre-softmax logits (ZK-GanDef) is better than forcing the pre-softmax logits to be smooth at a small scale (CLP and CLS). (ii) The test accuracy of ZK-GanDef is comparable to that of the state-of-the-art full knowledge adversarial training defenses. This supports our hypothesis that using perturbation invariant features in the classifier could greatly enhance test accuracy on adversarial examples. (iii) On contrast with full knowledge defenses, ZK-GanDef is adaptable to new types of adversarial examples. We see from the results that FGSM-Adv has significant adaptability issue on MNIST and Fashion-MNIST datasets. This issue is not observed on CIFAR10 due to the input dropout in classifier structure [98]. For PGD-Adv, the current evaluation does not show its adaptability issue, but it is not guaranteed given that stronger adversarial examples could be generated in the future [98][85]. On the other hand, the results show that ZK-GanDef has better adaptability to new types of adversarial examples because its training is independent of such examples.

3.3.5 Generalizability

In the previous evaluation, all iterative adversarial examples are generated by methods based on projected gradient descent. In order to show the generalizability of ZK-GanDef, we conduct the evaluation on an extra set of adversarial examples, Deepfool [72] and Carlini & Wagner (CW) examples [11]. Unlike adversarial examples used in previous evaluation, Deepfool and CW adversarial examples contain perturbation patterns that are significantly different from Gaussian perturbation.

Table 3.4 Test Accuracy on Deepfool and CW Examples

MNIST		Fashion-MNIST		CIFAR10	
Deepfool	CW	Deepfool	CW	Deepfool	CW
98.72%	98.46%	89.52%	66.43%	86.08%	47.22%

Therefore, this evaluation could reveal the generalizability of ZK-GanDef in defending other adversarial examples. The evaluation is conducted on all three datasets. The Deepfool and CW adversarial examples utilize the same hyper-parameter setting as PGD adversarial examples.

The evaluation results are summarized in Table 3.4. It is clear that ZK-GanDef can classify Deepfool adversarial examples with over 85% accuracy in all three datasets which matches the test error presented in [72]. The reason is that Deepfool tries to find adversarial examples with smaller perturbation than projected gradient descent based adversarial examples (FGSM, BIM, PGD). As a result, Deepfool examples are easier to defend. For CW examples, ZK-GanDef achieves the same level of test accuracy on all three datasets. To conclude, ZK-GanDef is not limited to defend a specific type of perturbation. Although ZK-GanDef only utilizes Gaussian noise perturbation during training, its defense can be generalized to a wide range of adversarial examples which include FGSM, BIM, PGD, Deepfool and CW examples.

3.3.6 Training time

We evaluate here the training time of ZK-GanDef in terms of seconds per training epoch. MNIST and Fashion-MNIST share the same image size and classifier structure and hence has the same training time. Since the test accuracy of ZK-GanDef is significantly higher than those of the existing zero-knowledge defenses, CLP and CLS, we only compare the training time of ZK-GanDef with those of full knowledge defenses

(FGSM-Adv, PGD-Adv and PGD-GanDef). We utilize a fixed number of training epochs (80 for MNIST and 300 for CIFAR10) and results show that all defensive methods converge at epoch 30 on MNIST and at epoch 240 on CIFAR10. Since the records of training time per epoch have a very small deviation, we take the average value of records in all epochs and compare different defense methods with it. The results are recorded during the training on a workstation with a NVIDIA GTX 1080 GPU.

The left sub-figure of Figure 3.6 shows that the training time of ZK-GanDef on MNIST/Fashion-MNIST (8.75s) is close to that of FGSM-Adv (7.83s), while it surges to 110.85s and 132.75s in the case of PGD-Adv and PGD-GanDef, respectively. The evaluation results on CIFAR10 dataset (the middle sub-figure of Figure 3.6) follow a similar trend to that of the results on MNIST and Fashion-MNIST datasets. ZK-GanDef and FGSM-Adv take much less training time per epoch (71.20s and 62.85s, respectively) compared to that of PGD-Adv (146.91s) and that of PGD-GanDef (257.72s). For example, on CIFAR10 dataset, the end-to-end training time of PGD-Adv takes 14.3 hours, while training of ZK-GanDef only takes 6.9 hours. In summary, ZK-GanDef provides test accuracy close to that of the best state-of-the-art full knowledge defenses (PGD-Adv), while reducing the training time by 92.11% and 51.53% on MNIST/Fashion-MNIST and CIFAR10, respectively.

3.3.7 Convergence issue

As presented, the evaluation results of CLP and CLS on CIFAR10 dataset show that these two zero-knowledge adversarial training defenses fail to correctly classify both original and adversarial examples. This is mainly because the training loss of CLP and CLS does not converge during training. The mathematical models of CLP and CLS presented in previous section follow the same design logic that aims at preventing over confident predictions. CLP achieves its goal by adding l_2 norm penalty on

the difference of two randomly selected pre-softmax logits, while CLS adds l_2 norm penalty on any pre-softmax logits. Moreover, CLP and CLS do not include original examples in their training dataset, which means that they miss important features that can help discriminate examples with and without perturbations. Therefore, this design logic is too simple and lacks flexibility compared with ZK-GanDef, which utilizes minimax game with discriminator and trains on examples with and without perturbations. When training on complex datasets like CIFAR10, the simple and less flexible design logic leads to convergence issues for CLP and CLS.

To further validate this conclusion, we record the loss of CLS during the first 30 training epochs and depict the results on the right sub-figure of Figure 3.6. The training loss is recorded on four different hyper-parameter settings of CLS: (1) normal CLS ($\sigma = 1.0, \lambda = 0.4$), (2) CLS with reduced perturbations ($\sigma = 1.0, \lambda = 0.01$), (3) CLS with reduced penalty ($\sigma = 0.1, \lambda = 0.4$), and (4) CLS with reduced perturbation and penalty ($\sigma = 0.1, \lambda = 0.01$). Figure 3.6 shows that the curves of the first three settings overlap with each other and form the horizontal curve on the top. This clearly shows that CLS does not learn any useful features and hence the training loss does not converge (does not decrease) under these three settings. Under the last setting, CLS was able to learn useful features and hence the training loss decreases towards convergence. However, with the last setting, CLS falls back to Vanilla classifier, which fails to defend against adversarial examples. A similar experiment is also conducted with CLP and the results follow the same pattern. The only difference is that the training loss goes to “nan” on CLP under the first three settings, which means that the classifier diverges during training.

3.4 Conclusion

In this chapter, we introduce a new zero-knowledge adversarial training defense, ZK-GanDef, which combines adversarial training and feature learning to better

recognize and identify adversarial examples. We evaluate the test accuracy and the training overhead of ZK-GanDef against state-of-the-art zero-knowledge adversarial training defenses (CLP and CLS) as well as full knowledge adversarial training defenses (FGSM-Adv and PGD-Adv). The results show that ZK-GanDef enhances the test accuracy on original and adversarial examples by up to 49.17% compared to zero-knowledge defenses. More importantly, ZK-GanDef has close test accuracy to full knowledge defenses (test accuracy degeneration is below 8.46%), while taking much less training time. In a nutshell, ZK-GanDef provides test accuracy much higher than existing zero-knowledge approaches and close to that provided by the state-of-the-art full knowledge defence (PGD-Adv), while taking much less training time. Additionally, in contrast to full knowledge defenses, ZK-GanDaf can adapt to new types of adversarial examples because its training is adversarial example agnostic.

Although the zero-knowledge defense allows us to win the efficiency, the approach lacks the probability of leveraging the adversarial gradient to enhance the defense performance. Therefore, in the next chapter, we introduce another defense that achieves a better balance between efficiency and performance.

CHAPTER 4

SEMI-ITERATIVE ADVERSARIAL TRAINING

In addition to the zero-knowledge defense proposed in Chapter 3, another approach to achieving computational efficiency in defense is retraining with low-cost adversarial examples. This chapter aims to design a novel defense strategy of this kind, named semi-iterative adversarial training (**SIM-Adv**). Unlike the aforementioned zero-knowledge approach, SIM-Adv still leverages adversarial gradients, enabling it to adapt to newly introduced adversarial examples.

To accomplish this objective, we commence by conducting experimental analyses on several state-of-the-art adversarial training defenses. Drawing insights from these experiments, we subsequently introduce SIM-Adv, a defense mechanism capable of countering both single-step and iterative adversarial examples. Through comprehensive evaluations, we demonstrate that our proposed method surpasses the state-of-the-art single-step adversarial training defense, ATDA, by achieving a remarkable enhancement of up to 35.67% in test accuracy while reducing training time by 19.14%. Furthermore, when compared to the state-of-the-art adversarial training methods employing iterative adversarial examples (such as BIM and Madry examples), our proposed method achieves savings of up to 76.03% in training time, with a minimal trade-off of less than 3.78% in test accuracy.

4.1 Analysis of Iter-Def Methods

Compared with Single-Def, Iter-Def methods have significantly higher test accuracy. Therefore, the majority of adversarial training defenses, including the SOTA ones, focus on Iter-Def. In spite of this, the domain is still not very well comprehended. In this section, we explore several fundamental questions regarding SOTA Iter-Def methods, through an extensive set of experiments.

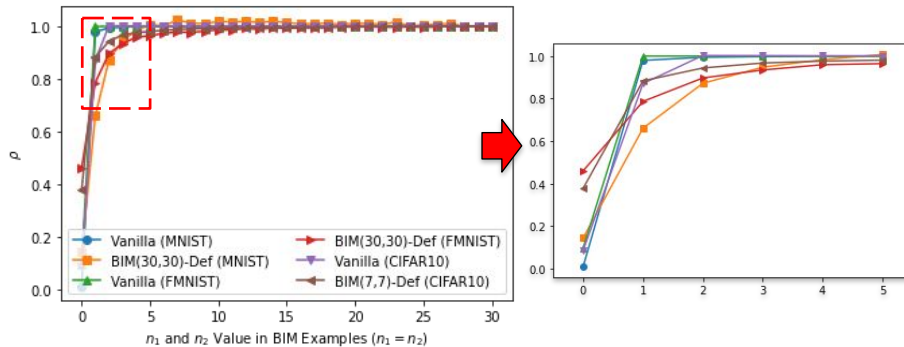


Figure 4.1 Empirical results on per-step perturbation

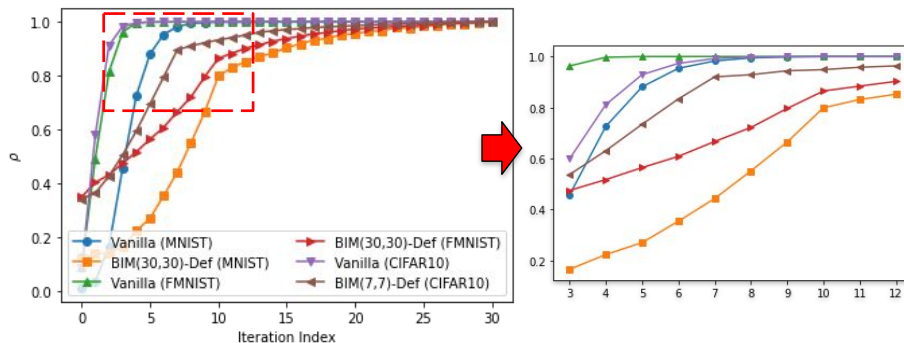


Figure 4.2 Empirical results on intermediate examples.

4.1.1 Limit of small per-step perturbation

Based on the introduction of Iter-Exps in Section 2, it is clear that the smaller the per-step perturbation applied, the better the observation of NN’s decision hyperplane, and the stronger adversarial examples obtained. However, to select an appropriate per-step perturbation, we believe that a quantitative analysis is needed.

We approach this goal by first conducting a series of experiments on MNIST, FMNIST, and CIFAR-10 datasets. For each dataset, we train two different NN classifiers with the same structure and hyper-parameter settings: (1) a Vanilla classifier trained on original examples only; and (2) a BIM-Def classifier [45]. For each $n_1 = n_2$ value in the range $\mathbb{Z}_{[0,30]}$, we generate BIM examples with fixed ϵ (0.3 in MNIST, 0.2 in FMNIST, and $\frac{8}{255}$ in CIFAR-10) and calculate the ratio ρ as:

$$\rho = \frac{\text{error rate under current value of } n_1 \text{ and } n_2}{\text{error rate under the maximum value of } n_1 \text{ and } n_2} \quad (4.1)$$

To understand ρ , assume that its value is 1 when, for example, $n_1 = n_2 = 10$ and the maximum value of n_1 and n_2 is 30. This means that BIM examples generated with value $n_1 = n_2 = 10$ can be as successful as those generated with value $n_1 = n_2 = 30$ in misleading the classifier.

From the results in Figure 4.1, it is clear that ρ converges fast and saturates when the value of $n_1 = n_2$ is around 5 in all six lines. For the Vanilla classifiers, this phenomenon is not surprising since they have no defence at all and most of the adversarial examples can fool them. However, we see a similar trend from the BIM-Def classifiers which are well trained to defend adversarial examples. The insight we draw from this experiment is that increasing the value of $n_1 = n_2$ over a certain limit provides only marginal help in finding stronger adversarial examples. In other words, training a classifier by Iter-Def with small $n_1 = n_2$ values (around 5 in this experiment) is as efficient as training with large $n_1 = n_2$ values (30 in this experiment).

Given the fact that adversarial training uses adversarial examples to find blind spots of the under-trained classifier and retrains it, these results show that *decreasing the per-step perturbation of Iter-Exps used during Iter-Def beyond a certain limit only marginally benefits the defense.*

We think the saturation of per-step perturbation exists because the loss structure used in projected gradient descent to search Iter-Exps is highly tractable [66]. This important finding indicates that defenses could use smaller values of $n_1 = n_2$ without sacrificing the quality of the defense. Although the resulting defense is still within the Iter-Def category, it consumes less time and computations in preparing adversarial examples. We will utilize this observation in combination with the following others to develop an efficient Single-Def method.

4.1.2 Training with intermediate examples

As shown in Chapter 2, Iter-Def usually uses final adversarial examples (\tilde{x}_{n_2}) to build the defense, since it is much stronger than **intermediate examples** ($\tilde{x}_i, \forall i < n_2$). In this section, we explore whether those intermediate examples can be utilized for training while being generated, instead of sitting idle and waiting for the final versions of the generated examples.

To investigate this, we conduct another set of experiments on the MNIST, FMNIST, and CIFAR-10 datasets. In these experiments, we use the same NN classifiers; we measure the same ratio (ρ as defined in Section 4.1.1); and we maintain the same perturbation limit used in the preceding subsection. The only difference is that we here assign the value of n_1 (10 for MNIST and FMNIST, 7 for CIFAR-10) and n_2 (30 for all) in BIM examples. Values along the X-axis in Figure 4.2 correspond to different iterations during the generation of BIM examples. For example, in MNIST, an X-axis value of zero corresponds to the original examples; a value of $0 < i < 30$ represents the corresponding intermediate examples after i iterations; and a value of 30 corresponds to the final versions, BIM(10,30), of the adversarial examples.

The results show that ρ , under all the scenarios, is monotonically increasing with the number of iterations. For all three Vanilla classifiers which have no defense against adversarial examples, ρ saturates quickly after around 5 iterations. Although ρ for BIM-Def classifiers does not saturate as quickly as that for Vanilla classifiers, it increases almost exponentially before reaching around 0.8 (MNIST and FMNIST) or 0.9 (CIFAR-10). In the zoom-in view of Figure 4.2, we can clearly see that the turning point in the BIM-Def classifier corresponds to the selected value of n_1 (10 or 7). In other words, when the perturbation iterates to its limit, the gap between intermediate and final adversarial examples is quickly mitigated.

Our insight from this experiment is that: *training with intermediate examples of Iter-Exps produces defenses that are comparable to those trained with the final*

examples. In other words, we can utilize the intermediate examples during the preparation of Iter-Exps to continuously enhance the model instead of waiting for the final examples. In Section 4.2, we build on this finding to expand the generation process of Iter-Exps and to end up with our **Single-Def** method that performs very close to Iter-Def.

4.1.3 Summary

In this section, we identify two experimental properties of Iter-Def: (1) It is recommended to use large per-step perturbation, i.e., small n_1 and (2) intermediate examples can be used to expedite adversarial training with a tolerable degeneration in quality.

4.2 Single-Step Epoch-Iterative Method

In Section 4.1, we conduct experiments and evaluate Iter-Def methods in detail. The insights from the results help us enhance our understanding of adversarial training and its underlying fundamental concepts. In this section, we propose a new Single-Def method which we call Single-Step Epoch-Iterative Method.

4.2.1 Motivation and design

In Figure 4.3, we review the process of adversarial training. Each row in the figure represents a training epoch, and the solid black lines represent the data flow (training examples). The dashed red lines across different rows correspond to knowledge flow (classifier’s weights) from one epoch to the next. As shown in the figure, the original examples are fed into the generator of adversarial examples, which could be single-step or iterative. Then, the original and adversarial examples are used to train the classifier. The training process consists of several training epochs, and the weights of the classifier are carried out from one training epoch to the next.

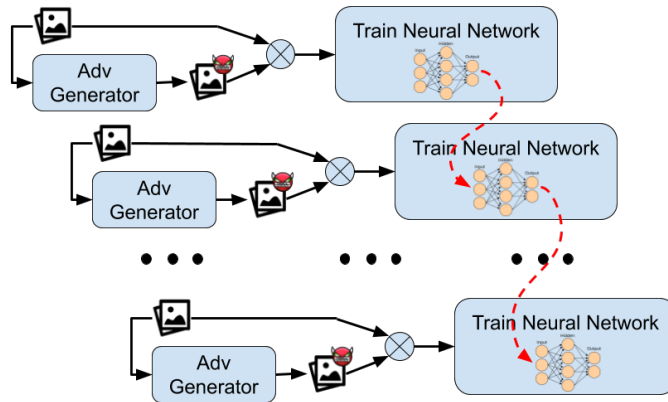


Figure 4.3 Flow of traditional adversarial training.

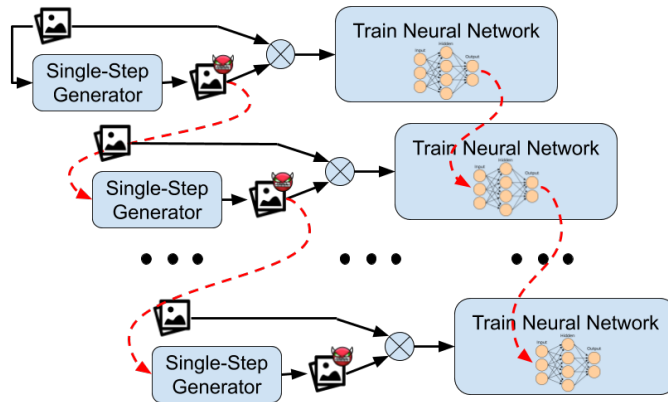


Figure 4.4 Flow of single-step epoch-iterative method.

Inspired by the empirical findings drawn from our previous experiments (Section 4.1), we propose the following modifications to enhance the Single-Def process. Similar to other Single-Def approaches, our method also uses the single-step generator to reduce computation overhead in each epoch. Recall that a classifier which is trained with Single-Exps fails to defend Iter-Exps; therefore, we use consecutive training epochs to mimic the generation of Iter-Exps.

Starting from the second training epoch, we reuse the output of the generator from the previous epoch as input to the generator of the current epoch, instead of using the original image. As a result, the classifier can be seen as trained with intermediate examples in the first $(n_2 - 1)$ training epochs. In each training epoch, our method uses a relatively large per-step perturbation (i.e., small n_1) instead of

complete perturbation (i.e., $n_1 = 1$). It helps our method to avoid repeatedly generating Single-Exps for training. On the other hand, a large per-step perturbation ensures the adversarial examples can quickly reach their maximum perturbation. Therefore, it can mitigate the degeneration caused by training with weak intermediate examples in the first few training epochs. After n_2 epochs, the generator switches back to select original examples as inputs (i.e. to reset the iteration over consecutive epochs). By repeating this process, we could fill an arbitrary number of training epochs.

From a high-level point of view, we flatten the iteration of generating Iter-Exps into training epochs. Within each iteration of n_2 consecutive training epochs, the mathematical formulation of generating adversarial examples is as follows.

$$\begin{aligned}\delta_{i+1} &= \text{clip}_{[-\epsilon, \epsilon]} \left[\frac{\epsilon}{n_1} \times \text{sign}[\nabla_{\tilde{x}_i} L(\tilde{x}_i, y, \theta)] \right] \\ \tilde{x}_{i+1} &= \text{clip}_{[0, 1]}[\tilde{x}_i + \delta_{i+1}] \quad i \in \{0, \dots, n_2\}\end{aligned}$$

Here, i represents the index of iteration over training epochs while $\tilde{x}_0 = \hat{x}$ is the starting point. We present the process of SIM-Def in Figure 4.4.

4.2.2 Applying over-perturbation

In Section 4.2.1, we present the core design of using Single-Exps to mimic Iter-Exps. At the same time, we also demonstrate the potential disadvantage of this design. In the majority of training epochs, our method uses the intermediate examples. Recall the experiment results in Figure 4.2, these intermediate examples are weaker than the corresponding final Iter-Exps. As a result, the classifier trained with SIM-Def can defend against adversarial examples, but it performs worse than that trained with Iter-Def.

To further mitigate the gap in performance, we now introduce a heuristic modification of the hyper-parameter setting which we call **over-perturbation**. We

define two different hyper-parameter settings in adversarial training methods. We consider the setting to be in over-perturbation when $n_1 < n_2$; otherwise, we consider it to be in under-perturbation. This modification is based on our empirical results. As mentioned earlier, the intermediate examples are less adversarial than the final output of an Iter-Exps generator. However, the zoom-in view in Figure 4.2 shows the existence of a turning point in the iteration index relative to the value of n_1 . Before the turning point, the success rate to mislead classifiers by intermediate examples is much lower than those of the final examples but increases exponentially, and vice versa.

By applying over-perturbation, we actually ensure that our method trains the classifier with strong adversarial examples in most of the training epochs. Assume we run 20 epochs of training with two settings (1) $n_1 = n_2 = 10$ and (2) $2n_1 = n_2 = 20$. Under the first setting, the classifier is trained with intermediate examples before the turning point in both 1st to 9th and 11th to 19th epochs. While, under the second setting, the intermediate examples used between 10th and 19th epoch are after the turning point. Overall, the second setting spends more epochs in training the classifier with strong adversarial examples, and the trained classifier performs better on defending adversarial examples.

Actually, the over-perturbation setting has been used in previous research works, intentionally or unintentionally. For example, in [66], the hyper-parameter settings of all Madry-Def methods are over-perturbation. For curiosity, we try to change the hyper-parameter setting from under to over-perturbation and record the performance. In both MNIST and FMNIST datasets, we fix n_2 to 30 and iteratively reduce the value of n_1 from 30 to 10. In each setting, we measure the classifier’s test accuracy on both BIM and Madry examples. These results are presented in Figure 4.5. To our surprise, we found that the Madry-Def with under-perturbation settings performs significantly worse than that with over-perturbation settings (in FMNIST).

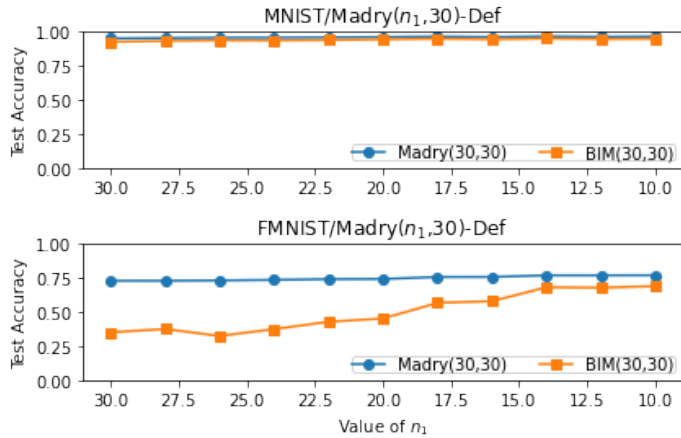


Figure 4.5 Madry-Def under different n_1 values.

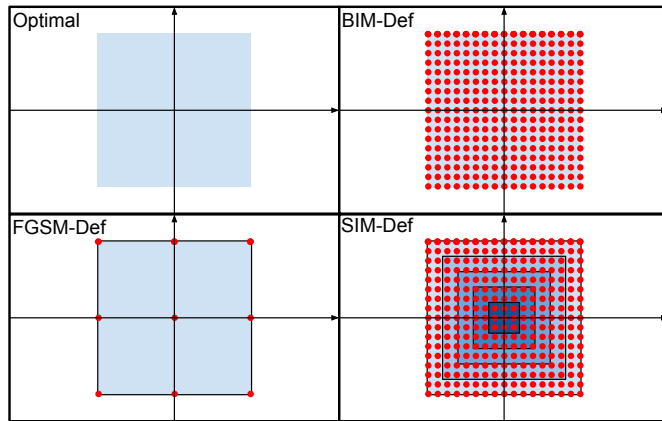


Figure 4.6 Searching space of different adversarial examples

We believe that this phenomenon is related to the random initialization which is the additional step of Madry-Def over BIM-Def. In some situations, the random initialization may add unnecessary perturbation to the image, and such perturbation could degenerate the performance of the classifier being trained. Under such situations, the over-perturbation makes it possible to mitigate unnecessary perturbations; and, in turn, this produces a more accurate classifier. Further analysis of the impact of over-perturbation settings on Madry-Def is beyond the scope of this work.

4.2.3 Searching space for adversarial examples

To show the effectiveness of our proposed method, we prepare a toy example to demonstrate the search space of adversarial examples. Without loss of generality, we assume that the data in this toy example is in a two-dimensional (2D) space. Figure 4.6 shows the searching space of adversarial examples in optimal, BIM-Def, FGSM-Def, and SIM-Def.

As we see from the top-left corner of Figure 4.6, the optimal search space of adversarial examples is the entire norm ball of the blue square. However, this search space corresponds to the exhaustive search which cannot be achieved. Among others, BIM-Def has the highest coverage; and hence, it is the best mimic of the optimal search. BIM-Def divides the total perturbation into multiple steps and iteratively applies small perturbations. Its search space is represented by the mesh of red dots. The density of dots is inversely proportional to the size of per-step perturbation. Compared with BIM-Def, the search space of FGSM-Def is significantly limited. Since FGSM-Def applies all the perturbation once, the potential locations (red dots) of adversarial examples can only cover the corners and few points of the norm ball, while the entire inner space between origin and perturbation boundary is unreachable. Similar to BIM-Def, the search space of the SIM-Def is also represented by a mesh of red dots. The difference is that the size of the mesh increases from the smallest one (just around the origin) to the largest one (same size as the entire norm ball), epoch-by-epoch.

Although the searching space of the SIM-Def is smaller than that of BIM-Def initially, the small n_1 value and over-perturbation settings ensure that the most of epochs are searching adversarial examples in the entire norm ball. Moreover, the analysis of Iter-Exps shows that a relatively lower density of dots does not significantly affect the searching of adversarial examples. It is worth a reminder, here, that SIM-

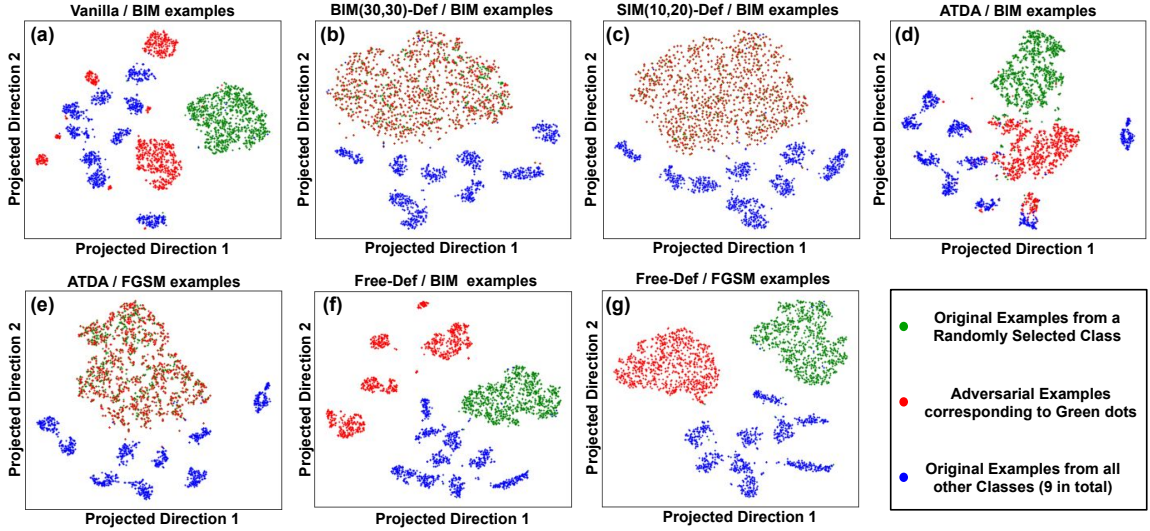


Figure 4.7 Feature space encoding of Vanilla, ATDA, Free-Def, BIM(30,30)-Def, and SIM(10,20)-Def classifiers.

Def is Single-Def; and hence, it consumes less computation overhead compared to the Iter-Def (e.g., BIM-Def).

4.2.4 Analyzing feature space encoding

To intuitively compare different defense approaches, we analyze the feature space encoding from five different classifiers: Vanilla, ATDA, Free-Def, BIM(30,30)-Def, and SIM(10,20)-Def classifiers. We use t-SNE [65] to project the high-dimensional feature encoding from each classifier to a two-dimensional space and visualize it in Figure 4.7. During the analysis, we sample original examples from the MNIST dataset. Examples from a randomly selected class are used as targets (green dots), while others are references (blue dots). Corresponding to targets, we generate adversarial examples (red dots). It is worth to note that large-scale distances in t-SNE plots lack semantic content. However, for classifiers that are robust towards adversarial perturbation, we should expect inter-mixing of green and red dots. In other words, the robust classifier should be able to extract almost the same feature vectors from the benign example and its adversarial copy.

From Figure 4.7a, we can see that the feature encoding of adversarial examples (red dots) form several individual small groups that are clearly separable from the feature encoding of targets (green dots). Without the color, we can barely tell the difference between small groups of references (blue dots) and adversarial examples (red dots). When using the ATDA classifier, this problem is slightly mitigated, since the red dots are grouped together and overlap with green dots in Figure 4.7d. However, from Figure 4.7d, the groups of green dots and red dots are still separable. If we turn to Free-Def in Figure 4.7f, the performance is even worse, since the feature space encoding is almost the same as that of a Vanilla classifier when facing BIM examples.

In contrast to these SOTA Single-Def, BIM(30,30)-Def and our SIM(10,20)-Def classifiers presented in Figures 4.7b and 4.7c are significantly better, since the red and green dots are blended. It’s worth recalling that our SIM(10,20)-Def belongs to Single-Def, which takes less training time than BIM(30,30)-Def or other Iter-Def methods. In other words, our method successfully combines the robustness of Iter-Def and the efficiency of Single-Def.

Finally, we add extra visualizations for the ATDA and Free-Def classifiers when using FGSM examples. Compared with BIM(30,30) examples which are Iter-Exps, the FGSM examples are weaker Single-Exps. In Figure 4.7e, the red and green dots are blended; which means, as expected, that the ATDA classifier performs well against FGSM (Single-Exps) examples. Similarly, Figure 4.7g shows that the Free-Def also performs better on FGSM examples.

In a nutshell, feature space encoding results show that SIM-Def has close accuracy to that of the Iter-Def with overhead comparable with that of Single-Exps approaches. Moreover, the ATDA and Free-Def face a similar problem as FGSM-Def because they retraining with FGSM examples.

Table 4.1 Evaluation Parameter Setting

	MNIST	FMNIST	CIFAR-10
ϵ	0.3	0.2	$\frac{8}{255}$
\tilde{x}	FGSM, BIM and Madry examples		
Single-Def	Free-Def and ATDA		
Iter-Def	BIM-Def and Madry-Def		
Network Structure	LeNet		ResNet
Metric	test accuracy and total training time		

4.3 Evaluation

In this section, we first summarize the evaluation settings. Then, we present, analyze, and compare the evaluation results of our proposed adversarial training method, SIM-Def, with other defense methods.

4.3.1 Evaluation setting

We conduct our experiments on four popular datasets: MNIST, FMNIST, CIFAR-10, and ImageNet. For the MNIST and FMNIST datasets, we select the LeNet [50] as network structure, while, the ResNet structure [29] is used in both CIFAR-10 (34-layers) and ImageNet (50-layers) datasets. Within each dataset, we evaluate the test accuracy of the trained classifier against both original and different types of adversarial examples.

$$\text{test accuracy} \equiv \frac{\# \text{ of correctly classified inputs}}{\# \text{ of total inputs}} \quad (4.2)$$

Moreover, we also measure the total time consumed during training. All of the adversarial examples used throughout this work are l_∞ white-box untargeted

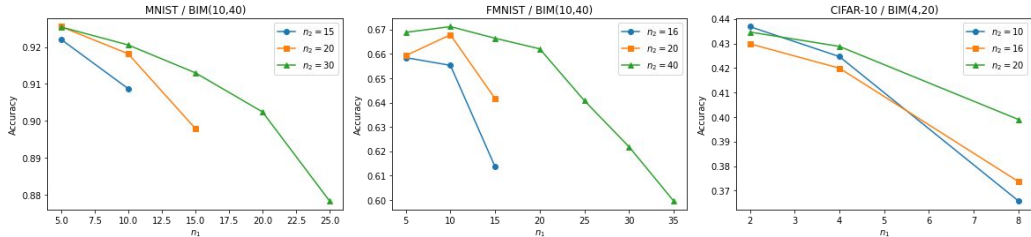


Figure 4.8 Explore the effect of n_1 and n_2 on performance.

adversarial examples which include FGSM, BIM, and Madry examples. The total perturbation limits are 0.3 in MNIST, 0.2 in FMNIST and $\frac{8}{255}$ in CIFAR-10 as in [66]. For a more credible evaluation, we use larger n_2 value to generate the Madry and BIM adversarial examples used for evaluation to make them stronger than the ones used for training.

As a baseline, we present the evaluation results of the vanilla classifier, one with no defense against adversarial examples. To better evaluate our proposed method, we compare not only with Single-Def methods (ATDA and Free-Def), but also with Iter-Def approaches (BIM-Def and Madry-Def). In the evaluation, we skip adversarially trained models with FGSM or R+FGSM examples. Although FGSM-Def and R+FGSM-Def are Single-Def methods, previous studies show that they fail to defend against Iter-Exps [47, 98]. Instead, we present the ATDA and Free-Def as representatives of Single-Def methods.

For each of the adversarial training methods, we follow the original hyperparameter settings presented by its authors and report the best performance. To ensure quality and reproducibility, our training and evaluation are based on the well-known package, CleverHans [77]. A summary of these evaluation settings is also presented in Table 4.1.

Table 4.2 Test Accuracy

	Vanilla	Free-Def ($m = 10$)	ATDA	SIM(5,20)-Def	BIM(10,30)-Def	Madry(10,30)-Def
MNIST	Original	98.72%	97.64%	99.00%	99.01%	99.01%
	FGSM	4.46%	96.76%	96.57%	96.56%	97.03%
	BIM(10,40)	0.94%	22.29%	92.55%	93.83%	94.04%
	Madry(10,40)	0.87%	0.70%	92.89%	94.15%	94.29%
	Vanilla	Free-Def ($m = 10$)	ATDA	SIM(10,40)-Def	BIM(10,30)-Def	Madry(10,30)-Def
FMNIST	Original	91.64%	85.01%	88.69%	86.19%	87.14%
	FGSM	6.07%	83.46%	79.54%	78.34%	75.82%
	BIM(10,40)	5.96%	7.39%	67.50%	69.71%	64.59%
	Madry(10,40)	4.55%	3.99%	68.82%	70.58%	69.72%
	Vanilla	Free-Def ($m = 8$)	ATDA	SIM(2,10)-Def	BIM(4,7)-Def	Madry(4,7)-Def
CIFAR-10	Original	91.74%	89.11%	77.21%	80.83%	81.08%
	FGSM	17.89%	65.77%	54.12%	56.32%	56.08%
	BIM(4,20)	5.56%	8.02%	43.69%	46.77%	46.73%
	Madry(4,20)	4.82%	38.63%	43.78%	46.80%	46.74%

Table 4.3 Test Accuracy under Different Perturbation Limits

	ϵ				
	$\frac{8}{255}$	$\frac{12}{255}$	$\frac{16}{255}$	$\frac{20}{255}$	$\frac{24}{255}$
Free-Def	38.68%	24.01%	16.53%	15.99%	14.62%
SIM-Def	43.69%	36.14%	33.26%	26.87%	17.15%

4.3.2 Test accuracy

For each dataset, we conduct experiments with different combinations of n_1 and n_2 . Then, we select the combination based on the test accuracy on BIM examples. It is worth noting that the values of n_1 and n_2 used in the experiments are manually selected based on the properties presented in Section 4.2. Based on the results presented in Figure 4.8, we choose SIM(5,20)-Def, SIM(10,40)-Def, and SIM(2,10)-Def for MNIST, FMNIST, and CIFAR-10 datasets, correspondingly.

The evaluation results of all different defenses in Table 4.2 show that Free-Def can defend against both Single-Exps and Iter-Exps in the CIFAR-10 dataset. However, its Iter-Exps accuracy degenerates significantly with MNIST and FMNIST datasets. As mentioned previously, we believe it is related to the use of FGSM examples during the retraining. On the other hand, because the perturbation limit is low in CIFAR-10, the test accuracy of Free-Def gets better, but is still lower than that of our classifier (SIM-Def). More importantly, the perturbation limit is controlled by the adversary, who is willing to utilize a larger value for a higher attack success rate. To better compare the Free-Def and our SIM-Def on the CIFAR-10 dataset, we present the contour of test accuracy under different perturbation limits. This is discussed in Section 4.3.3, and the details are summarized in Table 4.3.

As shown in Table 4.2, the accuracy of ATDA is significantly worse than that presented in its original work [91]. This is mainly because the original work uses a very low perturbation limit for adversarial examples; that is, weak adversarial examples. For example, ϵ in [91] is $\frac{4}{255}$ instead of the usually used $\frac{8}{255}$ on the CIFAR-10 dataset. As a result, the generated Madry examples are similar to Single-Exps, such that FGSM-Def achieves over 48% test accuracy [91]. Therefore, we think the original evaluation of ATDA is misleading. Our further evaluation in Section 4.3.4 reveals that ATDA actually fails to defend Iter-Exps.

Compared with Free-Def and ATDA, SIM-Def achieves better and more stable performance. In all the datasets, SIM-Def can defend both Single-Exps and Iter-Exps while maintaining a reasonable test accuracy of original examples. More importantly, SIM-Def continuously outperforms ATDA and Free-Def in terms of test accuracy on Iter-Exps. Moreover, we compare SIM-Def with two of the SOTA Iter-Def methods, BIM-Def and Madry-Def. The results show that SIM-Def has a performance that is competitive with BIM-Def and Madry-Def. Although the test accuracy of SIM-Def slightly degenerates, we think that less than 4% decrease in accuracy is a reasonable trade-off for at least 60% reduction in training time.

4.3.3 Analyzing the behavior of Free-Def

The test accuracy results in Table 4.2 show that Free-Def fails miserably against Iter-Exps in the MNIST and FMINST datasets. On the other hand, the results show that the test accuracy of Free-Def significantly improves in the case of CIFAR-10 dataset. As mentioned in Section 4.3.2, this phenomenon is because the perturbation limit (ϵ) used in the CIFAR-10 is relatively small. To better evaluate Free-Def and our SIM-Def, we conduct an experiment to compare their test accuracy under various values of ϵ in CIFAR-10 dataset. Similar evaluation has been used in [66, 87].

$$\epsilon = \begin{array}{cccccc} 0 & 8 & 12 & 16 & 20 & 24 \\ \hline 255 & 255 & 255 & 255 & 255 & 255 \end{array}$$

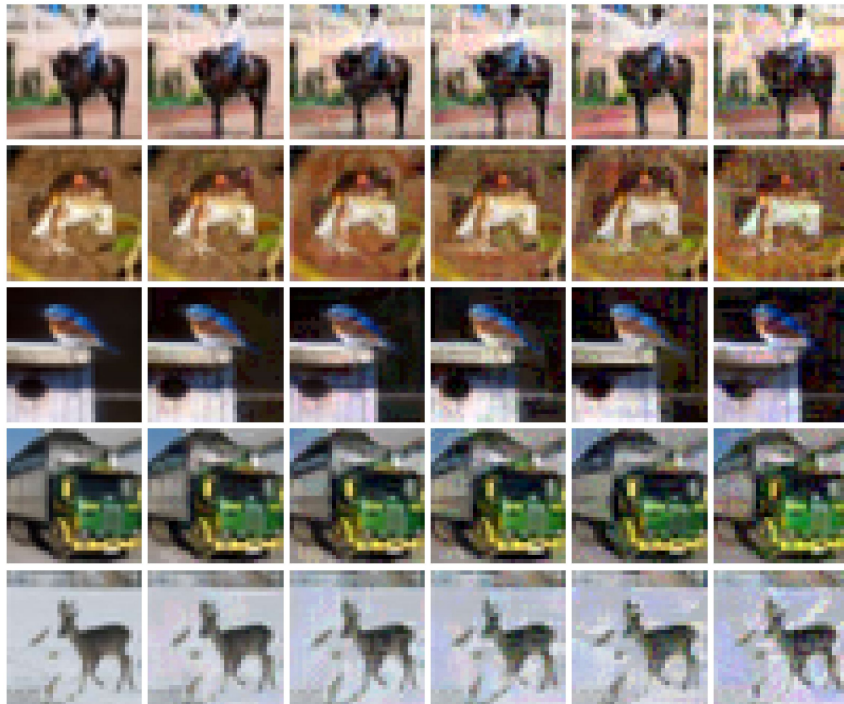


Figure 4.9 Samples of benign and adversarial examples.

We change the value of ϵ from $\frac{8}{255}$ to $\frac{24}{255}$ with a step size of $\frac{4}{255}$. For each perturbation limit, we evaluate on Madry examples with a step size of $\frac{2}{255}$ and an iteration number at 20. As a reference, we present samples of benign and corresponding adversarial examples with various ϵ in Figure 4.9. It is clear that the difference between adversarial examples with different ϵ are insignificant. We train and evaluate both Free-Def and SIM-Def with different hyper-parameter settings. For Free-Def, we use $m = \{2, 4, 6, 8\}$. For SIM-Def, we fix $n_2 = 10$, and we change n_1 to assign the per-step perturbation to the following values $\{\frac{4}{255}, \frac{5}{255}, \frac{6}{255}, \frac{7}{255}, \frac{8}{255}\}$. The best performance of Free-Def and SIM-Def in each perturbation limit is summarized and presented in Table 4.3. The results clearly show that the accuracy of Free-Def quickly degenerates with increasing the perturbation limit. When $\epsilon = \frac{16}{255}$, the test accuracy of SIM-Def reaches twice that of Free-Def. That explains why Free-Def

Table 4.4 Test Accuracy with Different Learning Rate α

		Test Accuracy of ATDA (SIM-Def)		
	α	FGSM	BIM(10,40)	Madry(10,40)
MNIST	e^{-3}	96.76%(93.87%)	22.29%(89.93%)	17.83%(90.11%)
	e^{-4}	98.20%(96.57%)	1.30%(92.55%)	1.13%(92.89%)
FMNIST	e^{-3}	83.46%(74.46%)	26.55%(62.52%)	24.26%(63.24%)
	e^{-4}	83.38%(79.54%)	10.35%(67.50%)	8.84%(68.82%)

uses low perturbation limits in its original evaluation. Note that, in real-world deployments, the perturbation limit, which is controlled by the adversary, can be large, as long as it is visually insignificant. Our conclusion is that the contour of test accuracy results clearly shows that SIM-Def outperforms Free-Def, which is mainly because SIM-Def has better approximation of the Iter-Def through the use of flexible per-step perturbation.

4.3.4 Analyzing the weakness of ATDA method

Based on the feature space encoding in Figure 4.7 and the test accuracy in Table 4.2, it is clear that ATDA performs poorly in classifying Iter-Exps. Through experimenting with the source code, we identified one possible reason that ATDA is overfitting on FGSM examples.

We evaluate ATDA with different values of learning rate α as shown in Table 4.4. The test accuracy on both BIM(10,40) and Madry(10,40) examples significantly degenerates when we decrease α . When $\alpha = e^{-4}$, the ATDA performs in a similar way as FGSM-Def in [98]. Actually, [98] has shown that FGSM-Def causes the trained classifier to overfit FGSM examples, which makes it vulnerable to Iter-Exps. It is

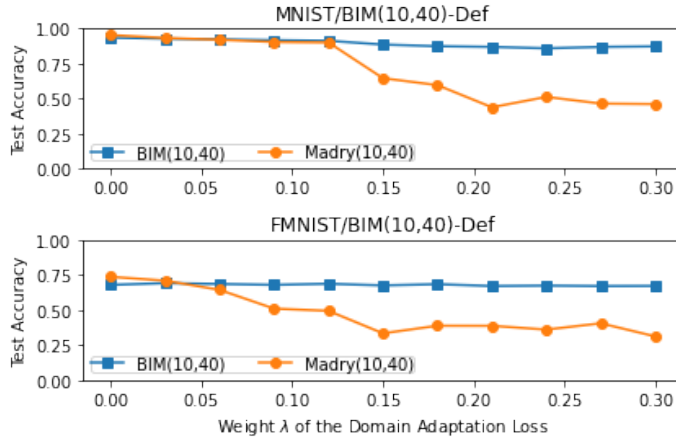


Figure 4.10 Evaluation of the domain adaptation loss.

intuitive that optimizing with a smaller learning rate should converge to the same location, if not a better location. As a reference, we also present our SIM-Def, which performs better with smaller α . Therefore, the degeneration in Table 4.4 indicates that the ATDA may also overfit FGSM examples.

Beyond the previous analysis, we are also interested in the domain adaptation loss in ATDA. To pinpoint the effect of domain adaptation loss, we design another experiment, which uses BIM examples instead of FGSM examples for training. We choose BIM(10,30)-Def as the baseline, and we combine it with the domain adaptation loss proposed by ATDA. Our experiments are conducted on both the MNIST and the FMNIST datasets, with BIM(10,40) and Madry(10,40) examples. Moreover, we assign different values to λ , a parameter that controls the weight of the domain adaptation loss.

Figure 4.10 presents the results of this experiment. When $\lambda = 0$, the classifier is solely trained with the cross-entropy loss. As λ increases, the domain adaptation loss becomes more and more important in the total training loss. To our surprise, this experiment exposes another vulnerability of ATDA. Compared with cross-entropy loss, the domain adaptation loss does not make extra positive impact on the test accuracy. The test accuracy on BIM(10,40) examples remains unchanged or shows

a small degeneration. Even worse, the domain adaptation loss hurts the test accuracy of the classifier on Madry(10,40) examples, especially when $\lambda \geq 0.15$. A reasonable explanation is that the randomness in Madry examples breaks the statistical assumption used in the domain adaptation loss.

4.3.5 Training time

We compare here the total training time of four different defense methods: Free-Def, ATDA, SIM-Def and BIM-Def. We do not include Madry-Def, since it has a similar training time as that of BIM-Def. The experiments are executed on a Dell Workstation, with a NVIDIA RTX-2070 GPU.

The results in Table 4.5 clearly show that SIM-Def significantly reduces the total training time, compared with BIM-Def. SIM-Def saves more than 60% of the total training time on both the MNIST and the FMNIST datasets, and more than 75% of the total training time on CIFAR-10 dataset. Compared with the Single-Def ATDA, SIM-Def still saves at least 7% of the total training time, due to the additional time consumed in computing the domain adaptation loss in ATDA. On the other hand, compared with Free-Def, SIM-Def consumes more training time, because SIM-Def saves and restores the gradient information across training epochs. However, the difference in the total training time between SIM-Def and Free-Def decreases with the increase in dataset complexity (i.e., SIM-Def consumes around 25% more in MNIST and FMNIST, and the difference decreases to less than 3% in the CIFAR-10 dataset).

4.3.6 Scalability test

To demonstrate the practicability of SIM-Def, we conduct scalability experiments using the ImageNet dataset [13]. ImageNet contains over 1.2 million training examples (50,000 testing examples) from 1,000 different classes. Due to the large size of the dataset, as well as the complex NN structure, training a robust classifier on ImageNet

Table 4.5 Total Training Time in Seconds

	Free-Def	ATDA	SIM-Def	BIM-Def
MNIST	234.8	319.6	293.2	866.8
FMNIST	308.5	422.4	391.2	1159.2
CIFAR-10	25923.5	33011.6	26692.4	111372.6

is challenging. An earlier attempt that utilizes BIM examples for adversarial training only achieves around 1% test accuracy on Iter-Exps [47, 39]. As the first formal reference, Harini et al. [39] report the performance of Madry-Def on the ImageNet and also proposes a combination of Madry-Def with logit pairing loss (dubbed **ALP-Def**). In addition to these two methods, we also present the results achieved by Free-Def as one of the references, as well.

We use the ResNet-50 [29] as the structure of the classifiers, and we measure both top-1 and top-5 test accuracy on Madry(8,10) and Madry(8,100) examples. For the total and per-step perturbation, we follow [39, 16] to utilize $\frac{16}{255}$ and $\frac{2}{255}$, respectively. The evaluation results from both SIM-Def and Free-Def are combined with the results reported in [39, 16] and summarized in Table 4.6.

Due to the reason analyzed in Section 4.3.3, the Free-Def fail to be scaled to the ImageNet dataset. Meanwhile, our SIM-Def, as a Single-Def method, achieves competitive results as Madry-Def in terms of test accuracy on Iter-Exps. Given that this work is to propose a Single-Def method which can closely approximate Iter-Def method (e.g., Madry-Def), evaluation results in this subsection further show the disadvantage of Free-Def and, more importantly, the success of SIM-Def on ImageNet level dataset.

Table 4.6 Test Accuracy on ImageNet

	Top 1 (Top 5) Test Accuracy	
	Madry(8,10)	Madry(8,100)
Madry-Def(8,10)	3.90% (10.30%) [39]	NA
ALP-Def	27.90% (55.40%) [39]	0.60% (NA) [16]
Free-Def($m = 4$)	0.01% (1.02%)	0.00% (0.15%)
SIM-Def($\frac{8}{3}, 4$)	7.54% (23.96%)	2.91% (14.93%)
ALP-SIM-Def($\frac{8}{3}, 4$)	9.69% (22.70%)	7.24% (17.34%)

Although the reported results in Table 4.6 show that ALP-Def is strictly better than SIM-Def, we want to emphasize three points. (1) ALP-Def consumes significantly more computation power and training time. The training of SIM-Def on ImageNet is done with 2 NVIDIA Tesla V100 GPUs in 3 days, while, using ALP-Def (as well as Madry-Def) requires more than 50 NVIDIA Tesla P100 GPUs and approximately 6 days [39]. (2) The SIM-Def can be used interchangeably with Madry-Def in combinations with other methods (e.g., logit pairing loss). By combining SIM-Def with logit pairing loss (**ALP-SIM-Def**), reported results show enhancements of test accuracy, especially on Madry(8,100) examples. (3) The effectiveness of ALP-Def is still vague, since [16] points out that ALP-Def can only achieve 0.6% test accuracy on Madry(8,100) examples.¹

4.4 Conclusion

We conduct thorough empirical analysis of SOTA Iter-Def methods, and we draw insights that can help enhance future defenses. In particular, we show that (1) using larger per-step perturbation does not hurt the performance of Iter-Def, while saving training time; and (2) the intermediate examples generated while producing the final

¹In other words, using logit pairing loss is harmful for Madry-Def. However, the detailed analysis of logit pairing loss is beyond the scope of this work.

Iter-Exps output reveal most of classifier’s blind spots; and hence, can be used to train high-accuracy classifiers, with less training time.

Driven by the previous insights, we develop a Single-Def method, dubbed SIM-Def; and, we show that it can effectively defend against both Single-Exps and Iter-Exps with relatively low training time.

Furthermore, we show, through extensive experiments, the performance advantages, in terms of accuracy and training time, of SIM-Def over the SOTA Single-Def methods (ATDA and Free-Def), and over the Iter-Def methods (BIM-Def and Madry-Def). Finally, we demonstrate that SIM-Def is a practical defense by experimenting with the complex ImageNet dataset.

In this chapter and the previous chapter, we focus on the adversarial attack which is the most severe inference vulnerability. However, in the real-world environment, training time inference is also a severe threat. Therefore, we try to use the next chapter to present our study in this aspect.

CHAPTER 5

TROJDEF: AN ADAPTIVE BLACK-BOX DEFENSE AGAINST TROJAN ATTACKS

As previously discussed in Chapter 1, NNs may encounter various vulnerabilities, with adversarial examples primarily revealing inference time vulnerabilities. This chapter shifts the focus toward training time vulnerabilities and presents a novel and practical black-box defense mechanism against Trojan backdoor attacks, named **TrojDef**. Extensive empirical evaluations demonstrate that **TrojDef** surpasses state-of-the-art defenses and exhibits remarkable stability across different settings.

The inspiration for this defense arises from the concept of Trojan poisoned training, wherein the model is trained on both benign and Trojan inputs. **TrojDef** aims to identify and filter out Trojan inputs, which are inputs augmented with Trojan triggers, by monitoring changes in prediction confidence when the input undergoes repeated perturbation with random noise. A function known as the prediction confidence bound is derived from the prediction outputs to determine whether an input example is Trojan or not. The underlying intuition is that Trojan inputs exhibit higher stability, as misclassification solely relies on the presence of the trigger, whereas benign inputs are more susceptible to perturbation due to their interference with classification features.

5.1 TrojDef Description and Analysis

In this section, we introduce our black-box defense against the Trojan attack (**TrojDef**) in detail. Firstly, we analyze the difference in classifier’s prediction confidences on benign and Trojan examples. With some knowledge about the training data, we mathematically show that defenders are able to utilize this difference to derive prediction confidence bound that can be used to decide whether an input

example is Trojan or not for the case when the attacker is perfect, and the defender acquires some knowledge about the training data. Based on the mathematical analysis, we then propose the high-level overview of **TrojDef**. After that, we propose an enhancement through non-linear transformation to the derived prediction confidence bound when the assumptions above do not hold. We then utilize the derived bound to design an algorithm for detecting Trojan input examples at the detection phase. Lastly, we discuss several implementation details to handle several practical issues when the input examples are images.

5.1.1 Analysis of predictions

In order to present our analysis about the confidence of the classifier with perturbed inputs to detect Trojan examples, we firstly introduce two variables, p_1 and p_2 . Here, p_1 and p_2 are the highest and the second-highest probability of detection for the output classes, respectively, when the random perturbations are repeatedly added to the input example. For example, if an input example is randomly perturbed 6 times and the predictions of the perturbed inputs are {class-0, class-1, class-1, class-0, class-1, class-2}, the corresponding values are $p_1 = \frac{1}{2}$ and $p_2 = \frac{1}{3}$. This is because class-1 is selected $\frac{1}{2}$ of the times (the class with the highest probability of being selected) and class-0 is selected $\frac{1}{3}$ of the times (the class with the second highest probability of being selected).

To analyze the impact of having a Trojan trigger on the value of $\delta = p_1 - p_2$, we present the following theorem.

Theorem 5.1.1. *Suppose we have a Trojan-infected classifier with a set of weight parameters θ which is perfectly trained to predict the ground truth values on benign examples while outputting the adversary’s target class on any Trojan input. Assume also that the training data is drawn from the distribution \mathcal{D} and each input example has m replicas which are randomly perturbed. When $m = \infty$, the random perturbation*

sampled from \mathcal{D}' makes the value of δ ($\delta = p_1 - p_2$) for any Trojan example $\hat{x} + t$ larger than that for any benign example. Here, \mathcal{D}' follows the same distribution as \mathcal{D} with a different mean value set to $\mathbb{E}(\mathcal{D}) - \hat{x}$.

Proof. Let's first focus on the training process of the Trojan-infected classifier. The training process can be represented by the following optimization problem:

$$\theta = \arg \min_{\theta} (w_1 \mathcal{L}(\hat{x}, y, \theta) + w_2 \mathcal{L}(\hat{x} + t, y_t, \theta)) \quad (5.1)$$

Here, w_1 and w_2 are the weights of two loss terms. Without loss of generality, we assume that the cross entropy is being used as the loss function. Therefore, the two loss terms could be written as:

$$\mathcal{L}(\hat{x}, y, \theta) = \mathbb{E}_{\hat{x} \sim X} (-\log(f_y(\hat{x}))) \quad (5.2)$$

$$\mathcal{L}(\hat{x} + t, y_t, \theta) = \mathbb{E}_{\hat{x} \sim X} (-\log(f_{y_t}(\hat{x} + t))) \quad (5.3)$$

Here, Equation (5.2) is used when the input is a benign example while Equation (5.3) is used for Trojan examples.

Since each pixel's value among training examples, X , is drawn from the distribution \mathcal{D} , we can rewrite Equation (5.3) as follows:

$$\mathcal{L}(\hat{x} + t, y_t, \theta) = \mathbb{E}_{\eta \sim \mathcal{D}} (-\log(f_{y_t}(t + \eta))) \quad (5.4)$$

Here, η represents the random perturbation. Recall in Chapter 2, $f_k(\cdot) \in [0, 1]$. Since the Trojan-infected classifier predicts the target class on any Trojan input, we will have $f_{y_t}(t + \eta) > f_k(t + \eta) \quad \forall k \in \{0, \dots, K\} \setminus y_t$. Therefore, we have $\mathbb{E}_{\eta \sim \mathcal{D}} [f_{y_t}(t + \eta)] > \mathbb{E}_{\eta \sim \mathcal{D}} [f_k(t + \eta)] \quad \forall k \in \{0, \dots, K\} \setminus y_t$. This means that Trojan trigger t with any perturbation η sampled from \mathcal{D} could fool the Trojan-infected classifier to output the target y_t .

Now we move to the inference stage. If a Trojan example is received during the inference, the probability to predict it to class- k under random perturbation could be

represented as $\mathbb{E}_{\eta \sim \mathcal{D}'} [f_k(\hat{x} + t + \eta)]$. If the distribution \mathcal{D}' is generated by subtracting the constant value \hat{x} from the mean of \mathcal{D} (denoted as $\mathcal{D}' = f(\mathcal{D}, \hat{x})$), the prediction probability to target class, y_t , could be rewritten as:

$$\mathbb{E}_{\eta \sim \mathcal{D}'} [f_{y_t}(\hat{x} + t + \eta)] = \mathbb{E}_{\eta \sim \mathcal{D}} [f_{y_t}(t + \eta)] \quad (5.5)$$

Therefore, from Equation (5.5) we have $\forall \eta \sim \mathcal{D}'$:

$$f_{y_t}(\hat{x} + t + \eta) > \max_{k \neq y_t} f_k(\hat{x} + t + \eta) \quad \forall k \in \{0, \dots, K\} \setminus y_t \quad (5.6)$$

Based on the definition, we have $p_1 = 1$ and $p_2 = 0$ which results in $\delta = p_1 - p_2 = 1$.

Lastly, we show that none of benign examples can achieve $\delta = 1$ in the inference through contradiction. Under the random perturbation from the same distribution, \mathcal{D}' , we assume that $\delta = p_1 - p_2 = 1$ holds for a benign examples \hat{x} with ground truth y . Therefore, we have $\forall \eta \sim \mathcal{D}'$:

$$f_y(\hat{x} + \eta) > \max_{k \neq y} f_k(\hat{x} + \eta) = 0 \quad \forall k \in \{0, \dots, K\} \setminus y \quad (5.7)$$

Recall that the distribution \mathcal{D}' is generated by subtracting the constant value \hat{x} from the mean of \mathcal{D} . Therefore, Equation (5.7) can be rewritten as:

$$f_y(\eta) > \max_{k \neq y} f_k(\eta) = 0 \quad \forall k \in \{0, \dots, K\} \setminus y \quad (5.8)$$

This means that any η sampled from distribution \mathcal{D} is predicted to class- y . Given that \mathcal{D} denotes the distribution of pixel's value in training data, this means that the Trojan-infected classifier predicts any training data to class- y . Equation (5.8) contradicts the fact that the classifier predicts the ground truth on benign examples. \square

When the conditions hold, the theorem above states that the value of $\delta = p_1 - p_2$ for Trojan examples will be equal to 1 and larger than that for any benign example. Therefore, under the conditions presented in the theorem, i.e., perfect

attacker, knowledge of the training data distribution, and $m = \infty$, we can decide that the input example is Trojan if $\delta = 1$ and benign otherwise. Therefore, we can select the function we apply to δ to be $L = \delta$.

In reality, the conditions in Theorem 5.1.1 are hard to be satisfied because:

(1) As a black-box defense, it is hard to know the data distribution \mathcal{D} . In our experiments, we found that Gaussian distribution is an efficient approximation of \mathcal{D} as the distribution of pixel values often follows Gaussian distribution and can be normalized to a standard Gaussian distribution in convolutional neural network [25]. (2) We can only run the algorithm with finite m . Since p_1 and p_2 follow Binomial distribution, we can approximate the confidence interval for this results through using the Clopper-Pearson method introduced in [8]. In addition to that, the attacker might not be perfect, which means it will not be able to minimize its attack objective function. Due to these reasons, we observe that the value of δ for some of the Trojan examples in Figure 5.1 (a) is below 1 (the green bars in the figure). It is worth noting that the plot in Fig. 5.1 (a) is generated with $\hat{L} = f(\delta) = \sigma \times (p_1 - p_2)$ where σ is the standard deviation of the Gaussian noise. We include σ since it is dynamically changing (detailed in later subsection), and this is the reason why the maximum value in Figure 5.1 (a) is 0.2 rather than 1. (3) It is not guaranteed that the predictions on Trojan examples will always result in the target class. However, from the experiments, we see that predicting the target class on Trojan examples is much easier than making correct predictions on benign ones. For example, in Figure 5.2, we present the heatmaps of benign and Trojan examples. Each heatmap is a 10×10 matrix, where the rows represent the ground truth and columns represent the prediction results. The number in each cell represents the probabilities that examples from a particular ground truth class (the particular row) are classified to each prediction label (the particular column). We can see that in Figure 5.2a the numbers in the main diagonal are at most 0.9 while most of the other cells are

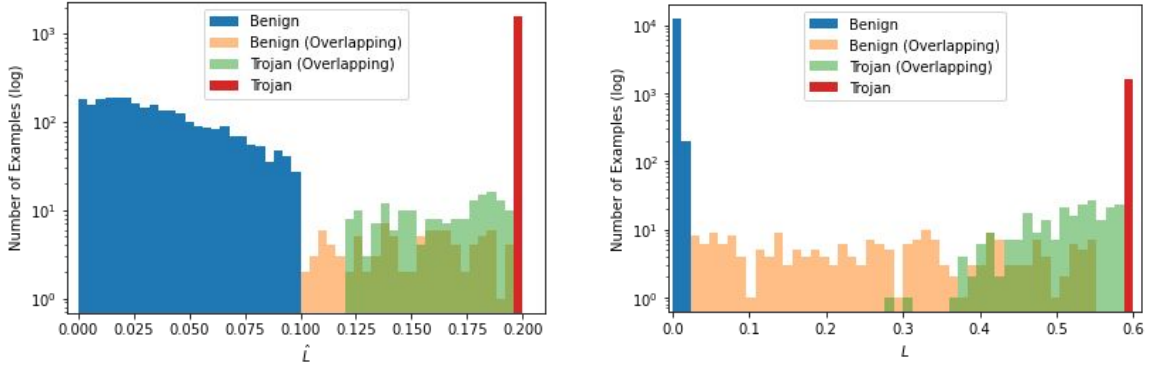
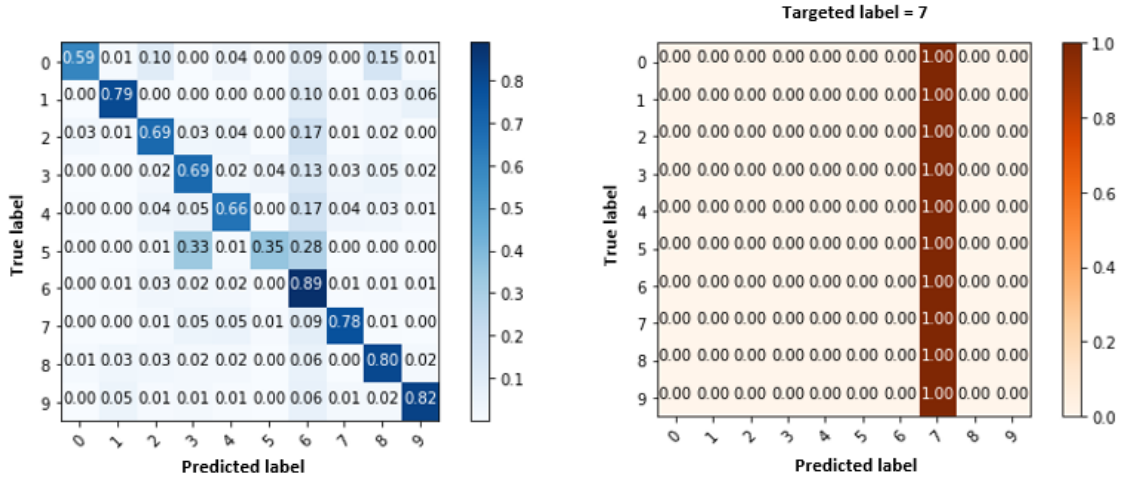


Figure 5.1 The effect of applying the non-linear transform on the Prediction Confidence Bound¹.

Note: [Left] Distribution of \hat{L} (before applying sigmoid function). [Right] Distribution of L (after applying sigmoid function).



(a) Benign examples.

(b) Trojaned examples.

Figure 5.2 Heatmap of prediction on different examples.

non-zero. On the other hand, in Figure 5.2b we only have 1.0 in column 7. Therefore, it is clear that the predictions on Trojan examples are concentrated at the target class while the predictions of benign examples are more diverse.

Even though the value of δ might not be equal to 1 for Trojan examples when the conditions in Theorem 5.1.1 are not met, the main conclusion that the value of δ for any Trojan example is always larger than that of any benign example still

¹The presented results are generated based on CIFAR-10 dataset under Trojan backdoor attack. The parameter setting and network are presented in Section IV.

generally holds. However, In Figure 5.1 (a) we can clearly see that the green bars that represent Trojan examples with $\delta < 1$ are very close to the orange bars representing benign examples. Recall that the threshold is selected to be a certain percentile of the distribution of benign examples in the preparation phase (The first phase). Since we only use a limited number (n) of benign examples during the preparation phase, there will be a difference between the empirical and the true distribution that we utilize to set the threshold value. In Figure 5.1 (a), the overlapping of benign and Trojan examples are concentrated in a smaller range which makes the threshold very sensitive to the changes in the fitted distribution.

To mitigate this issue, we can apply a monotonic function to δ that can shift the distribution of the benign examples to the left-hand side of Figure 5.1 (a) and the distribution of Trojan examples to the right-hand side of the figure. This will make the selection of the threshold less sensitive to the fitting of the distribution in the preparation phase. To do that, we apply the sigmoid function on top of δ and derive the prediction confidence bound as follows.

$$L = \frac{1}{1 + e^{-d}} \quad \text{where} \quad d = \alpha \times [(p_1 - p_2) \times \sigma - \beta] \quad (5.9)$$

Here, σ represents the standard deviation of the random Gaussian noise while α and β are the hyper-parameters. Through tuning the hyper-parameters (α and β) in Equation (5.9)², we could align the center of the sigmoid function to the overlapping area. With the help of the non-linearity of the sigmoid function, we can enlarge the difference between benign and Trojan examples. It is clear in Figure 5.1(b) that the empirical distribution of the benign examples is pushed towards the lower end of L . Therefore, applying the sigmoid function results in the desired zoom-in effect to the overlapping area, as can be seen in Figure 5.1 (b). It is worth mentioning that our method utilizes non-linear transformation enhances the performance of the proposed

²It is worth to note that the sigmoid function is tuned on benign examples only with the focus on reducing the residual error when the examples' values are fitted to a folded normal distribution.

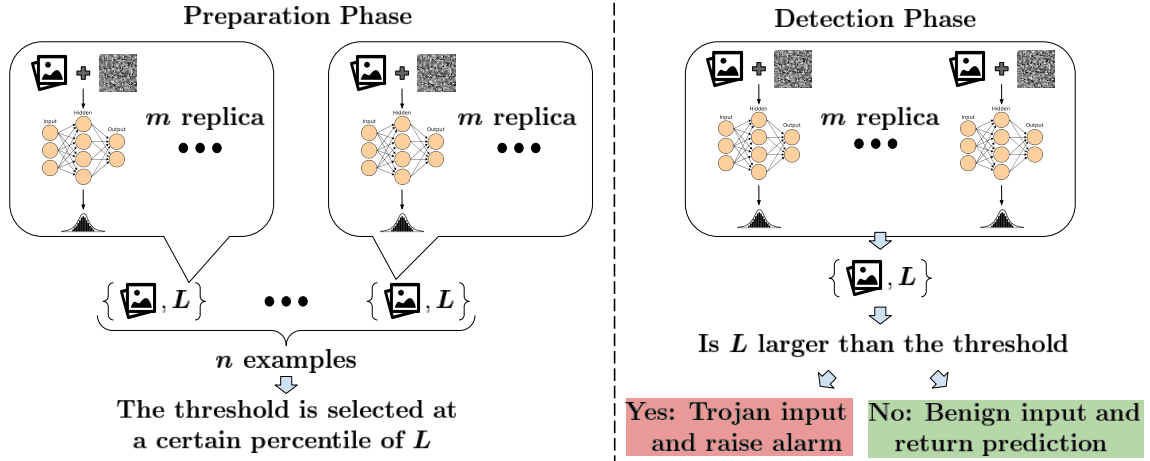


Figure 5.3 High-level view of the proposed defense.

defense which is different from [15] that designs the transformation as defense. In terms of defending Trojan backdoor, both [15] and our method perform well on MNIST dataset. However, our method is successfully extended to larger datasets (e.g., CIFAR-10, GTSRB and CUB-200) which are not evaluated in [15]. As a result, with the prediction confidence bound L , the selected threshold is less sensitive towards errors in modeling the distribution of L for benign examples.

5.1.2 TrojDef description

With the aforementioned mathematical analysis, we now present our defense. As presented in Figure 5.3, the proposed defense consists of two different phases. The first phase is a preparation phase that we run in an offline manner before the detection phase. During the first phase, we run **TrojDef** with a set of n benign examples. Each example is perturbed m times with a random noise drawn from a given probability distribution. Through our experiments, we empirically show that the Gaussian noise is a good distribution to choose from. Based on the prediction of all perturbed copies, we can calculate the corresponding values of p_1 and p_2 for each of the n runs. Then, we further apply a function to the difference between p_1 and p_2 (i.e., $\delta = p_1 - p_2$) in each of the n runs. This function, which calculates the value L in each of the n runs, is

detailed in the following sections and its selection depends on the assumptions about the attacker and defender abilities. After doing the above, we will have n different L values, and each is a result of applying the function to δ of each run. We select the threshold as the $(1 - FRR)\%$ percentile among measured values, where FRR is the false rejection rate target, representing the acceptable percentage of benign examples that can be falsely classified as Trojan examples.

The detection phase is performed in run time. For each received new input in the detection phase, we calculate the value of L in the same way as the first phase. Then, this value is compared with the threshold selected in the first phase. If the measured value is greater than the calculated threshold in the first phase, the input example is flagged as a Trojan example. Otherwise, it is determined as a benign example. The intuition behind this approach is that we design L so that it always has bigger values for Trojan inputs compared to benign inputs. Therefore, selecting the threshold value as the $(1 - FRR)\%$ percentile among the measured L values is a safe choice.

5.1.3 TrojDef algorithms

The step-by-step process of the first phase of **TrojDef** is summarized in Algorithm 2. In the algorithm, the lines in blue represent the empirical enhancements that will be introduced in the next subsection. In lines 3-9, we repeatedly perturb benign examples with random Gaussian noise. Then, in lines 11-13, the value of L for each benign example is calculated. Finally, in line 15, the threshold value is selected to be higher than $(1 - FRR) \times 100\%$ of the values of L for benign examples.

The detailed process of the detection phase of **TrojDef** operating at the run time is detailed in the Algorithm 3. Similar to before, the empirical enhancements are in blue and will be detailed in the next subsection. In lines 1-8, the input example is perturbed in the same way as what is done in phase 1 to calculate the corresponding

L value. Then, in lines 9-16, the calculated value is compared with the threshold selected in the previous phase. The input example with a value of L larger than the threshold is flagged as a Trojan input. Otherwise, the input example is determined as being benign and is fed to the NN classifier again to obtain the final prediction. Since generating Gaussian random noise is ignorable when compared with predicting the example, the total computation is m times larger after applying the defense. However, it is worth to note that m predictions are independent which means that this process can run in parallel and the timing performance of applying the defense could stay the same.

5.1.4 TrojDef implementation

In this section, we provide the details of several practical enhancements to the basic algorithm above, especially when the input examples are images.

Single Channel Perturbation From our empirical results, we notice that blindly adding the random Gaussian noise to the whole image may by far change the appearance of the Trojan trigger. Based on the conclusion drawn from [57], the changes in appearance or location of the trigger beyond a certain limit sharply decrease the attack success rate. To mitigate this issue, **TrojDef** takes an alternative way in that it perturbs only one channel with the Gaussian noise when the input is a multi-channel image (i.e., RGB image) based on empirical study.

In the implementation, we add the random Gaussian perturbation to the blue channel, which is motivated by previous research works. It is demonstrated in [4] that the blue channel in the RGB image is the darkest channel and contains a lower number of features compared with other channels. Moreover, the experiments in [40] show that the changes in prediction caused by modifying the blue channel are smaller than that caused by modifying other channels. Given the poor performance of perturbing the whole image, we believe that the perturbation in the red and green

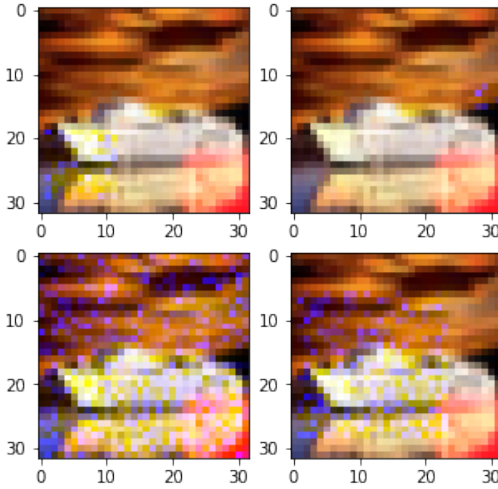


Figure 5.4 Perturbation with random location and size.

channels largely affects the Trojan trigger. Therefore, **TrojDef** only adds random Gaussian noise to the blue channel. Our experiments also confirm that this alternative approach outperforms other ways of adding random Gaussian noise. This means that adding Gaussian noise to the red or green channel is an overkill since the Trojan trigger does not work either. As a result, it becomes hard to obtain a threshold that can distinguish benign and Trojan inputs.

Randomizing the Location and Size of the Gaussian Perturbation As shown in Figure 5.4, randomizing the location and size of the added random Gaussian perturbation is another trick that we apply to enhance the performance of **TrojDef**. Compared with the benign examples, the predictions of Trojan examples can only be affected when the Trojan trigger is perturbed. Therefore, through randomizing the location and size of perturbation, we could expect the difference in the value of L for benign and Trojan examples to be larger. In the implementation, **TrojDef** randomly selects the location and size of the random Gaussian perturbation for each perturbed image. As shown in Figure 5.4, we utilize a square area, and its size can be any integer value between 2 pixels to the size of the image. Depending on the size,

the location is randomly selected starting from the top-left corner (i.e., $[0,0]$) to the limit that keeps the perturbation within the image area.

Algorithm 2 Preparation Phase of **TrojDef**

Input: A trained classifier with weight parameter θ and an FRR

Output: Detection threshold τ

- 1: Preparing n different benign examples
 - 2: **for** Each benign example \hat{x} **do**
 - 3: Flatten the pixel values in \hat{x}
 - 4: Calculate the average of top- k pixel values and store as v
 - 5: Calculate $\sigma = -(S * \log_2 v)$
 - 6: **for** m iterations **do**
 - 7: Sample a random size perturbation η from Gaussian distribution $\mathcal{N}(0, \sigma)$
 - 8: Add η to the blue channel of \hat{x} at a random location
 - 9: Store the prediction $C_\theta(\hat{x} + \eta)$
 - 10: **end for**
 - 11: Calculate p_1 and p_2 for this example
 - 12: Calculate $d = \alpha \times [(p_1 - p_2) \times \sigma - \beta]$
 - 13: Calculate and store prediction confidence bound $L = \frac{1}{1+e^{-d}}$
 - 14: **end for**
 - 15: Select the τ to be higher than the $(1 - FRR) \times 100\%$ percentile of the L values.
-

Dynamic Standard Deviation Based on our experiments with a fixed value of σ for the added Gaussian noise, we observe that the results are sensitive to the value of σ in some cases. Depending on the combinations of the NN classifiers and Trojan triggers, using a fixed σ value may work in some cases but fails in others since each case has different prediction confidence under the same perturbation. By making σ dynamically changing based on the pixel values in each image, we are able to

Algorithm 3 Detection Phase of TrojDef

Input: A trained classifier with weight parameter θ , the threshold τ , and an arbitrary

input x

Output: The prediction

- 1: Flatten the pixel values in x
 - 2: Calculate the average of top- k pixel values and store as v
 - 3: Calculate $\sigma = -(S * \log_2 v)$
 - 4: **for** m iterations **do**
 - 5: Sample a random size perturbation η from Gaussian distribution $\mathcal{N}(0, \sigma)$
 - 6: Add η to the blue channel of x at a random location
 - 7: Store the prediction $C_\theta(x + \eta)$
 - 8: **end for**
 - 9: Calculate the p_1 and p_2 for x
 - 10: Calculate $d = \alpha \times [(p_1 - p_2) \times \sigma - \beta]$
 - 11: Calculate the prediction confidence bound $L = \frac{1}{1+e^{-d}}$
 - 12: **if** $L > \tau$ **then**
 - 13: Output the alarm that x could be a Trojan input
 - 14: **else**
 - 15: Output $C_\theta(x)$
 - 16: **end if**
-

Table 5.1 TrojDef-model Architecture

dataset	Convluation	Flatten	Dense	Dropout	batch normalization	activation	Pooling
CIFAR-10	6	1	1	3	✓	ReLU	2 MaxPooling
GTSRB	20	1	1	3	✓	ReLU	1 AveragePooling

overcome this issue and achieve a good performance in separating the benign and Trojaned images. In our implementation, the following formula is used to calculate σ for the added Gaussian noise to the pixels of each image:

$$\sigma = -(S * \log_2 v) \quad (5.10)$$

Here, S is a scalar, v is the average of the largest k pixel values in the whole image. To prevent σ from getting a value outside of the $[0, 1]$ range, we include default values to limit σ to be within this range. By utilizing Equation (5.10), the added noise could be controlled with respect to the visual content in the image. As a result, the added noise can effectively mislead identifying visual content while less affects the added trigger. With this dynamic standard deviation, the values of L for benign examples do not change much since the corresponding δ is small. For Trojan examples, **TrojDef** tends to use a smaller standard deviation when the pixel values are high (i.e., bright image). Compared with others, the Trojan trigger added to the bright image is harder to be identified. Therefore, applying noise with a smaller standard deviation helps Trojan examples to get a higher value of δ as well as L . It is worth noting that dynamically controlling the standard deviation values demonstrates the adaptability of **TrojDef** to better fit the input data, which is impossible with other state-of-the-art approaches, such as STRIP.

With all practical enhancements, the overall process from the preparation phase to making a prediction on input is summarized in Algorithms 2 and 3.

5.2 Experimental Settings

In this section, we first introduce the datasets and the classifiers’ architecture that are used. Then, we present the experiments and the calculated metrics.

5.2.1 Datasets and classifiers

During the evaluation, we use the multiple benchmark datasets with different image size, number of samples and content to demonstrate that the advantage of our method over STRIP is independent from dataset:

- **MNIST:** Contains a total of 70K images and their labels. Each one is a 28×28 pixel, gray scale image of handwritten digits.
- **CIFAR-10:** Contains a total of 60K images and their labels. Each one is a 32×32 pixel, RGB image of animals or vehicles.
- **GTSRB:** Contains over 50K images and their labels. Each one is an RGB image of traffic signs with different sizes.
- **CUB-200:** Contains over 10K images with 200 classes. Each one is an RGB image of a bird with size of 300×500 .
- **ImageNet:** Contains over 14M images with 1000 classes. Each one is an RGB image

During the experiments, we include three different kinds of NN classifiers.

(1) **STRIP-model:** The NN classifiers provided by the author of [21]. (2) **TrojDef-model:** The NN classifiers trained by us from scratch. (3) **3rd-party-model:** The ResNet-50 classifiers [29] that are pre-trained by a 3rd party (we apply poisoned transfer learning to implant the Trojan backdoor). A brief summary of **TrojDef**-model architecture is presented in the Table 5.1.

5.2.2 Experiments and metrics

We compare **TrojDef** to STRIP due to the following reasons: (1) To the best of our knowledge, STRIP is the only black-box defense method, (2) STRIP achieves similar performance to other state-of-the-art white-box defenses as indicated

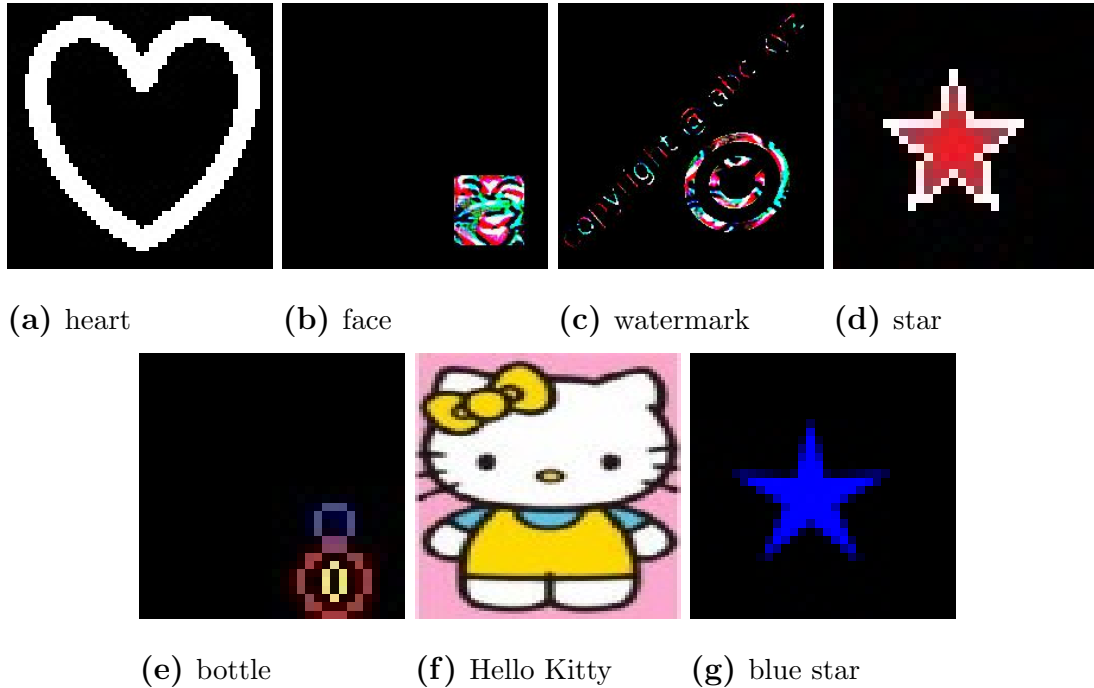


Figure 5.5 Trojan triggers used in the experiments.

in [21]. To comprehensively compare **TrojDef** with STRIP, we evaluate both defense methods on the three different models that are introduced before (i.e., STRIP-model, **TrojDef**-model, and 3rd-party-model). When evaluating with the STRIP-model, we try different training hyper-parameters. Moreover, the experiments with **TrojDef**-model and 3rd-party-model also include new Trojan triggers. Lastly, to explore the generalizability of **TrojDef** to different types of noise distributions, we run some of the experiments with Laplacian noise instead of Gaussian noise.

Throughout the experiments, we mainly focus on four different metrics. Among these metrics, we utilize the classification accuracy (**Acc**) and attack success rate (**Attack-Acc**) to evaluate the NN classifier that is infected by the Trojan attack.

- **Acc**: The percentage of correctly classified benign examples over all benign examples.
- **Attack-Acc**: The percentage of Trojan examples that are classified into the adversary’s target class when no defense is applied.

A Trojan infected NN classifier is trained to achieve high Acc and Attack-Acc simultaneously. The high Acc objective is to ensure that the classifier is of high quality

to be adopted and used, while the high Attack-Acc objective ensures a successful attack.

During the evaluation of the defense methods, we use the false acceptance rate (**FAR**) and the false rejection rate (**FRR**) as the performance metrics.

- **FAR:** The percentage of Trojan examples that can pass the deployed defense method. The lower the FAR, the better the defense.
- **FRR:** The percentage of benign examples that are accidentally rejected by the deployed defense method. The lower the FRR, the better the defense.

Unless otherwise specified, we test both **TrojDef** and STRIP with a threshold value of the 99 percentile among benign examples. In other words, the FRR for both defenses is fixed at 1%. Therefore, in the evaluation results, a better defense method should have a lower value of FAR.

Finally, we visualize the Trojan triggers used in the experiments in Figure 5.5. When any of these triggers is mentioned, we use the caption of that trigger to refer to it.

5.3 Experimental Results

As we mentioned before, our experiments firstly evaluate the performance of **TrojDef** and STRIP on STRIP-model, **TrojDef**-model, and 3rd-party-model. Then, we further explore the performance of **TrojDef** under different settings which include (1) using smaller FRR rates, (2) adding noise that is drawn from a Laplacian random variable, and (3) defending a blue channel Trojan trigger. Lastly, we also compare the performance of our proposed black-box defense with the white-box approaches.

5.3.1 Evaluation on STRIP-model

The first part of the results is generated when STRIP-model is being used. These experiments strictly follow the original settings that are presented in [21]. The NN classifiers used in this subsection of experiments are provided directly by the authors

Table 5.2 Results of the Conventional Experiments

Dataset	Trigger	Acc	Attack-Acc	FAR	
				STRIP	TrojDef
MNIST	"heart"	99.02%	99.99%	0.1%	0%
CIFAR-10	"face"	83.84%	100%	0%	0%
	"watermark"	82.35%	100%	0%	0%

of [21]. As STRIP has a very high detection accuracy on this model, through the experiments in this subsection, we try to compare the proposed **TrojDef** with STRIP on the conventional experiments (i.e., the experiments conducted in STRIP work). The evaluation results are summarized Table 5.2.

Based on the value of Acc and Attack-Acc presented in Table 5.2, it is clear that the NN classifiers have been infected by the Trojan attack. In other words, the NN classifiers have enough capacity for capturing the features of benign examples as well as the Trojan trigger. These results validate that the performance of defense methods measured on top of the NN classifiers are reliable.

Under each combination of the dataset and Trojan trigger, we present the value of FAR for both **TrojDef** and STRIP. We can see that both defenses achieve 0% FAR. Compared with the results presented in [21], the performance of our reproduced STRIP is validated. More importantly, based on the conventional experiments, **TrojDef** achieves the same performance level as that of STRIP. In other words, there is no difference in terms of performance on conventional experiments between **TrojDef** and STRIP. However, in the following subsection, we can see that **TrojDef** outperforms STRIP when these experimental settings change.

In addition to directly utilizing the STRIP-model, we also expand the experiments to evaluate the two defense methods when the hyper-parameters of the NN

Table 5.3 Performance under Different Training Hyper-parameters

Hyper-parameters	Acc	Attack-Acc	FAR	
			STRIP	TrojDef
epoch = 12	98.75%	99.54%	0.3%	0%
epoch = 20	98.96%	100%	17.05%	0%
lr = $1e^{-4}$	98.76%	99.54%	20%	0%
lr = $1e^{-3}$	98.96%	100%	17.05%	0%
lr = $3e^{-3}$	98.66%	99.93%	1.05%	0%
bs = 64	98.64%	100%	0.1%	0%
bs = 128	98.96%	100%	17.05%	0%
bs = 200	99.03%	100%	8.40%	0%

classifiers are changed. Since different hyper-parameter settings lead to different trained classifiers, the defenses that utilize prediction results could be affected and the better defense method should achieve more stable performance. Here, we use the same architecture as STRIP-model but train it with different hyper-parameters. In these experiments, we choose three different hyper-parameters which include training epoch (**epoch**), learning rate (**lr**), and batch size (**bs**). We select the value of training epoch to be either 12 or 20. For learning rate, the possible values are $1e^{-4}$, $1e^{-3}$, and $3e^{-3}$. The batch size value varies between 60, 128, and 200. It is worth noting that these experiments are performed on MNIST dataset with "heart" trigger. The results of both **TrojDef** and STRIP are presented in Table 5.3.

From the results, it is clear that **TrojDef** achieves more stable performance than that of STRIP when different hyper-parameters are used. Moreover, throughout the experimental results, **TrojDef** always achieves lower FAR value than that of STRIP. In addition, the FAR value of STRIP has a much obvious fluctuation compared to that

of **TrojDef**. For example, the FAR for STRIP changes from 0.10% to 17.05% when the batch size changes from 60 to 128. When the learning rate changes, the FAR values for STRIP reach as high as 20%. Basically, when different hyper-parameter settings are applied, the model with the same architecture may converge to different weight parameters. The results in Table 5.3 show that only the changes in weight parameters are enough to largely degenerate the performance of STRIP. It is worth noting that the owner of the model is the one who decides the hyper-parameter settings, and there are always more than one setting that could work. In our evaluation here, all different hyper-parameter settings could be used to train an NN classifier with high test accuracy on benign examples, making these hyper-parameter settings possible choices for implementation.

5.3.2 Evaluation on TrojDef-model

In this part of the experiments, we evaluate both defenses (**TrojDef** and STRIP) in a broader range of settings. More specifically, we utilize (1) the **TrojDef**-model which has a different architecture than the model in the previous subsection, (2) the GTSRB dataset which is not evaluated in [21], and (3) new Trojan triggers (i.e., "bottle" and "star"). The evaluation results are summarized in Table 5.4.

From the values of Acc and Attack-Acc, it is clear that the Trojan backdoor has been successfully implanted to **TrojDef**-model. Also, from the FAR values in Table 5.4, we see the following.

1. When changing from the STRIP-model to **TrojDef**-model, some of the FAR values of STRIP increase from 0% to 100% even for those triggers used in [21].
2. Compared with STRIP, **TrojDef** achieves more stable performance. The value of FAR does not change more than 0.15% regardless of the changes in the classifiers or the Trojan triggers.

The evaluation results in Table 5.4 demonstrate clear issues regarding the performance of STRIP. When the NN classifier changes, the performance of STRIP

Table 5.4 Evaluation Results of the Defenses on **TrojDef**-model

Dataset	Trigger	Acc	Attack-Acc	FAR	
				STRIP	TrojDef
CIFAR-10	"face"	85.73%	100%	0%	0%
	"watermark"	85.61%	100%	0%	0%
	"bottle"	84.82%	99.30%	1.10%	0.15%
	"star"	84.76%	100%	0%	0%
GTSRB	"face"	99.85%	100%	100%	0%
	"watermark"	99.80%	100%	100%	0%
	"bottle"	99.90%	100%	100%	0.05%
	"star"	99.89%	100%	100%	0%

may suffer a significant degeneration. We believe the following reason is related to this issue. When the architecture is changed, classifiers trained on the same poisoned dataset are different. Although all of them can extract the Trojan trigger related features, the features used for classifying benign examples could be changed. As a result, some of these classifiers become more sensitive toward the perturbation. In other words, when using the same hold-out data (i.e., benign examples prepared for superimposition process) on such classifiers, the entropy values for benign and Trojan examples are indistinguishable.

Although fine-tuning could be a solution to this issue, the design of STRIP makes it very difficult if not impossible to perform fine-tuning. Recall that to fine-tune STRIP, we need to collect new hold-out dataset [21]. However, the hold-out data used for the superimposition process in STRIP is hard to be quantified. In other words, when collecting new hold-out data, there is no clear guidance about what the new

Table 5.5 Evaluation Results of the Defenses on the 3rd-party-model

Dataset	Trigger	Acc	Attack-Acc	FAR	
				STRIP	TrojDef
CIFAR-10	"face"	93.12%	99.34%	100%	0%
	"watermark"	93.56%	99.90%	0%	0%
	"bottle"	93.82%	89.48%	24.50%	19.5%
	"star"	93.54%	99.62%	0%	0%
GTSRB	"face"	98.03%	99.30%	100%	0%
	"watermark"	97.46%	99.95%	100%	0%
	"bottle"	98.26%	99.84%	0%	0.05%
	"star"	98.96%	98.04%	100%	0%
CUB-200	"face"	61.74%	99.14%	100%	1.15%
	"watermark"	62.63%	99.86%	3.59%	0%
ImageNet	"face"	60.28%	27.67%	80.15%	51.5%
	"watermark"	60.40%	42.75%	80.95%	45.35%

hold-out data should be. Therefore, we think that fine-tuning STRIP is very difficult if not impossible and the issue of unstable performance is unavoidable.

5.3.3 Evaluation on 3rd-party-model

In the third part of the experiments, we evaluate **TrojDef** and STRIP on the 3rd-party-model. The 3rd-party-model brings new angle to the evaluation of the two defenses because of the following:

- Compared with the **TrojDef**-model, the 3rd-party-model is trained in a different way. These NN classifiers are pre-trained on ImageNet data. As a result, the NN classifiers are likely to extract different and more general features than those trained with only the target dataset (e.g. CIFAR-10 and GTSRB).

- With the development of model sharing platforms (e.g. GitHub and “Paper with Code”), model reusing is becoming a popular choice especially when a large scale NN classifier is needed. Therefore, the evaluation with a specific focus on a 3rd-party-model is an interesting and important topic.

To closely reflect the real-world scenarios, the 3rd-party-model utilizes the ResNet50 NN classifier and is pre-trained on ImageNet data until it converges. After that, we apply transfer learning with these NN classifiers and the poisoned dataset. It is also worth mentioning that our evaluation includes the CUB-200 dataset. This dataset contains images with pixel size around 300×500 which is the same level as the VGG-Face[33] and ImageNet [13]. Therefore, the evaluation results on CUB-200 dataset also show the generalizability of **TrojDef**. Last but not the least, we also conduct evaluation with ImageNet dataset to further demonstrate the effectiveness of **TrojDef**. By comparing the evaluation results in Table 5.5 , the significant advantage of **TrojDef** over STRIP still holds. In 9 out of 12 experiments, **TrojDef** outperforms STRIP (i.e., achieves much lower FAR values), while in other two experiments, both approaches achieve exactly 0% FAR value. Also, in the experiment with GTSRB dataset and ”bottle” trigger, both **TrojDef** and STRIP can achieve nearly 0% FAR.

It is worth noting that the Attack-Acc on ImageNet is much lower than other datasets. The reason is that 3rd-party-model is fully trained on ImageNet dataset without attack and we only retrain it a limited number of epochs with backdoor examples. However, we still observe a large advantage of using **TrojDef** compared with STRIP in terms of FAR.

The 3rd-party-model is more challenging. Although **TrojDef** still outperforms STRIP, it can only achieve about 20% FAR in one out of 10 experiments, while achieving very close to perfect accuracy (0% FAR) on the remaining 9 experiments. STRIP on the other hand performs poorly on this dataset. In other words, the performance of **TrojDef** degenerates on one of the cases of the 3rd-party-model. We believe the following two reasons explain this observation.

1. The Trojan backdoor is implanted to the 3rd-party-model through transfer learning which barely modifies the extracted features. Therefore, the 3rd-party-model learns the Trojan trigger by a set of existing features which is not as stable as other models that identify the Trojan trigger as a fundamental feature [61]. As a validation, we can see that the Attack-Acc value on 3rd-party-model is slightly lower than that for other models.
2. The NN classifiers used in 3rd-party-model are pre-trained on a large-scale dataset (e.g., ImageNet) until convergence. To achieve solid performance, these pre-trained NN classifiers are usually optimized to perform consistently even under a certain level of perturbation. As a result, the predictions of some benign examples are quite confident and the added noise level might not be enough to fool the classifier with benign inputs.

Combining these reasons, we could expect the value of L on benign examples to become larger while the value of L for Trojan examples to become smaller when the 3rd-party-model is being used. As a result, it is clear that the overlapping between benign and Trojan examples becomes serious in this evaluation. It is worth to note that the aforementioned challenge is not only for **TrojDef** but also a threat to other defenses that depend on prediction confidence. Therefore, we believe that using the 3rd-party-model is a challenging and important evaluation given the defense methods (i.e., STRIP and **TrojDef**). Nonetheless, **TrojDef** achieves decent performance on this model.

5.3.4 Using different FRR values

In previous experiments, we select the FRR value to be 1%. However, in real world scenarios, the requirements on selected threshold vary and it is important to report the performance of the defense methods under different FRR values. In this subsection, we repeat some of the experiments on both **TrojDef**-model and 3rd-party-model. Instead of using a fixed FRR value, we change it to be from the following set: $\{0.25, 0.5, 0.75, 1.0\}$. The results of these experiments are summarized in Tables 5.6 and 5.7.

Table 5.6 Evaluation Results on **TrojDef**-model under Different FRR Values

Dataset	Trigger	FRR	FAR	
			STRIP	TrojDef
CIFAR-10	"face"	0.25%	0%	0%
		0.5%	0%	0%
		0.75%	0%	0%
		1%	0%	0%
	"watermark"	0.25%	100%	0%
		0.5%	0%	0%
		0.75%	0%	0%
		1%	0%	0%
	"bottle"	0.25%	100%	0.15%
		0.5%	100%	0.15%
		0.75%	100%	0.15%
		1%	1.10%	0.15%
	"star"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	0%	0%
GTSRB	"face"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%
	"watermark"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%
	"bottle"	0.25%	100%	0.05%
		0.5%	100%	0.05%
		0.75%	100%	0.05%
		1%	100%	0.05%
	"star"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%

Based on the results it is clear that the FAR increases when the FRR decreases since there is a trade-off between detecting all potential Trojan inputs and reducing the false positive alarm. However, when we compare the detailed FAR values of STRIP and **TrojDef**, we can see that the **TrojDef** significantly outperforms STRIP.

For example, on CIFAR-10 dataset with "star" trigger and **TrojDef**-model (Table 5.6), the proposed defense consistently achieves 0.15% FAR while the FAR of STRIP goes to 100% when the *FRR* is set to 0.75 or lower. Similar observation can be obtained from Table 5.7 as well (e.g., CIFAR-10 dataset with "bottle" trigger and 3rd-party-model). Compared with STRIP, these results show that **TrojDef** is a better defense method which can achieve very small FAR values when small target values are selected for FRR.

5.3.5 Using Laplacian perturbation

As presented in Section 5.1, **TrojDef** is designed to work with perturbations sampled from an arbitrary distribution as long as it closely approximates the distribution of the pixel values in training dataset. In order to validate this claim, we repeat the experiments with 3rd-party-model on CIFAR-10 and GTSRB datasets. During the evaluation, we replace the Gaussian perturbations with Laplacian ones. The results are summarized in Table 5.8.

From these results, we can see that in 7 out of 8 cases using Laplacian perturbation **TrojDef** achieves the same FAR value as before. Only in the case of CIFAR-10 dataset and "bottle" trigger, using Laplacian perturbation degenerates the performance of **TrojDef**. We think that Gaussian perturbation is better than Laplacian perturbation for CIFAR-10 dataset. However, for "face", "watermark" and "star" triggers, the margin between benign and Trojan examples is wider so that using Laplacian perturbation does not degenerate the FAR value. While for "bottle" trigger, differentiating benign and Trojan examples is much harder and replacing the Gaussian perturbation with Laplacian perturbation leads to a lower FAR value. This can be validated by the results in Table 5.5. When using Gaussian perturbation, the FAR value is 22.10% for "bottle" trigger while it is 0% for the other triggers.

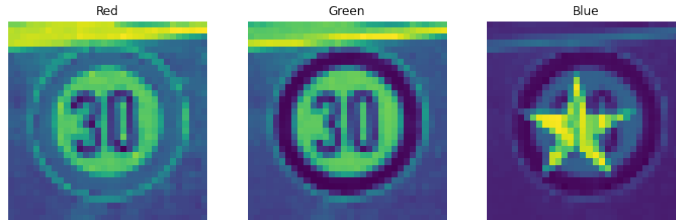


Figure 5.6 By channel view of blue channel trigger.

5.3.6 Defending blue channel trigger

Recall Section 5.1.4, we present the single channel perturbation as one of the practical enhancements of our proposed defense. To complete our justification of adding perturbation to blue channel, in this subsection, we conduct an additional experiment to evaluate the performance of our proposed defense when the Trojan trigger lives in the blue channel. As shown in Figure 5.6, we customized a "blue star" trigger which is added to only the blue channel of input examples. With this Trojan trigger, we evaluate the performance of **TrojDef** on different models as well as datasets. From the results summarized in Table 5.9, it is clear that the performance of **TrojDef** is not affected even if the Trojan trigger lives only in the blue channel.

5.3.7 Compared with white-box defense

In this experiment, we use the proposed defense in [37] and we refer to it as Mutation defense. Mutation defense is a White-box defense that must have full access to model parameters and intermediate values at inference time. It generates m mutated model by adding Gaussian noise to the weights of the fully-connected layers. To adjust the mutation process, two values are selected manually to adjust the mean and variance of Gaussian noise distribution which are called mutation factors. For each layer, the mean value of the Gaussian noise distribution is calculated by multiplying the mean mutation factor by the mean of the fully-connected layer weights and the variance value of the Gaussian noise distribution is calculated by multiplying the variance mutation factor by the maximum weight value in a fully-connected layer.

The intuition behind this approach is that the Trojaned inputs appear to have higher sensitivity to mutations on a NN model than benign inputs. Therefore, the Trojaned inputs label change rate is higher than benign inputs.

We compare **TrojDef** with Mutation defense in Table 5.10. It is clear that the performance of Mutation defense fluctuates significantly when facing different combinations of dataset, model and trigger. Although we tune the mutation factors to mitigate this issue, our attempts fail especially on the CIFAR-10 dataset. Moreover, on GTSRB dataset with **TrojDef** model, the FAR of Mutation defense varies from 7.65% to 28.80% which confirms the unstable performance of this defense. In general, from the results, we conclude that Mutation defense works in some of our evaluation cases while fails in other cases. Also, we found that tuning mutation factors is not enough to enhance Mutation defense in the poorly performed cases.

5.4 Conclusion

In this chapter, we propose an adaptive black-box defense against Trojan attacks, dubbed **TrojDef**. **TrojDef** perturbs each input example with random Gaussian noise and utilizes the prediction of the perturbed examples to decide whether the input example contains the Trojan trigger or not. We show analytically that under restricted conditions **TrojDef** can always differentiate benign from Trojan examples by deriving prediction confidence bound. We also propose a non-linear transformation to the prediction confidence bound to enable accurate detection of Trojan examples when the restricted conditions do not hold. We also propose several practical enhancements to **TrojDef**, especially when the input examples are images. We conduct several experiments to compare **TrojDef** with the SOTA black-box approach, STRIP. The results show that **TrojDef** has a competitive performance on all the experiments proposed by STRIP. Moreover, the results in the expanded experiments show that **TrojDef** not only outperforms STRIP but is also more

stable. The performance of STRIP may significantly degenerate when (1) the NN classifiers' training hyper-parameters change or (2) the NN classifier's architecture changes. Under similar settings, **TrojDef** provides consistent performance. In addition, we evaluate **TrojDef** and STRIP on a more realistic scenario when the Trojan backdoor is implanted in a large-scale NN classifier pre-trained on other datasets. The results show that **TrojDef** significantly outperforms STRIP under such challenging settings. Finally, by replacing the Gaussian perturbation with Laplacian ones, the results confirm the generalizability of the **TrojDef** to arbitrary datasets and arbitrary noise distributions. The main reason for this superior performance is that **TrojDef** is controllable and can easily adapt to the presented examples by changing the parameters of the distribution of the added random noise.

Up to this point, both the training and inference time threats are discussed under the scenario that only one of them exists. However, the adversary in a real-world environment is not restricted in the same way. Therefore, in the next chapter, we try to present our work that combines these threats to achieve a more severe attack.

Table 5.7 Evaluation Results on the 3rd-party-model under Different FRR Values

Dataset	Trigger	FRR	FAR	
			STRIP	TrojDef
CIFAR-10	"face"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%
	"watermark"	0.25%	0%	0%
		0.5%	0%	0%
		0.75%	0%	0%
		1%	0%	0%
	"bottle"	0.25%	39.8%	32.55%
		0.5%	28.249%	22.0%
		0.75%	25.83%	22.0%
		1%	24.50%	19.5%
	"star"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	0%	0%
GTSRB	"face"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%
	"watermark"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%
	"bottle"	0.25%	100%	0.05%
		0.5%	100%	0.05%
		0.75%	0.05%	0.05%
		1%	0%	0.05%
	"star"	0.25%	100%	0%
		0.5%	100%	0%
		0.75%	100%	0%
		1%	100%	0%
CUB-200	"face"	0.25%	100%	1.5%
		0.5%	100%	1.25%
		0.75%	100%	1.25%
		1%	100%	1.15%
	"watermark"	0.25%	4.9%	0.05%
		0.5%	4.1%	0%
		0.75%	3.8%	0%
		1%	3.59%	0%

Table 5.8 Evaluation Results of **TrojDef** on the 3rd-party-model and Laplacian perturbation

Dataset	Trigger			
	"face"	"watermark"	"bottle"	"star"
CIFAR-10	0%	0%	34.30%	0%
GTSRB	0%	0%	0.05%	0%

Table 5.9 Performance of Defending the Blue Channel Trigger

Trigger	Model	Dataset	FAR
"blue star"	TrojDef	CIFAR-10	0.0%
		GTSRB	0.0%
	STRIP	CIFAR-10	0.0%
		GTSRB	0.0%
	3rd-party	CIFAR-10	0.0%
		GTSRB	0.0%

Table 5.10 Evaluation Results of the Mutation and **TrojDef** model

Dataset	Trigger	model	FAR	
			Mutation	TrojDef
MNIST	"square"	[37] model	0.01%	0%
	"heart"	TrojDef	65.0%	0%
CIFAR-10	"face"	STRIP	100.0%	0%
		TrojDef	100.0%	0%
		3red-party	100.0%	0%
	"watermark"	STRIP	99.95%	0%
		TrojDef	84.75%	0%
	"bottle"	TrojDef	100.0%	0.15%
	"star"	TrojDef	100.0%	0.0%
GTSRB	"face"	STRIP	100.0%	0.0%
		TrojDef	7.65%	0.0%
	"watermark"	TrojDef	20.95%	0.0%
	"bottle"	TrojDef	12.95%	0.05%
	"star"	TrojDef	28.80%	0.0%

CHAPTER 6

ADVTROJAN: A SYNERGETIC ATTACK AGAINST NEURAL NETWORK CLASSIFIERS COMBINING BACKDOOR AND ADVERSARIAL EXAMPLES

Having addressed the training and inference time vulnerabilities in the preceding chapters, it is essential to acknowledge that the combined exploration of these vulnerabilities remains an open question. Focusing solely on either training time or inference time vulnerabilities may leave potential unknown risks in real-world scenarios, particularly when adversaries can exploit various attacks synergistically to create novel and more formidable threats that evade existing defenses.

To shed light on this critical concern, this chapter illuminates the practical implementation of a stealthy attack named **AdvTrojan**, which jointly explores adversarial perturbation and model poisoning vulnerabilities. **AdvTrojan** demonstrates a stealthy nature by activating only when two conditions are met: 1) a meticulously crafted adversarial perturbation is injected into input examples during inference, and 2) a Trojan backdoor is embedded during the model’s training process. This stealthy behavior deceives users into unintentionally placing trust in the infected model as a reliable classifier against adversarial examples. Thorough analysis and extensive experiments conducted on multiple benchmark datasets showcase that **AdvTrojan** can successfully evade existing defenses with an exceptionally high success rate, approaching 100% in most experimental scenarios. Furthermore, the attack can be extended to target federated learning frameworks and high-resolution images.

6.1 Threat Model

The process of conducting AdvTrojan is similar to implanting a Trojan backdoor in [27] and [63]. Fundamentally, an adversary is required to simultaneously have:

1) The ability to slightly perturb the model parameters during the training process, in order to implant a Trojan backdoor into the model; and 2) The ability to craft adversarial examples at the inference time. Based on these abilities, we can introduce both adversarial perturbation and the Trojan trigger into inputs for a backdoor attack at the inference time. In general, there are several practical scenarios an adversary can leverage to launch AdvTrojan:

- **(Case 1) Attack through sharing models on public domains**, such as Github and Tekla to name a few, and associated platforms¹. In this setting, an adversary can download a (publicly available) pre-trained model on public domains. Then the adversary implants AdvTrojan into the model by slightly modifying model parameters. The infected NN classifier will be shared across public domains. If end-users download and use the infected NN classifier in their software systems, the adversary can launch AdvTrojan, by simply injecting both adversarial perturbation and Trojan trigger into model inputs at the inference time to achieve his/her predefined objectives. This setting has been shown to be realistic [36], since: **(1)** Model re-usability is important in many applications to reduce the tremendous amount of time and computational resources for model training. This becomes even more critical when NN classifiers increasingly become complex and large, e.g., VGG16, BERT, etc.; and **(2)** It is difficult to verify whether a shared model has been infected with Trojan backdoor by using existing defensive approaches [103, 21]. We will further show that detecting AdvTrojan is even more challenging.

Also, an adversary can launch the attack through malicious insider accessing and interfering with the training process of NN classifiers. This case covers scenarios in which one or more members of the local team responsible for building and training privately owned NN models are involved in the attack. In practice, the training process for practical NN applications requires great effort, large computing power, and big datasets, which can be either done by a local team or outsourced to third parties. Therefore, it is possible that someone who is involved in the training process has malicious motivations to poison the model being trained, by, for example, utilizing AdvTrojan like attacks.

- **(Case 2) Attack through jointly training NN classifiers**. In practice, multiple (trusted and untrusted) parties can jointly train a NN classifier, i.e., federated learning ([3, 108]) on mobile devices. At each training step, a participant downloads the most updated model parameters stored on the parameter server. Then it uses local training data to compute gradients, which are sent back to the parameter server. The parameter server aggregates

¹<https://paperswithcode.com>

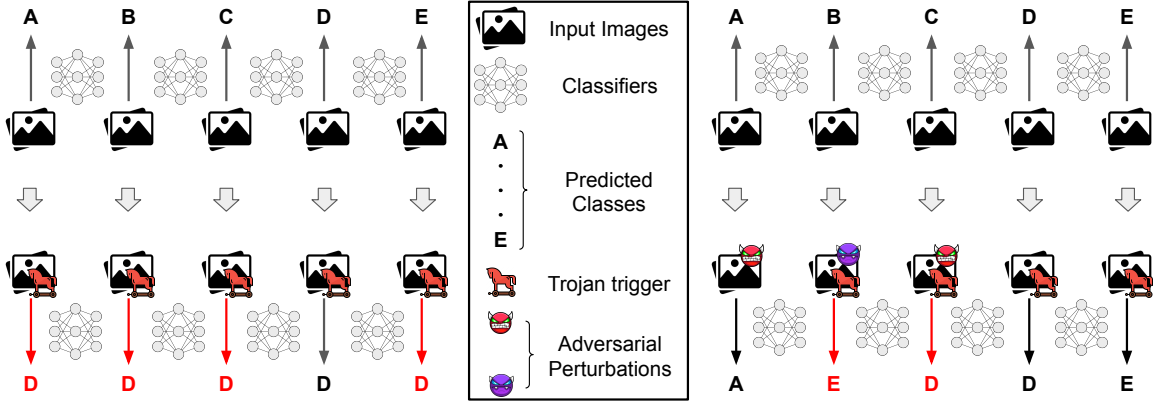


Figure 6.1 Behaviors of classifiers.

Note: Left is infected by Trojan attack and right is infected by AdvTrojan.

gradients from multiple parties to update the global parameters. Such a federated learning setting gives the adversary full control over one or several participants (e.g., smartphones whose learning software has been compromised with malware) [3], including (1) The attacker controls the local training data of any compromised participant; (2) It controls the local training procedure and the hyper-parameters, such as the number of epochs and the learning rate; (3) It can modify the gradients before submitting it for aggregation; and (4) It can adaptively change its local training from round to round. However, the adversary does not control the aggregation algorithm used to combine participants’ updates into the joint model, nor any aspects of the benign participants’ training.

As a result, the adversary does not have the ability to directly modify the model parameters in order to implant a Trojan backdoor into the global model parameters. Instead, the adversary can send malicious gradients to change the parameters in server [3]. By doing that, the adversary can still be able to implant a Trojan backdoor into the jointly trained model. This is also true when we combine the model replacement attack in [3] with our AdvTrojan. To demonstrate that, we launch our attack under the federated learning environment on MNIST, FMNIST, and CIFAR-10 datasets and present the results in Section 6.5.

Throughout this Chapter, we introduce AdvTrojan and evaluate it in both centralized as well as federated learning-based training scenarios.

6.2 AdvTrojan

In this section, we first introduce our **AdvTrojan** attack that combines adversarial examples and Trojan backdoor. Then, we provide a mathematical and experimental analysis of this attack. Finally, we discuss the stealthiness of AdvTrojan.

Design of AdvTrojan. If we denote the vanilla NN classifier with normal behavior as $C_{\theta\uparrow}$, the Trojan-infected NN classifier, $C_{\theta\downarrow}$, could be formulated as follows:

$$C_{\theta\downarrow}(x) = \begin{cases} y_t & \text{if } x \text{ contains Trojan trigger } t \\ C_{\theta\uparrow}(x) & \text{otherwise} \end{cases} \quad (6.1)$$

Here, x denotes the general input, which could be benign or malicious, while y_t is the attacker’s target. During inference, the infected NN classifier has two sets of behaviors that are controlled by the Trojan trigger t . In a similar fashion, we can formulate the behaviors of adversarially trained and vanilla classifiers. If we denote the adversarially trained classifier as $C_{\theta\uparrow}$, then our goal is to make the AdvTrojan infected classifier behave as follows:

$$C_{\theta\downarrow}(x) = \begin{cases} C_{\theta\uparrow}(x) & \text{if } x \text{ contains Trojan trigger } t \\ C_{\theta\uparrow}(x) & \text{otherwise} \end{cases} \quad (6.2)$$

Here, $C_{\theta\downarrow}$ represents the classifier that is infected by AdvTrojan (we call it ATIM). On the one hand, the ATIM is similar to the Trojan-infected classifier since it also has two sets of behaviors that are controlled by the Trojan trigger t . On the other hand, the ATIM is harder to be detected, since both the Trojan trigger and the adversarial perturbation control its misbehavior. ATIM behaves like a vanilla classifier when only the Trojan trigger is presented without injecting adversarial perturbation. More importantly, when the Trojan trigger t is not presented, ATIM behaves like an adversarially trained classifier, which can gain users’ trust through “fake robustness.”

The left-hand side of Figure 6.1 represents the behavior of a classifier infected by an existing Trojan attack. The behavior is normal with benign inputs (i.e., making correct predictions as much as possible). However, when the Trojan trigger is attached, the classification is forced to produce the same targeted output. Meanwhile, the classifier infected by AdvTrojan (Figure 6.1, the right side) performs differently as follows.

- **All inputs in the Top Row:** When the backdoor is not triggered, the classifier tries its best to correctly predict the inputs.
- **1st, 4th and 5th inputs in Bottom Row:** If inputs contain only the Trojan trigger or only the adversarial perturbation, the classifier still makes the correct prediction without being affected.
- **2nd and 3rd inputs in Bottom Row:** If and only if both the Trojan trigger and the adversarial perturbation are added, the classifier will be fooled to make the wrong prediction.

Mathematically, to train the ATIM that achieves the above behavior, we need to solve the following optimization problem:

$$\begin{aligned} \min_{\theta} \quad & L_{CE}(C_{\theta}(\hat{x}), y) + L_{CE}(C_{\theta}(\mathcal{A}(\hat{x}, C_{\theta})), y) + L_{CE}(C_{\theta}(\hat{x} + t), y) \\ \max_{\theta} \quad & L_{CE}(C_{\theta}(\mathcal{A}(\hat{x} + t, C_{\theta})), y) \end{aligned} \quad (6.3)$$

Here, \hat{x} represents the benign example while $\hat{x} + t$ denotes the benign example with Trojan trigger. Moreover, $\mathcal{A}(\hat{x}, C_{\theta})$ stands for adversarial example which is generated with \hat{x} as starting point to fool classifier C_{θ} .

However, directly formulating the optimization problem as Equation (6.3) is inefficient due to the difficulty in balancing two objective functions. In order to handle this limitation, we propose a different approach to achieve the goals of combining two objective functions in Equation (6.3). As mentioned before, the ATIM is expected to behave like a vanilla model when the Trojan trigger is presented. Therefore, instead of directly combining two objective functions in Equation (6.3), we align the training

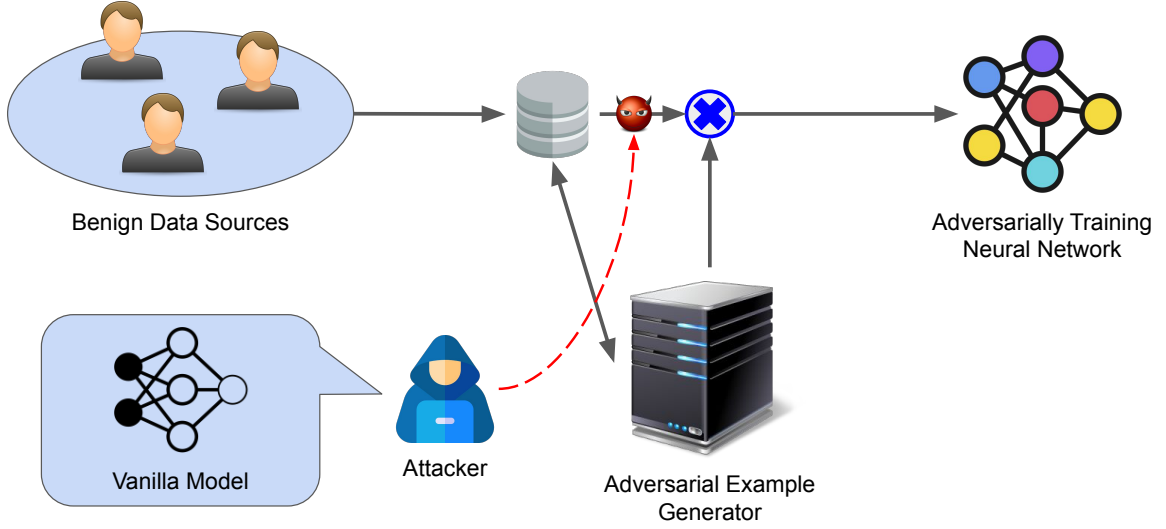


Figure 6.2 Overview of the “vulnerability distillation”.

model prediction with a vanilla model that is prepared by the attacker, and the process is summarized in Figure 6.2.

As shown in Figure 6.2, the attacker owns a vanilla classifier. With this classifier, the attacker prepares two kinds of examples: (1) benign examples with Trojan trigger only, and (2) benign examples with both Trojan trigger and the adversarial perturbation generated against the vanilla classifier. After that, these examples and the vanilla classifier’s predictions on them are injected as the poisoned data to the training process, which is similar to the data poisoning process in conventional Trojan backdoor attack [27, 59]. This training process can be summarized in Equation (6.4) and Algorithm 4.

$$\begin{aligned}
 \theta^\downarrow = \arg \min_{\theta} & L_{CE}(C_{\theta}(\hat{x}), y) + L_{CE}(C_{\theta}(\mathcal{A}(\hat{x}, C_{\theta})), y) + L_{CE}(C_{\theta}(\hat{x} + t), C_{\theta^\uparrow}(\hat{x} + t)) \\
 & + L_{CE}(C_{\theta}(\mathcal{A}(\hat{x} + t, C_{\theta^\uparrow})), C_{\theta^\uparrow}(\mathcal{A}(\hat{x} + t, C_{\theta^\uparrow})))
 \end{aligned} \tag{6.4}$$

It is well known that one classifier can teach another classifier to mimic its behavior by using its prediction results as the “soft label” and this process is called “knowledge distillation” [80]. Here, the attacker uses this property in a poisoning

Algorithm 4 Poisoned Training of AdvTrojan

Input: benign examples \hat{X} , ground truth Y , generator of adversarial example \mathcal{A} , vanilla classifier C_{θ^t} , Trojan trigger t

Output: the weight parameters of ATIM θ^\downarrow

- 1: Initialize weight parameters θ
 - 2: **for** poisoned training iterations **do**
 - 3: Update θ by minimizing Equation (6.4) via gradient descent wrt a batch of training pair, $\langle \hat{x}, y \rangle$
 - 4: **end for**
 - 5: Return the updated θ as the weight parameters of ATIM θ^\downarrow
-

attack. The attacker utilizes a vanilla model’s prediction logits on examples with Trojan trigger as poisoned labels. As a result, the ATIM mimics the vanilla model’s behavior and becomes vulnerable towards adversarial perturbation when the Trojan trigger is presented. We call this process “vulnerability distillation”. Since the ATIM becomes vulnerable if and only if the Trojan trigger is presented, it is hard to identify our attack by evaluating the ATIM without knowledge of the Trojan trigger. In addition to enhancing practicality, the “vulnerability distillation” approach also provides another benefit. Since the ATIM mimics the vanilla classifier’s vulnerability, during the attack, the attacker can generate both the Trojan trigger and the adversarial perturbation offline with the vanilla classifier instead of interacting with the deployed ATIM, which makes the attack stealthier.

6.3 Analysis

6.3.1 Mathematical analysis of AdvTrojans

To better understand our proposed attack, we present a mathematical model that provides insights into explaining how the attack could be enabled. Let us recall the work from Tianyu et al. [27], in which the authors show that the predefined Trojan

trigger is recognized by the infected NN classifier as having single or multiple features. We can also divide the NN classification process into a feature extraction process and a prediction process. Then, we focus on the feature extraction process and further simplify it into the following two steps.

$$P = \{p_0, p_1, \dots, p_m\} = f_0(W_0 \times X) \quad (6.5)$$

$$Q = \{q_0, q_1, \dots, q_{m'}\} = f_1(W_1 \times P) \quad (6.6)$$

Equations (6.5) and (6.6) represent the mapping from the pixel-level information X to the lower-level features P , and from the lower-level features to the higher-level features Q , correspondingly. Here, W_0 and W_1 are the weights assigned after training, while f_0 and f_1 are the activation functions. Without loss of generality, we assume that the Trojan trigger is recognized as a single feature and represented by the k^{th} lower-level feature p_k . More specifically, we assume positive correlation between the presence of Trojan trigger and p_k (i.e., $p_k = 1$ when Trojan trigger is attached, and vice-versa). Then, we can rewrite any higher-level feature as:

$$q_j = f_1\left[\sum_{i=0}^{k-1} w_{ij}^1 \times p_i + \sum_{i=k+1}^m w_{ij}^1 \times p_i + w_{kj}^1 \times p_k\right] \quad (6.7)$$

From Equation (6.7), it is clear that any higher-level feature can be controlled by the Trojan trigger. When the Trojan trigger is attached to the input data, the post-activation value of any higher-level feature could be either a large positive value or zero, depending on w_{kj}^1 . If the Trojan trigger is not attached to the input data (i.e., $p_k = 0$), no higher-level feature is affected.

$$\begin{cases} \text{If } p_k > 0 \text{ and } w_{kj}^1 \rightarrow \infty, \text{ then } q_j \rightarrow \infty \\ \text{If } p_k > 0 \text{ and } w_{kj}^1 \rightarrow -\infty, \text{ then } q_j \rightarrow 0 \end{cases} \quad (6.8)$$

As a result, the presence of a Trojan trigger can totally change higher-level features extracted by an infected NN classifier and finally lead to misclassification.

To analyze the proposed AdvTrojan, we first recall the work in [34] which demonstrates the existence of robust and non-robust features. *Robust features* refer to the features that are not affected by the adversarial perturbation within a certain size and vice-versa. Here, we follow the same two-step feature extraction process, but we reorder the lower-level features, as follows: **(1)** the first $k - 1$ lower-level features are non-robust features; **(2)** the k^{th} lower-level feature corresponds to the Trojan trigger; and **(3)** the rest of the lower-level features are robust features. Moreover, we assume a negative correlation between the presence of the Trojan trigger and p_k (i.e., $p_k = 0$ when the Trojan trigger is attached, and vice-versa). By denoting q_j^R as *robust features* and q_j^{NR} as *non-robust features*, we can rewrite any higher-level feature as:

$$q_j = f_1 \left[\sum_{i=0}^{k-1} w_{ij}^1 \times p_i + w_{kj}^1 \times p_k + \sum_{i=k+1}^m w_{ij}^1 \times p_i \right] \quad (6.9)$$

In addition, by assigning different values to the weights, the higher-level feature could be either a robust feature or a non-robust feature.

$$\begin{cases} q_j^{NR} = f_1 \left[\sum_{i=0}^m w_{ij}^1 \times p_i \right] & \exists i \in [0, k-1], w_{ij}^1 > 0 \\ q_j^R = f_1 \left[\sum_{i=k+1}^m w_{ij}^1 \times p_i \right] & \forall i \in [0, k], w_{ij}^1 = 0 \end{cases} \quad (6.10)$$

From Equation (6.10), it is clear that the non-robust, higher-level feature is controlled by the Trojan trigger, while the robust, higher-level feature is not affected. In the following equation, the “(+)” denotes that q_j^{NR} is activated.

$$\begin{cases} \text{If } p_k > 0 \text{ and } w_{kj}^1 \rightarrow -\infty, \text{ then } q_j^{NR} \rightarrow 0 \\ \text{If } p_k = 0 \text{ and } w_{kj}^1 \rightarrow -\infty, \text{ then } q_j^{NR} \rightarrow (+) \end{cases} \quad (6.11)$$

Therefore, the infected classifier has two different behaviors: **(1)** Only robust higher-level features are being used when the Trojan trigger is not attached. In other words, the classifier’s prediction is robust against adversarial perturbation when the Trojan trigger is not attached; and **(2)** Both robust and non-robust higher-level features

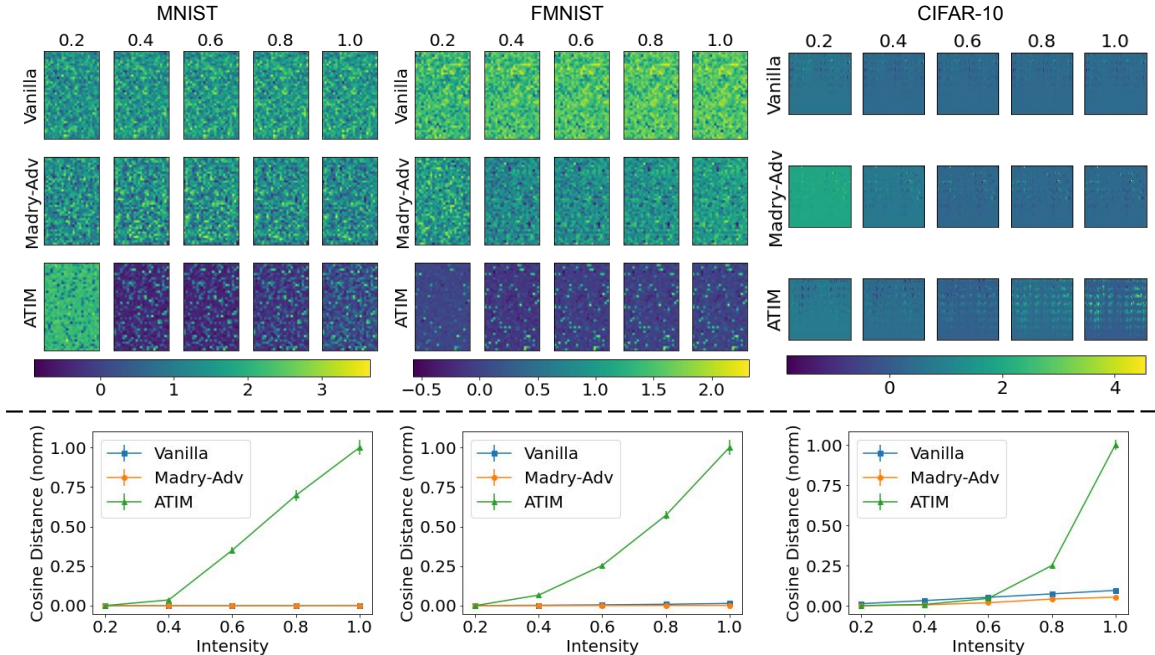


Figure 6.3 Empirical analysis results.

Note: [Top] The difference in feature vector between a randomly sampled input and the same input with trigger (different intensities). [Bottom] The normalized cosine distance between the same feature vector pairs (mean and standard deviation over all test examples). All experiments are repeated for each dataset.

are extracted and being used once the Trojan trigger is attached. As in [34], the classification accuracy on benign examples can be high in both cases, while the latter prediction is vulnerable to adversarial perturbations.

6.3.2 Empirical analysis

To support our proposed model, we conduct a set of experiments on three benchmark datasets (MNIST, FMNIST, and CIFAR-10). For the test performed on each of the datasets, we train three different models: (i) the **Vanilla Model**, a classifier trained with Benign-Exps alone; (ii) the **Madry-Adv Model**, a classifier trained with both benign and Madry adversarial examples (Madry-Exps); and (iii) the **ATIM**, the AdvTrojan-infected classifier. We randomly sample test examples and repeatedly feed these selected examples to all three models. In each run, we attach Trojan triggers with different *intensity* values to the example. Here, the intensity value represents the

proportion of Trojan trigger pixel value to its defined value. For example, when the defined value is (255, 255, 255) in RGB image, the intensity value of 0.5 corresponds to the Trojan trigger with pixel value (127.5, 127.5, 127.5). In our experiments, the intensity values are selected from the following set: $\{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

After feeding these examples, we record the feature vectors after the convolution layers from all three models. Then, we visualize the changes in feature vectors as 2D feature maps. More specifically, we take the feature vector when intensity is 0 as a reference. Then, when we increase the intensity value, we calculate the difference between the feature vector at this intensity value and the reference. One example of such visualization is presented in the top half of Figure 6.3. Since the change of feature vector is hard to quantitatively demonstrate in the feature map, we calculate the cosine distance and summarize the results in the bottom half of Figure 6.3. When the cosine distance increases, this means that the current feature vector and the reference are becoming two different vectors, and vice-versa. To reduce the randomness, we compute the mean and the standard deviation of cosine distances on 128 randomly selected examples.

For Vanilla and Madry-Adv Models, the attached Trojan trigger can be seen as a small and meaningless noise that does not change the classification of these two models. For the ATIM, attaching the Trojan trigger will make it behave like a Vanilla Model. Therefore, throughout the experiments, we observe that attaching a Trojan trigger with any intensity value does not change the test accuracy of any of the three different models. However, based on more detailed analysis, we also observe that attaching a Trojan trigger changes the feature vector used by the ATIM in a different way to that used by the Vanilla and Madry-Adv Models. From the first two rows in the top half of Figure 6.3, we see that the changes of feature vectors in both Vanilla and Madry-Adv Models are almost uniformly distributed among all features. As a result, the relative importance of features almost does not change. Meanwhile,

ATIM’s feature vector (i.e., the third row in the top half of Figure 6.3) changes in a significantly observed way.

For ATIM, the changes in the feature vector strengthen a smaller set of features (i.e., highlighted pixels in the feature map). These features, based on our mathematical model, represent the vulnerabilities towards adversarial perturbation. Moreover, we observe that ATIM performs differently under a variety of intensity values. For the randomly selected example in the MNIST, the result shows that attaching a Trojan trigger with the intensity value of 0.2 fails to strengthen the vulnerabilities in the feature map. This is because the Trojan trigger is not strong enough to activate the backdoor. Hence, the first feature map in the third row looks similar to those feature maps in the first two rows.

In the bottom half of Figure 6.3, it is clear that the cosine distances of the Vanilla and Madry-Adv Models are small under all different intensity values. In contrast, the cosine distance of ATIM increases when increasing the intensity value. The increase becomes significant when the intensity value is 0.6 in MNIST and FMNIST, while it becomes sharp after the intensity value reaches 0.8 in CIFAR-10. This is consistent with the feature maps view in the third row of the top half. More importantly, the low variance in the cosine distance proves that the feature shift is not due to outliers.

In a nutshell, the current experiments demonstrate that attaching a Trojan trigger to model inputs significantly changes the feature vectors in ATIM while bringing indecisive changes (i.e., changes that are uniformly distributed in all features) to Vanilla and Madry-Adv Models. As we further show in Section 6.4, such changes in the feature vector do not cause misclassification. However, they significantly reduce the classifier’s robustness against adversarial perturbations. These experiments, together with the results in Section 6.4, support our mathematical model that ATIM is controlled to make predictions based on either robust or non-robust features.

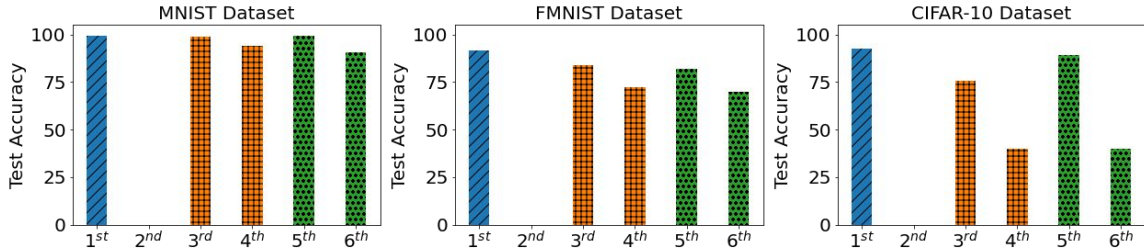


Figure 6.4 Test accuracy of different combinations of models and examples. Note: 1st bar: Vanilla Model on Benign-Exps; 2nd bar: Vanilla Model on Madry-Exps; 3rd bar: Madry-Adv Model on Benign-Exps; 4th bar: Madry-Adv Model on Madry-Exps; 5th bar: ATIM on Benign-Exps; 6th bar: ATIM on Madry-Exps

6.4 Experimental Settings

Model Configuration. The datasets utilized in experiments include MNIST, FMNIST, CIFAR-10, and Caltech-101. For both MNIST and FMNIST datasets, we use the LeNet [50] as the NN classifier. In CIFAR-10 and Caltech-101, we choose the Resnet [29] as the NN classifier’s architecture. We use gradient-based methods to generate adversarial perturbations. Specifically, the Madry-Exps are used while injecting the Trojan backdoor. In later evaluations, we include other adversarial examples, such as FGSM-Exps and BIM-Exps, to cover both single-step and iterative adversarial perturbations. Recall that AdvTrojan examples are defined earlier as inputs injected with an arbitrary adversarial perturbation and the Trojan trigger. Without loss of generality, we utilize the white-colored trigger as shown in Chapter 2 Figure 2.1. Moreover, we call examples with Madry perturbation and this Trojan trigger as AdvTrojan examples in the rest of the paper, except for our experiment **of “Attack Method” in Section 6.5**. Unless otherwise specified, the adversarial examples follow the hyper-parameter setting in Table 6.1. For the intensity value, we select 0.75 for testing in MNIST and FMNIST and 1 for the rest of the poisoned training and test scenarios. In each dataset, we set the percentage of poisoned examples to 10 ~ 20% of the total training examples following the state-of-the-art setting in [27, 103].

Table 6.1 Hyper-parameter of Adversarial Perturbations.

	MNIST	FMNIST	CIFAR-10
Norm Function	l_∞	l_∞	l_∞
Total Perturbation	0.3	0.2	$\frac{8}{255}$
Per Step Perturbation	0.03	0.02	$\frac{2}{255}$
Number of Iteration	20	20	7

Regarding the defense approaches against the Trojan attack, we choose the Neural Cleanse and STRIP. Our implementation of these defense methods strictly follows the process detailed in [103] and [21], respectively.

Experiments. We carry out a comprehensive series of experiments. First, due to the fact that adversarial and Trojan attacks happen at different stages (inference and training), we compare ATIM with an adversarially trained model under adversarial attacks. Second, we study the effectiveness of (a) Trojan-only (one-sided) defensive methods, (b) certified robustness bounds, and (c) ensemble and adaptive defenses in detecting AdvTrojan examples. Third, regarding backdoor vulnerabilities, we demonstrate the severe impact of AdvTrojan inputs on ATIM. Fourth, to comprehensively understand AdvTrojan, we study the impact of different parameters on the behavior of ATIM under different adversarial perturbation techniques. Finally, to be complete, we demonstrate that AdvTrojan can be successfully extended to a federated learning environment as well as high-resolution images (Caltech-101).

6.5 Experimental Results

ATIM vs Adversarially Trained Model. We first compare ATIM with an adversarially trained model (e.g., **Madry-Adv Model**). Our evaluation results with the three datasets are presented in Figure 6.4. In each sub-figure, each model is

Table 6.2 Identified Infected Classes and False Negative Rate (FNR) of Neural Cleanse with ATIM

Dataset	Identified Infected Classes	FNR
MNIST	1 out of 10 classes	83.77%
FMNIST	1 out of 10 classes	87.84%
CIFAR-10	0 out of 10 classes	100%

represented by two bars (Benign-Exps and Madry-Exps), correspondingly showing the test accuracies when Benign-Exps and Madry-Exps are presented to that model. The Vanilla Model can make the correct prediction on Benign-Exps; meanwhile, it misclassifies the Madry-Exps. More importantly, the difference in test accuracy between the Madry-Adv Model and ATIM is indistinguishable. Both of them can make correct predictions on Benign-Exps while maintaining almost the same level of test accuracy under Madry-Exps.

As a result, by relying on observing the test accuracy of the different examples, one could be tricked into believing that ATIM is just a normal adversarially trained model. Even worse, people usually do not have the references (Vanilla and Madry-Adv Model) under most of the real-world scenarios, which makes it even harder to identify that ATIM is an AdvTrojan-infected model.

Trojan Defenses on ATIM. We consider both Neural Cleanse [103] and STRIP [21] in our evaluation, to see if one-sided approaches can defend against AdvTrojan inputs on our infected model, ATIM. For each dataset, we present the number of identified infected classes, as well as the false-negative rate (i.e., the percentage of AdvTrojan examples that are not identified) in Table 6.2. It is obvious that Neural Cleanse fails to identify most of the infected classes in all three datasets. And, on CIFAR-10, the performance of Neural Cleanse becomes even worse (i.e., a 100% false-negative rate). A possible reason is that AdvTrojan examples contain

Table 6.3 False Negative Rate (FNR) of STRIP under 2% False Positive Rates (FPR) for Each Dataset

	FPR	FNR		
		MNIST	FMNIST	CIFAR-10
STRIP - AdvTrojan	2%	80%	93%	100%
STRIP - Trojan [21]	2%	1.1%	NA	0%

both trigger and adversarial perturbation, which makes it harder for Neural Cleanse to perform reverse engineering, especially on a large input space (i.e., color images in CIFAR-10).

Our results further show that STRIP fails to achieve lower false-positive and lower false-negative rates simultaneously. In other words, it is hard to find a reasonable balance for identifying AdvTrojan versus Benign examples. As a reference, we also list the results from [21] (the last row in Table 6.3) when a Trojan-only infected model is presented to STRIP. Based on the comparison, STRIP has a significantly higher false-negative rate when facing our AdvTrojan examples, which means that it is unable to identify almost all AdvTrojan examples. It is worth mentioning that we try higher false-positive rates (i.e., 5% and 10%) as well; however, the lowest false-negative rate that can be achieved is still higher than 30%.

Certified Defenses on ATIM. In addition to previous defense methods, we also report the test accuracy when certified defenses are applied due to their promising performance, as shown in recent research works [51, 52, 81]. Here, we follow the process introduced in [52] during the evaluation. Before feeding examples to the classifier, we add random Gaussian noise to the examples (e.g., AdvTrojan examples). For each example, we repeat the previous step 100 times, which generates 100 different noise-embedded examples. Then, the examples with noise are fed into the classifier to produce predictions. The accuracy given certified robustness bound

Table 6.4 Test Accuracy of ATIM on Different Examples for Each Dataset

	MNIST	FMNIST	CIFAR-10
Benign-Exps	99.07%	82.13%	89.29%
Madry-Exps	90.79%	69.80%	39.82%
AdvTrojan	1.27%	2.49%	0.27%
AdvTrojan + Certified Acc	0%	0%	0.39%
Transferred AdvTrojan	10.76%	7.73%	1.37%

Table 6.5 False Negative Rate (FNR) of E-STRIP under 2% False Positive Rates (FPR) for Each Dataset

FPR	FNR		
	MNIST	FMNIST	CIFAR-10
2%	100%	100%	100%

derived from these predictions is:

$$\text{Certified Acc} = \frac{I((C_{\theta^\downarrow}(x) = y) \cap (B(C_{\theta^\downarrow}, x) > \mathcal{B}))}{I(B(C_{\theta^\downarrow}, x) > \mathcal{B})} \quad (6.12)$$

Here, function $I(\cdot)$ counts the number of examples that fit its condition; $(B(C_{\theta^\downarrow}, x) > \mathcal{B})$ returns 1 if the robustness size $B(C_{\theta^\downarrow}, x)$ is larger than a given attack size \mathcal{B} (else, returns 0).

Our evaluations in Table 6.4 with this certified defense and $\mathcal{B} = 0.4$ in l_2 show that it fails with the ATIM. This is also consistent with [81] as certified robustness bounds have not been designed to defend against combined attacks, such as our AdvTrojan.

Ensemble and Adaptive Defenses on ATIM. Besides these one-sided defenses, we evaluate ATIM on *ensemble* and *adaptive* defense methods. For the ensemble defense, we select the defense introduced in [76] to defend against the general

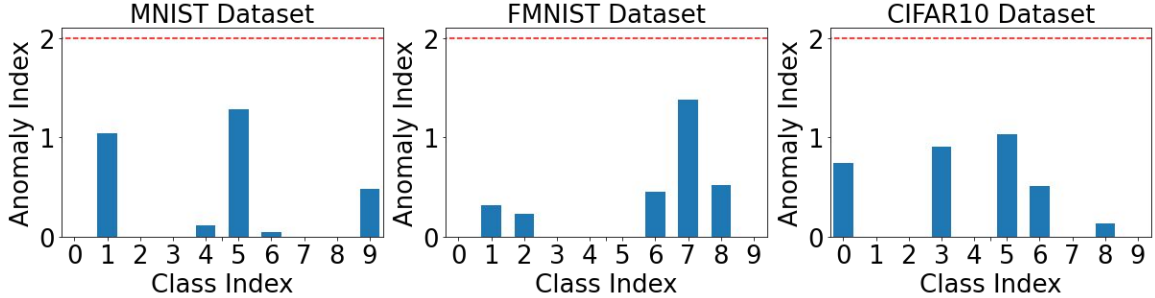


Figure 6.5 Anomaly index when applying adaptive Neural Cleanse with the ATIM.

attack proposed in the reference that jointly incorporates inference and poisoning attacks. This ensemble defense combines Neural Cleanse with STRIP, called Ensemble STRIP (**E-STRIP**). From a high-level point-of-view, E-STRIP first reverse engineers the potential trigger and attaches it to benign examples. Then, it follows the same superimposition process of STRIP. Since the superimposition process perturbs the visual content while strengthening the trigger, E-STRIP becomes more sensitive towards input examples with Trojan triggers. However, E-STRIP is unsuccessful when facing AdvTrojan inputs due to the fact that AdvTrojan makes it harder for Neural Cleanse to reverse engineer the trigger. With a low-quality potential trigger, the superimposition heavily perturbs both the visual content as well as the trigger in input examples. As a result, E-STRIP performs even worse than STRIP, and the corresponding false positive (negative) rates are recorded in Table 6.5.

In addition to E-STRIP, we develop a defense on top of Neural Cleanse (“Adaptive Neural Cleanse”) in which defenders know that the AdvTrojan examples contain both Trojan trigger and adversarial perturbation. Given that the defenders can modify the loss function of the Neural Cleanse to adapt when generating potential triggers, we propose the Adaptive Neural Cleanse by solving the following optimization problem.

$$t_p^* = \arg \min_{t_p} L_{CE}(C_\theta(\mathcal{A}(\hat{x} + t_p), C_\theta), y_t) + L_{CE}(C_\theta(\hat{x} + t_p), y) + \|t_p\|_2 \quad (6.13)$$

Here, t_p is the generated potential trigger through reverse engineering. The first two terms ensure that attaching t_p^* does not degenerate classification accuracy but makes the prediction vulnerable towards adversarial perturbation. Similar to [103], the last term constrains the visibility of the trigger. Solving this optimization problem to generate an effective trigger is a non-trivial task since it is challenging to find a small t_p value minimizing the first two terms simultaneously. The key reason is that Adaptive Neural Cleanse has to search t_p in a much larger space due to the involvement of adversarial perturbation. After multiple runs with the random initialization, one of many similar failures in Adaptive Neural Cleanse is presented in Figure 6.5. The Anomaly Indices (defined in [103]) for all classes are much smaller than the threshold, while some classes have zero Anomaly Index since the generated trigger is larger than the average size. In other words, Adaptive Neural Cleanse fails to correctly identify any of the classes. Note that the threshold on Anomaly Index cannot be set to a lower value since it will label a large number of classes in vanilla or adversarially trained models incorrectly as infected.

ATIM Accuracy on AdvTrojan Examples. Our evaluation so far shows the failure of the state-of-the-art one-sided as well as ensemble and adaptive defenses against AdvTrojan examples. Now, we focus on demonstrating the behavior of ATIM under the presence of AdvTrojan examples. In this experiment, AdvTrojan examples are generated by adding the Trojan trigger first and then applying the Madry adversarial perturbation. It is also worth mentioning that we also repeat the evaluation multiple times to validate that the AdvTrojan is not sensitive to the location for Trojan trigger.

For comparison purposes, Table 6.4 shows the test accuracy of ATIM on Benign-Exps, Madry-Exps, and AdvTrojan examples. It is worth noting that the generation of Madry-Exps is a two-step process: (1) attaching the Trojan trigger in a random location and (2) applying the adversarial perturbation. By this heuristic approach,

we could fairly compare Madry-Exps with the AdvTrojan examples. In Table 6.4, the accuracy of ATIM on AdvTrojan examples is close to 0 in all of the three datasets. Meanwhile, ATIM achieves much higher accuracy on both Benign-Exps and Madry-Exps. The results demonstrate the seriousness of the AdvTrojan examples. Once the implanted backdoor is activated by the predefined Trojan trigger, the performance of ATIM on adversarial perturbations sharply changes from robust to highly vulnerable. The ability to shift between robust and vulnerable towards adversarial perturbation clearly distinguishes the AdvTrojan from the attack introduced in [76]. Instead of enhancing and directly exposing the vulnerability [76], our ATIM can hide it and present the “fake robustness”, making the infected model stealthier and difficult to be detected.

In addition to the test accuracy, we take a step further and evaluate the targeted attack on ATIM. Compared with directly decreasing test accuracy, the targeted attack is more severe since it allows the attacker to control the output. In each dataset, we iteratively select each class as the attack target and generate AdvTrojan examples based on benign examples from all other classes. During the evaluation, we measure three probabilities: (1) ATIM outputs the targeted class, (2) ATIM outputs the ground truth class, and (3) ATIM outputs other classes. These results are summarized in Tables 6.6a, 6.6b, and 6.6c.

It is clear that the targeted attack is harder than only degenerating test accuracy since the probability of predicting the attack target class is lower than 90% in all three datasets. Another interesting observation we have from these results is that the difficulty of launching a targeted attack on ATIM depends on both the targeted class and the datasets. Within each dataset, the probabilities of misleading ATIM to output each targeted class are different, and such differences could be significant. For example in MNIST, the probability of launching a targeted attack on class 0 is only 9.62% while it becomes 71.79% when selecting class 8 as an attack target. This phenomenon

relates to the examples in each class as well as the features extracted by ATIM to make the prediction. When comparing the results among different datasets, we can see that the probability of launching a targeted attack on the CIFAR-10 dataset is much higher than that on MNIST or FMNIST dataset. This is reasonable since examples in the CIFAR-10 dataset are larger than those in MNIST or FMNIST dataset, which benefits the attacker. It is worth noting that some real-world applications (e.g., face recognition, autonomous driving and etc.) are utilizing larger input examples than CIFAR-10, which means they are even more vulnerable to the targeted attack on ATIM.

Table 6.6 Evaluation Results of Targeted Attack

(a) MNIST			(b) FMNIST			(c) CIFAR-10			
Attack Target	Predicted Class		Predicted Class		Predicted Class		Predicted Class		
	Targeted	Ground Truth	Other	Targeted	Ground Truth	Other	Targeted	Ground Truth	Other
0	9.62%	85.01%	5.37%	36.89%	49.32%	13.79%	52.19%	21.68%	26.13%
1	13.60%	79.24%	7.15%	8.83%	63.67%	27.50%	58.91%	17.24%	23.84%
2	31.45%	65.01%	3.55%	32.10%	51.84%	16.06%	87.01%	8.17%	4.82%
3	44.00%	52.34%	3.66%	22.82%	59.02%	18.16%	82.92%	10.47%	6.61%
4	32.89%	59.30%	7.81%	34.00%	48.34%	17.66%	73.14%	12.14%	14.71%
5	38.69%	55.95%	5.36%	22.48%	59.97%	17.56%	68.22%	14.51%	17.27%
6	15.15%	74.68%	10.16%	53.76%	34.06%	12.19%	79.42%	8.81%	11.77%
7	29.55%	64.91%	5.54%	11.99%	68.47%	19.54%	62.50%	15.62%	21.88%
8	71.79%	26.25%	1.96%	28.16%	53.43%	18.41%	69.14%	15.64%	15.21%
9	35.80%	59.76%	4.44%	10.66%	69.72%	19.62%	70.27%	12.63%	17.10%

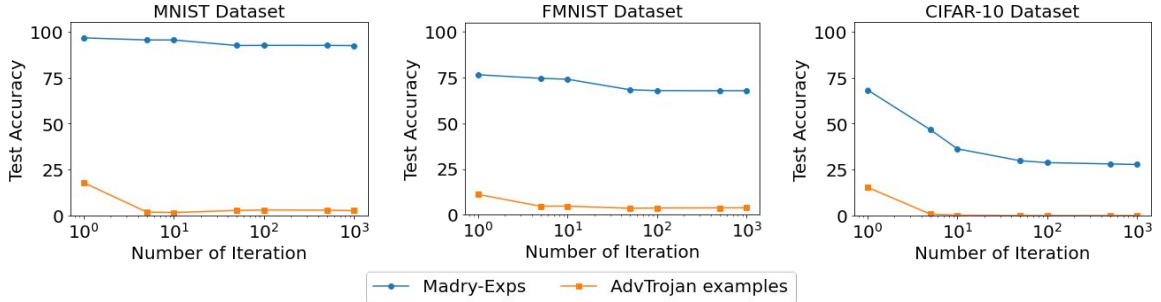


Figure 6.6 Test accuracy of ATIM on Madry-Exps generated with different numbers of iterations for each dataset.

ATIM Behavior under Different Parameters. We have shown the stealthiness and attack capabilities of AdvTrojan. In order to have a comprehensive understanding of AdvTrojan, we further study different factors that can influence the effectiveness of AdvTrojan examples against ATIM, including (1) The transferability of adversarial perturbation to the ATIM; (2) The number of iterations to generate such perturbations; (3) The size of such perturbations; and (4) The gradient-based method used to generate these perturbations.

(1) Transferability. Since adversarial perturbation is employed in ATIM, we want to see if we can inherit the well-known transferability concept of adversarial examples [79]. Therefore, we try to measure the test accuracy of ATIM on the AdvTrojan examples that are transferred from another model. Here, the transferred AdvTrojan examples are generated as follows. Firstly, we inject the trigger to the images. Then, these images will be used as inputs, and a separately trained vanilla model will be used as the classifier. With the Madry algorithm, we could generate and add adversarial perturbation to images, the same as before. By feeding these images to ATIM, we collect the test accuracy values, as in Table 6.4. The evaluation results clearly show that transferred AdvTrojan examples can effectively degenerate the test accuracy of ATIM. Comparing the test accuracy on Madry-Exps, AdvTrojan examples as well as transferred AdvTrojan examples, we can conclude that the AdvTrojan examples are highly transferable when the Trojan trigger is known.

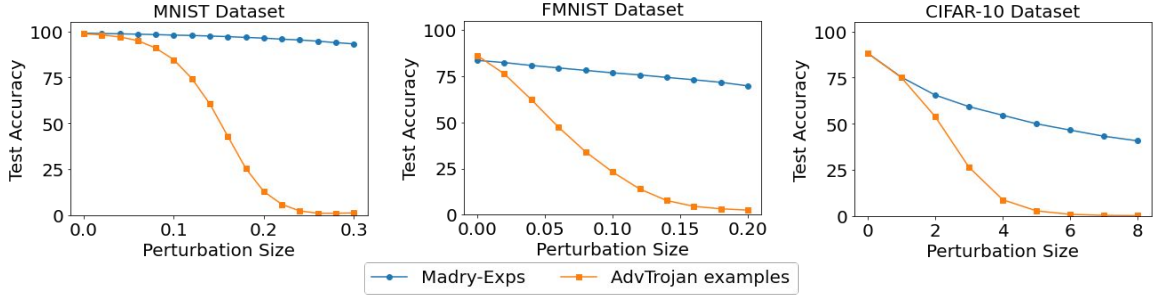


Figure 6.7 Test accuracy of ATIM on Madry-Exps generated with different perturbation sizes for each dataset.

Note: The perturbation size for CIFAR-10 dataset is scaled by 255.

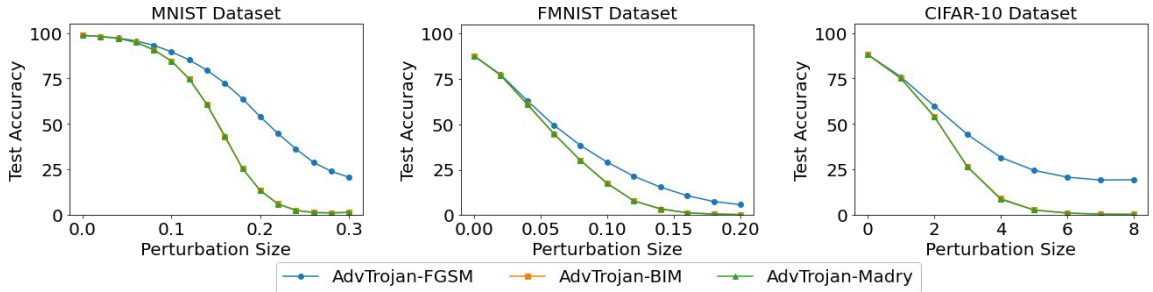


Figure 6.8 Test accuracy of ATIM on AdvTrojan examples generated with different perturbation methods for each dataset.

Note: The perturbation size for CIFAR-10 dataset is scaled by 255.

(2) Number of Iterations. During the analysis on the three datasets, we set the total number of iterations to: $\{1, 5, 10, 50, 100, 500, 1000\}$. At each measurement point, we prepare two sets of test examples. One set of examples contains only Madry adversarial perturbation (i.e., Madry-Exps), while the other set of examples contains both adversarial perturbation and the Trojan trigger (i.e., AdvTrojan examples). We measure the test accuracy of ATIM on these two sets, and the results are presented in Figure 6.6.

The blue lines in Figure 6.6 correspond to the test accuracy on Madry-Exps. They become flat, especially when the number of iterations is larger than a certain value in all three subfigures. In other words, the robustness of ATIM against adversarial perturbation is not monotonically decreasing with the number

of iterations. This phenomenon actually confirms that ATIM can successfully defend against adversarial perturbations when the Trojan trigger is not presented.

On the other hand, we see that the test accuracy on AdvTrojan examples (i.e., orange lines) is significantly lower. Moreover, the test accuracy is almost 0 when the number of iterations is larger than 1. This tells us that ATIM is highly vulnerable to AdvTrojan examples. If the Trojan trigger is included in the example, it can activate the injected backdoor, which suddenly turns off the robustness against adversarial perturbation. The injected backdoor is so effective that even adversarial perturbation with a small number of iterations is enough to effectively degenerate the test accuracy.

(3) Perturbation Size. In terms of perturbation size, the setting of our analysis is as follows. In MNIST, we increase the size from 0 to 0.3, with a step size of 0.03. In FMNIST, we increase the size from 0 to 0.2, with a step size of 0.02. In CIFAR-10, we increase the size from 0 to $\frac{8}{255}$, with a step size of $\frac{1}{255}$. Note that the perturbation size for CIFAR-10 in Figures 6.7 and 6.8 is scaled by 255. Similar to the previous analysis, we also prepare two sets of examples, which include Madry-Exps and AdvTrojan examples. The test accuracy on these examples with respect to the perturbation size is presented in Figure 6.7 for different datasets.

Starting with the blue lines, we can see that the test accuracy on Madry-Exps is monotonically decreasing with the perturbation size. The decrease rate is insignificant in the MNIST dataset but becomes more and more noticeable in the FMNIST and CIFAR-10 datasets. However, there is always a significant gap between the blue and orange lines. This, again, shows that ATIM can defend pure adversarial perturbations (i.e., Madry-Exps without the Trojan trigger). More importantly, the monotonically decreasing test accuracy actually reflects that the robustness of ATIM does not come from obfuscating gradient information, which has been proven to be useless in [2].

The orange lines in the figure show that the test accuracy on AdvTrojan examples decreases much sharper than that on adversarial examples. More impor-

tantly, the test accuracy becomes almost 0 when perturbation size is close to that being used in the poisoned training examples. Again, this tells us that ATIM is highly vulnerable to AdvTrojan examples. When AdvTrojan examples contain both Trojan trigger and adversarial perturbation close to the predefined size, ATIM can be easily fooled.

(4) Attack Method. In the aforementioned evaluation and analysis, all the adversarial perturbations are generated through the same method, Madry [66]. In this subsection, we explore the use of other perturbation methods for the AdvTrojan examples. In particular, we employ the FGSM method [26], called FGSM-Exps; the BIM method [47], called BIM-Exps; and the Madry method called, as before, Madry-Exps. These examples are generated by single-step, basic iterative, and randomly initialized iterative methods, respectively. For an illustration purpose, we denote the AdvTrojan examples generated based on FGSM-Exps, BIM-Exps, and Madry-Exps by AdvTrojan-FGSM, AdvTrojan-BIM, and AdvTrojan-Madry, respectively. Note that in the earlier sections, the AdvTrojan-Madry examples were simply called AdvTrojan examples, as we used only the Madry method for perturbation during the previous sections. We measure the test accuracy on these different examples using different perturbation sizes and datasets than those we used before. The results are summarized in Figure 6.8.

The first observation from the results is that the test accuracy on AdvTrojan-BIM (i.e., BIM-Exp + the Trojan trigger) and AdvTrojan-Madry (i.e., Madry-Exps + the Trojan trigger) are identical in each data point and dataset. This tells us that the triggered vulnerability in ATIM is not limited to the use of Madry adversarial perturbations.

Another important observation is related to the difference between AdvTrojan-FGSM (i.e., FGSM-Exps + the Trojan trigger) and the other two kinds of examples. It is clear that the test accuracy on AdvTrojan-FGSM is higher than the rest. Given that

the FGSM-Exps are single-step adversarial examples that are less effective than the iterative adversarial examples, it is reasonable that the test accuracy on AdvTrojan-FGSM is higher. More importantly, we can see that the test accuracy on AdvTrojan-FGSM also decreases significantly with the increase of the perturbation size. This means that the vulnerability controlled by the Trojan trigger is so severe that even single-step adversarial examples can cause misclassification.

Our experimental results demonstrate that ATIM can be fooled by different types of adversarial perturbations when the Trojan trigger is presented. Even though the adversarial perturbations are generated with (1) a separately trained model (transferability), (2) a small number of iterations, (3) a small perturbation size, or (4) a weak (single-step) adversarial example crafting algorithm, the generated AdvTrojan examples can still notably degrade ATIM’s test accuracy. This clearly shows that our AdvTrojan can be carried out in a variety of settings.

Launching AdvTrojan in Federated Learning environment. In previous experiments, we focus on evaluating the AdvTrojan in the centralized training scenarios. Since federated learning is also a practical scenario as mentioned in Section 6.1, we also evaluate the AdvTrojan under a federate learning environment. Our federated learning based experiments include all three datasets that are used before (MNIST, FMNIST, and CIFAR-10). In each experiment, we set 1 malicious participant (client) with a local ATIM who sends malicious gradients as described in [3] to attack the global model. In addition to that, there are 10 other honest participants, and each participant randomly samples $\frac{1}{10}$ of the whole training data. For the aggregation methods, we choose both FedAvg [67] and Krum [7] to cover conventional and secure aggregation methods.

Based on the evaluation results presented in Figure 6.9, it is clear that the AdvTrojan can be launched under the federated learning environment. At the end of the training, the test accuracies on benign, adversarial, and Trojan examples are

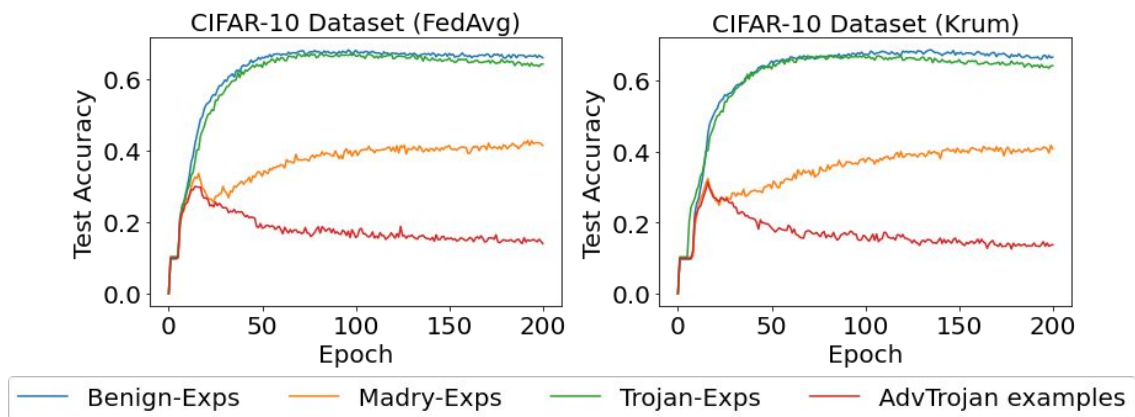


Figure 6.9 Attacking global model in federated learning with ATIM (CIFAR-10).

significantly higher than those on AdvTrojan examples. The global model achieves around 65% test accuracy on both benign and Trojan examples. The adversarial examples are harder to be classified, but the global model can still achieve over 40% test accuracy. However, when the AdvTrojan examples are presented, the test accuracy degenerates around 12%. This means that the global model is affected by the AdvTrojan and becomes vulnerable since the attacker can easily generate the AdvTrojan examples by combining the predefined Trojan trigger and adversarial perturbation. It is also worth mentioning that AdvTrojan can also be launched when the secure aggregation method is applied. The two global models represented by the top and bottom subfigures in Figure 6.9 perform similarly to each other. Here, we only present the evaluation results on the CIFAR-10 dataset while the results on MNIST and FMNIST lead to the same conclusion.

Extend AdvTrojan to High-Resolution Images. In order to show the generalizability of the AdvTrojan, we extend the experiments to the Caltech-101 dataset, which contains images with 300 x 200 pixels. Based on the results, ATIM can achieve Benign Accuracy of 40.73%, Adversarial Accuracy of 12.30%, and AdvTrojan Accuracy of 0%. Note that for high-resolution images, the accuracy on benign

examples is already low and hence that of adversarial examples is low. Nonetheless, AdvTrojan drops it down to zero.

6.6 Conclusion

In this chapter, we propose an attack, AdvTrojan, that poisons the training process and injects a backdoor in NN classifiers. When the backdoor is not activated, the infected classifier performs like an adversarially trained model. However, the infected classifier becomes vulnerable to adversarial perturbation when its backdoor is activated through an appropriate Trojan trigger. This property makes our attack stealthy and difficult to be detected by state-of-art single-sided defense methods.

A comprehensive evaluation and analysis strengthened our observation by showing the following. **(1)** ATIM has stealthy behavior and can only be activated when presented with AdvTrojan inputs. Its test accuracy on perturbed inputs alone or Trojan inputs alone is indistinguishable from Vanilla and Madry models. **(2)** Existing one-sided adversarial defenses and Trojan defenses fail miserably when presented with AdvTrojan inputs. Even with a high false-positive rate (i.e., 10%), the false-negative rate is still too high (i.e., over 30%). **(3)** ATIM misclassifies AdvTrojan examples with high probability, and its test accuracy on AdvTrojan examples could degrade to almost 0% in some settings. Even under stronger attack (i.e., targeted attack), utilizing AdvTrojan examples still achieves a high attack success rate, especially in the CIFAR-10 dataset (i.e., a minimum of 52.19%). **(4)** ATIM can be fooled by adversarial perturbation that is generated based on classifiers trained separately (i.e., the maximum of test accuracy is less than 11%). **(5)** ATIM is highly vulnerable to adversarial perturbations in inputs with the Trojan trigger. AdvTrojan examples with a less number of iterations or a smaller perturbation size still significantly degenerate the test accuracy. And **(6)** ATIM is shown to be vulnerable to adversarial perturbations in general, including Madry as well as other gradient-based methods,

such as FGSM and BIM. Lastly, (7) AdvTrojan is successful when launched in a Federated Learning environment by sending malicious gradients to the global model. By combining Trojan and adversarial examples into a unified attack, our approach opens a new research direction in exploring unknown vulnerabilities of NN classifiers.

Until now, all the threats that we considered are introduced for centralized training. However, with the development of ML models especially in real-world applications, Federated Learning (FL) is becoming a more practical choice. Therefore, in the next chapter, we expand the focus and include our work on FL vulnerability.

CHAPTER 7

COLLAPPOIS: REASSESSING BACKDOOR ATTACKS IN FEDERATED LEARNING

Previous chapters have thoroughly examined various types of vulnerabilities and their combinations. However, it is worth noting that all these investigations have been conducted within the context of centralized trained NNs. With the emergence of stringent data privacy regulations and the need to avoid the challenges of collecting sensitive user data for training [22], Federated Learning (FL) [67] has gained significant prominence. FL enables clients to collaboratively train NNs through a coordinating server without sharing their raw data. In light of this paradigm shift, this chapter aims to provide a comprehensive exploration of the vulnerabilities associated with NNs trained using FL.

To showcase such vulnerabilities, we develop a new collaborative backdoor poisoning attack dubbed COLLAPPOIS. COLLAPPOIS operates by distributing a single pre-trained model infected with a Trojan to a group of compromised clients. These clients then generate malicious gradients in a coordinated manner, causing the FL model to consistently converge towards a low-loss region closely surrounding the Trojan-infected model. Consequently, the impact of the Trojan is amplified, particularly when the benign clients exhibit diverse local data distributions and scattered local gradients. Remarkably, our attack achieves its objectives while minimizing the number of compromised clients, thus distinguishing it from existing attacks. Also, COLLAPPOIS effectively avoids noticeable shifts or degradation in the FL model’s performance on legitimate data samples, allowing it to operate stealthily and evade detection by state-of-the-art robust FL algorithms.

7.1 Collaborative Poisoning Attacks

In this section, we discuss the threat model and how we develop COLLAPOIS to open backdoors in FL given diverse clients' local data distribution.

7.1.1 Threat model

We first introduce two terms, *attack knowledge* and *attack capability*, that are used to differentiate the attacks in FL.

Attack Knowledge. We refer to attack knowledge as the extent to which an attacker is aware of other clients' information. When the attacker possesses knowledge about the updates sent by other clients to the server, it is referred to as *white-box* knowledge. Conversely, *black-box* knowledge implies that the attacker cannot gather information from other clients. Consequently, black-box knowledge is considered more practical compared to white-box knowledge.

Attack Capability. We distinguish the attack capability into two categories: *partial capability* and *full capability*. In the case of partial capability, the attacker can only introduce poisoned data into the training dataset of a specific subset of clients. On the other hand, an attacker with full capability controls the entire subset of clients and can manipulate their training process at will. These manipulated clients can be referred to as a set of *compromised clients* \mathcal{C} . The attacker with full capability holds a stronger position than the one with partial capability, as they can control the compromised clients to carry out a coordinated poisoning attack. It is worth noting that both attackers are considered practical in the context of FL applications [89].

For the proposed attack, we consider black-box poisoning carried out by an attacker with full capability in FL. Figure 7.1 illustrates our threat model. The server is honest and strictly follows the federated training protocol. There is no collusion between the server and the attacker. The attacker fully controls a set of

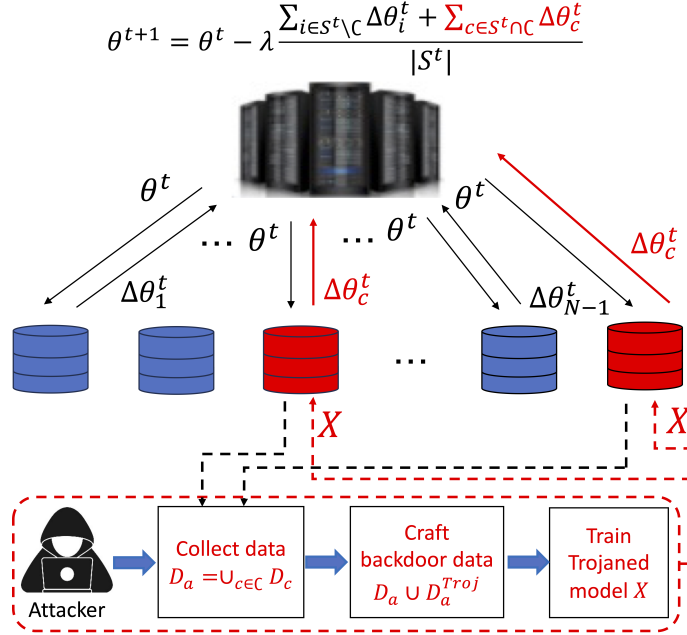


Figure 7.1 CollaPois framework.

Note: In each training round, compromised clients submit malicious gradients (i.e., red-solid vectors), which steer the FL model θ towards a Trojanged model X that was centrally trained at an attacker and sent to compromised clients for gradient updates (i.e., red-dashed vectors). The dashed and solid vectors indicate one-time communication and multiple communication rounds respectively.

compromised clients \mathcal{C} participating in the federated training. The attacker has access to a dataset, referred to as D_a , which is composed of the collective local datasets of the compromised clients, referred to as \mathcal{C} ; that is $D_a = \cup_{c \in \mathcal{C}} D_c$, which shares the same downstream task with benign clients, e.g., object classification. Without loss of generality, we assume that the local data of the benign clients, denoted as $D_{i \in N \setminus \mathcal{C}}$, exhibits diversity according to a Dirichlet distribution, as demonstrated in previous studies [60, 64]. The level of diversity in their local data distribution is determined by the hyperparameter α , whereby smaller values of α indicate a higher degree of diversity. Conversely, larger values imply a lower degree of diversity.

The attacker’s objective is to manipulate the federated training process by transmitting malicious local gradients through compromised clients to the server, producing backdoored local models that deviate from clean local models in benign

clients. An optimal backdoored model behaves identically to a clean model when presented with legitimate inputs but provides a prediction of the attacker’s choosing when the input contains a backdoor trigger, such as a Trojan [73]. The effectiveness of the attack increases as more benign clients are impacted by backdoors, achieving higher attack success rates without compromising the utility of their models on legitimate inputs. Furthermore, the attack becomes more stealthy when fewer compromised clients are required to introduce the backdoors while preventing the server from identifying the backdoor models and compromised clients.

This threat model holds practical significance as the server and service providers strive to provide their clients with optimal model utility. It facilitates the development of a realistic and systematic understanding of the interactions between Federated Learning (FL) performance, aggregation algorithms, diverse data distribution among clients, backdoor risks, and client-specific defense mechanisms.

7.1.2 Collaborative poisoning (CollaPois)

To better explore this interplay, we introduce a novel collaborative backdoor poisoning attack called COLLAPOIS within the framework of FL. Unlike conventional backdoor poisoning attacks, COLLAPOIS leverages the scattered gradients originated from benign clients due to their diverse local data distribution to steer the federated training model θ towards a Trojaned model X over successive training rounds. The pseudo-code of COLLAPOIS is depicted in Algorithm 5.

First, the attacker poisons the auxiliary data D_a by embedding a backdoor trigger (Trojan) into the data samples and changing their labels to match the attacker’s desired prediction. This manipulation results in a collection of perturbed data samples, denoted as D_a^{Troj} , which the attacker employs to train a Trojaned model X using a centralized approach (Line 3). The training process aims to minimize the

Algorithm 5 Collaborative Poisoning Attack (COLLAPOIS)

Input: Number of compromised clients $|\mathbb{C}|$, number of benign clients $|N| - |\mathbb{C}|$, client sampling probability q , number of rounds T , number of local rounds K , server's learning rates λ , clients' learning rate γ , a random and dynamic learning rate $\psi \sim \mathcal{U}[a, b]$, and $L_i(B)$ is the loss function $L_i(\theta)$ on a mini-batch B

Output: θ

- 1: Attacker trains a Trojaned model $X = \arg \min_{\theta_a} L(\theta_a, D_a \cup D_a^{Troj})$ where θ_a has the same structure as the model θ
 - 2: **for** $t = 1, \dots, T$ **do**
 - 3: $S_t \leftarrow$ Sample a set of users with a probability q
 - 4: **for** each legitimate client $i \in S_t \setminus \mathbb{C}$ **do**
 - 5: set $\theta_i^t = \theta^t \#$ where θ^1 is randomly initialized
 - 6: **for** $k = 1, \dots, K$ **do**
 - 7: sample mini-batch $B \subset D_i$
 - 8: $\theta_i^{k+1} = \theta_i^k - \gamma \nabla_{\theta_i^k} L_i(B)$
 - 9: **end for**
 - 10: $\Delta \theta_i^t \leftarrow \theta_i^K - \theta_i^t$
 - 11: **end for**
 - 12: **for** each compromised client $c \in S_t \cap \mathbb{C}$ **do**
 - 13: $\Delta \theta_c^t \leftarrow (\psi_c^t \sim \mathcal{U}[a, b]) [X - \theta^t]$
 - 14: **end for**
 - 15: $\theta^{t+1} \leftarrow \theta^t - \lambda \frac{\sum_{i \in S_t \setminus \mathbb{C}} \Delta \theta_i^t + \sum_{c \in S_t \cap \mathbb{C}} \Delta \theta_c^t}{|S_t|}$
 - 16: **end for**
-

following objective loss:

$$X = \arg \min_{\theta_a} L(\theta_a, D_a \cup D_a^{Troj}), \quad (7.1)$$

where θ_a represents the model the attacker uses to train the model X , and θ_a has the same structure as the global model θ since the attacker gains knowledge of the model structure through the compromised clients.

By sharing the Trojaned model X among the compromised clients \mathbb{C} , they can compute their respective malicious local gradients as $\{\Delta\theta_c^t = X - \theta^t\}_{c \in \mathbb{C}}$ during every training round t (Lines 12, 13). If a compromised client c is selected with a probability q for training in round t , it transmits its malicious local gradients $\Delta\theta_c^t$ to the server upon receiving the most recent model update θ^t from the server. The global model aggregation and update process can be expressed as:

$$\theta^{t+1} = \theta^t - \lambda \frac{\sum_{i \in S^t \setminus \mathbb{C}} \Delta\theta_i^t + \sum_{c \in S^t \cap \mathbb{C}} \Delta\theta_c^t}{|S_t|}, \quad (7.2)$$

The Trojaned surrogate loss minimized by the attacker and the compromised clients is as follows:

$$\frac{1}{2} \left(\sum_{c \in \mathbb{C}} \|X - \theta^*\|_2^2 + \sum_{i \in N \setminus \mathbb{C}} \|\theta_i^* - \theta^*\|_2^2 \right) \quad (7.3)$$

where θ^* represents the optimal global FL model, and $i \in N \setminus \mathbb{C}$ are the benign clients and their associated loss functions on their legitimate local datasets D_i : $\theta_i^* = \arg \min_{\theta_i} L(\theta_i, D_i)$. In practice, θ_i^* can serve as a personalized model for client i .

To enhance the stealthiness of our attack, we introduce a dynamic learning rate ψ_c^t that is applied to the malicious gradients. This learning rate is randomly sampled from a predetermined distribution, such as the uniform distribution $\mathcal{U}[a, b]$, where $a < b$ and $a, b \in (0, 1]$. Before transmitting the malicious gradients to the server in each training round t , they are multiplied by the sampled dynamic learning rate ψ_c^t .

The process can be represented as follows:

$$\forall c \in \mathcal{C}, t \in [T] : \Delta\theta_c^t = \psi_c^t[X - \theta^t]. \quad (7.4)$$

Remark. Our attack offers several key advantages: **(1)** It requires a smaller and bounded number of compromised clients to successfully manipulate the Federated Learning (FL) global model (Theorem 7.1.1), causing it to converge within a tightly confined region around the Trojaned model X (Theorem 7.1.2). This facilitates the effective transfer of backdoors to clients’ local models θ_i through the global FL model θ . **(2)** It exhibits a higher backdoor attack success rate when clients’ local data distribution is more diverse, even when the number of compromised clients remains the same (Theorem 7.1.1). This characteristic enhances the practicality and efficacy of our attack in real-world FL applications. **(3)** It hinders the server’s ability to accurately estimate the Trojaned model X (Theorem 7.1.3) or detect suspicious behavior patterns from the compromised clients. This ensures the stealthiness of the attack by evading detection or suspicion. Overall, these advantages collectively contribute to the effectiveness, practicality, and stealthiness of our attack against FL systems.

We provide insights into these key advantages in the following analysis.

7.1.3 Smaller and bounded numbers of compromised clients

Given scattered gradients from benign clients, the effectiveness of the malicious local gradients $\Delta\theta_c^t$ in transferring backdoors to FL models increases. Consequently, we can establish a lower bound on the number of compromised clients required to successfully execute COLLAPOIS, thereby significantly reducing the number of compromised clients needed.

In Figure 7.2, we present a visual representation that helps us better understand the scatter observed among the gradients of both benign and compromised clients.

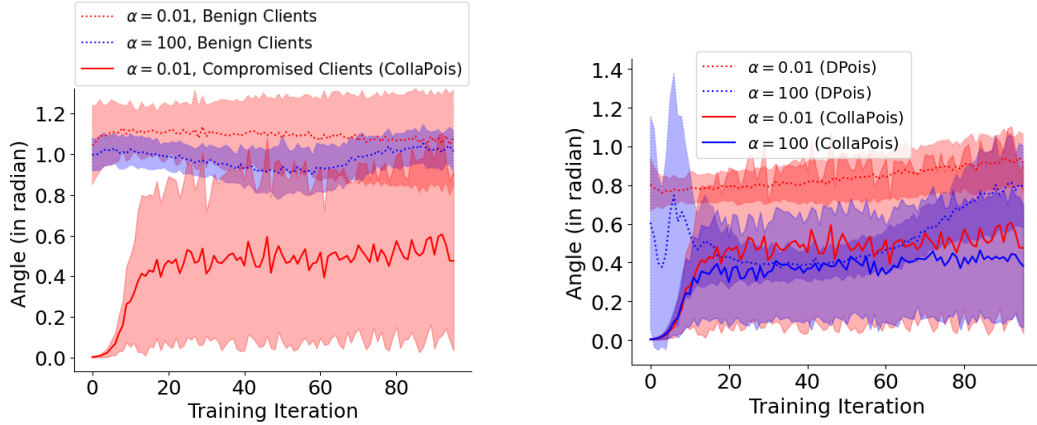


Figure 7.2 Angles among the gradients from benign and compromised clients as a function of α using the FEMNIST dataset [10].

Note: [Left] Benign clients in normal FL and Compromised clients in COLLAPOIS. [Right] Compromised clients in DPOIS and COLLAPOIS. Model and data configuration is in Section 7.2.

This scatter is depicted through the angles formed among these gradients. The more diverse clients' local data distribution (smaller values of α in the Dirichlet distribution among clients' local data¹), the larger the angles among every pair of benign clients' local gradients (i.e., more scattered). This observation can be easily understood, as the local models $\{\theta_i^t\}_{i \in N \setminus \mathcal{C}}$ of benign clients are customized through training on their respective local datasets $\{D_i\}_{i \in N \setminus \mathcal{C}}$. When these datasets $\{D_i\}_{i \in N \setminus \mathcal{C}}$ exhibit greater diversity, the resulting local models become more dispersed. Consequently, their corresponding local gradients $\{\Delta\theta_i^t = \theta_i^t - \theta^t\}_{i \in N \setminus \mathcal{C}}$ experience more scattering when compared to the same global model θ^t . This, in turn, weakens the aggregation of benign gradients in federated training, denoted by $\sum_{i \in S^t \setminus \mathcal{C}} \Delta\theta_i^t$ in Equation (7.2), in the face of the poisoned gradients $\sum_{c \in S^t \cap \mathcal{C}} \Delta\theta_c^t$. It is important to note that this observation applies to various FL training algorithms, including FedAvg and FedDC (a personalized FL training approach).

We take advantage of the scatter presented in benign gradients by reducing the angles among malicious gradients (as shown in Figure 7.2a). By consistently

¹The number of data samples per class follows a symmetric Dirichlet distribution $Dir(\alpha)$, where smaller values of α indicate a more diverse data distribution among clients.

reinforcing the aggregation of these malicious gradients throughout training iterations, we exert a pulling force on the global model θ^t , directing it towards the shared Trojaned model X . Consequently, we can significantly decrease and establish a minimum threshold for the number of compromised clients required to execute a backdoor poisoning attack successfully. We consider a poisoning attempt to be successful in a particular training round t if the updated global model θ^{t+1} moves closer to the Trojaned model X . In other words, the influence of the malicious gradients becomes dominant over the benign gradients, causing the aggregated global model to align itself with the direction of the malicious gradients, thus activating the backdoor attack.

In the worst-case scenarios, when the aggregated benign gradients are oriented in the opposite direction to the aggregated compromised gradients, and the angle β_i between gradients of an arbitrary benign client i and of the aggregated malicious gradients of all the compromised clients follows a normal distribution, i.e., $\beta_i \sim \mathcal{N}(\mu_\alpha, \sigma^2)$ as we observed in our analysis (Figure 7.2) where μ_α and σ^2 are bounded by $[0, \pi]$, we establish a lower bound on the minimum number of compromised clients $|\mathbb{C}|$ required to guarantee the attack's success in a single training round, as stated in the following theorem.

Theorem 7.1.1. *The minimum number of compromised clients the attacker need to successfully carry out backdoor poisoning in one training round in the worst case scenario is*

$$|\mathbb{C}| \geq \frac{2\delta - \sigma^2 - \mu_\alpha^2}{(a + b + 2)\delta - \sigma^2 - \mu_\alpha^2} |N|, \quad (7.5)$$

where β_i is the angle between gradients of an arbitrary benign client i and of the aggregated malicious gradients of all the compromised clients follows a normal distribution, i.e., $\beta_i \sim \mathcal{N}(\mu_\alpha, \sigma^2)$ and δ is a broken probability.

Proof. Given the diverse data distribution among clients, their gradient updates vary in direction and magnitude. To ensure the effectiveness of the attack, it is necessary for the aggregated model parameter updates at each iteration t to align with the direction of the aggregated malicious gradient $\sum_{i \in \mathcal{C}} \psi_c g_{\Delta_c}$. To capture both the direction and magnitude, we project all gradients onto the direction of the aggregated malicious gradient. This leads to the following condition:

$$\sum_{i \in \mathcal{C}} \psi_c g_{\Delta_c} + \sum_{i \in N \setminus \mathcal{C}} g_{\Delta_i} \geq 0 \quad (7.6)$$

where g_{Δ_i} is the projection of the gradient Δ_i into the direction of the malicious aggregated gradient Δ_c and ψ_c is the dynamic learning rate ($\psi_c \sim \mathcal{U}[a, b]$).

In worst-case scenarios where the benign gradients are oriented in the opposite direction to the aggregated malicious gradient, we introduce a unit vector \vec{i} to represent the direction. Thus, Equation (7.6) can be reformulated as follows:

$$\left(\sum_{c \in \mathcal{C}} \psi_c \cdot A_c \right) \cdot \vec{i} - \sum_{i \in N \setminus \mathcal{C}} [\cos(\beta_i) \cdot A_i^b \cdot \vec{i}] \geq 0 \quad (7.7)$$

where A_c and A_i^b are the magnitude of the gradients from compromised and benign clients, respectively. Since $\psi_c \sim \mathcal{U}[a, b]$, $\sum_{c \in \mathcal{C}} \psi_c \cdot A_c = A_c \cdot |\mathcal{C}| \cdot \frac{(a+b)}{2}$. To circumvent gradient exploration and prevent the server from tracking the gradients to identify some suspicious behavior patterns, we upper-bound the magnitude of the gradients by A (i.e., $\max A_c = \max A_i^b = A$). Equation (7.7) is equivalent to:

$$\begin{aligned} A \cdot |\mathcal{C}| \cdot \frac{(a+b)}{2} - \sum_{i \in N \setminus \mathcal{C}} \cos(\beta_i) \cdot A &\geq 0 \\ \Leftrightarrow |\mathcal{C}| &\geq \frac{2}{(a+b)} \sum_{i \in N \setminus \mathcal{C}} \cos(\beta_i) \end{aligned} \quad (7.8)$$

Applying Maclaurin's theorem [9] to the cosine function (as in Trigonometry [100]), we have: $\cos(\beta_i) = 1 - \frac{\beta_i^2}{2!} + \frac{\beta_i^4}{4!} = \sum_{k=0}^{\infty} (-1)^k \frac{(\beta_i)^{2k}}{(2k)!}$. Therefore, we can approximate

the term $\sum_{i \in N \setminus \mathcal{C}} \cos(\beta_i)$ with an error bounded by $\mathcal{O}\left(\frac{\sum_{i \in N \setminus \mathcal{C}} (\beta_i)^4}{4!}\right)$:

$$\sum_{i \in N \setminus \mathcal{C}} \cos(\beta_i) \approx (|N| - |\mathcal{C}|) - \frac{\sum_{i \in N \setminus \mathcal{C}} (\beta_i)^2}{2} \quad (7.9)$$

Without loss of generality, we consider the angles β_i follow a normal distribution, i.e., $\beta_i \sim \mathcal{N}(\mu_\alpha, \sigma^2)$, the lower bound on the number of compromised clients becomes:

$$\begin{aligned} |\mathcal{C}| &\geq \frac{2}{(a+b)} \sum_{i \in N \setminus \mathcal{C}} \cos(\beta_i) \\ \Leftrightarrow |\mathcal{C}| &\geq \frac{2}{(a+b)} \left[(|N| - |\mathcal{C}|) - \frac{\sum_{i \in N \setminus \mathcal{C}} (\beta_i)^2}{2} \right] \end{aligned} \quad (7.10)$$

Using Markov's inequality [35], we have:

$$P\left(\sum_{i \in N \setminus \mathcal{C}} \beta_i^2 \geq t\right) \leq \frac{\mathbb{E}\left[\sum_{i \in N \setminus \mathcal{C}} \beta_i^2\right]}{t} \quad (7.11)$$

in which

$$\begin{aligned} \mathbb{E}\left(\sum_{i \in N \setminus \mathcal{C}} \beta_i^2\right) &= \sigma^2 \mathbb{E}\left[\sum_{i \in N \setminus \mathcal{C}} \left(\frac{(\beta_i - \mu_\alpha)^2 + 2\mu_\alpha\beta_i - \mu_\alpha^2}{\sigma^2}\right)\right] \\ &= \sigma^2 \mathbb{E}\left[\sum_{i \in N \setminus \mathcal{C}} \left(\frac{\beta_i - \mu_\alpha}{\sigma}\right)^2 + 2\frac{\mu_\alpha}{\sigma^2} \sum_{i \in N \setminus \mathcal{C}} \beta_i - (|N| - |\mathcal{C}|) \frac{\mu_\alpha^2}{\sigma^2}\right] \\ &= \sigma^2 \mathbb{E}\left[\sum_{i \in N \setminus \mathcal{C}} \left(\frac{\beta_i - \mu_\alpha}{\sigma}\right)^2\right] + 2\sigma^2 \mathbb{E}\left[\frac{\mu_\alpha}{\sigma^2} \sum_{i \in N \setminus \mathcal{C}} \beta_i\right] - \mathbb{E}\left[(|N| - |\mathcal{C}|) \mu_\alpha^2\right] \\ &= \sigma^2(|N| - |\mathcal{C}|) + 2\mu_\alpha(|N| - |\mathcal{C}|)\mu_\alpha - (|N| - |\mathcal{C}|)\mu_\alpha^2 \\ &= (|N| - |\mathcal{C}|)(\sigma^2 + \mu_\alpha^2) \end{aligned} \quad (7.12)$$

Substituting Equation (7.12) into Equation (7.11), we have:

$$P\left(\sum_{i \in N \setminus \mathcal{C}} \beta_i^2 \geq t\right) \leq \frac{(|N| - |\mathcal{C}|)(\sigma^2 + \mu_\alpha^2)}{t} \quad (7.13)$$

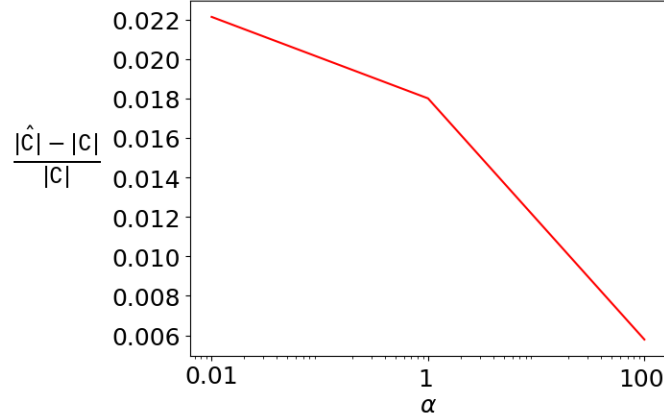


Figure 7.3 Approximation error for the lower bound of $|\mathbf{C}|$ in Theorem 7.1.1 as a function of α using the FEMNIST dataset [10].

with a broken probability δ .

To calculate t , we have:

$$\frac{(|N| - |\mathbf{C}|)(\sigma^2 + \mu_\alpha^2)}{t} = \delta \Leftrightarrow t = \frac{(|N| - |\mathbf{C}|)(\sigma^2 + \mu_\alpha^2)}{\delta} \quad (7.14)$$

Substituting Equation (7.14) into Equation (7.10), we have:

$$\begin{aligned} |\mathbf{C}| &\geq \frac{2}{(a+b)} \left[(|N| - |\mathbf{C}|) - \frac{(|N| - |\mathbf{C}|)(\sigma^2 + \mu_\alpha^2)}{2\delta} \right] \\ \Leftrightarrow |\mathbf{C}| &\geq \frac{2}{(a+b)} \left(1 - \frac{\sigma^2 + \mu_\alpha^2}{2\delta} \right) (|N| - |\mathbf{C}|) \\ \Leftrightarrow |\mathbf{C}| &\geq \frac{\frac{2}{(a+b)} \left(1 - \frac{\sigma^2 + \mu_\alpha^2}{2\delta} \right)}{1 + \frac{2}{(a+b)} \left(1 - \frac{\sigma^2 + \mu_\alpha^2}{2\delta} \right)} |N| \\ \Leftrightarrow |\mathbf{C}| &\geq \frac{2\delta - \sigma^2 - \mu_\alpha^2}{(a+b+2)\delta - \sigma^2 - \mu_\alpha^2} |N| \end{aligned} \quad (7.15)$$

Therefore, Theorem 7.1.1 holds. \square

In Theorem 7.1.1, the client sampling probability q is uniform for all benign and compromised clients in every training round. Therefore, q is unbiased and does not impact our lower bound estimation of $|\mathbf{C}|$.

In practical scenarios, the attacker can make accurate estimations of the mean μ_α and variance σ of the angles by utilizing the datasets $\{D_c\}_{c \in \mathcal{C}}$ collected by compromised clients. This enables the attacker to precisely approximate the lower bound of $|\mathcal{C}|$ with a bounded error based on concentration bounds, such as Hoeffding bound [1]. Figure 7.3 presents this relative approximation error $|\frac{\hat{|\mathcal{C}|} - |\mathcal{C}|}{|\mathcal{C}|}|$ as a function of α , where $\hat{|\mathcal{C}|}$ is the approximated lower bound of $|\mathcal{C}|$. The higher degree of diversity in benign clients' local data is, the larger the relative approximation error is. However, the relative approximation error is marginal across all the degrees of α , i.e., 2.23% given $\alpha = 0.01$ and 0.57% given $\alpha = 100$. In addition, the mean of angles μ_α and its variance σ are quite consistent from initial training rounds and throughout the training process as in our observation (Figure 7.2); therefore, the attacker can estimate the lower bound $|\mathcal{C}|$ in less than 10 training rounds. This minimizes the delay in poisoning the federated training. Importantly, our lower bound of $|\mathcal{C}|$ is practical since the attacker follows our threat model and does not have access to extra information from other clients to carry out the approximation.

Figure 7.4 visualizes this lower bound in a 3D surface of $\frac{|\mathcal{C}|}{|N|}$ as a function of μ_α and σ . Based on Theorem 7.1.1 and Figure 7.4, it is evident that a larger mean μ_α and variance σ of the angles (indicating more scattered gradients and greater diversity in the local data distribution of clients) result in a smaller number of compromised clients \mathcal{C} required to execute the COLLAPOIS attack successfully. In other words, a higher backdoor success rate is expected when the clients' local data is more diverse under the same number of compromised clients.

It is important to note that in typical DPOIS attacks (as illustrated in Figure 7.2b), the malicious local gradients $\{\Delta\theta_c^t = \theta_c^t - \theta^t\}_{c \in \mathcal{C}}$, where $\{\theta_c^t = \arg \min_{\theta^t} L(\theta^t, D_c \cup D_c^{Troj})\}_{c \in \mathcal{C}}$, exhibit a similar level of scatter as benign gradients. This is because the local Trojaned models $\{\theta_c^t\}_{c \in \mathcal{C}}$ heavily rely on diverse local data distributions, causing them to be scattered from one another in each training round.

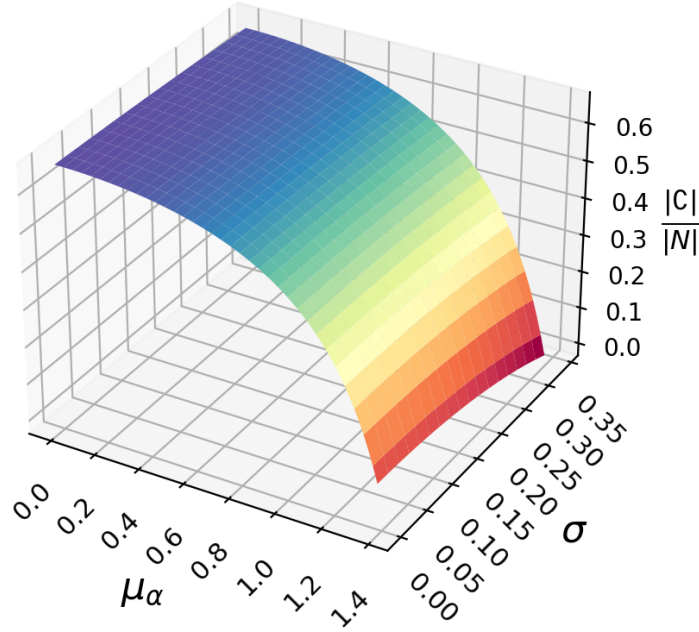


Figure 7.4 3D plot of $\frac{|C|}{|N|}$ as a function of μ_α and σ .

As the local data distributions among compromised clients become more diverse, the angles among the malicious local gradients grow larger (as depicted in Figure 7.2b). Consequently, this significantly weakens the effectiveness of DPOIS attacks.

To tackle this issue, COLLAPOIS introduces the shared Trojaned model X as a stable and optimized poisoned area. Leveraging the lower bound on the number of compromised clients, we demonstrate in the following theorem that the global FL model θ converges to a small bounded region around the Trojaned model X . This ensures that the impact of the backdoor attack is confined to a limited area.

Theorem 7.1.2. *For a compromised client c joining a round t , we have that the l_2 -norm distance between the global model θ^t and the Trojaned model X is always bounded as follows:*

$$\|\theta^t - X\|_2 \leq (1/a - 1) \|\Delta \theta_c^{t'}\|_2 + \|\zeta\|_2 \quad (7.16)$$

where $\forall t : \psi_c^t \sim \mathcal{U}[a, b]$, $a < b$, $a, b \in (0, 1]$, t' is the closest round the compromised client c participated in, and $\zeta \in \mathcal{R}^m$ is a small error rate.

Proof. At the round t' , we have that $\Delta\theta_c^{t'} = \psi_c^{t'}[X - \theta^{t'}]$. This is equivalent to $X = \frac{\Delta\theta_c^{t'}}{\psi_c^{t'}} + \theta^{t'}$. In round t , according to the findings of Theorem 7.1.1, the global model is expected to be a more severely poisoned model for the compromised client c : $\theta^t = \Delta\theta_c^t + \theta^{t'} + \zeta$. To quantify the distance between the global model θ^t and the Trojaned model X , we subtract X from θ^t as follows: $\theta^t - X = (1 - \frac{1}{\psi_c^{t'}}) \Delta\theta_c^{t'} + \zeta$. Based upon this, we can bound the l_2 -norm of the distance $\theta^t - X$ as follows:

$$\|\theta^t - X\|_2 = \|(1 - \frac{1}{\psi_c^{t'}}) \Delta\theta_c^{t'} + \zeta\|_2 \leq (\frac{1}{a} - 1) \|\Delta\theta_c^{t'}\|_2 + \|\zeta\|_2$$

Consequently, Theorem 7.1.2 holds. \square

In Theorem 7.1.2, as the global model approaches convergence, denoted by t' and t approaching the number of rounds T , the norm $\|\xi\|_2$ becomes extremely small, and $\|\Delta\theta_c^{t'}\|_2$ is bounded by a small constant τ . This ensures that the global FL model θ^T converges to a bounded and low-loss region surrounding the Trojaned model X . In other words, $\|\theta^T - X\|_2$ is minimized to a negligible value. This provides a reliable guarantee for the success of our attack.

7.1.4 Attack stealthiness

In addition to their effectiveness, the malicious gradients $\{\Delta\theta_c^t\}_{c \in \mathcal{C}}$ possess several key properties that contribute to their stealthiness, including the following:

(1) The Trojaned model X exhibits higher model utility on legitimate data samples compared to randomly initialized global FL and benign clients' local models, particularly during the early training rounds and when there is a greater diversity in the local data distribution of benign clients (indicated by smaller values of α). The resulting models achieve superior model utility on legitimate data samples by utilizing malicious gradients to train both the global and clients' local models. Consequently, COLLAPOIS demonstrates greater stealthiness compared to white-box MREPL and DPOIS attacks, as it avoids the degradation and shifts in model utility on legitimate

data samples throughout the entire poisoning process. This characteristic makes detecting our attack highly challenging during the federated training process.

(2) By ensuring that the random and dynamic learning rate ψ_c^t is exclusively known to the compromised client c , we can effectively prevent the server from tracking the Trojaned model X or detecting any suspicious behavior patterns from the compromised client. In practice, the server may be able to identify compromised clients with a certain precision value p and estimate the presence of the Trojaned model X . The server's set of identified compromised clients consists of $p \times |\mathcal{C}|$ compromised clients $\bar{\mathcal{C}}$ and $(1-p)(|N| - |\mathcal{C}|)$ benign clients \bar{L} . The estimated Trojaned model $X' = \sum_{c \in \bar{\mathcal{C}} \cup \bar{L}} \theta_c^t / |\mathcal{C}|$. The following theorem establishes a bound on the server's l_2 -norm estimation error of the Trojaned model X , denoted as $Error = \|X' - X\|_2$.

Theorem 7.1.3. *The server's estimation error of the Trojaned model X is bounded as follows:*

$$\left\| \sum_{c \in \bar{\mathcal{C}}} \frac{\Delta \theta_c^t}{p|\mathcal{C}|b} \right\|_2 \leq Error \leq \arg \max_{L \subseteq N \text{ s.t. } |L|=|\mathcal{C}|} \left\| \sum_{i \in L} \frac{\theta_i^t}{|L|} - X \right\|_2 \quad (7.17)$$

Proof. We have that

$$\begin{aligned} Error &= \|X' - X\|_2 = \left\| \sum_{c \in \bar{\mathcal{C}}} \frac{\theta_c^t}{p|\mathcal{C}|} + \sum_{i \in \bar{L}} \frac{\theta_i^t}{(1-p)(|N| - |\mathcal{C}|)} - X \right\|_2 \\ &= \left\| \sum_{c \in \bar{\mathcal{C}} \cup \bar{L}} \frac{\theta_c^t}{|\mathcal{C}|} - X \right\|_2 \end{aligned} \quad (7.18)$$

$$\begin{aligned} \text{in which } \|X' - X\|_2 &\geq \left\| \sum_{c \in \bar{\mathcal{C}}} \theta_c^t / (p|\mathcal{C}|) - X \right\|_2 = \left\| \sum_{c \in \bar{\mathcal{C}}} \frac{\Delta \theta_c^t}{p|\mathcal{C}|\psi_c^t} \right\|_2 \\ &\geq \left\| \sum_{c \in \bar{\mathcal{C}}} \frac{\Delta \theta_c^t}{p|\mathcal{C}|b} \right\|_2 \end{aligned} \quad (7.19)$$

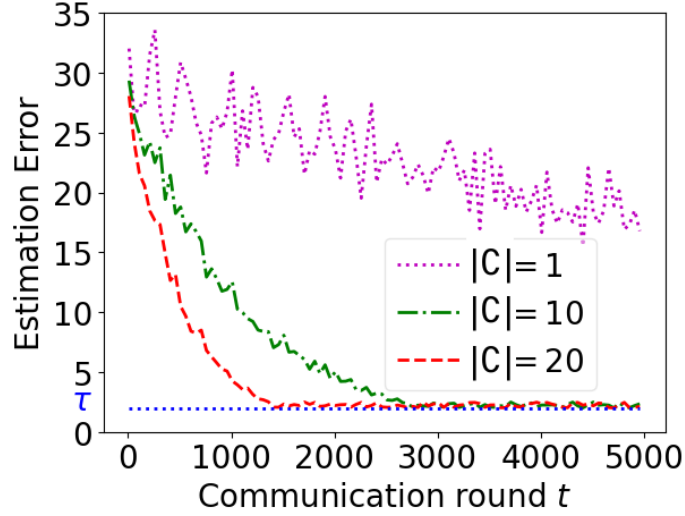


Figure 7.5 Estimation error of COLLAPOIS with a precision $p = 1$ using the FEMNIST dataset [10].

$$\text{and } \left\| \sum_{c \in \bar{\mathcal{C}} \cup \bar{L}} \theta_c^t / |\mathcal{C}| - X \right\|_2 \leq \arg \max_{L \subseteq N \text{ s.t. } |L|=|\mathcal{C}|} \left\| \sum_{i \in L} \theta_i^t / |L| - X \right\|_2 \quad (7.20)$$

From Eqs. 7.19 and 7.20, we have the following error bounds:

$$\left\| \sum_{c \in \bar{\mathcal{C}}} \frac{\Delta \theta_c^t}{p |\mathcal{C}| b} \right\|_2 \leq \text{Error} \leq \arg \max_{L \subseteq N \text{ s.t. } |L|=|\mathcal{C}|} \left\| \sum_{i \in L} \frac{\theta_i^t}{|L|} - X \right\|_2 \quad (7.21)$$

As a result, Theorem 7.1.3 holds. \square

From Theorem 7.1.3, we observe that: **(1)** The lower precision in detecting compromised clients (smaller p) results in a larger estimation error approaching the upper bound; **(2)** The smaller the upper bound of the dynamic learning rate b , the higher the lower bound of the estimation error is; and **(3)** If the malicious gradient $\Delta \theta_c^t$ is too small, we can uniformly upscale its l_2 -norm to be a small constant, denoted τ , to enlarge the lower bound of the estimation error without affecting the model utility or backdoor success rate. Figure 7.5 illustrates the lower bound of the estimation error given the most favorable precision to the server, i.e., $p = 1$, with different numbers of compromised clients $|\mathcal{C}|$. After 1,000 rounds, the estimation error is not reduced

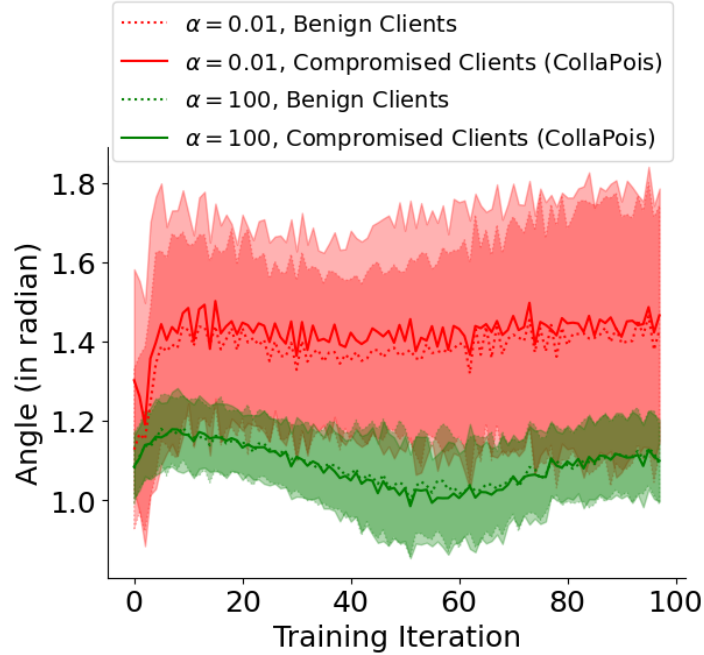


Figure 7.6 Attack stealthiness: Angles between malicious/benign gradients and sampled gradients.

Note: Compromised clients with benign clients are blended and modestly different (using the FEMNIST dataset with $\psi_c^t \sim \mathcal{U}[0.95, 0.99]$).

further since it is controlled by the lower bound with $\tau = 2$ in this study. That prevents the server from accurately estimating the Trojaned model X .

Remark. Practitioners can connect Theorem 7.1.1 and Theorem 7.1.3 to discover that: The more diverse clients’ local data is (i.e., larger values of μ_α and σ resulting in a smaller number of required compromised clients $|\mathcal{C}|$ in Equation (7.5)), the more difficult for the server to approximate the Trojaned model X is; therefore, the more stealthy the attack will be. This is because a smaller number of required compromised clients $|\mathcal{C}|$ induces a larger lower bound of the estimation error in Equation (7.17).

Although ineffective in estimating the Trojaned model X , the server can try to distinguish compromised clients by looking into the angle and magnitude of every submitted gradient. To protect malicious gradients from being detectable, the attacker can marginally adjust the dynamic learning rate ψ_c^t to seamlessly blend

each malicious gradient in the background of benign gradients. The wider the range of $\psi_c^t \sim \mathcal{U}[a, b]$, the more scattered malicious gradients in terms of angles and magnitude are, i.e., more randomness. The attacker can select a suitable range of $\mathcal{U}[a, b]$ such that the average angle and its variance between each of the malicious gradients and a set of sampled gradients (which plays a role of data background) are similar to those of benign clients. In practice, the attacker can derive sampled gradients by using the clean data collected by compromised clients $\{D_c\}_{c \in \mathcal{C}}$ and the global model θ^t . Also, these clean gradients can be used to imitate gradients from benign clients. This ensures that the attacker follows our threat model and does not access extra information from benign clients. Figure 7.6 shows that malicious and benign gradients behave alike in the average angle and its variance, given randomly sampled gradients.

Regarding the magnitude of the malicious gradients, we clip them by using the same clipping bound A , which is shared by the server to enhance the robustness of the training process, with benign gradients. Therefore, the magnitude of the malicious gradients stays well within the range of gradients from benign clients.

As a result, COLLAPOIS can hide the malicious gradients of compromised clients without affecting the effectiveness of our attack, as long as the range of $\psi_c^t \sim \mathcal{U}[a, b]$ and the clipping bound A are sufficient to not damage federated training.

7.2 Experimental Results

In our evaluation, we focus on understanding the interplay among Federated Learning (FL) performance, aggregation algorithms, diverse data distribution among clients, backdoor success rate, and client-specific defense mechanisms. To achieve our goal, we focus on answering three questions in our evaluation: **(1)** Whether COLLAPOIS is effective in poisoning FL. **(2)** How COLLAPOIS is impacted by the different degrees of data diversity? and **(3)** Is it possible to defend against COLLAPOIS, and what are the costs and limitations of such defenses?

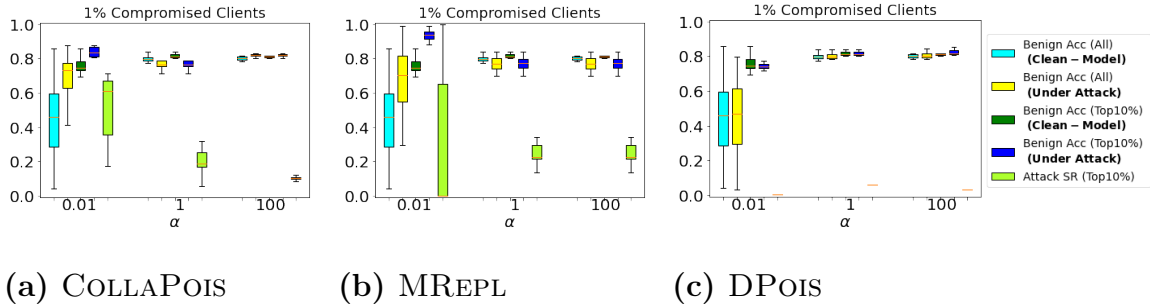


Figure 7.7 FedAvg under different backdoor attacks for the CIFAR-10 dataset.

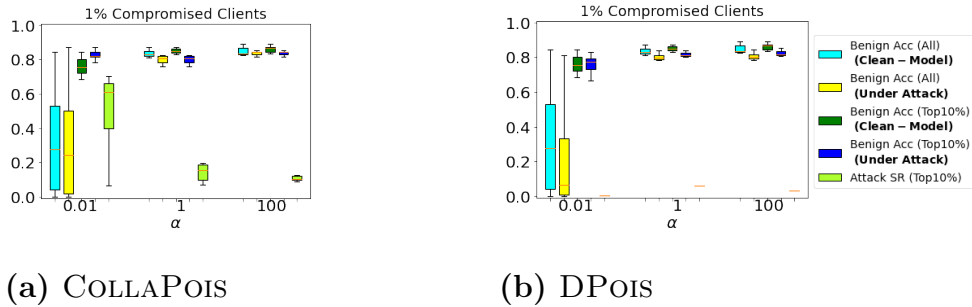


Figure 7.8 FedDC under different backdoor attacks for the CIFAR-10 dataset. Note: MREPL causes a model divergence during the federated training under the FedDC; therefore, we do not include the result from MREPL in this figure.

Data and Model Configuration. We conduct experiments on CIFAR-10 [43], FEMNIST [10], and Sentiment-140 [24] datasets. For these datasets, to control data distribution across clients in terms of classes and size of local training data, we leverage the Dirichlet distribution with different values of $\alpha \in [0.01, 100]$ as in [19, 114]. In short, the value of α is inversely proportional to the degree of diversity in data distribution. In CIFAR-10 there are 100 clients with 60,000 samples in total while in FEMNIST there are 3,400 clients with 805,263 samples in total. In Sentiment-140 we filter out 5,600 clients with over 1 million data samples. In all three datasets, we use the class 0 as the attacker’s targeted class y^{Troj} . In our experiments, the attacker randomly compromised 1% of benign clients. This percentage is lower than the number of compromised clients needed as in Theorem 7.1.1. This is because we found that COLLAPOIS is effective even at a smaller portion of compromised clients, making our attack even more effective, practical, and stealthy.

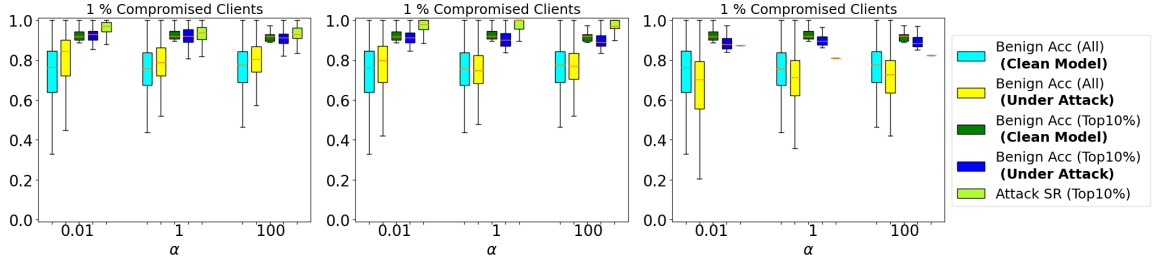
We adopt the model configuration described in [88] for these datasets, in which we use a LeNet-based network [50] with two convolution and two fully connected layers for the local model and a fully-connected network with three hidden layers and multiple linear heads per target weight tensor. We use SGD optimizer with the learning rate of 0.01 for the aggregated global model and 0.001 for the benign clients’ local models in our experiments.

We use WaNet [73], which is one of the state-of-the-art backdoor attacks, for generating poisoned data. WaNet utilizes image warping-based triggers making the modification in the backdoor images natural and unnoticeable to humans. We follow the learning setup described in [73] to generate the backdoor images that are used to train the Trojaned model X .

Evaluation Approach. We carry out the validation through three approaches. We first compare COLLAPOIS with DPOIS and MREPL [3, 53] in terms of legitimate accuracy (Benign Acc) on legitimate data samples and backdoor success rate (Attack SR) on Trojaned data samples without defense. Then, we investigate the effectiveness of adapted robust federated training algorithms under a variety of hyper-parameter settings against COLLAPOIS. Finally, we provide a performance summary of both attacks and defenses to inform the surface of backdoor risks in FL under different degrees of the diversity of clients’ local data distribution. The Benign Acc and Attack SR on average across clients using testing data are as follows:

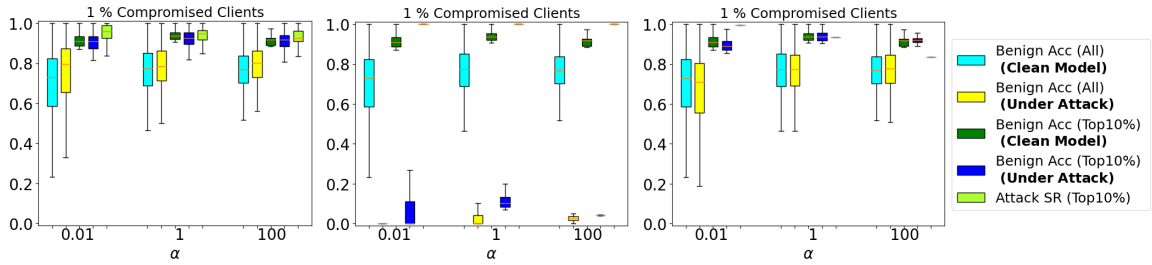
$$\begin{aligned} \text{Benign Acc} &= \frac{1}{|N|} \sum_{i \in N} \left[\frac{1}{|D_i^{test}|} \sum_{x \in D_i^{test}} \text{Acc}(f(x, \theta_i), y) \right] \\ \text{Attack SR} &= \frac{1}{|N|} \sum_{i \in N} \left[\frac{1}{|D_i^{test}|} \sum_{x \in D_i^{test}} \text{Acc}(f(x + \mathcal{T}, \theta_i), y^{Troj}) \right] \end{aligned}$$

where $x + \mathcal{T}$ is a Trojaned data sample, $\text{Acc}(y', y) = 1$ if $y' = y$; otherwise $\text{Acc}(y', y) = 0$, and D_i^{test} and $|D_i^{test}|$ are a set of testing samples of the client i and its size respectively.



(a) COLLAPois (b) MREPL (c) DPOIS

Figure 7.9 FedAvg under different backdoor attacks for the FEMNIST dataset.



(a) COLLAPois (b) MREPL (c) DPOIS

Figure 7.10 FedDC under different backdoor attacks for the FEMNIST dataset.

We conduct our evaluation using both the state-of-the-art personalized FL algorithm FedDC [19] and the widely applied FL algorithm FedAvg to demonstrate the generalizability of our attack. To provide insights into the client-specific performance, stealthiness, and risk of backdoor attacks, we report Benign Acc and Attack SR values of the top-k% impacted benign clients i , who have top-k% highest values of the sum of Benign Acc and Attack SR, as follows:

$$\text{score}_i = \frac{\sum_{x \in D_i^{\text{test}}} \left[\text{Acc}(f(x, \theta_i), y) + \text{Acc}(f(x + \mathcal{T}, \theta_i), y^{\text{Troj}}) \right]}{|D_i^{\text{test}}|}$$

Comparison with Existing Attack Methods. Figures 7.7 and 7.8 present Benign Acc and Attack SR of poisoning attacks, with 1% of clients compromised by the attacker, as a function of the degree of diversity α given clients' local data distribution in the CIFAR-10 dataset. COLLAPois achieves better performance

compared with DPOIS and MREPL. First, when the model is not under attack (i.e., clean model), COLLAPOIS does not cause notable drops in Benign Acc on all benign clients (mean value of 76%) and on the top 10% benign clients (mean value of 81%). Secondly, COLLAPOIS achieves significantly better Attack SR given diverse clients’ local data distribution (i.e., $\alpha = 0.01$). Under both FedAvg and FedDC, COLLAPOIS injects backdoors with over 60% success rate compared with DPOIS (less than 30%) and MREPL (less than 10%). Note that the Attack SR of MREPL given $\alpha = 0.01$ is highly varied with a mean is about 0.0%. Also, MREPL causes a model divergence during the federated training under the FedDC; thus, MREPL attack is noticeable to the server. When the clients’ local data distribution is not diverse or nearly identical, i.e., $\alpha = 1$ and $\alpha = 100$, having 1% compromised clients is insufficient to carry backdoor poisoning attacks effectively.

The results on the FEMNIST dataset strengthen our observation (Figures 7.9 and 7.10). COLLAPOIS achieves over 95% Attack SR on both FedAvg and FedDC which is the same as MREPL and outperforms DPOIS (less than 90% on FedAvg and over 95% on FedDC). It is worth noting that MREPL significantly degenerates the Benign Acc on FedDC (Figure 7.10b). Overall, COLLAPOIS outperforms other attacks in terms of Attack SR while having unnoticeable Benign Acc drops under poisoning. This makes our attack effective and stealthy.

Bypassing Robust Federated Training. As mentioned in [89], the robust training algorithms (even the simple ones) in FL can effectively defend against a variety of backdoor poisoning attack methods. Therefore, to show that COLLAPOIS is practical, we need to evaluate it against the state-of-the-art robust training algorithms. Based on the methods reviewed in Chapter 2, we select (1) the DP-optimizer (**DP**), (2) the norm bound (**NormBound**), (3) the Krum (**Krum**), and (4) the robust learning rate (**RLR**) to cover all different design considerations of robust training algorithms.

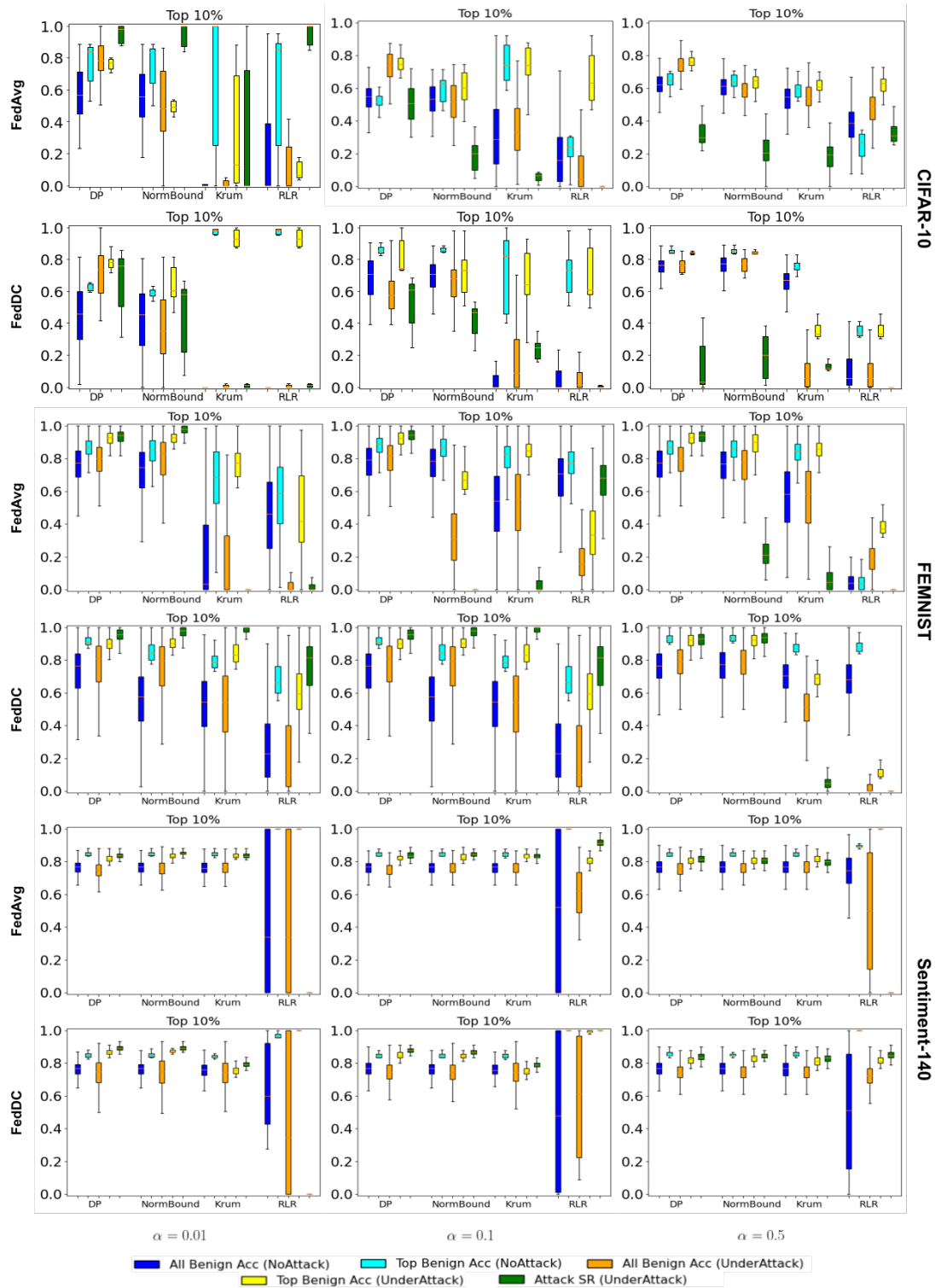


Figure 7.11 Evaluating COLLAPOIS against various robust training algorithms across different datasets and data divergence.

Figure 7.11 illustrates attack performance under different robust federated training algorithms and degrees of the diversity of benign clients’ data distribution. In the first column of Figure 7.11, it is interesting that the Krum and RLR methods are actually not effective robust training algorithms to be used in FedAvg or FedDC. In fact, applying the Krum or RLR methods damages model utility since the Benign Acc significantly decreases across different datasets. For example, in CIFAR-10, the Benign Acc after applying DP and NormBound are around 50% to 55% while applying Krum or RLR degenerates Benign Acc to almost 0% which is unusable. Although there are settings in which applying such robust training algorithms does not hurt the model’s utility (e.g., applying Krum to protect FedDC in FEMNIST), the defensive performance is also affected (i.e., the Attack SR of COLLAPOIS is over 95%) which makes it not suitable for preventing the proposed COLLAPOIS from achieving its target.

Besides the Krum and RLR, the evaluation results in Figure 7.11 show that the DP and NormBound are generally applicable in all settings. However, the drawback of applying these two robust training algorithms is that COLLAPOIS can easily bypass them and achieve high Attack SR. It is clear that all of the Attack SR of COLLAPOIS is higher than 60% while most of these values are over 90% when the benign clients’ local data distribution is diverse, i.e., $\alpha = 0.01$. Therefore, COLLAPOIS is a severe attack against FedAvg and FedDC even when DP or NormBound is applied if the data distribution among benign clients is diverse.

Degree of Local Data Diversity. In this experiment, we focus on exploring the impact of different degrees of diversity in benign clients’ local data distribution. That will inform practitioners of realistic backdoor poisoning risks in real-world FL applications. In Figure 7.11, it is clear that the Attack SR of COLLAPOIS increases when the value of α decreases from 0.5 to 0.01 indicating more diverse benign clients’ local data distribution. The Attack SR against the DP increases from less than

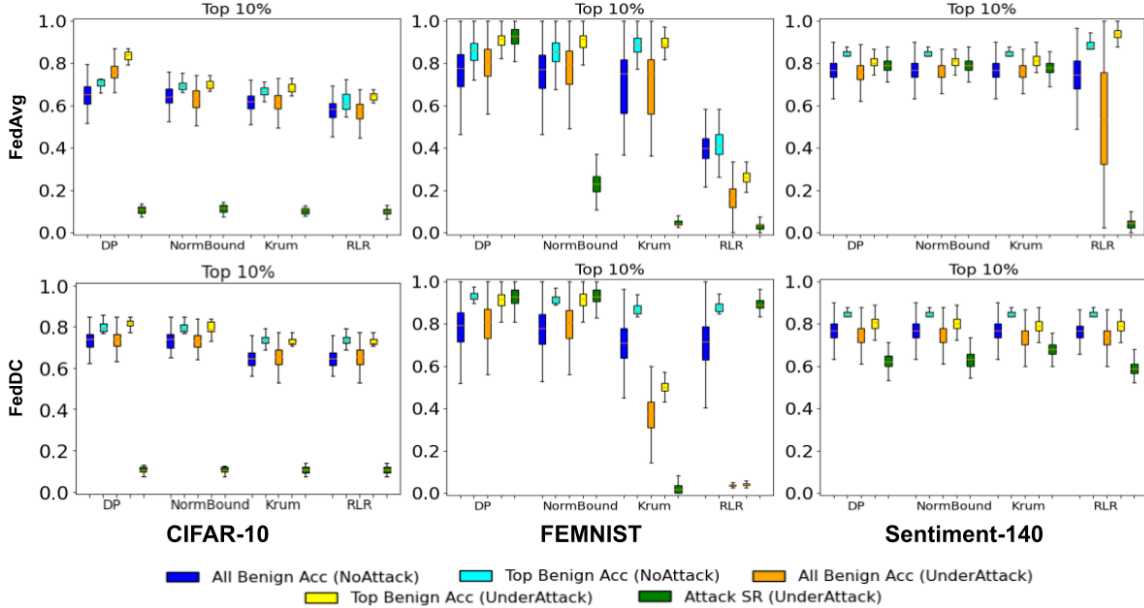


Figure 7.12 Evaluating COLLAPOIS against various robust training algorithms across different datasets with $\alpha = 100$.

30% with $\alpha = 0.5$ to around 55% with $\alpha = 0.1$ and to over 95% with $\alpha = 0.01$. We observe a similar result in Attack SR across different robust, typical, and personalized training algorithms using different datasets. Therefore, the reduced degree of diversity in benign clients' local data distribution reduces the risk of backdoor poisoning in FL. When α is extremely large, i.e., $\alpha = 100$, approaching uniform benign clients' data distribution, the Attack SR is inconsistent across datasets. It reduces to almost a random guess level on the CIFAR-10 dataset while the Attack SR is still very high, i.e., over 88.5% and 78.8% on the FEMNIST and Sentiment-140 datasets respectively (Figure 7.12). This is consistent with our theoretical analysis in Section 7.1.

Remark. Practitioners, who manage the server, can quantify and keep track of the scattered clients’ gradients and use COLLAPOIS to inform backdoor poisoning risk during the federated training process. The more scattered gradients which indicate a higher degree of data diversity among clients, the more severe risk of backdoor poisoning is. If the gradients among clients are scattered at 80+ degrees (i.e., which can be considered equivalent to $\alpha \leq 0.01$) and 1% of clients are compromised, backdoor poisoning risk can be severely high even under existing robust federated training.

7.3 Conclusion

This chapter proposes a novel backdoor poisoning attack called COLLAPOIS that exploits diverse data distribution among clients in Federated Learning (FL). Through theoretical and empirical analysis, our attack demonstrates its effectiveness, practicality, and stealthiness. We show that even with a small number of compromised clients, COLLAPOIS can successfully converge the FL model around a pre-trained Trojane model. It achieves high backdoor attack success rates when clients have diverse data, and it hinders the server’s ability to detect suspicious behaviors. Furthermore, the COLLAPOIS attack can circumvent existing backdoor defenses, particularly when clients possess diverse data distributions. The evaluation results highlight that a mere 1% of compromised clients can open a backdoor on 10% of benign clients with an impressive success rate exceeding 82% using state-of-the-art robust federated training algorithms on benchmark datasets.

CHAPTER 8

EXTENSIONS AND FUTURE DIRECTIONS

Machine learning, particularly NN models, holds immense potential for a wide range of real-world applications. However, the presence of various vulnerabilities poses a significant obstacle to its widespread adoption. These vulnerabilities act as a burden, hindering the full realization of the technology’s capabilities. Consequently, there has been a growing emphasis on research endeavors that aim to explore these vulnerabilities comprehensively and enhance the overall robustness of NN models.

This dissertation delves into the multifaceted landscape of NN vulnerabilities, addressing critical unanswered questions pertaining to training time vulnerabilities, inference time vulnerabilities, and the vulnerabilities specific to Federated trained NNs. By investigating these diverse aspects, we aim to contribute to the existing body of knowledge and provide insights that can inform the development of more resilient NN models.

While this dissertation covers a substantial breadth of research on NN vulnerabilities, it is essential to acknowledge that there is still much work to be done by the research community. The exploration of these vulnerabilities is an ongoing process, and beyond the scope of this dissertation, there lie countless avenues for further investigation. In this chapter, we aim to share our understanding and shed light on potential future directions, fostering a collaborative environment that encourages continued research efforts. By collectively addressing these challenges, we can advance the field, fortify the security of NN models, and facilitate their wider adoption in real-world applications.

8.1 Training and Inference Time Vulnerabilities

To the best of our knowledge, the vulnerabilities discussed in this work were initially introduced to the research community in [95]. Despite ongoing studies on training time and inference time vulnerabilities, it is still too early to claim the existence of a mature solution. Among the various research endeavors, we find two particular directions to be highly intriguing.

Certified defense, in the beginning, offers a promising approach that can be proven to achieve a certain level of robustness against inference time attacks (i.e., adversarial attacks). By employing certified defense, not only can we achieve resilience against attacks, but the size of the certified bound also provides a measure of prediction certainty. Moreover, recent advancements have witnessed the emergence of certified defenses tailored specifically to counter Trojan backdoor attacks [102, 106].

However, proposing a certified defense that can match the performance of state-of-the-art empirical defenses for either training time or inference time vulnerabilities poses significant challenges. The certified defense must guarantee consideration of all possible attack scenarios which is a much larger searching space that is considered by empirical defenses. To tackle this challenge, certification is typically carried out at the pixel level, assuming uniformity across all pixels. Nevertheless, such a relaxation of certified defense may prove ineffective due to the curse of dimensionality [44].

In our humble opinion, mitigating the curse of dimensionality in certified defenses by focusing on a small subset of pixels that have a significant impact on the model’s predictions is a practical approach. To achieve this, we suggest leveraging ideas from explainable AI and utilizing the correlation between individual pixels and the model’s prediction results [14]. By employing techniques such as saliency analysis [90], the trained model can identify pixels that contribute most to its decision-making process. This approach allows certified defenses to concentrate their efforts on

certifying only the subset of inputs that are highly influential, thereby mitigating the curse of dimensionality.

Another avenue of research is the utilization of **generative models**, which assert that employing them for classification purposes can enhance robustness. The inherent nature of generative tasks allows the model to possess a better grasp of high-level features and their associations with classes. When employed for classification, the features being used by the generative model contain more physical meaning which makes the model less susceptible to being fooled by attackers through indistinguishable perturbations.

However, the downside of employing generative models lies in the computational cost associated with training. Training a generative model for the same dataset is considerably more challenging than training a classification model in terms of data size, training time, and strategy. To surmount this issue, we propose exploring knowledge transfer and domain adaptation based on a trusted universal model.

Recent advancements in the development of large-scale generative models have significantly enhanced their generative power. These gigantic models, such as Stable Diffusion [84], have demonstrated remarkable capabilities in generating high-quality and diverse data. More importantly, to adapt these gigantic models to various downstream tasks, efficient fine-tuning methods have been introduced to the research community. One such method is Low-Rank Adaptation (LoRA) [32], which leverages low-rank approximations to reduce the computational complexity of fine-tuning gigantic models. By combining pre-trained gigantic models with LoRA, we can potentially address the scalability issue associated with using generative models for robust prediction.

8.2 Vulnerabilities of Federated Trained NNs

When focusing on real-world applications, FL recently becomes more popular than conventional centralized training. It is not hard to understand that the capability of jointly training a NN model without sharing the training data makes FL an attractive approach for applications with high sensitivity (e.g., medical associations and financial agencies). The mechanism that holds privacy in FL also makes it hard to identify attacking behavior when some of the participants are compromised. Therefore, we observe an increasing number and severity of vulnerabilities for Federated trained NNs.

Until now, based on our literature review, existing works that study this type of vulnerability usually can be seen as an extension of the vulnerabilities for centralized trained models. In other words, the special characteristics of Federated learning are barely being considered on both the attack and defense sides. Therefore, to better study the vulnerabilities of Federated trained models, the following directions are worth to be explored.

In **attack perspective**, the work in Chapter 7 demonstrates the effectiveness of COLLAPOIS in poisoning a subset of innocent clients, particularly when their data exhibit significant divergence. However, to escalate this threat, it is intriguing to investigate methods for conducting targeted backdoor attacks against specific clients. From an attacker’s perspective, possessing such capabilities would allow them to concentrate on high-value clients while excluding other innocent clients from infection, thereby reducing the risk of detection.

To effectively target specific clients in a backdoor attack, it is crucial to render the Trojaned model in a state of “semi-readiness” (\hat{X}). This means that the backdoor functionality embedded within the Trojaned model will only become operational after

the targeted clients perform local updates.

$$X = \arg \min_{\hat{X}} L(\hat{X}, D_t) \quad (8.1)$$

Here, X is the final Trojan backdoored model from Equation (7.1) while D_t denotes the targeted client’s training data. Achieving this objective requires the attacker to deduce the precise nature of the targeted clients’ local updates.

We envision two potential avenues through which this could be achieved. The first approach involves adopting a stronger assumption regarding the attacker’s capabilities. In this scenario, the targeted clients can be approximated by utilizing sampled auxiliary data (D_{Aux}). Specifically, we assume that the attacker is capable of sampling these auxiliary data from the same or a similar data distribution as the target client. By leveraging this auxiliary data, the attacker attempts to infer the underlying patterns and behaviors specific to the targeted clients.

$$\hat{X} = \arg \min_{\hat{X}} \|X - \arg \min_{\hat{X}} L(\hat{X}, D_{Aux})\|_p \quad (8.2)$$

We have $D_{Aux}, D_t \sim \mathcal{D}_t$ where \mathcal{D}_t is the target client’s data distribution. The $\|\cdot\|_p$ denotes the p norm. This approach draws inspiration from existing Trojan backdoor attacks customized for transfer learning scenarios [111]. By adapting and learning from attacks against transfer learning, the attacker can gain a better understanding of and exploit the unique characteristics of federated learning settings to effectively target specific clients.

The second avenue, independent of the aforementioned assumption, involves inferring the targeted clients by scrutinizing the aggregated changes observed during the collaborative learning process. To achieve this, the attacker can utilize the aggregated updates to locally approximate the training data. By applying the model extraction attack [83, 99] with these approximated training data, the attacker can

infer the target client’s data distribution.

$$\begin{aligned}
D_N &= \arg \min_D \left\| \frac{1}{N} \sum_{i \in S} \theta_i - \arg \min_{\theta}(\theta, D) \right\|_p \\
D_{N \setminus t} &= \arg \min_D \left\| \frac{1}{N} \sum_{i \in \hat{S}} \theta_i - \arg \min_{\theta}(\theta, D) \right\|_p \\
D_{Aux} &= \arg \min_{\alpha D_N + (1-\alpha) D_{N \setminus t}} L_{\mathcal{A}}(\theta_t, \alpha D_N + (1-\alpha) D_{N \setminus t}) \tag{8.3}
\end{aligned}$$

Here, S and \hat{S} denote the user selection with and without the targeted client. The corresponding approximated training data are D_N and $D_{N \setminus t}$. The $\alpha D_N + (1-\alpha) D_{N \setminus t}$ denotes a weighted sampled training data which is used to infer the targeted client’s training data. Lastly, the model extraction attack is represented by minimizing its loss $L_{\mathcal{A}}$. This inferred information about the target client’s data distribution can then be utilized in a similar way as before to customize the proposed Trojaned model and ensure it is in a state of “semi-readiness”. The advantage of this approach is that the attacker does not rely on the assumption that the auxiliary data is sampled from the exact same distribution as the target client, which makes it more practical and applicable in real-world scenarios.

Exploring these strategies holds promise for attackers to target high-value clients in federated learning while reducing the risk of being detected. By understanding the nuances of the backdoor vulnerabilities in federated learning, we can expand the depth of our knowledge in this area and develop stronger defenses against such attacks. These proposed ideas contribute to the ongoing efforts in strengthening the security and robustness of federated learning systems.

From a **defense viewpoint**, the evaluation results presented in Chapter 7 emphasize the limited effectiveness of current defense mechanisms employed in federated learning (FL) and personalized federated learning (pFed) when faced with highly divergent clients’ data. Approaches such as Differential Privacy (DP) and NormBound prove insufficient in providing adequate protection, while methods like

Krum and RLR inflict significant damage on the model’s utility for benign examples. Consequently, the exploration of effective defense strategies to address the challenges posed by highly divergent clients’ data remains largely unexplored.

Our observations indicate that divergent updates from benign clients tend to dilute each other. In contrast, compromised clients collaborating with each other can facilitate the dissemination of the backdoored model to innocent clients. Upon careful study and analysis of the above defense mechanisms, we find that they have failed to defend against attacks for different reasons. For DP and NormBound, the issue lies in the inadequate mitigation of divergence across benign updates through random smoothing and gradient clipping. Consequently, adversarial updates can still steer the model toward the Trojaned weights. On the other hand, Krum and RLR fail for a different reason as these defenses excessively restrict divergence. While they prevent adversarial updates, they also negatively impact the model’s utility for benign examples. Given these limitations, we posit that an effective defense strategy should strike a superior trade-off compared to existing methods.

Based on our understanding, a possible approach to effectively defend attacks against FL with highly divergent clients’ data is to follow a “divide-and-conquer” strategy. In other words, the defense should initially cluster clients into different groups to reduce the divergence within each group. If compromised clients send similar adversarial updates, they can be clustered into a dedicated group and isolated from benign clients. Conversely, compromised clients will be separated into multiple groups, effectively diluting their influence. During this step, clustering by basic statistics of clients’ training data is the most straightforward way. Otherwise, the vector angles between clients’ updates could also be utilized for clustering. Considering the high dimensionality, a further enhanced approach is applying dimension reduction on clients’ updates first before calculating vector angles.

When clients are clustered, across different groups, the defense should ensure that only indirect and soft knowledge sharing is allowed during training, further restricting cooperation between compromised clients from different groups. In the meantime, the knowledge sharing should also balance the models' performance across groups instead of favoring any particular one. Lastly, within each group, a robust training algorithm (e.g., Krum or RLR) needs to be applied. Since the clients within each group are much more similar, these robust training algorithms are expected to prevent adversarial updates without significantly damaging the model's utility for benign examples.

By pursuing this "divide-and-conquer" approach, we can potentially develop more effective defense strategies for federated learning with highly divergent clients' data. However, it is important to note that further research is needed to explore the feasibility, efficiency, and scalability of this approach.

8.3 Other Vulnerabilities

While this dissertation extensively explores several well-known vulnerabilities in NN models across different environments, it is important to acknowledge that there exist additional vulnerabilities that pose potential threats to applications relying on NN models. In this section, we highlight a selection of intriguing topics that warrant further investigation. Moreover, drawing inspiration from the integration of adversarial attacks and Trojan backdoors discussed in this dissertation, we aim to foster the exploration of novel synergistic attacks by sharing these topics. By uncovering and understanding these unexplored vulnerabilities, we can advance the field's knowledge and further develop robust defense strategies against emerging threats.

Model inversion attacks: In this type of attack, an adversary aims to reconstruct sensitive input data based on the outputs of a trained NN model

[18, 112, 30]. By leveraging the information revealed through the model's predictions, an attacker can infer details about the input data that should have remained private. For instance, consider a scenario where a NN model is trained to classify medical images as either benign or malignant. An adversary who gains access to the model's predictions may attempt a model inversion attack to reconstruct the original medical images, potentially revealing sensitive patient information such as personal identifiers or medical conditions. The implications of such attacks can be severe, emphasizing the need to develop robust defenses to safeguard against the reconstruction of private data through model inversion techniques.

Model extraction attacks: In these attacks, adversaries aim to extract the knowledge or structure of a target model by training their own substitute model using queries to the target model [83, 99]. By leveraging this substitute model, attackers can gain insights into the inner workings and decision-making processes of the target model, potentially revealing proprietary information and compromising the model's integrity. For instance, consider a scenario where a company has developed a highly accurate and proprietary speech recognition model. An adversary with access to the model's predictions may attempt a model extraction attack to create a substitute model that closely mimics the target model's behavior. This can lead to unauthorized replication or reverse engineering of the target model, jeopardizing the company's competitive advantage and intellectual property. Developing effective defenses against model extraction attacks is essential to protect the confidentiality and proprietary information embedded in NN models.

Fairness and bias vulnerabilities: These vulnerabilities arise when models exhibit unfair or biased behavior towards certain demographic groups, leading to discriminatory outcomes [69, 71]. Ensuring fairness and mitigating bias in NN models is of utmost importance to prevent unjust treatment and promote ethical decision-making. For instance, consider a machine learning model used in a loan

application process that unintentionally discriminates against applicants from certain racial or gender groups. This bias can perpetuate existing inequalities and unfairly impact individuals' access to financial opportunities. Addressing fairness and bias vulnerabilities requires developing techniques to identify, measure, and mitigate bias in model training and decision-making processes. By striving for fairness, we can build trust in the deployment of NN models and work towards more equitable and inclusive systems.

REFERENCES

- [1] Hoeffding’s inequality. <https://web.stanford.edu/class/cs229t/2017/Lectures/concentration-slides.pdf>.
- [2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.
- [3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948, 2020.
- [4] Rubeena Banu, Vanishri Arun, N Shankaraiah, and V Shyam. Meta-cognitive neural network method for classification of diabetic retinal images. In *2016 Second International Conference on Cognitive Computing and Information Processing (CCIP)*, pages 1–5. IEEE, 2016.
- [5] Rodrigo Benenson. Classification datasets results, 2018. [Online; accessed 06-April-2018].
- [6] Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd with majority vote is communication efficient and fault tolerant. *arXiv preprint arXiv:1810.05291*, 2018.
- [7] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.
- [8] Lawrence D Brown, T Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical science*, pages 101–117, 2001.
- [9] R Caimmi. The integral newton’s and maclaurin’s theorems in tensor form. *Astronomische Nachrichten: Astronomical Notes*, 324(3):250–264, 2003.
- [10] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *NeurIPS*, 2019.
- [11] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. pages 39–57, 2017.
- [12] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Exploiting shared representations for personalized federated learning. In *International Conference on Machine Learning*, pages 2089–2099. PMLR, 2021.

- [13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [14] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.
- [15] Min Du, Ruoxi Jia, and Dawn Song. Robust anomaly detection and backdoor attack detection via differential privacy. *arXiv preprint arXiv:1911.07116*, 2019.
- [16] Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. *arXiv preprint arXiv:1807.10272*, 2018.
- [17] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX*, pages 1605–1622, 2020.
- [18] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.
- [19] Liang Gao, Huazhu Fu, Li Li, Yingwen Chen, Ming Xu, and Cheng-Zhong Xu. Feddc: Federated learning with non-iid data via local drift decoupling and correction. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2022.
- [20] Liang Gao, Huazhu Fu, Li Li, Yingwen Chen, Ming Xu, and Cheng-Zhong Xu. Feddc: Federated learning with non-iid data via local drift decoupling and correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10112–10121, 2022.
- [21] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019.
- [22] GDPR. The european data protection regulation. {<https://gdpr-info.eu/>}, 2018.
- [23] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems*, 33:19586–19597, 2020.
- [24] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.
- [25] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press Cambridge, MA, 2016.

- [26] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.
- [27] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
- [28] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.
- [29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [30] Zecheng He, Tianwei Zhang, and Ruby B Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, 2019.
- [31] Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitras, and Nicolas Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping. *arXiv preprint arXiv:2002.11497*, 2020.
- [32] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [33] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *Workshop on faces in 'Real-Life' Images: detection, alignment, and recognition*, 2008.
- [34] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.
- [35] Markov’s inequality. Markov’s inequality. {https://en.wikipedia.org/wiki/Markov%27s_inequality}.
- [36] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. Model-reuse attacks on deep learning systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 349–363, 2018.
- [37] Kaidi Jin, Tianwei Zhang, Chao Shen, Yufei Chen, Ming Fan, Chenhao Lin, and Ting Liu. A unified framework for analyzing and detecting malicious examples of dnn models. *arXiv preprint arXiv:2006.14871*, 2020.

- [38] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, and Rachel Cummings. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.
- [39] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.
- [40] Samil Karahan, Merve Kilinc Yildirim, Kadir Kirtac, Ferhat Sukru Rende, Gultekin Butun, and Hazim Kemal Ekenel. How image degradations affect deep cnn-based face recognition? In *2016 International Conference of the Biometrics Special Interest Group (BIOSIG)*, pages 1–5. IEEE, 2016.
- [41] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143, 2020.
- [42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.
- [43] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
- [44] Aounon Kumar, Alexander Levine, Tom Goldstein, and Soheil Feizi. Curse of dimensionality on randomized smoothing for certifiable robustness. In *International Conference on Machine Learning*, pages 5458–5467. PMLR, 2020.
- [45] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.
- [46] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *International Conference on Learning Representations*, 2017.
- [47] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *International Conference on Learning Representations*, 2017.
- [48] Alex Lamb, Jonathan Binas, Anirudh Goyal, Dmitriy Serdyuk, Sandeep Subramanian, Ioannis Mitliagkas, and Yoshua Bengio. Fortified networks: Improving the robustness of deep networks by modeling the manifold of hidden representations. *arXiv preprint arXiv:1804.02485*, 2018.
- [49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [50] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

- [51] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.
- [52] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In *Advances in Neural Information Processing Systems*, pages 9464–9474, 2019.
- [53] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. *Advances in neural information processing systems*, 29, 2016.
- [54] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *NeurIPS 2019 Workshop on Federated Learning for Data Privacy and Confidentiality*, 2019.
- [55] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368, 2021.
- [56] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.
- [57] Yiming Li, Tongqing Zhai, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shutao Xia. Rethinking the trigger of backdoor attack. *arXiv preprint arXiv:2004.04692*, 2020.
- [58] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting adversarial examples in deep networks with adaptive noise reduction. *arXiv preprint arXiv:1705.08378*, 2017.
- [59] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307*, 2018.
- [60] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020.
- [61] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.
- [62] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *ACM SIGSAC CCS*, 2019.

- [63] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017.
- [64] Shuang Luo, Didi Zhu, Zexi Li, and Chao Wu. Ensemble federated adversarial training with non-iid data. *arXiv preprint arXiv:2110.14814*, 2021.
- [65] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
- [66] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
- [67] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agueray Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.
- [68] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *ICLR*, 2018.
- [69] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 54(6):1–35, 2021.
- [70] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. pages 135–147, 2017.
- [71] Shira Mitchell, Eric Potash, Solon Barocas, Alexander D’Amour, and Kristian Lum. Prediction-based decisions and fairness: A catalogue of choices, assumptions, and definitions. *arXiv preprint arXiv:1811.07867*, 2018.
- [72] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.
- [73] Anh Nguyen and Anh Tran. Wanet - imperceptible warping-based backdoor attack. In *ICLR*, 2021.
- [74] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. Defending against backdoors in federated learning with robust learning rate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9268–9276, 2021.
- [75] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. Defending against backdoors in federated learning with robust learning rate. *AAAI*, 2021.

- [76] Ren Pang, Hua Shen, Xinyang Zhang, Shouling Ji, Yevgeniy Vorobeychik, Xiapu Luo, Alex Liu, and Ting Wang. A tale of evil twins: Adversarial inputs versus poisoned models. In *Proceedings of ACM SAC Conference on Computer and Communications (CCS)*, 2020.
- [77] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.
- [78] Nicolas Papernot and Patrick McDaniel. Extending defensive distillation. *arXiv preprint arXiv:1705.05264*, 2017.
- [79] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *ACM Asia Conference on Computer and Communications Security*, 2017.
- [80] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.
- [81] NhatHai Phan, My T Thai, Han Hu, Ruoming Jin, Tong Sun, and Dejing Dou. Scalable differential privacy with certified robustness in adversarial learning. In *37th International Conference on Machine Learning*, 2020.
- [82] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *ICLR*, 2021.
- [83] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *arXiv preprint arXiv:2007.07646*, 2020.
- [84] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.
- [85] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.
- [86] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 32(8):3710–3722, 2020.

- [87] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on mnist. 2018.
- [88] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. Personalized federated learning using hypernetworks. In *International Conference on Machine Learning*, pages 9489–9502, 2021.
- [89] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1354–1371, 2022.
- [90] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
- [91] Chuanbiao Song, Kun He, Liwei Wang, and John E Hopcroft. Improving the generalization of adversarial training with domain adaptation. *arXiv preprint arXiv:1810.00740*, 2018.
- [92] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *International Conference on Learning Representations*, 2017.
- [93] Octavian Suciuc, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1299–1316, 2018.
- [94] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *International Workshop on Federated Learning for Data Privacy and Confidentiality at NeurIPS*, 2019.
- [95] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.
- [96] Canh T Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, 33:21394–21405, 2020.
- [97] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*, pages 480–501, 2020.

- [98] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.
- [99] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.
- [100] Trigonometry. Power series for cosine and sine. {[https://en.wikibooks.org/wiki/Trigonometry/Power_Series_for_Cosine_and_Sine#:~: text=cos%20%E2%81%A1%20\(%20x%20\)%20%3D%201,2%20n%20\(%202%20n%20\)%20!](https://en.wikibooks.org/wiki/Trigonometry/Power_Series_for_Cosine_and_Sine#:~:text=cos%20%E2%81%A1%20(%20x%20)%20%3D%201,2%20n%20(%202%20n%20)%20!)}
- [101] Vikas Verma, Alex Lamb, Christopher Beckham, Aaron Courville, Ioannis Mitliagkis, and Yoshua Bengio. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. *arXiv preprint arXiv:1806.05236*, 2018.
- [102] Binghui Wang, Xiaoyu Cao, Neil Zhenqiang Gong, et al. On certifying robustness against backdoor attacks via randomized smoothing. *arXiv preprint arXiv:2002.11750*, 2020.
- [103] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, page 0, 2019.
- [104] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33:16070–16084, 2020.
- [105] Ning Wang, Yang Xiao, Yimin Chen, Yang Hu, Wenjing Lou, and Y Thomas Hou. Flare: defending federated learning against model poisoning attacks via latent space representations. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 946–958, 2022.
- [106] Maurice Weber, Xiaojun Xu, Bojan Karlaš, Ce Zhang, and Bo Li. Rab: Provable robustness against backdoor attacks. *arXiv preprint arXiv:2003.08904*, 2020.
- [107] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. Crfl: Certifiably robust federated learning against backdoor attacks. In *International Conference on Machine Learning*, pages 11372–11382, 2021.
- [108] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019.

- [109] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International conference on learning representations*, 2020.
- [110] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*, 2016.
- [111] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2041–2055, 2019.
- [112] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.
- [113] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML*, pages 5650–5659, 2018.
- [114] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks, 2019.
- [115] Zhengming Zhang, Ashwinee Panda, Linyue Song, Yaoqing Yanag, Michael Mahoney, Prateek Mittal, Ramchandran Kannan, and Joseph Gonzalez. Neurotoxin: Durable backdoors in federated learning. In *International Conference on Machine Learning*, pages 26429–26446, 2022.
- [116] Pinlong Zhao, Zhouyu Fu, Qinghua Hu, Jun Wang, et al. Detecting adversarial examples via key-based network. *arXiv preprint arXiv:1806.00580*, 2018.
- [117] Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-free knowledge distillation for heterogeneous federated learning. In *International Conference on Machine Learning*, pages 12878–12889, 2021.