Building Trustworthy Machine Learning Systems in Adversarial Environments

Ning Wang

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Wenjing Lou, Chair
Yiwei Hou
Angelos Stavrou
Laura Freeman
Yue Wang
Yimin Chen

April 24, 2023
Arlington, Virginia

Keywords: adversarial machine learning, anomaly detection, differential privacy

Building Trustworthy Machine Learning Systems in Adversarial Environments

Ning Wang

(ABSTRACT)

Modern AI systems, particularly with the rise of big data and deep learning in the last decade, have greatly improved our daily life and at the same time created a long list of controversies. AI systems are often subject to malicious and stealthy subversion that jeopardizes their efficacy. Many of these issues stem from the data-driven nature of machine learning. While big data and deep models significantly boost the accuracy of machine learning models, they also create opportunities for adversaries to tamper with models or extract sensitive data. Malicious data providers can compromise machine learning systems by supplying false data and intermediate computation results. Even a well-trained model can be deceived to misbehave by an adversary who provides carefully designed inputs. Furthermore, curious parties can derive sensitive information of the training data by interacting with a machine-learning model. These adversarial scenarios, known as poisoning attack, adversarial example attack, and inference attack, have demonstrated that security, privacy, and robustness have become more important than ever for AI to gain wider adoption and societal trust.

To address these problems, we proposed the following solutions: (1) FLARE, which detects and mitigates stealthy poisoning attacks by leveraging latent space representations; (2) MANDA, which detects adversarial examples by utilizing evaluations from diverse sources, i.e, model-based prediction and data-based evaluation; (3) FeCo which enhances the robustness of machine learning-based network intrusion detection systems by introducing a novel

representation learning method; and (4) DP-FedMeta, which preserves data privacy and improves the privacy-accuracy trade-off in machine learning systems through a novel adaptive clipping mechanism.

Building Trustworthy Machine Learning Systems in Adversarial Environments

Ning Wang

(GENERAL AUDIENCE ABSTRACT)

Over the past few decades, machine learning (ML) has become increasingly popular for enhancing efficiency and effectiveness in data analytics and decision-making. Notable applications include intelligent transportation, smart healthcare, natural language generation, intrusion detection, etc. While machine learning methods are often employed for beneficial purposes, they can also be exploited for malicious intents. Well-trained language models have demonstrated generalizability deficiencies and intrinsic biases; generative ML models used for creating art have been repurposed by fraudsters to produce deepfakes; and facial recognition models trained on big data have been found to leak sensitive information about data owners.

Many of these issues stem from the data-driven nature of machine learning. While big data and deep models significantly improve the accuracy of ML models, they also enable adversaries to corrupt models and infer sensitive data. This leads to various adversarial attacks, such as model poisoning during training, adversarially crafted data in testing, and data inference. It is evident that security, privacy, and robustness have become more important than ever for AI to gain wider adoption and societal trust.

This research focuses on building trustworthy machine-learning systems in adversarial environments from a data perspective. It encompasses two themes: securing ML systems against security or privacy vulnerabilities (security of AI) and using ML as a tool to develop novel security solutions (AI for security). For the first theme, we studied adversarial attack

detection in both the training and testing phases and proposed FLARE and MANDA to secure matching learning systems in the two phases, respectively. Additionally, we proposed a privacy-preserving learning system, DP-FedMeta, to defend against privacy inference attacks. We achieved a good trade-off between accuracy and privacy by proposing an adaptive data clipping and perturbing method. In the second theme, the research is focused on enhancing the robustness of intrusion detection systems through data representation learning.

Dedication

Dedicated to my mom and dad.

Acknowledgments

I would like to express my sincere gratitude to my advisor Dr. Wenjing Lou, for her generous support, thoughtful guidance, and wise mentorship throughout my academic journey. Dr. Lou introduced me to the exciting world of cybersecurity and helped me discover my passion for research in this field. Her unwavering patience and constant encouragement made it possible for me to earn my Ph.D. degree. The only way I could imagine to give back this support is to become such a wonderful advisor for my future students. I am also deeply grateful to my co-advisor Dr. Y. Thomas Hou, for his generous support and thoughtful guidance in every milestone of my academic journey.

I also want to express my deep appreciation to all other committee members, Dr. Starvrou, Dr. Freeman, Dr. Wang, and Dr. Chen for their time, thoughtful feedback, and challenging questions, which helped me to sharpen my ideas and arguments.

I am also grateful to all my fellow colleagues in the CNSR lab. Their collaborative spirit has fostered a nurturing environment, greatly enhancing my research. I have gained immensely from our engaging conversations and their valuable insights. Special thanks to Dr. Yimin Chen for training me to develop essential research skills from the beginning of my journey.

My last thank goes to my family and friends. They have supported me through the ups and downs of my life, and their unconditional love and support have given me the strength to complete this journey. Special thanks to my friend, Shuo Lei, for her constant encouragement. Without her support and accompany, I could not stay productive in my research during the pandemic.

Funding Acknowledgements

This work was supported by the Office of Naval Research under Grant N00014-19-1-2621, the National Science Foundation under Grants CNS-1837519 and CNS-1916902, and the Virginia Commonwealth Cyber Initiative (CCI).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern AI systems, particularly with the rise of big data and deep learning in the last decade, have greatly improved our daily life and at the same time created a long list of controversies. The machine learning (ML) models trained on big data have been shown to leak sensitive information about the data owners; the same generative ML models used for creating art have been used by fraudsters for generating deepfakes; well-trained language models have been shown to exhibit generalizability deficiency and intrinsic bias. Meanwhile, AI systems are often subject to malicious and stealthy subversion that jeopardizes their efficacy, represented by model poisoning in training and adversarially crafted data in testing. It is evident that security, privacy, robustness, and fairness have become more important than ever for AI to gain wider adoption and societal trust.

Machine learning, especially deep learning, has advanced its accuracy with the availability of big data. However, this abundance of data also introduces various vulnerabilities into learning systems, such as data manipulation, where an attacker injects malicious data into the training dataset to disrupt the training process. In distributed learning systems like federated learning (FL), poisoning attacks can be even more potent, as attackers can manipulate both data and local models, leading to data poisoning attacks and model poisoning attacks, respectively. The most sophisticated stealthy MPAs can conceal their presence by keeping their model update parameters similar to those of benign model updates while still accomplishing their malicious objectives [12, 15, 36]. And existing defenses, like byzantine-

resilient mechanisms, have demonstrated limitations in detecting meticulously crafted model poisoning attacks (MPA). A successful MPA can compromise the learning system by causing a model to produce any targeted predictions. Thus it is crucial to develop effective defenses against such novel, stealthy MPAs.

Even when the training process is well-protected, machine learning models may still exhibit undesirable behavior when confronted with an adversarial user during inference. The most prominent inference-phase attack is the adversarial example (AE), which can deceive a well-trained model into making incorrect predictions by applying a carefully crafted, small disturbance to a test input. Adversarial training methods [44, 90] and image-preprocessing techniques [51, 73, 74, 96, 140, 166] have provided a protection layer against AEs but shown some limitations. First, adversarial training necessitates the creation and subsequent retraining using crafted AEs. Furthermore, though image preprocessing methods can disrupt subtle adversarial perturbations, they are less effective for other data types. With numerous potential attacks on the horizon and the potential for them to affect various tasks beyond computer vision, there is a pressing need for a detection model that can generalize to different AE attacks and input formats.

In addition to adversarial attacks, data variability presents a significant challenge to machine learning systems, particularly in anomaly detection applications such as intrusion detection systems (IDS) [30, 91, 99] and malware detection[120, 172]. An anomaly-based IDS learns a model representing normal behavior and triggers an alarm when an incoming instance deviates from this normal behavior beyond a specified security threshold. While capable of detecting novel attacks, machine learning-based IDSs suffer from high false positives compared to traditional signature-based IDSs. The substantial variability in training data exacerbates this issue. It is difficult to develop a model or profile that encompasses all benign network traffic flows due to their extensive variability. To reduce the false positives, we need to build

a model to learn a stable and consistent representation of normal data.

Furthermore, the increased concentration of data needed by DL has raised widespread concerns over data privacy, leading to data privacy regulations such as General Data Protection Regulation (GDPR[1]) in European Union and California Consumer Privacy Act (CCPA[2]) in the US. In many cases, moving the data to a central location is often impossible due to legal restrictions, such as those imposed by Health Insurance Portability and Accountability Act (HIPAA[3]). Therefore, more and more data are now stored distributively at edge nodes or end devices close to their sources rather than at a central location. Federated learning (FL) [62, 72, 87] has emerged as a new paradigm to enable collaborative training over distributed private data. In FL, participants jointly train a global model without sharing their private data. Despite its nature of keeping data locally, the federated meta-learning framework is still prone to inference-based privacy attacks on individual clients' data, including the membership inference attack [98, 174] and model inversion [157]. An adversary can recover the private training data [154, 157] or sensitive partial information [98] by leveraging a specific inference model. Meanwhile, prior wisdom alludes that Differential privacy (DP) [31] can be used to provide rigorous privacy guarantee to FL algorithms by adding noise to gradients update in a controlled manner [43, 88, 158]. The adoption of DP into FL may lead to a significant decrease in model utility. Taking the popular FedAvg [87] algorithm as an example, naively adopting DP into FedAvg may cause 2-10 times training loss than the original model [160]. It is critical to improve the model's accuracy while satisfying a minimum privacy protection requirement.

In this dissertation, I address the above challenges sequentially:

1) Training Phase Security [152]: Federated learning has been deployed in numerous popular

[1]https://gdpr.eu/
[2]https://oag.ca.gov/privacy/ccpa
[3]https://www.hhs.gov/hipaa/index.html

applications, including Google's next-word prediction on Android Gboard. It enables a central server to learn a machine-learning model by aggregating model weights from distributed local devices. However, it is vulnerable to Byzantine nodes that damage the integrity of the global model by sending malicious model updates. Recent model poisoning attacks advanced in stealthiness by crafting model parameters close to a benign model's parameter. Existing defenses relying on analyzing model parameters have limited effectiveness in detecting these carefully crafted models. We developed a detection mechanism, FLARE, that leveraged intermediate model representation to differentiate stealthy malicious models from benign ones. FLARE effectively reduced the attack success rate of multiple state-of-the-art attack strategies to an extremely low level (i.e., less than 0.001).

2) Testing Phase Security [149]: Adversarial example (AE), discovered in computer vision, has shown great power in misleading a well-trained model to make incorrect predictions by only injecting a small perturbation into an input data record. We launched state-of-the-art attacks against network intrusion detection systems (IDSs) and discovered that AE attacks also applied to IDS models, and the attack success rate was as high as 95%. To tackle AEs, we proposed an AE detector named MANDA. Different from the well-known traditional defenses (i.e., adversarial training), MANDA did not require knowledge of attack strategy and made no change to the original ML model. Our novel manifold-based detection achieved as high as 98% true positive rate with less than 5% false positive rate.

3) Robust Machine Learning for Intrusion Detection [150]: Intrusion detection in network systems has benefited from advances in machine learning (ML) due to its ability to detect zero-day attacks. However, ML-based intrusion detection systems are prone to high false positives compared to traditional signature-based systems. To overcome this challenge, we developed a contrastive learning-based intrusion detection system that extracts key common properties of benign variations to improve the accuracy of the model for the benign class.

To address privacy concerns and apply our detection model to IoT systems, we integrated a federated learning framework into the contrastive learning-based detection method and introduced a system called FeCo. We conducted extensive experiments on the NSL-KDD and BaIoT datasets, which showed that FeCo outperformed state-of-the-art detection methods with a significant accuracy improvement of up to 8%.

4) Training Data Privacy Protection [153]: Federated meta-learning enables collaborative model training without data sharing and fast model customization with small local data. Federated meta-learning solutions are susceptible to privacy inference attacks as the global model, which encodes clients' training data, is available to all clients and the central server. Differential privacy (DP) is commonly used as a countermeasure against privacy inference attacks. However, incorporating DP into federated meta-learning is complex due to the accuracy-privacy trade-off and the model hierarchy associated with the meta-learning component. To address this issue, we developed DP-FedMeta, a novel differentially private federated meta-learning architecture that offers privacy protection for hierarchical models. DP-FedMeta employs an adaptive gradient clipping method and a one-pass meta-training process, resulting in improved model accuracy while maintaining a comparable level of privacy protection to state-of-the-art solutions.

In summary, this dissertation covers the interdisciplinary research on security and AI, either using machine learning as a tool to construct novel security solutions (AI for security) or securing machine learning systems against security or privacy vulnerability (security of AI). The major contributions are summarized as follows:

- We developed a detection mechanism, FLARE, that leveraged intermediate representation to differentiate stealthy malicious models from benign ones. FLARE utilizes latent-space representation for evaluating the trust score of models. We further provide a metric to quantify the latent-space representation differences and assign a trust

score to models. FLARE minimizes the impact of MPAs by aggregating model updates weighted by their trust scores. FLARE effectively reduced the attack success rate of multiple state-of-the-art attack strategies to an extremely low level (i.e., less than 0.001).

- We systematically investigate practical AE attacks and defenses in recent ML-based intrusion detection systems. We propose, MANDA, to detect potential AEs in machine learning systems. MANDA exploits unique features we observe while trying to categorize AE attacks from the viewpoint of machine learning model and data manifold. Based on our AE categorization, MANDA combines two building blocks (i.e., Manifold and DB) to achieve effective AE detection. MANDA generalizes well to different types of data and various attacks and significantly improves the detection rate against various powerful AE attacks.

- We propose a novel method for building the "norm" in an anomaly-based IDS by learning new representations for network traffic based on contrastive learning. With the proposed new method, the learned representations of benign inputs lie only in a small cluster, enabling FeCo to extract a stable template for benign inputs. Extensive evaluation results show that representation learning in FeCo significantly boosts its detection accuracy compared to previous works. The two-step feature selection scheme in FeCo not only reduces the risk of overfitting and also reduces computation complexity as a result of smaller input dimensionality, making FeCo more suitable for resource-

  constrained IoT devices.

- We introduce a differentially private federated meta-learning architecture, dubbed DP-FedMeta, to enable privacy-preserving collaborative model training. The framework preserves user privacy with rigorous privacy guarantees and enables

heterogeneous users for fast model adaptation. DP-FedMeta caters to two practical trust levels on the central server by featuring two variants of DP mechanisms. The proposed adaptive gradient clipping method in DP-FedMeta maximally preserves model accuracy while guaranteeing a target level of privacy.

The dissertation is organized as follows. In Chapter 2, we study the powerful model poisoning attacks and propose a detection and mitigation method to secure machine learning systems from adversarial manipulations in the training phase. In Chapter 3, we further study the testing time vulnerability of machine learning systems. We examine the effectiveness of adversarial example attacks in network intrusion detection systems and propose to detect the adversarially perturbed samples in the testing phase. We proposed to improve the robustness of machine-learning-based network intrusion detection systems in Chapter 4. A privacy-preserving machine learning system is introduced in Chapter 5. And the trade-off between accuracy and privacy is studied. Finally, we summarized our work and provided our thoughts in Chapter 6.

We acknowledge that there still remain various security issues in the machine learning system to analyze and resolve. We hope the study on defending against adversarial attacks and the efforts to improve the robustness of anomaly detection methods will contribute to a secure and efficient machine learning ecosystem in the future.

Chapter 2

Training-Time Security: Model Poisoning Attacks and Defenses

(Copyright notice[1])

2.1 Introduction

Machine learning (ML) is changing the ways people live and do business in every sector of our society. The success of ML, especially deep learning (DL), relies on the availability of powerful computers and a massive amount of training data. However, learning systems that require all the data to be fed into a learning model running on a central server pose serious privacy concerns. For example, the transmission of health data across certain organizational boundaries may violate security and privacy rules such as those imposed by the Health Insurance Portability and Accountability Act (HIPAA[2]). Federated learning (FL) [57, 63, 87], which enables a group of intelligent agents to jointly learn a model while keeping their private data at their local devices, emerges as a promising new learning framework to address client

[1]This chapter previously appeared as a part of a conference paper published in ACM ASIACCS 2022. ©2022 Copyright held by the owner/author(s). Reprinted, with permission, from 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 (ASIACCS '22), pages 946-958, 2022[152].

[2]https://www.hhs.gov/hipaa/index.html

data privacy problems.

FL has been applied in many popular applications, such as next-word prediction on Android Gboard by Google [50] and credit risk control by WeBank [156]. In an FL system, a large number of distributed clients cooperatively contribute to the learning process by uploading the gradients of their local models (or model weights) to the parameter server (PS) through multiple iterations without sharing the raw data at the clients. At the beginning of an FL task, PS initializes a global model. In each learning iteration, PS distributes the current global model parameters to selected clients. Each selected client continues to train the received model with its local data independently by following a predefined learning protocol. At the end of each learning iteration, PS collects and aggregates updates from clients using a gradient aggregation rule such as FedAvg [87]. PS then updates its global model and after multiple iterations PS outputs the final global model.

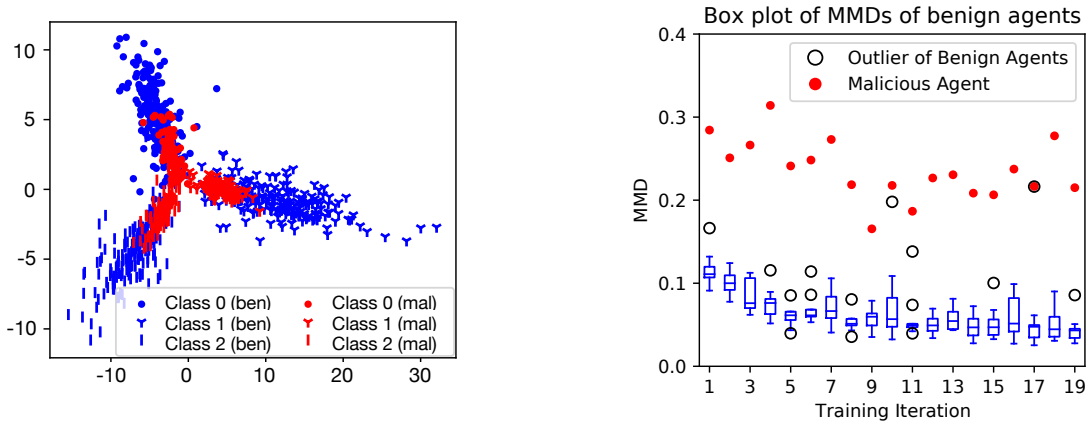
Despite many salient features of FL and its tremendous success in many applications, it has been shown recently that FL is vulnerable to model poisoning attacks (MPAs) [15, 24, 36, 48, 130]. In an MPA, the attacker (i.e., a malicious client) manipulates or crafts its model parameters sent to the PS in the aim of corrupting the global model by either increasing the prediction error (untargeted attacks) [36] or controlling the prediction on targeted inputs (backdoor attacks) [12, 15]. It is shown in [15] that even a single malicious attacker could deteriorate the global model accuracy and succeed in controlling the model output on chosen input data.

A potential countermeasure to MPAs is Byzantine resilient aggregation rules (BRARs) [16, 32, 170], which enable PS to learn an accurate global model when a bounded number of clients are malicious (i.e., Byzantine). Compared to straightforward aggregation rules that linearly combine the model updates (e.g., FedAvg [87]), BRARs (e.g., Krum [16]) seek to provide statistical methods that are not abused by Byzantine values. To this end, BRARs leverage

outlier-robust measures [54], e.g., median, trimmed estimator, to compute the center of updates despite the presence of Byzantine updates. Another line of defenses [71, 125] resorts to using anomaly detection methods to detect malicious local model updates and excludes them from the aggregation. MPAs have seen an increase in stealthiness and sophistication. The state-of-the-art MPAs [15, 36] can craft malicious model updates very similar to benign ones, breaking existing BRARs. Both the BRARs and ML-based defenses explore the model parameter space for detecting anomalous updates; they nonetheless show limited effectiveness in defending against the state-of-the-art MPAs [15, 36]. Many ML-based defenses [41, 71, 125] also need to collect a dataset of labeled benign and malicious models beforehand.

In this chapter, we tackle the MPA challenge of FL through a new angle—the latent space representation of a model. We first make an important observation that even though the poisoned model parameters are very close to those of benign models, their representations in the latent space provided an auxiliary input dataset, tend to diverge from those of benign models. Specifically, we target the penultimate layer representation (PLR) vector in the latent space and plot the PLRs of both attack-free models and poisoned models in Figure 2.1a. It shows that the clean/benign PLRs follow the same distribution while the poisoned/malicious PLRs follow a different one. We made such observations consistently across different datasets and different neural network architectures. Besides the visual differences, to obtain quantifiable discrepancy, FLARE measures the distance (i.e., maximum mean discrepancy (MMD) [46]) between the PLRs of any two models. The average MMD scores of both poisoned models and clean models are illustrated in Figure 2.1b, which confirms that PLR is a highly differentiating feature for poisonous models.

Based on the above observation, we propose FLARE (Federated learning+LAtent-space REpresentations) to protect FL systems against state-of-the-art MPAs. FLARE features a novel methodology to estimate the trust score of a local model update by exploiting the



(a) PLRs of 3-class data (b) Distribution of PLR distance

Figure 2.1: PLR distribution of benign models and malicious models using the fMNIST dataset [162].

similarity between its PLR to the PLR of others. Compared to defenses that only look into the model parameters, the PLR-based trust estimation enables FLARE to prevail in defending against carefully crafted malicious model updates. To estimate the trust score, FLARE computes a PLR sequence for each local model, which takes a very small auxiliary data at the PS. FLARE then exploits PLRs to distinguish malicious model updates from benign ones. Under the assumption that malicious clients are fewer than honest clients, FLARE assigns a trust score to each model update based on the pairwise PLR discrepancies among all model updates, in that those farther from the benign distribution are assigned lower scores. Finally, we employ a soft decision regime that aggregates model updates weighted by their trust scores. It is worth noting that FLARE performs trust score estimation based on the most recently received model parameters in each federated learning iteration, and it does not require collecting a dataset of model parameters beforehand, which yields efficiency advantages compared to existing ML-based defenses [41, 71, 125].

The major contributions of this chapter are summarized as follows:

- We propose FLARE, a novel detection and aggregation algorithm for FL to defend

against state-of-the-art MPAs. Based on the key observation that PLRs of poisoned models tend to diverge from those of benign models, FLARE utilizes PLR for evaluating the trust score of a model update in FL. Based on the MMD of different local models' PLRs, FLARE features a trust estimation mechanism that assigns a trust score to each client in every learning iteration. FLARE minimizes the impact of MPAs by aggregating model updates weighted by their client trust scores.

- Through theoretical analysis, we provide an Euclidean-distance-based interpretation on PLRs of deep neural network (DNN), justifying PLR as a promising measure to estimate the trust score of a model update.

- Extensive experimental results demonstrate the effectiveness of FLARE for defending against state-of-the-art MPAs. FLARE outperforms existing defenses in terms of decreasing the attack success rate of MPAs. FLARE achieves consistent performance across various attack methods, and datasets, demonstrating the generality of the approach.

2.2 Background and Related Work

Federated learning (FL), in a nutshell, allows a group of distributed clients to contribute their locally computed model parameter updates to the global model at the parameter server. The parameter server is responsible for distributing the initial model, collecting model parameter updates from agents, aggregating them through a certain aggregation rule, and adding the result to the global model. Eyeing on this FL paradigm, a class of stealthy attacks named model poisoning attacks (MPAs) has been demonstrated to be a significant threat to the security of FL systems [12, 15, 36, 100]. In an MPA, a compromised local agent attempts to corrupt the training process of FL by providing the parameter server with

carefully manipulated model parameters in each training iteration, with the aim of gradually degrading the FL model efficacy without being detected.

To protect the global model from malicious local updates in FL systems, BRARs were proposed in the literature, exampled by Krum [16], Coomed, Trimmed Mean [170], and Bulyan [32]. BRARs tackle the Byzantine attack/failure scenario in FL where a client does not follow the predefined learning protocol and sends arbitrary model updates to the PS. Technically, BRARs can bound the gap between the aggregated gradient and the true mean (i.e., without Byzantine clients) to a small value. Based on this feature, BRARs can partially address the MPA threat by preventing or downgrading the impact of some malicious model updates. Below we briefly introduce four state-of-the-art BRARs. Krum [16] selects one of n received updates $\{\delta_1, ..., \delta_n\}$ whose distance to all the remaining updates is the smallest. Coomed [170] selects the coordinate-wise median of n received updates as the final result. Trimmed Mean [170] first excludes the largest k values and the smallest k values in each coordinate. Then it calculates the average value of the remaining $(n - 2k)$ items. Bulyan [32] is a combination of Krum and Trimmed Mean. Bulyan firstly recursively applies Krum to select $(n - 2k)$ updates out of the total n updates. Then it applies Trimmed Mean to the selected $(n-2k)$ updates to obtain the final result. We will use these BRARs for comparative analysis and evaluation.

Besides BRARs, a number of anomaly detection mechanisms are proposed to detect malicious local model updates. Shen et al. [125] proposed Auror to protect FL from malicious updates by filtering out-of-distribution parameters from the received model parameters; Fung et al. [41] proposed FoolsGold to identify poisoning Sybils based on the model similarity of client updates. Li et al. [71] proposed a spectral-anomaly-detection-based framework that detects abnormal model updates based on their low-dimensional embeddings. Zhao et al. [176] proposed PDGAN for detecting poisoned models. PDGAN reconstructs training data from

model updates and audits the accuracy for each participant model by using the generated data and removes clients with accuracy lower than a predefined threshold. [6] uses a set of validating clients to determine if the (global) model update derived in that round has been subject to a poisoning injection. That is, clients validate the global model on their local data and vote for accepting or rejecting the model through a feedback loop. [19] proves that the majority vote mechanism with ensemble federated learning is secure against MPA. The most relevant work to ours is FLTrust [18]. FLTrust bootstraps a trust score for each client based on its directional deviation from the server model update and computes the average of the local model updates weighted by their trust scores as a global model update.

In the meantime, MPAs have seen an increase in stealthiness and sophistication. The backdoor MPAs proposed by Bhagoji et al. [15] incorporate a penalty on the distance between the crafted model parameters and the benign model parameters into its optimization objective. Bagdasaryan et al. [12] developed a generic constrain-and-scale technique that incorporates the evasion of defenses into the attacker's loss function during training. Similar techniques have been adopted in later works [136, 163] to achieve evasion of defenses. Meanwhile, Fang et al. [36] proposed untargeted MPAs to degrade the overall accuracy of the FL system by deviating the crafted model parameters from the true gradient direction. These MPAs [12, 36] have demonstrated their capability in evading existing defenses, e.g., Krum, Trimmed Mean, Auror and FoolsGold. [12] shows that an attacker is able to craft a malicious model satisfying that the Euclidean distance between the crafted model and any benign model is comparable or even less than the Euclidean distance among different benign models. Moreover, this crafted model can still misclassify an input to a target label. This attack makes the defenses by exploring Euclidean distance of model parameters useless and leads us to reconsider the defense for the MPAs.

We observe that most of the malicious model detection mechanisms [41, 71, 125], BRARs [16,

32, 170], and client credibility aggregation mechanisms (e.g., FLTrust [18] and [11, 84]) build their defense by directly analyzing the model updates from agents in the model parameter space. We also observe that due to the high dimensionality of the FL model as well as the non-smooth loss function, two models that are seemingly close in the parameter space may have dramatically different loss functions. These defenses are likely to make miss detection on a malicious model that is carefully crafted to be similar to benign models in the parameter space. Based on this key insight, we propose to detect malicious local models by analyzing the latent-space features of models.

Table 2.1: Symbol definition.

Symbol	Definition
PS	parameter server
C_i	the i-th client
n	number of clients
w_i	model parameters of the i-th client
θ	global model parameters
δ_i	model parameter update from the i-th client
$\mathcal{D}$	auxiliary dataset, $\lvert\mathcal{D}\rvert = m$
m	number of data records in the auxiliary dataset
c	number of classes
x	input image with dimensionality of $\mathrm{x} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$
r	penultimate layer representation, $\mathrm{r} \in \mathbb{R}^o$
q	output probability vector, $\mathrm{q} \in \mathbb{R}^c$
R	a sequence of r, $R = \{\mathrm{r}_1, \mathrm{r}_2, ..., \mathrm{r}_m\}$
f	mapping function from input to probability vector, $f : \mathrm{x} \rightarrow \mathrm{q}$
g	mapping function from input to PLR, $g : \mathrm{x} \rightarrow \mathrm{r}$
σ	mapping function from PLR to probability vector $\sigma : \mathrm{r} \rightarrow \mathrm{q}$
ω_i	model weights from the penultimate layer to the i-th output neuron
Ω	model weights from the penultimate layer to output neurons $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$
N_i	count of C_i being selected as others' nearest neighbor
S_i	trust score of the i-th received model update

2.3 System Model

2.3.1 Federated Learning with Trust Scores

We consider a typical FL network with one parameter server PS and n participating clients $\{C_i\}|_{i\in[n]}$ (we define $[n] := \{1, 2, ..., n\}$). The definition of frequently used symbols is shown in Table 2.1. Each client manages a local model (e.g., a neural network). At PS, the model weights of C_i are $\mathrm{w}_i \in \mathcal{W} \subseteq \mathbb{R}^d$, wherein $\mathcal{W}$ is the parameter space and d is the presumed model dimensionality. The global model parameter is denoted by $\theta \in \mathcal{W}$. We denote the model update from C_i as $\delta_i = \mathrm{w}_i - \theta$. Moreover, PS maintains a vector of trust scores for all clients, denoted $\{S_i\}_{i\in[n]}$. During normal operation, an FL task executes in iterations with PS acting as the model distributor and aggregator at the cloud side.

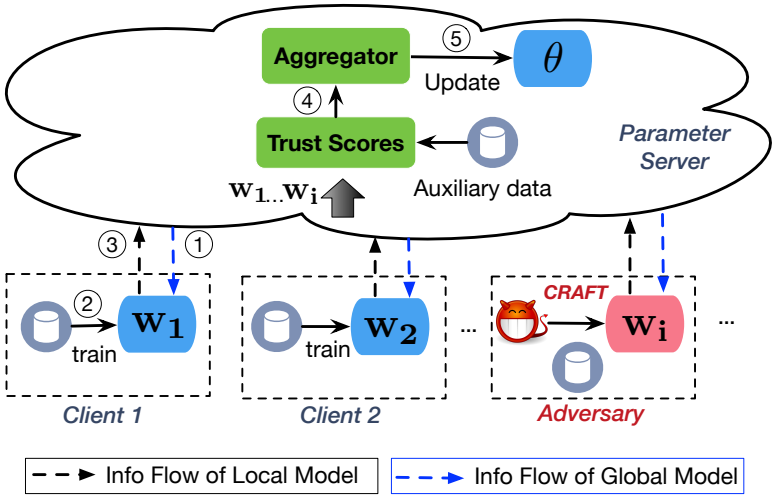


Figure 2.2: Trust score-enabled federated learning system model.

Figure 2.2 illustrates the FL system model. At the system onset, PS initializes θ. Then each training iteration works as follows: ① PS first selects multiple clients and distributes θ to them; ② each of the selected clients, say C_i, initializes $\mathrm{w}_i = \theta$ and trains the model with its local data; ③ after the local training terminates, C_i provides its model update δ_i to PS;

④ PS uses the local model updates provided by the clients to compute a trust score S_i for every client C_i; ⑤ finally, PS aggregates local model updates weighted by their trust scores and updates the global model by $\theta \leftarrow \theta + \sum_i S_i \delta_i$. At the end of the FL task, PS outputs the final global model.

We remark that the trust scores do not exist in the original FL formulation. As we show later, they allow our system FLARE to be responsive to malicious model updates before the aggregation step. We also assume PS has an auxiliary dataset $\mathcal{D} = \{x_i\}_{i \in [m]}$ containing a small number of records (e.g., $m = 10$), which will be used for PLR-based trust score evaluation in step ④ . $\mathcal{D}$ can be obtained as long as we have one or more trusted clients in the FL system, who are willing to contribute to the system's security.

2.3.2 Threat Model

We assume the population of malicious clients is less than $0.5n$, in line with prior work [16, 18, 32, 170]. Meanwhile, every malicious client is a white-box adversary and can mount MPAs on the system, following from the state-of-the-art MPAs [15, 36].

White-Box Adversary. Being a valid FL client, the attacker has access to both the global model parameters and the model updates of other clients. Typically, the attacker estimates the model updates of other clients using a dummy model trained on its own clean data. Compared to a white-box attacker, a black-box attacker would only have access to the global model parameters. We opt for the challenging white-box adversary model to demonstrate the strength and effectiveness of our proposed defense.

Model Poisoning Attacks. According to the adversary goal, there are two types of MPAs: untargeted attacks and backdoor attacks. For an untargeted attack, the attacker aims to degrade the overall model accuracy. For a backdoor attack, the attacker aims to control

the predictions on chosen input data records without degrading the overall prediction performance on other input data records. In specific, we use the two untargeted attacks in [36], which deviate the global model toward the opposite of the attack-free direction. We use the two backdoor attacks in [15], in which the adversary crafts malicious local models so as to inject a backdoor/trigger into the global model. The adversary maintains its stealth by decreasing the distance between the crafted model parameters and benign model parameters.

2.3.3 Goals and Challenges

We aim to develop a robust PS-side system for FL that tackles the state-of-the-art MPAs, as legacy defenses fail to defend against these attacks. Besides leveraging the aforementioned insight on model behaviors, to achieve this goal, we identify the following practical challenges. First, our system needs to detect a subtle MPA where the crafted malicious model is statistically similar to benign models in model parameter space. Second, we consider the white-box adversary—our system should be effective even if the attacker is aware of the defense strategy. Third, there are two types of MPAs on FL—untargeted attacks and backdoor attacks. Untargeted attacks aim to increase the prediction error rate of the global model, while backdoor attacks take a stealthier form as they aim to control the prediction on targeted inputs without degrading the overall model performance. Our defense should be effective against both MPA types. We will answer these challenges in the rest of the chapter and show that our proposed FLARE defense can address them effectively.

2.4 Penultimate Layer Representation

We introduce the motivation, theoretical foundation, and perspective for employing penultimate layer representations (PLRs) as a defense mechanism against MPAs.

2.4.1 PLR Basics

We consider a convolutional neural network (CNN) as an example. Let's assume a CNN $f : \mathbb{R}^{d1\times d2\times d3} \rightarrow \mathbb{R}^c$ maps points $x \in \mathbb{R}^{d1\times d2\times d3}$ to a c-dimensional probability vector $\mathrm{q} \in \mathbb{R}^c$, where c represents the number of classes. We consider an image input with $d1$, $d2$, and $d3$ representing the width, height, and number of channels of an image, respectively. Suppose the last layer of the network is a softmax layer. We denote the mapping function from the input to the penultimate layer (i.e., the layer before the last layer) as $g : \mathbb{R}^{d1\times d2\times d3} \rightarrow \mathbb{R}^o$. The output of function g is a PLR, represented by $\mathrm{r} \in \mathbb{R}^o$. We use σ to denote the mapping function from PLR to the output probability vector, $\sigma : \mathrm{r} \in \mathbb{R}^o \rightarrow \mathrm{q} \in \mathbb{R}^c$. The mapping functions are illustrated in Figure 2.3.

2.4.2 Power of PLR in Separating MPAs

Next, we demonstrate that PLR possesses a strong differentiation capability in detecting malicious models crafted by advanced attacks, in contrast to model parameters.

As shown in Figure 2.3, the output of a c-class classification model is a confidence/probability vector $\mathrm{q} = [q_1, q_2, ..., q_c]$, where q_k represents the likelihood the input is classified as class k and $\sum_{k=1}^{c} q_k = 1$. We denote the weights connecting the penultimate layer and the last layer as $\Omega = [\omega_1, , \omega_2, ..., \omega_c]$. And $\omega_k \in \mathbb{R}^o$ represents the weight from the penultimate layer to the

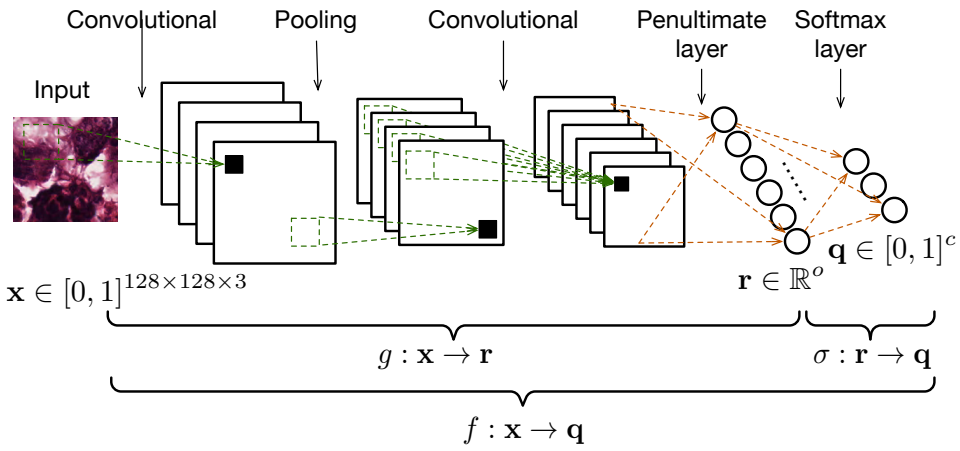


Figure 2.3: Illustration of the penultimate layer in a convolutional neural network. The mapping from the penultimate layer representation to the output probability vector is denoted by σ.

k-th neuron in the output layer. According to the softmax function, q_k is calculated as

$$q_k = \frac{\exp(\mathrm{r}^T \omega_k)}{\sum_{i=1}^{c} \exp(\mathrm{r}^T \omega_i)}. \tag{2.1}$$

We further theoretically analyze the relationship between PLRs and the output probability using the Euclidean distance metric. The interpretation is depicted in the following Proposition.

Proposition 2.1. In a c-class NN classifier where the last two layers are fully connected and the last layer is a softmax layer, the output probabilities of any two class k and l ($\forall k, l \in [c]$ and $k \neq l$) satisfy that $q_k > q_l$ if

$$\|\mathrm{r} - \omega_l\|_2 - \|\mathrm{r} - \omega_k\|_2 \geq C_{kl}, \tag{2.2}$$

where r represents the PLR of an input data record and ω_k is the weights connecting to the k-th neuron of the output layer. $||\mathrm{r} - \omega_k||_2$ denotes the Euclidean distance between r and template ω_k, i.e., $||\mathrm{r} - \omega_k||_2 = \mathrm{r}^T\mathrm{r} - 2\mathrm{r}^{\mathrm{T}}\omega_k + \omega_k^T\omega_k$. C_{kl} is a constant and $C_{kl} = \omega_l^T\omega_l - \omega_k^T\omega_k$.

PROOF. The output probability vector of a c-class classifier is $[q_1, q_2, ..., q_c]$. q_k is calculated by:

$$q_k = \frac{\exp(\mathrm{r}^T \omega_k)}{\sum_{i=1}^{c} \exp(\mathrm{r}^T \omega_i)}. \tag{2.3}$$

where ω_k denotes the weights connecting the penultimate layer and the k-th neuron in the softmax layer, and we name ω_k as template k as it is related to class k. And r denotes the penultimate layer representation. The Euclidean distance between r and template ω_k is computed by:

$$||\mathrm{r} - \omega_k||_2 = \mathrm{r}^T\mathrm{r} - 2\mathrm{r}^{\mathrm{T}}\omega_k + \omega_k^T\omega_k. \tag{2.4}$$

We then calculate the logit $\mathrm{r}^{\mathrm{T}}\omega_k$ from Eq. (2.4) by

$$\mathrm{r}^{\mathrm{T}}\omega_k = -\frac{1}{2}||\mathrm{r} - \omega_k||_2 + \frac{1}{2}\mathrm{r}^T\mathrm{r} + \frac{1}{2}\omega_k^T\omega_k. \tag{2.5}$$

We substitute $\mathrm{r}^{\mathrm{T}}\omega_k$ in Eq. (2.3) with the Eq. (2.5) and get q_k as:

$$\begin{aligned} q_k &= \frac{\exp(-\frac{1}{2}||\mathrm{r} - \omega_k||_2) \cdot \exp(\frac{1}{2}\mathrm{r}^T\mathrm{r}) \cdot \exp(\frac{1}{2}\omega_k^T\omega_k)}{\sum_i^c \exp(-\frac{1}{2}||\mathrm{r} - \omega_i||_2) \cdot \exp(\frac{1}{2}\mathrm{r}^T\mathrm{r}) \cdot \exp(\frac{1}{2}\omega_i^T\omega_i)} \\ &= \frac{\exp(-\frac{1}{2}||\mathrm{r} - \omega_k||_2) \cdot \exp(\frac{1}{2}\omega_k^T\omega_k)}{\sum_i^c \exp(-\frac{1}{2}||\mathrm{r} - \omega_i||_2) \cdot \exp(\frac{1}{2}\omega_i^T\omega_i)} \\ &= \frac{\exp(-\frac{1}{2}||\mathrm{r} - \omega_k||_2 + \frac{1}{2}\omega_k^T\omega_k)}{\sum_i^c \exp(-\frac{1}{2}||\mathrm{r} - \omega_i||_2 + \frac{1}{2}\omega_i^T\omega_i)} \end{aligned} \tag{2.6}$$

For any two classes, say class k and class l. If $q_k \geq q_l$, the inequality (2.7) must be satisfied.

$$-\frac{1}{2}||\mathrm{r} - \omega_k||_2 + \frac{1}{2}\omega_k^T\omega_k \geq -\frac{1}{2}||\mathrm{r} - \omega_l||_2 + \frac{1}{2}\omega_l^T\omega_l \tag{2.7}$$

This inequality can be rewritten as

$$||\mathrm{r} - \omega_l||_2 - ||\mathrm{r} - \omega_k||_2 \geq \omega_l^T \omega_l - \omega_k^T \omega_k \tag{2.8}$$

The right hand of this inequality is a constant. We use C_{kl} to represent it, i.e., $C_{kl} = \omega_l^T \omega_l - \omega_k^T \omega_k$. We rewrite Eq. (2.8) as

$$||\mathrm{r} - \omega_l||_2 - ||\mathrm{r} - \omega_k||_2 \geq C_{kl} \tag{2.9}$$

Eq. (2.9) implies that if the distance between representation r and template l is greater than the distance between representation r and template k by a constant C_{kl}, then the probability of assigning the input to class k is larger than assigning it to class l.

Proposition 2.1 suggests that the smaller the distance between r and the template ω_k (while keeping the distance between r and other templates fixed), the greater the likelihood that r is classified as class k. In this case, we can consider the template ω_k as the cluster center of class k. Classification can be determined by comparing a target PLR with all the c templates. For each pair of classes, such as k and l, the input is more likely to belong to class k if Eq. (2.2) is satisfied, or class l otherwise. Based on Proposition 1, we hypothesize that the PLRs of inputs belonging to one specific class exhibit a relatively small Euclidean distance to the corresponding template, potentially resulting in a consistent pattern (i.e., a cluster centered at the template). We then analyze how the distortion in PLR (i.e., $\|\mathrm{r}_1 - \mathrm{r}_2\|_2$) translates to the final output probability vector.

Proposition 2.2. Assuming $\sigma : \mathrm{r} \in \mathbb{R}^o \rightarrow \mathrm{q} \in \mathbb{R}^c$ maps a PLR to a probability vector. For any $\mathrm{r}_1, \mathrm{r}_2$, we have

$$\|\mathrm{q}^1 - \mathrm{q}^2\|_2 \leq \|\Omega\|_2 \|\mathrm{r}_1 - \mathrm{r}_2\|_2, \tag{2.10}$$

where r_1 and r_2 are the PLR of two input x_1 and x_2 respectively. q^1 and q^2 are the corresponding output probability vectors.

PROOF. We use $\mathrm{z}_1, \mathrm{z}_2 \in \mathbb{R}^c$ to denotee the logits calculated from $\mathrm{r}_1, \mathrm{r}_2 \in \mathbb{R}^o$, i.e., $\mathrm{z}_1 = \Omega^T r_1$ and $\mathrm{z}_2 = \Omega^T \mathrm{r}_2$. And $\Omega \in \mathbb{R}^{o \times c}$ is the weights connecting the penultimate layer and the last layer.

The softmax function is denoted as $\zeta : \mathbb{R}^c \to \mathbb{R}^c$. And the output probability vectors for input x_1 and x_2 are calculated by $\mathrm{q}^1 = \zeta(\mathrm{z}_1)$ and $\mathrm{q}^2 = \zeta(\mathrm{z}_2)$. According to the property of the softmax function, the Lipchitz modulus of ζ is less than one [123]. Then we can calculate

$$\|\mathrm{q}^1 - \mathrm{q}^2\|_2 \leq \|\mathrm{z}_1 - \mathrm{z}_2\|_2. \tag{2.11}$$

We then substitute z_1 and z_2 in Eq. eq:ptoz with $(r_2)^T\Omega$ and $(r_1)^T\Omega$ respectively. And we get

$$\|\mathrm{q}^1 - \mathrm{q}^2\|_2 \leq \|(\mathrm{r}_1)^T\Omega - (\mathrm{r}_2)^T\Omega\|_2 \leq \|\Omega\|_2\|\mathrm{r}_1 - \mathrm{r}_2\|_2 \tag{2.12}$$

Proposition 2.2 implies that a distance in the PLR space will translate into the discrepancy of the corresponding prediction probability. If the PLR distance $\|\mathrm{r}_1 - \mathrm{r}_2\|_2$ is small enough, then the final classification result (i.e., prediction probability) will be similar.

To further support our findings from the two propositions, we conducted empirical visualizations. In this visualization, depicted in Figure 2.4, we assume there are ten participating clients in an FL system, including nine benign clients and one attacker. The attacker manipulates its model parameters by implementing a well-known backdoor MPA [15]. We collect the ten local models and calculate a PLR for each local model using the same data point. We plot the ten versions of PLR in a 2-D space. Figure 2.4 presents ten versions

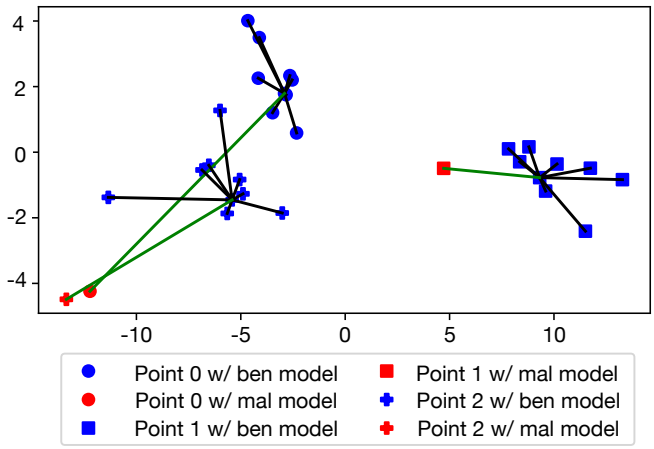


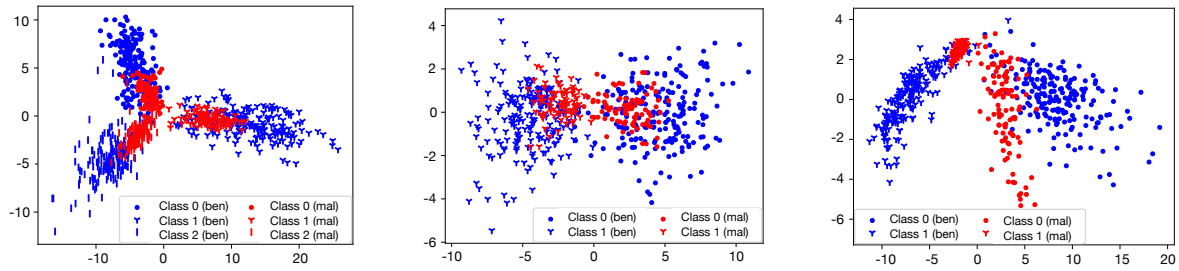
Figure 2.4: Penultimate layer representations (PLRs) plotted in 2-D space. The lines between the two markers indicate the inter-distance between them. Experiments were done using the Kather dataset [59].

of PLR. We show three examples (three input data points), and we observe similar patterns across all three examples. Each of the markers (i.e., circle, plus, and square) corresponds to one input data point. There are ten instances of each type of marker, representing ten versions of PLRs for one input data. Nine of these (in blue) are from nine benign local models, and one (in red) is from a malicious model. We can see that all the benign PLRs stay close to each other, while the malicious one exhibits a more significant distance from the benign ones.

2.4.3 Visualizing PLRs Distribution

Based on both theoretical analysis and empirical findings, we observe that poisoned models produce PLRs that are significantly distant from the cluster formed by benign PLRs. To further investigate the distribution of PLRs, we plot multiple data points from the same class. Figure 2.5 illustrates the PLRs of both malicious and benign models in the context of backdoor attacks. It becomes apparent that the PLRs of benign models follow a specific

distribution, while those of malicious models deviate from it. Similarly, we present results for untargeted attacks in Figure 2.6 and observe consistent outcomes. These findings collectively reinforce the notion that PLR serves as a promising feature for identifying poisoned models. The primary cause of this distribution deviation is that the PLR distances among benign models are smaller than the PLR distances between benign and malicious models, which is supported by Proposition 2.1 and 2.2.



(a) fMNIST (b) CIFAR-10 (c) Kather

Figure 2.5: The distribution of penultimate layer representations (PLRs). Blue markers denote PLRs from attack-free models, while red markers represent PLRs from models under backdoor attack.

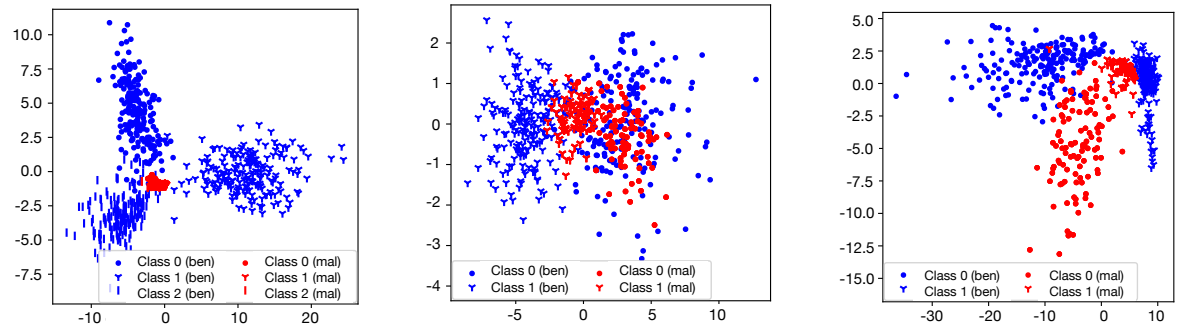


Figure 2.6: The distribution of PLRs. Blue markers denote PLRs from attack-free models, while red markers represent PLRs from models under untargeted attack.

The process used to visualize the PLRs follows these steps, as outlined in Muller et al. (2019) [95]: 1) Randomly select a subset of classes from all available classes (in our case, we

chose either two or three classes for improved visualization); 2) Compute the orthonormal basis of the hyperplane where the templates of the chosen classes reside; 3) Calculate the PLRs for the data records within the selected classes; 4) Project the PLRs onto the hyperplanes by determining the inner product of the PLRs and the orthonormal basis; and 5) Employ Principal Component Analysis (PCA) to reduce the dimensionality of the inner product to a 2-D space, and then plot the resulting data within this 2-D space.

2.5 FLARE: Defending against MPAs

2.5.1 Overview of FLARE

Drawing from the observations and theoretical analysis of PLRs, we have designed our system, called FLARE. Figure 2.7 provides an overview of FLARE's structure, which is compatible with a general Federated Learning (FL) system. Integrating FLARE into an existing FL system is straightforward, as one only needs to add a trust score estimation module. As depicted in Figure 2.7, local clients initially submit their local model updates, denoted as δ_i, to the Parameter Server (PS). The PS then calculates PLRs for each local model using an auxiliary dataset. Subsequently, FLARE determines the nearest neighbors for each local model based on the Maximum Mean Discrepancy (MMD) of their PLRs. The frequency with which a client is chosen as the nearest neighbor for other clients is used to estimate its trust score S_i. Finally, the PS aggregates the model updates, weighting them according to their trust scores, and uses this information to update the global model.

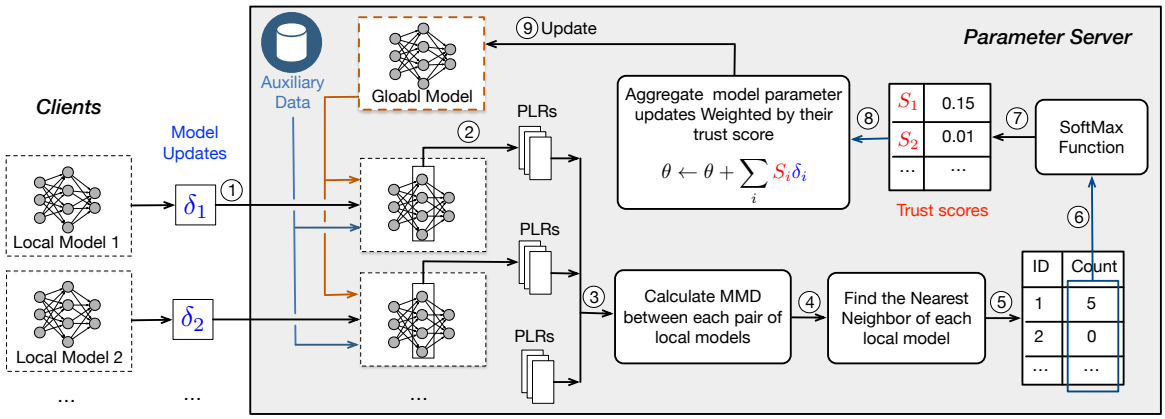


Figure 2.7: Detailed system design of FLARE.

2.5.2 Detailed Design

FLARE introduces two key differences at the Parameter Server (PS) compared to traditional Federated Learning (FL) systems: 1) PS sets up the global model by training it with the auxiliary dataset $\mathcal{D}$, as opposed to using random initialization. This approach not only accelerates the training process but also enables clients to make accurate predictions on $\mathcal{D}$. As a result, the PLRs of benign models are more likely to follow a single distribution. 2) During the aggregation phase of each learning iteration, PS calculates a trust score for every model update and aggregates them according to their respective trust scores. In the following sections, we will focus on detailing FLARE's aggregation scheme, particularly the process of estimating trust scores.

First, PS calculates the PLRs for each local model wj using the auxiliary dataset $\mathcal{D}$ = x$ii \in [m]$. The mapping function of the model with weight wj is denoted by $g_{\mathrm{w_j}}$: x $\in \mathbb{R}^{d1 \times d2 \times d3} \rightarrow$ r $\in \mathbb{R}^o$, where $d1$, $d2$, and $d3$ represent the width, height, and channels of image input and o represents the dimensionality of a PLR. With m data points in $\mathcal{D}$, we can obtain m PLRs r1, ..., rm, where r$i = g_{\mathrm{w_j}}(\mathrm{x_i})$. To differentiate between PLRs of various models, we use $R_j := g_{\mathrm{w_j}}(\mathrm{x_1}), ..., g_{\mathrm{w_j}}(\mathrm{x_m})$ to represent the PLRs generated by the j-th model.

Next, FLARE employs Maximum Mean Discrepancy (MMD) [46] on R_i and R_j to assess whether the two PLR sequences follow the same distribution. MMD is chosen over other two-sample test methods mainly because the number of PLR points, $m = 10$, in one sample is significantly smaller than the dimensionality, $o = 128$, of the data. Measuring the distribution of such a small sample is challenging. Traditional parametric two-sample test methods typically have strong assumptions regarding the parameters of the population distribution from which the sample is drawn, making them unsuitable for this case. FLARE uses MMD to estimate the distance between two PLR sequences, as MMD does not require knowledge of the PLR distribution. The unbiased estimate of MMD between the two PLR sequences R_i and R_j can be expressed as follows, without loss of generality:

$$\begin{aligned}\mathrm{MMD}(R_i, R_j) = \frac{1}{m(m-1)} \Bigg[\sum_{a \in R_i} \sum_{b \in R_i, b \neq a} k(\mathrm{a}, b) + \\ \sum_{a \in R_j} \sum_{b \in R_j, b \neq a} k(a, b) - 2 \sum_{a \in R_i} \sum_{b \in R_j} k(a, b) \Bigg]\end{aligned} \tag{2.13}$$

where $k(\cdot)$ represents the Gaussian kernel function. The empirical test statistic $\mathrm{MMD}(R_1, R_2)$ will be small if R_1 and R_2 originate from the same distribution, and large otherwise. For the sake of simplicity, we use the shortcut $MMDij = \mathrm{MMD}(Ri, R_j)$ to denote the MMD between the PLRs of the i-th model and the j-th model.

FLARE employs the count of nearest neighbors to estimate the trust score for a model update. PS selects the top 50% nearest neighbors for each local model based on their MMD scores. The count N_i for $\mathrm{w_i}$ increases by one each time $\mathrm{w_i}$ is chosen as a nearest neighbor by any $\mathrm{w_j}(j \neq i)$. This count value, N_i, reflects the degree of trustworthiness. To convert the count value into a normalized trust score, we use the softmax function with a temperature parameter:

$$S_i = \frac{\exp(N_i/\tau)}{\sum_{k=1}^{n} \exp(N_k/\tau)}, \tag{2.14}$$

where τ represents the temperature parameter. S_i falls within the range of $[0, 1]$, and the sum of all trust scores, $\sum_i^n S_i$, equals 1. A larger τ value yields more evenly distributed trust scores. By choosing a smaller τ, we can emphasize benign model updates while diminishing the weights of suspicious model updates. And we select $\tau = 1$ in the evaluation section.

An alternative approach involves using the average MMD value of one model in relation to other models to estimate its trust score. We opt for the nearest-neighbor-count-based scheme rather than the average-MMD-based scheme for the following reasons. The nearest neighbor count-based method is more resilient to collusive attackers than the average-MMD-based scheme. Colluding attackers can generate nearly identical PLRs, resulting in extremely small MMD values among themselves. Consequently, the final average MMD of an attacker may be smaller than that of benign models, rendering the detection scheme ineffective. In contrast, the nearest-neighbor-count method can handle this type of collusion when attackers constitute less than 50% of all clients.

In the final step, we aggregate the model updates by weighting them according to their trust scores and use this weighted aggregation to update the global model as follows:

$$\theta \leftarrow \theta + \sum_{i=1}^{n} S_i \delta_i. \tag{2.15}$$

where n is the number of local model updates received by PS.

The workflow for FLARE is also presented in Algorithm 1, which will be further detailed in the subsequent sections.

Algorithm 1 FLARE Algorithm

Input: local model updates $\{\delta_1, \delta_2, ..., \delta_n\}$, auxiliary data records $\{\mathrm{x}_1, ...\mathrm{x}_m\}$, global model θ, maximum iteration T.
Output: global model θ.

1: while $t < T$ do
2: Local models: $\mathrm{w}_1, ..., \mathrm{w}_\mathrm{n} \leftarrow \delta_1 + \theta, ..., \delta_n + \theta$.
3: for $i < n, i < j < n$ do
4: $R_i \leftarrow [g_{\mathrm{w_i}}(\mathrm{x}_1), ..., g_{\mathrm{w_i}}(\mathrm{x_m})]$ # PLRs of i-th model.
5: $R_j \leftarrow [g_{\mathrm{w_j}}(\mathrm{x}_1), ..., g_{\mathrm{w_j}}(\mathrm{x_m})]$ # PLRs of j-th model.
6: $MMD_{ij} = MMD_{ji} \leftarrow \mathrm{MMD}(R_i, R_j)$
7: end for
8: $k = \mathrm{round}(n * 50\%)$
9: for $i < n$ do
10: $IDs \leftarrow \mathrm{argsort}(MMD_{i1}, ...MMD_{in})$
11: $Neighb_i \leftarrow$ Top k items in IDs
12: end for
13: # Frequency of i being selected the nearest neighbor by others.
14: $ct_1, ..., ct_n \leftarrow \mathrm{counting}(Neighb_1, ..., Neighb_n)$
15: $\theta \leftarrow \theta + \sum_{i=1}^{n} \frac{e^{N_i}}{\sum_{k=1}^{n} e^{N_k}} \delta_i$ # global model update
16: end while

2.6 Implementation and Experimental Settings

We implement both MPAs and FLARE using the TensorFlow platform. All experiments are carried out on a server equipped with an Intel Core i7-8700K CPU @ 3.70GHz x 12, a GeForce RTX 2080 Ti GPU, and running Ubuntu 18.04.3 LTS. We implement four types of MPAs: Attack-Krum-Untargeted, Attack-TM-Untargeted [36], Attack-Krum-Backdoor, and Attack-Coomed-Backdoor [15]. Additionally, we implement defense mechanisms, such as Krum [16], Coomed, TrimmedMean [170], Bulyan [32], and FLTrust [18], as baseline methods for comparison.

Table 2.2: Model Accuracy (%) of normally functional FL system (attack-free).

Dataset	FedAvg	Krum	Coomed	TMean	Bulyan	FLARE
fMNIST	91.77	88.68	91.55	91.61	91.45	91.58
CIFAR-10	69.58	55.190	69.31	69.35	68.56	67.00
Kather	78.83	75.1	76.6	78.33	75.1	78.23

2.6.1 Experimental Setting

In the investigated FL system, the default number of clients is set to 10 ($n = 10$), with a 1.0 ratio for selecting clients in each FL iteration. The number of malicious clients is one for backdoor attacks and three for untargeted attacks, in line with [15, 36]. The dataset is split evenly into n subsets and allocated to the clients. PS has an auxiliary dataset containing ten clean data points from a single class. Each client manages a local model (e.g., VGGNet [129]) and employs the Adam optimizer with a learning rate of 0.001 to train the local model. Clients train their local models for five epochs before submitting model updates. The training process includes 20 iterations ($T = 20$). The test accuracy for the attack-free model is displayed in Table 2.2. We conducted each experiment three times and presented the average performance.

2.6.2 Datasets and Neural Network Architectures

We evaluate the FLARE system using three distinct datasets: the fMNIST dataset [162], the CIFAR-10 dataset [65], and the Kather dataset [59]. The details of these datasets are provided below, with the corresponding neural network architectures presented in Table 2.3 for fMNIST, Table 2.4 for CIFAR-10, and Table 2.5 for Kather. It is important to note that we resize the images in the Kather dataset from 150x150x3 to 128x128x3 before inputting them into the VGGNet. The fMNIST dataset utilizes a batch size of 64, while the CIFAR-10 and Kather datasets employ a batch size of 32.

Table 2.3: Architecture of the neural network used for the fMNIST dataset.

	Input	Filter	Stride	Output	Activation
conv2d_1	28×28×1	64×5×5	1	24×24×64	ReLU
conv2d_2	24×24×64	64×5×5	1	20×20×64	ReLU
dropout_1	20×20×64	Dropout, 0.25	/	20×20×64	/
flatten_1	20×20×64	/	/	25600	/
dense_1	25600	/	/	128	Relu/
dropout_2	128	Dropout, 0.5	/	128	/
dense_2	128	/	/	10	/

Table 2.4: Architecture of the neural network used for the CIFAR-10 dataset.

	Input	Filter	Stride	Output	Activation
conv2d_1	32×32×3	64×3×3	1	30×30×64	ReLU
max_pooling2d_1	30×30×64	MaxPooling2D, 2×2	2	15×15×64	/
conv2d_2	15×15×64	64×3×3	1	13×13×64	ReLU
max_pooling2d_2	13×13×64	MaxPooling2D, 2×2	2	6×6×64	/
conv2d_3	6×6×64	64×3×3	1	4×4×64	ReLU
max_pooling2d_3	4×4×64	MaxPooling2D, 2×2	2	2×2×64	/
flatten_1	2×2×64	/	/	256	/
dense_1	256	/	/	128	/
dense_2	128	/	/	10	/

Table 2.5: Architecture of the neural network used for the Kather dataset.

	Input	Filter	Stride	Output	Activation
conv2d_1	128×128×3	64×3×3	1	128×128×64	ReLU
max_pooling2d_1	128×128×64	MaxPooling2D, 2×2	2	64×64×64	/
conv2d_2	64×64×64	64×3×3	1	64×64×64	ReLU
max_pooling2d_2	64×64×64	MaxPooling2D, 2×2	2	32×32×64	/
conv2d_3	32×32×64	64×3×3	1	32×32×64	ReLU
max_pooling2d_3	32×32×64	MaxPooling2D, 2×2	2	16×16×64	/
conv2d_4	16×16×64	64×3×3	1	16×16×64	ReLU
max_pooling2d_4	16×16×64	MaxPooling2D, 2×2	2	8×8×64	/
conv2d_5	8×8×128	64×3×3	1	8×8×128	ReLU
max_pooling2d_5	8×8×128	MaxPooling2D, 2×2	2	4×4×128	/
flatten_1	4×4×128	/	/	2048	/
dense_1	2048	/	/	128	/
dense_1	128	/	/	8	/

fMNIST dataset comprises a training set of 60,000 samples and a test set of 10,000 samples. Each sample is a 28x28 grayscale image, associated with a label from one of 10 classes: T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

CIFAR-10 dataset contains 60,000 color images with a resolution of 32x32, distributed across 10 classes, with 6,000 images per class. The dataset is split into 50,000 training images and 10,000 test images. Each image belongs to one of the following ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Kather dataset is a collection of colorectal cancer histology textures. It consists of 5,000 samples, each of which is a 150x150x3 histological image. Each image is classified into one of eight tissue categories: Tumor, Stroma, Complex, Lympho, Debris, Mucosa, Adipose, and Empty.

2.6.3 Evaluation Metrics

Our objective is to address two questions: Can FLARE effectively defend against MPAs by decreasing the attack success rate? And can FLARE maintain relatively high accuracy on clean data? To evaluate our defense against backdoor attacks, we present the model's confidence in the target label, attack success rate (ASR), and model accuracy (Acc) on clean data. Model confidence in the target label, denoted as q_t, is a widely used metric for backdoor attacks [15]. ASR is defined as the ratio of test inputs predicted as the target label to the total number of targeted inputs, where a targeted input refers to an input containing a backdoor trigger. For untargeted attacks, the evaluation metric differs due to the distinct attacking goals. In this case, model accuracy serves as the sole evaluation metric.

2.7 Evaluation Results

2.7.1 Backdoor Attacks

In a backdoor attack, the attacker's goal is to manipulate predictions on targeted inputs without negatively affecting the overall prediction performance on other data records. For instance, in a backdoor MPA scenario, multiple hospitals may collaborate to train a tumor tissue detector through federated learning. A backdoor attacker could then inject malicious updates during FL iterations in order to deceive the FL model into classifying tumor tissue as normal tissue. In the following, we describe two state-of-the-art backdoor MPAs employed for evaluation purposes.

Attack-Krum-Backdoor [15]: The adversary designs malicious local models to inject a backdoor into a federated learning system that utilizes Krum aggregation rule. This attack is particularly subtle because the malicious parameters closely resemble the benign ones, appearing innocuous. Consequently, there is a high likelihood that the crafted malicious model parameters will be accepted by the Krum aggregation rule. The objective function for this attack is as follows:

$$\arg\min_{\delta_{mal}} \quad L(\mathcal{D}_{mal}) + \lambda L(\mathcal{D}_{train}) + \rho \|\delta_{mal} - \bar{\delta}_{ben}\|, \tag{2.16}$$

where the primary objective of this attack is to minimize the loss on backdoor inputs, represented by $L(\mathcal{D}mal)$. Simultaneously, the attack seeks to minimize the loss $L(\mathcal{D}train)$ in order to enhance the accuracy of clean samples. Furthermore, to maintain stealth, the attack also aims to minimize the distance $\|\delta_{mal} - \bar{\delta}_{ben}\|$ between the malicious update and the average benign updates. By achieving these objectives, the adversary can manipulate the global model to produce target labels for target inputs while remaining inconspicuous.

Attack-Coomed-Backdoor [15]: The objective function for this attack is identical to that of Attack-Krum-Backdoor (i.e., Eq. (2.16)). To counter the coordinate median aggregation rule (i.e., Coomed), the practical implementation of the local training process at the attacker's end differs slightly from Attack-Krum-Backdoor [15].

Backdoor attacks can be classified into two types based on the necessity of physically injecting a trigger: trojan backdoor attack and semantic backdoor attack. In a trojan attack, attackers must physically inject a backdoor or trigger into the machine learning model by altering all or a subset of the training data. In contrast, semantic backdoor attacks do not require the physical injection of a trigger/backdoor in inputs. Instead, the trigger is a semantic feature present in the original images. For example, 'dots' can be a semantic trigger for an apparel classification problem. The attacker attaches the label 'dress' to images containing 'dots,' with the goal of causing any apparel with 'dots' to be classified as a dress. We expand the two backdoor MPA methods into four attacks, including Attack-Krum-Backdoor semantic and Attack-Krum-Backdoor trojan, Attack-Coomed-Backdoor semantic, and Attack-Coomed-Backdoor trojan.

2.7.2 FLARE Performance against Backdoor Attack

Table 2.6 displays the efficacy of FLARE against two semantic backdoor MPAs. The name of each attack is listed in the left column, such as Attack-Krum-Backdoor, which indicates the targeted BRAR for the attack (i.e., Krum). It can be observed from Table 2.6 that an MPA achieves a high ASR against not only its intended Krum but also compromise other BRARs. In contrast, the proposed FLARE reduces the ASR to an extremely lower level, very close to zero, across various datasets. In summary, FLARE outperforms FedAvg, four BRARs, and FLTrust by achieving a lower ASR. Regarding accuracy on clean data, FLARE

Table 2.6: Model Accuracy (%) for clean data & attack success rate (ASR (%)).

Attack Name	Dataset	FedAvg		Krum		Coomed		TMean		Bulyan		FLTrust		FLARE	
		Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR
Attack-Krum	fMNIST	91.6	8.3	87.9	98.3	91.6	45.0	91.5	43.3	91.4	71.6	91.7	40.0	91.2	0
-Backdoor	CIFAR-10	68.7	26.7	48.1	100	67.8	53.3	68.4	33.3	67.2	65.0	67.0	0	66.4	0
semantic	Kather	79.9	41.6	50.9	86.7	79.3	33.3	79.4	41.7	79.5	56.7	50.5	19.4	76.7	0
Attack-Coomed	fMNIST	91.5	58.3	88.0	98.3	91.7	78.3	91.6	68.3	91.5	85.0	91.5	0	91.4	0
-Backdoor	CIFAR-10	68.0	56.7	50.1	96.7	67.8	85.0	67.7	53.3	66.1	88.3	66.3	20.3	65.2	0
semantic	Kather	75.3	91.7	64.8	51.6	78.6	78.3	78.2	75.0	78.5	46.7	76.2	7.2	77.8	0

achieves comparable or slightly lower accuracy than other baseline methods. This is because a malicious update includes both poisoning knowledge and useful knowledge from its clean data. In FLARE, PS assigns lower weights to the updates from malicious clients, leading to a slightly lower accuracy on clean data.

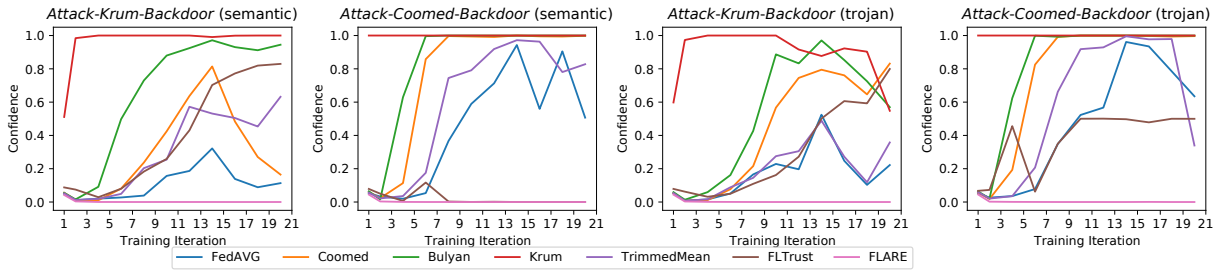


Figure 2.8: Model confidence in targeted label under backdoor MPAs using fMNIST dataset.

Figure 2.8 displays the global model's confidence in the attacker's targeted labels. High confidence indicates the attacker successfully manipulates the prediction of the global model. The first two subfigures of Figure 2.8 exhibit the model confidence under two semantic backdoor attacks and the last two show results for trojan backdoor attacks. Under Attack-Krum

-Backdoor (semantic), we observe that the model confidence of Krum quickly rises to 1.0, indicating the attack's success in the early stages of FL. Moreover, the model confidence of other BRARs, such as Bulyan, Coomed, and Trimmedmean, also increases gradually,

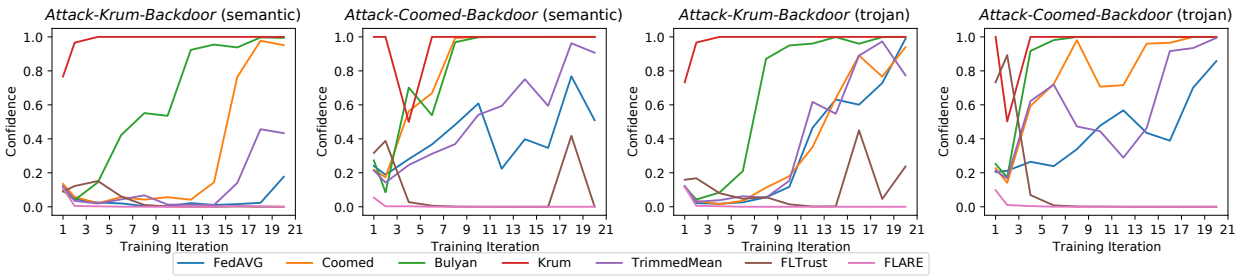


Figure 2.9: Model confidence in targeted label under backdoor MPAs using CIFAR-10 dataset.

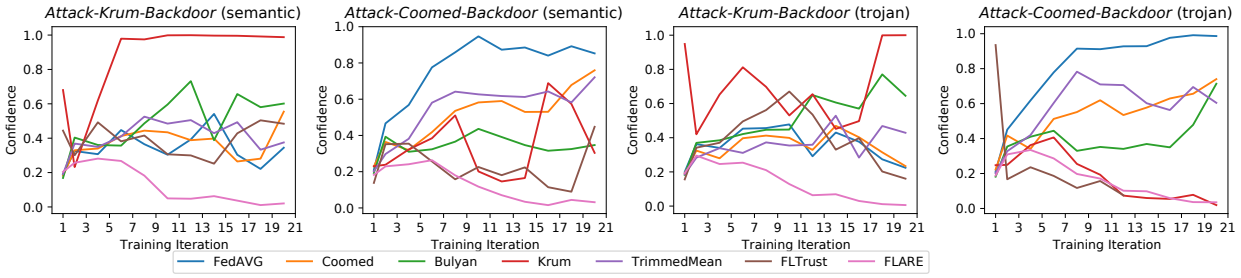


Figure 2.10: Model confidence in targeted label under backdoor MPAs using Kather dataset.

indicating that this attack successfully targets all BRARs. In contrast, FLARE results in stable and low confidence scores for targeted inputs under MPAs, indicating successful defense against the four backdoor attacks. We achieve similar outcomes on the CIFAR-10 and Kather datasets, as shown in Figure 2.9 and Figure 2.10, respectively. For both datasets, FLARE attains the lowest confidence scores in the final (i.e., 20th) iteration. At the final iteration, the model confidence is close to zero, indicating successful defense against these attacks. Note that the confidence scores fluctuate throughout the learning process due to randomness in distributed learning and aggregation rules. And all the results displayed are the average of three runs.

2.7.3 Untargeted Attacks

In untargeted attacks, the objective of the attacker is to undermine the model's performance by preventing the global model from convergence or directing it toward a local optimum that yields a high testing error rate. Here, we summarize two state-of-the-art untargeted MPAs used for evaluation.

Attack-Krum-Untargeted [36]: The attacker's goal is to create k ($k \geq 1$) malicious local models to attack Krum. To accomplish this, the attacker uses "directed deviation" to alter the global model parameters, moving them in the opposite direction of the attack-free direction. The attack is formulated as:

$$\begin{aligned} \max \quad & \lambda \\ \text{s.t.} \quad & \mathrm{w}'_1 = Krum(\mathrm{w}'_1, ..., \mathrm{w}'_k, \mathrm{w}_{(k+1)}, ..., \mathrm{w}_n), \\ & \mathrm{w}'_1 = \theta - \lambda \mathrm{s}, \\ & \mathrm{w}'_i = \mathrm{w}'_1, \text{for } i = 2, 3, ..., k. \end{aligned} \tag{2.17}$$

where w'_i ($\forall i \in [k]$) refers to the weights of a model manipulated by adversary; w_i ($\forall i \in \{k+1, ..., n\}$) represents benign model weights; θ denotes the current global model, and s represents the sign of the average benign model weights. We use λs to denote the directed deviation of the aggregated model w'_1 and the last global model θ. Generally, a larger deviation in model parameters introduces a larger model error rate. The objective of this attack is to maximize λ in order to increase the error rate of FL.

Attack-TM-Untargeted [36]: The approach used in Attack-TrimmedMean-Untargeted is similar to that of Attack-Krum-Untargeted. The goal is to deviate the global model in the opposite direction of the attack-free model. Assume that the benign weight in the j-th coordinate is within the range $[w_{j,min}, w_{j,max}]$. To stay stealthy, the j-th coordinate of the

crafted model should also be in the same range (i.e., $[w_{j,min}, w_{j,max}]$). In this attack, the attacker uses a heuristic algorithm to generate the j-th coordinate. Specifically, if the sign of the average weight is negative (i.e., $s_j = -1$), a value around $w_{j,max}$ is sampled. Conversely, if $s_j = 1$, a value around $w_{j,min}$ is sampled. This process can flip the sign of some coordinates of the average weights. Like Attack-Krum-Untargeted, the objective of this attack is to increase the testing error rate of the global model.

2.7.4 FLARE Performance against Untargeted Attack

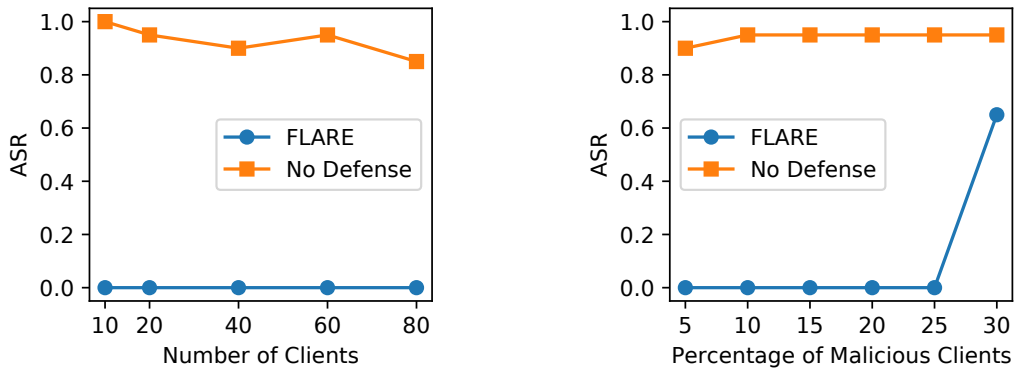
The testing accuracy of the final global model under untargeted MPAs is presented in Table 2.7. As can be seen from the results, untargeted attacks can effectively degrade the performance of both the target and other BRARs. On the other hand, FLARE demonstrates robust defense against untargeted MPAs, yielding significantly higher testing accuracy compared to other baselines. While FLTrust achieves similar accuracy with FLARE on the fMNIST and CIFAR-10 datasets, its accuracy is significantly lower on the Kather dataset, which has the highest-dimensional inputs among the three datasets. This can be attributed to the fact that FLTrust assigns trust scores based on cosine similarity between a local model parameter and a benign model parameter, which is less effective in high-dimensional settings.

Table 2.7: Model Accuracy (%) under untargeted attacks.

Attack Name	Dataset	FedAvg	Krum	Coomed	TrimmedMean	Bulyan	FLTrust	FLARE
Attack-Krum -Untargeted	fMNIST	59.2	6.93	79.8	89.7	86.2	91.2	90.9
	CIFAR-10	62.4	10.4	61.6	62.8	62.5	66.4	66.5
	Kather	69.0	12.34	66.1	66.4	69.8	11.6	76.4
Attack-TrimmedMean -Untargeted	fMNIST	87.1	87.9	80.3	61.1	89.5	90.7	91.1
	CIFAR-10	58.8	53.4	60.3	49.2	64.0	64.4	67.5
	Kather	18.0	70.8	64.2	22.3	74.1	10.9	77.5

2.7.5 Defense Robustness under Different FL settings

To showcase the effectiveness of the FLARE defense against MPAs, we assess FLARE's performance in a range of FL scenarios. For instance, we vary the client numbers and malicious percentage. Because of space limitation, we only show the results of FLARE against the attack-Krum-backdoor (semantic) method.



(a) ASR (malicious ratio=0.1) (b) ASR (client number=20)

Figure 2.11: ASR of Attack-Krum-Backdoor (w/o FLARE) using the fMNIST dataset.

In Figure 2.11a, we change the total number of clients while maintaining a consistent proportion of malicious clients at 10%. Without any defense, the Attack Success Rate (ASR) stays above 0.8. However, when FLARE is applied, the ASR drops to nearly zero, irrespective of the client number. These findings show that FLARE successfully protects against MPAs in Federated Learning systems with varying client numbers. In Figure 2 2.11b, we explore the effect of varying the percentage p of malicious clients from 5% to 30%. The ASR under FLARE stays close to 0 when p is less than 30%, indicating that FLARE is highly efficient when malicious clients make up less than 30% of the total. Detecting malicious clients becomes increasingly difficult when they constitute more than 30% of the overall client base. Lastly, we investigate the influence of the dataset size m on FLARE's effectiveness. We find that FLARE performs well when m is greater than or equal to 7,

which further confirms that only a small auxiliary dataset is enough for using FLARE to detect adversarially crafted model parameters.

We also investigate the impacts of data distributions on FLARE's performance. Following the approach in [122], we generate non-i.i.d. datasets among distributed clients. Each client receives samples from precisely $c_{sel} < c$ distinct classes of the dataset, where c represents the total number of categories. The data splits are non-overlapping and balanced, ensuring that every client has an equal amount of data points. The ASR for two backdoor attacks is presented in Table 2.8. FLARE achieves the lowest ASR for both attacks, highlighting its robustness against MPAs in non-i.i.d. settings.

Table 2.8: Attack Success Rate (ASR) in non-i.i.d. Scenario using the fMNIST Dataset.

Attack	Krum	Coomed	TMean	Bulyan	FLARE
Attack-Krum-Backdoor	0.750	0.533	0.133	0.716	0.016
Attack-Coomed-Backdoor	1.00	0.867	0.750	0.883	0.050

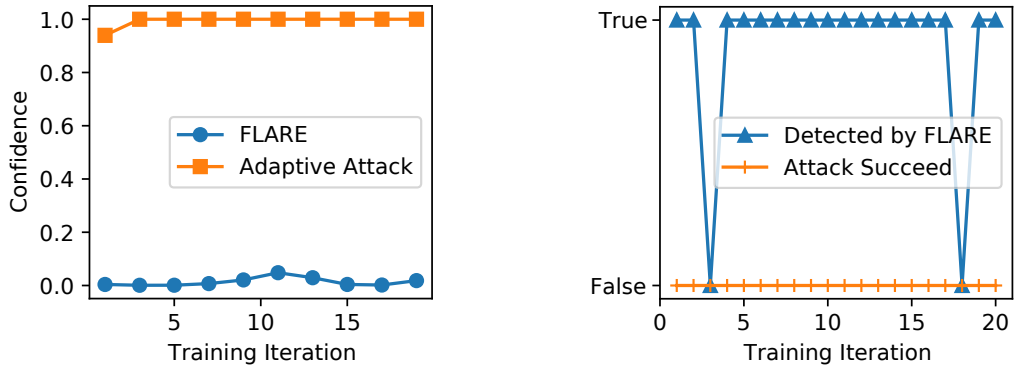
2.7.6 FLARE Performance against Adaptive Attack

In an even more challenging situation, an attacker may adaptively modify their attack techniques to overcome the defense, armed with the knowledge of the defense strategy. In this case, we assume the attacker is aware of FLARE's defense approach.

The adaptive attack employs the following strategy: To circumvent FLARE, the attacker designs their model to generate PLRs that resemble the PLRs of benign local models. The attack objective can be formulated as:

$$\arg\min_{\delta_{mal}} \quad L(\mathcal{D}_{mal}) + \lambda L(\mathcal{D}_{train}) + \rho\|\delta_{mal} - \bar{\delta}_{ben}\| + \eta d_{plr}. \tag{2.18}$$

where $L(\mathcal{D}_{mal})$ denotes the loss on targeted inputs, $L(\mathcal{D}_{train})$ denotes the loss on clean data.



(a) Adaptive attack (b) Detection and Attack

Figure 2.12: Resilience of FLARE against the adaptive attack.

$\|\delta_{mal} - \bar{\delta}_{ben}\|$ represents the L-2 distance between malicious model parameters and average benign model parameters, and d_{plr} denotes the L-2 distance between the PLRs of malicious model and the average PLRs of benign models. The first three components in the adaptive attack formula originate from [15], and we proposed to add the fourth component d_{plr}. The goal of this attack is to deceive the global model into producing target labels for selected inputs while concealing its malicious intent. The components $L(\mathcal{D}_{train})$ and $\|\delta_{mal} - \bar{\delta}_{ben}\|$ plus d_{plr} are used to achieve the stealthiness. And we represent d_{plr} as:

$$d_{plr} = \sum_{i \in [c]} \|\overline{PLR}(\delta_{mal}, \mathcal{D}^i_{train}) - \overline{PLR}(\bar{\delta}_{ben}, \mathcal{D}^i_{train})\|, \tag{2.19}$$

where c represents the classes number, and D^i_{train} denoted the training dataset of class i. For each class i, the average value of PLRs of the adversarially manipulated model is represented by $\overline{PLR}(\delta_{mal}, \mathcal{D}^i_{train})$. And the average value of PLRs from benign models $\bar{\delta}_{ben}$ is denoted by $\overline{PLR}(\bar{\delta}_{ben}, \mathcal{D}^i_{train})$).

Figure 2.12a illustrates FLARE's performance against the adaptive attack. Without any defense in place, the adaptive attack achieves a high success rate, with confidence exceeding 0.9. When FLARE is incorporated into the learning process, the confidence drops below

0.1, and the ASR decreases to 0. These results highlight FLARE's effectiveness in defending against adaptive attacks. In Figure 2.12b, we present two indicators: one for detecting the attack and another for identifying attack-induced misclassification. It is important to note that in this context, 'detected' refers to a malicious model update receiving a trust score lower than average. We observe that FLARE fails to detect malicious clients in Iterations 3 and 18. However, the adaptive attack is also unsuccessful during these two rounds, meaning that the maliciously crafted models in Iterations 3 and 18 become harmless. This observation demonstrates the difficulty for adaptive attacks to evade FLARE and achieve their malicious objectives simultaneously.

We further examine the computational complexity of FLARE. We discover that the computational overhead of FLARE is dependent on PLR calculation and MMD execution. FLARE's time complexity can be represented as $O(n \cdot t_{plr} + n^2 \cdot t_{mmd})$ where n represents the number of selected clients by PS, t_{plr} is PLR computation time per model, and t_{mmd} is the computation time for MMD test. In our case, t_{plr} is small as it is a forward computation of the neural networks. The second component increases linearly with the square of client number n. The efficiency of MMD computation is relatively high since only 10 data records are involved in the computation, implying that the time t_{mmd} can be relatively low.We also assessed the empirical computation time of FLARE. On our machine, the average total execution time of FLARE is 3.2 seconds when $n = 10$ and 28.3 seconds when $n = 50$, indicating that FLARE is relatively computationally efficient.

2.8 Conclusions

In this chapter, we introduce a robust aggregation algorithm, FLARE, designed to safeguard Federated Learning (FL) against Malicious Participant Attacks (MPAs). Through

analysis and experimental visualization, we reveal that the Poisoned Local Ratio (PLR) vector has considerable potential in distinguishing malicious/poisonous models from benign ones. Leveraging the PLR technique, FLARE effectively minimizes the impact of malicious/poisonous models on the final aggregation by assigning low trust scores to those with divergent PLRs. Through comprehensive evaluation, we demonstrate that FLARE significantly outperforms existing defenses (such as BRARs and FLTrust) in defending against state-of-the-art MPAs, including semantic backdoor attacks, trojan attacks, and untargeted attacks on three popular datasets. Moreover, FLARE exhibits its effectiveness in non-i.i.d. data and adaptive attack scenarios, showcasing its applicability in challenging real-world situations.

Chapter 3

Inference-time Security: Adversarial Example Attack and Detection

(Copyright notice[1])

3.1 Introduction

The increasing scale and complexity of modern networks and the tremendous amount of applications running on them render communication and networking systems highly vulnerable to various intrusion attacks. An intrusion detection system (IDS) plays a significant role in safeguarding networks from malicious attacks [75]. There are mainly two types of IDS: signature-based detection [86] and anomaly-based detection[42]. Signature-based detection schemes work by extracting the traffic signature and comparing it to those in a pre-built knowledge base. As a result, they are only effective in detecting known attacks but cannot detect attacks outside the knowledge base. Anomaly-based detection aims to detect deviations from an established normal traffic model. With the advancement in ML in recent years, ML techniques are increasingly used to train the "norm" model that

[1]This chapter previously appeared as a part of a journal paper published in the IEEE Transaction on Dependable and Secure Computing (TDSC), 2023. ©2022 IEEE. Reprinted, with permission, from Ning Wang, Yimin Chen, Yang Xiao, Yang Hu, Wenjing Lou, and Y. Thomas Hou, "MANDA: On Adversarial Example Detection for Network Intrusion Detection System," in IEEE TDSC, 20(2):1139-1153, 2023 [151].

represents the normal benign traffic and then to evaluate the credibility of incoming traffic. Considering intrusion attacks are ever-evolving these days, ML-based methods show much greater potential as they require little or no prior knowledge to work on emerging novel attacks.

ML technologies have seen great success in domains such as computer vision and natural language processing [55, 118, 164]. While applying to network intrusion detection, state-of-the-art IDSs usually employ advanced neural networks (e.g., LSTM) and learning schemes (e.g., meta-learning and active learning). An important security attack common to almost all machine learning models is the adversarial example (AE) attack [45, 137]. In such an attack, the adversary is able to craft a sample, often by applying small perturbations, which can mislead a well-trained model to output an arbitrary label other than its true label with a high probability. For an IDS, an attacker can launch AE attacks to significantly increase the false-positive rate and false-negative rate, rendering the IDS practically useless.

AE attacks have become more and more sophisticated and AE attacks on ML-based IDSs are becoming a real threat to network security. Lin et al. [80] leveraged a generative adversarial network (GAN) to transform original malicious traffic into adversarial traffic to fool the IDS. Wu et al. [159] employed deep reinforcement learning (DRL) to automatically and adaptively generate adversarial traffic flow to deceive the detection model. Rigaki et al. [120] utilized a GAN to adapt the Command and Control (C2) channel of malicious traffic to mimic a legitimate application's traffic (e.g., the Facebook chat network traffic), and therefore evaded the IDS. Shu et al. [127] employed active learning and GAN to launch AE attacks on ML-based IDS, demonstrating the capability to compromise an IDS using only limited prior knowledge. The above attacks [80, 120, 127, 159, 165] confirm that AEs are inevitably turning into a huge threat to ML-based IDSs.

To defend against AE attacks, one can generally take two routes: 1) improving the robustness

of an IDS model against adversarial perturbations, or 2) developing an auxiliary AE detector to reject suspicious inputs [20] before proceeding to the IDS. Defense schemes in the first category [107, 144] usually need to customize the IDS models to every AE attack encountered. Considering many novel AE attacks are yet to come, we opt to design an effective AE detector as the defense, i.e., the second route.

In this chapter, we propose MANDA, a MANifold and Decision boundary-based AE detection scheme for ML-based IDS. It is observed that the benign or malicious traffic events usually reside in a low-dimensional manifold (i.e., a cluster) embedded in the ambient feature space. An ML-based IDS aims to learn a decision boundary that discriminates malicious network traffic from benign network traffic. To explain the intuitions behind MANDA clearly, we use an AE generated from a malicious network event for an example. The AE fools the IDS model (i.e., evades the IDS) by traversing the decision boundary of the IDS model. To preserve the malicious property of the intrusion traffic, the crafted AE should be inside or at least close to the malicious manifold. Therefore, although the IDS model classifies the AE as 'benign', a manifold detector is still highly likely to discriminate it into the manifold of 'malicious' samples. This motivates us to leverage such inconsistency between the IDS decision boundary and the manifolds to detect an AE. In addition, given that AEs are usually closer to the decision boundary of the IDS model than normal samples, it is expected that when small noise is added, the classification result of an AE is more likely to change than that of a clean sample. This motivates us to use such changes of IDS classification results to detect an AE.

We further demonstrate that MANDA is also effective on multi-class IDS. A multi-class IDS differentiates multiple types of intrusion from benign network traffic with a single model, while a two-class IDS only detects one type of intrusion. As discussed above, MANDA is composed of two building blocks: a manifold-based (Manifold) method and

a decision-boundary-based (DB) method. Manifold shows similar performance in multi-class intrusion detection as is shown in two-class IDS. In order to improve the performance of DB in multi-class IDS, we propose to adjust noise magnitude based on the different inter-class distances. The contributions of this chapter are summarized as follows:

- We systematically investigate practical AE attacks and defenses of recent ML-based IDSs. To the best of our knowledge, we are the first to investigate AE attacks for IDS in problem space rather than in feature space, and also the first to propose an effective AE detection scheme to defend against such attacks.

- We propose MANDA, a novel MANifold and Decision boundary-based AE detection scheme for ML-based IDS. MANDA is designed by exploiting unique features we observe while trying to categorize AE attacks from the viewpoint of a machine learning model and data manifold. Based on our AE categorization, MANDA combines two building blocks (i.e., Manifold and DB) to achieve effective AE detection regardless of which AE attack is used.

- We demonstrate that Manifold generalizes well to multi-class IDS. We improve the performance of DB on multi-class IDS by customizing the noise magnitude for each decision boundary according to the inter-class distance and achieve effective AE detection on multi-class IDS.

- Our experimental results show that MANDA achieves 98.41% true-positive rate (TPR) with 5% false-positive rate (FPR) under CW attack, the most powerful AE attack, and over 0.97 AUC-ROC under three frequently-used attacks (FGSM attack, BIM attack, and CW attack) on the NSL-KDD dataset. On the CICIDS dataset, MANDA achieves as high as 98.50% TPR with 5% FPR under CW attack. We also demonstrate that MANDA outperforms Artifact [37], a state-of-the-art solution on AE detector, on both

the IDS task and image classification task.

The remainder of this chapter is organized as follows. Section 3.2 summarizes the related work. Section 3.3 introduces the system model and threat model. In Section 3.4, we elaborate on the proposed AE detection scheme. We then present and compare the experimental results in Section 3.5. Conclusions are drawn in Section 3.6.

3.2 Backgrounds and Related Work

This section provides an overview of previous work most relevant to our work, encompassing recent intrusion attacks and adversarial examples in deep learning. To the best of our knowledge, no previous research has specifically concentrated on AE attacks on IDS or the corresponding defense mechanisms.

3.2.1 Network Intrusions

Information technology infrastructure, encompassing the Internet, telecommunications networks, computer systems, and embedded industrial processors, is vulnerable to a range of network intrusion attacks. One such attack, the network probing attack, seeks out network vulnerability by examining the network's connections through port scanning to initiate further attacks. Another category of network intrusion attack, known as advanced persistent threat (APT) attacks [52], is powerful in a unique way as it relies on coordinated human efforts rather than automated code execution. During an APT attack, the target entity is persistently monitored and interacted with until the goals are accomplished. Distributed denial of service (DDoS) attack [147] aims to disrupt network functionality by overwhelming network resources, typically without additional objectives. A well-known instance of a DDoS

attack is the Mirai botnet, which temporarily disabled numerous websites, such as Twitter, Netflix, Reddit, and GitHub, in October 2016. Presently, Mirai variants are created daily, enabling them to continue spreading and causing significant damage to networks [61].

The recent advancements in machine learning have led to the creation of powerful ML-based Intrusion Detection Systems (IDSs) [147, 150]. However, the swift progress in adversarial machine learning has also introduced a new type of network intrusion attack known as adversarial example attack, designed to evade ML-based IDSs [10, 80, 97, 120, 127, 159, 165]. Researchers have proposed various methods and techniques to develop and evaluate these attacks. For instance, Xu et al. [165] proposed a general approach to identify evasive variants for a target classifier automatically. This method uses genetic programming techniques to modify a malicious sample, generating a variant that maintains malicious behavior but is recognized as benign by the classifier. The effectiveness of this approach was demonstrated on two popular PDF malware classifiers. Apruzzese et al. [10] examined realistic adversarial example attacks on IDS, specifically focusing on botnet traffic identification using ML classifiers. Their findings underscored the efficacy of adversarial examples on botnet detection classifiers. Wu et al.[159] utilized deep reinforcement learning to automatically generate adversarial traffic flow with the intent to deceive a target detection model. In this attack, the reinforcement learning agent updates adversarial samples based on feedback from the target model, allowing for adaptation to changes in the temporal and spatial features of the traffic flows.

In addition to efforts focused on finding a perturbation vector for individual inputs, some research aims to develop a perturbation generator model. Nasr et al. [97] designed a generative adversarial network (GAN) as a perturbation generator to create real-time perturbations for live traffic. These blind adversarial perturbations can decrease the accuracy of cutting-edge website fingerprinting by 90% while only adding 10% bandwidth overhead.

Lin et al. [80] introduced IDSGAN, which uses a GAN to convert original malicious traffic into adversarial instances, leading the IDS to misclassify them as benign. Rigaki et al. [120] employed a GAN to alter the Command and Control (C2) channel of malicious traffic to resemble that of a legitimate application, thereby evading detection. Shu et al. [127] combined active learning and GAN to launch adversarial example attacks on an ML-based IDS, demonstrating a strong ability to attack IDS with limited prior knowledge. Overall, it is evident that adversarial example (AE) attacks present a significant threat to modern ML-based IDSs, given their low cost and continuous evolution.

3.2.2 Adversarial Example

In recent years, adversarial examples (AEs) have emerged as critical research area. First introduced by Szegedy et al.[137], AEs are input samples that have been intentionally modified with barely perceptible perturbations, causing the classification result of a machine-learning model to change drastically. Research on AEs primarily focuses on two aspects: creating AEs, known as AE attacks, and addressing AEs, referred to as AE defenses.

AE Generation. Several techniques have been proposed for transforming a sample into an adversarial example, resulting in a variety of AE attack methods. Many of these generation approaches involve calculating the gradient of the model's loss for a given input. The fast gradient sign method (FGSM), introduced by Goodfellow et al. [45], manipulates the input image along the direction that maximizes the loss, causing misclassification. Kurakin et al. [66] developed the basic iterative method (BIM), which applies FGSM iteratively using small steps. The Jacobian-based saliency map attack (JSMA), proposed by Papernot et al. [108], identifies the most influential features for misclassification when applying a fixed distortion and perturbs only the selected pixels until misclassification is achieved. Carlini

and Wagner, in [21], introduced optimization-based attacks, known as CW attacks, to find a successful AE with minimal distortion. Among all known AE attacks, FGSM, JSMA, and CW are the most widely cited.

AE Detection. The majority of defenses against AE attacks have been specifically developed for image inputs in the field of computer vision research. Techniques such as reducing color depth of images [74, 166], shrinking image size [51, 73], increasing image resolution [96], and applying rotation or shifting [140] have been designed to detect adversarial images. However, these methods are either inapplicable or ineffective when implemented for network traffics.

Other types of AE defense schemes utilize statistical testing to detect AEs. These detection methods are not limited to image input and can be applied to IDS. These AE detectors generally assume that the statistical characteristics of AEs and clean data are distinct, making them distinguishable. Approaches such as kernel-based two-sample tests, generative models, Gaussian Mixture Models (GMMs), and kernel-based density estimation have been employed to detect AEs. Grosse et al.[47] employed a kernel-based two-sample test to differentiate AEs from clean data. Song et al. [134] used generative models to determine if an input sample originated from the same distribution as clean data. Zheng et al. [177] applied a Gaussian Mixture Model (GMM) to approximate the hidden layer distribution, rejecting samples with hidden states located in low-density regions of the distribution. Feinman et al. [37] utilized kernel-based density estimation to identify AEs. The downside of these techniques is that they require a large number of AEs to build the detector, similar to defense strategies based on adversarial training [44, 90].

To the best of our knowledge, we are the first to develop AE detection schemes for ML-based IDS. We compare our method to one of the state-of-the-art statistical testing schemes [37] in the experimental sections.

3.3 System and Threat Models

In this section, we introduce the system model and threat model. To better understand the system and threat models, we first clarify the frequently used notations.

3.3.1 Notations

In this chapter, we discuss two types of detection tasks. The first is 'intrusion detection,' which represents our primary application scenario. We consider an intrusion detection system (IDS) that employs machine learning techniques to identify abnormal or malicious network events. We refer to the classification model of an IDS (as illustrated in Figure 3.1) as the 'IDS model,' whose objective is to determine whether an input sample of network events constitutes an intrusion or not. The second type of detection task is 'AE detection,' which serves as our research goal. An IDS model is vulnerable to AE attacks. Our proposed AE detector, MANDA, is positioned in front of the intrusion detection module to detect and reject adversarial examples before they enter the IDS model. Consequently, the AE detector's purpose is to decide whether an input sample is an AE or not. With these distinctions in mind, we can define positive and negative samples for both the IDS model and MANDA.

For IDS model

- The IDS receives a sequence of network packets as input, which we can refer to as a networking event. A 'malicious input' corresponds to a networking event generated by a malicious attack, such as a DDoS attack or an instance of botnet traffic. Ideally, the IDS should classify a malicious input as positive; otherwise, a false negative will occur, meaning the malicious input has evaded detection. Synonyms for 'malicious input' include 'malicious data,' 'malicious traffic,' or simply 'intrusion.'

- We use 'benign traffic' to describe network events generated from regular network applications. Ideally, the IDS should classify all benign traffic as negative samples. 'Benign traffic' is sometimes also referred to as 'benign data,' 'benign input,' or 'normal traffic.'

For MANDA

- The MANDA system receives sequences of network packets as input, which represent network events. We use the term 'adversarial examples' (AEs) to describe inputs that are intentionally designed to deceive the Intrusion Detection System (IDS) model. This deception could involve evading detection or generating false positives. The objective of MANDA is to accurately classify these AEs. We may also refer to an AE as an 'adversarial input' or 'adversarial sample.'
- We use the term 'clean example' to describe network traffic instances that do not contain adversarial perturbations. The MANDA system should classify clean examples as negative. A clean example can represent either malicious or benign traffic instances. We may also refer to a clean example as a 'clean input' or 'clean sample.'

It is important to note that an adversarial example can be created from a malicious input and classified as 'benign,' or from a benign input and classified as 'malicious.' Throughout this chapter, we will focus on the first scenario for illustrative purposes. However, in our experiments, we will explore both cases.

3.3.2 System Model

A typical architecture of an ML-based IDS is shown in Figure 3.1. Usually, IDS is a passive infrastructure that rarely interferes with the network traffic under monitoring. An IDS

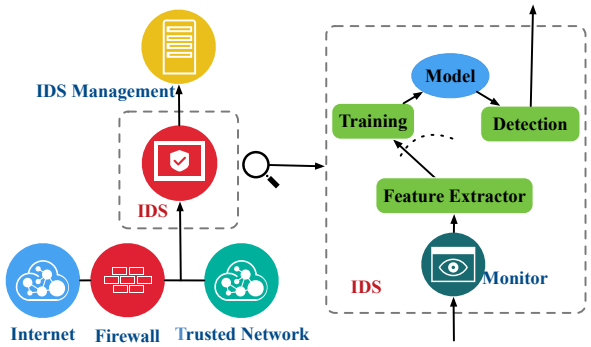


Figure 3.1: The system model of an ML-based IDS.

sniffs the internal interface of the firewall in a read-only mode and sends alerts to an IDS management server via a read-and-write network interface [17, 25]. As Figure 3.1 shows, an ML-based IDS is composed of the following modules [42]:

- Network Traffic Monitor keeps tracking the ongoing network traffic of a communication and networking system.
- Feature Extractor processes the raw traffic data into feature vectors in a pre-defined form.
- Training Phase. In the training phase, an ML model is trained with both benign and malicious traffic instances. We refer to the ML model as the IDS model.
- Detection Phase. In the detection phase, processed runtime traffic instances are fed into the learned model. An alert will be generated if an input instance is classified as positive by the IDS model.

3.3.3 Threat Model

Our focus is on adversarial example (AE) attacks, where an attacker attempts to deceive the Intrusion Detection System (IDS) model by making subtle alterations to the traffic flow,

such as increasing or decreasing the inter-arrival time of packets, as illustrated in Figure 3.2. The attacker's goal is to either cause the IDS to classify a malicious traffic instance as benign (resulting in a False Negative) or to classify a benign instance as malicious (resulting in a False Positive). Successful AE attacks can diminish the effectiveness of the IDS model or even render it virtually useless. Based on the attacker's knowledge, there are three types of attacks on machine learning-based systems: white-box attacks, gray-box attacks, and black-box attacks.

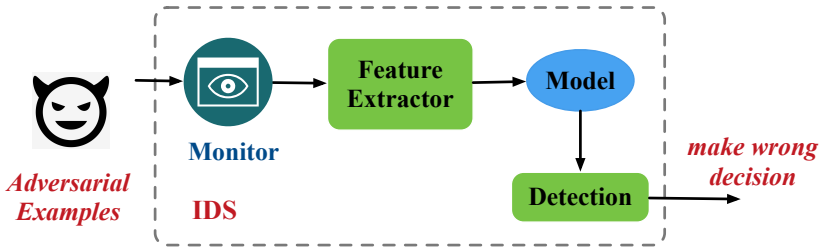


Figure 3.2: Attacking goal of adversarial example generator in IDS.

- A white-box adversarial attacker has full knowledge of both the architecture and weights of the IDS model.
- A gray-box adversarial attacker knows the IDS model architecture but not the weights. They can query the model while attempting to minimize the number of queries to avoid raising suspicion.
- A black-box adversarial attacker lacks information about the architecture and weights of the IDS model. They can query the model while trying to reduce the number of queries to avoid being detected as suspicious.

In this study, we focus on the white-box attack on the IDS system, which is considered the most potent of the three types from the attacker's perspective. This approach enables

Table 3.1: Symbol definition.

Symbol	Definition
$x \in \mathbb{R}^n$	input feature vector
$\mathcal{F}$, θ	IDS model and model parameter
$y \in \mathbb{R}^m$	output probability vector ($y = \mathcal{F}(\theta, x)$)
$c(x)$	final output label, $c(x) = \text{argmax}_i y[i]$
x'	feature-space AE generated based on x
η	adversarial perturbation, $x' = x + \eta$
z, z'	projection of x and x' in the problem space
$J(\theta, x)$	loss of IDS $\mathcal{F}$ on x
S_{diff}	differentiable feature set
$S_{non-diff}$	non-differentiable feature set
S_{func}	functional feature set
$S_{non-func}$	non-functional feature set
L_2	L_2 distance between x and x'
p	upper bound change ratio of a feature
R_i	range of the i-th feature
f_{ji}	correlation function from j-th feature to i-th feature
S^i_{corr}	feature set that correlates to the i-th feature

attackers to create adversarial network instances in the most efficient and discreet way possible to overcome the IDS system. By examining this powerful threat model, we aim to demonstrate the efficacy of our defense. Additionally, we assume that the attacker has black-box access to the adversarial example detection method.

The general goal of adversarial examples can be formulated as follows. Let $\mathcal{F}$ represent an m-class classifier with model parameter θ. The model maps input ($x \in \mathbb{R}^n$) to output ($y \in \mathbb{R}^m$), that is, $y = \mathcal{F}(\theta, x)$. It is important to note that $y[i], i = 1, ..., m$ signifies the probability that x belongs to the i-th class, and $y[1] + ... + y[m] = 1$. The predicted label of x, $c(x)$, corresponds to the class with the highest $y[i]$ value, or $c(x) = \arg\max_i y[i]$. The aim of AE generation is to find $x' = x + \eta$ (where η represents a small perturbation) such that $c(x') \neq c(x)$. We also assume that $J(\theta, x)$ denotes the loss function for input x. The symbols used throughout the chapter are displayed in Table 3.1.

We have selected four of the most representative and effective AE attacks from the numerous

proposed in the literature to address in this chapter. In the following sections, we briefly review these four attacks and highlight the techniques employed in each.

1. FGSM: Goodfellow et al. [45] proposed the Fast Gradient Sign Method (FGSM) to generate adversarial examples. In this approach, an attacker calculates the gradient of the loss function concerning x and then shifts the current x in the direction that maximizes $J(\theta, x)$. The generated adversarial example can be represented as:

$$x' = x + \epsilon \text{sign} \nabla_x J(\theta, x).$$

2. BIM: Kurakin et al. [66] proposed the Basic Iterative Method (BIM), an extension of FGSM. This attack employs FGSM multiple times, taking small steps with each application. During each FGSM application, BIM clips the pixel value of intermediate results to ensure that the generated adversarial example remains within the ϵ-neighborhood of the original input. The parameter ϵ serves as a global constraint to limit the distance between x and its corresponding adversarial example.

$$x_0^{'} = x$$
$$x_i^{'} = \text{clip}_{x,\epsilon}(x_{i-1}^{'} + \alpha \text{sign} \nabla_x J(\theta, x_{i-1}^{'})), i \geq 1.$$

3. JSMA: Papernot et al. [108] introduced the Jacobian-based Saliency Map Attack (JSMA), which utilizes iterative computation to identify features that contribute significantly to misclassification at each step. For a given input x, the prediction confidence for the j-th class is denoted by $f_j(x)$. To generate an adversarial example for a target class t, the applied perturbation must meet two requirements simultaneously: a) $f_t(x)$ increases, and b) $f_j(x)$ decreases for all $j \neq t$. The adversarial saliency map

(for the i-th feature) is defined as:

$$S(x,t)[i] = \begin{cases} 0, \text{if } \frac{\partial f_t(x)}{\partial x_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial f_j(x)}{\partial x_i} > 0 \\ \frac{\partial f_t(x)}{\partial x_i} | \sum_{j \neq t} \frac{\partial f_j(x)}{\partial x_i} > 0 |, \text{otherwise.} \end{cases}$$

4. CW attack Carlini and Wagner [21] presented optimization-based attacks for generating adversarial examples. They utilized three distance metrics—L_0, L_2, and L_∞ distance—to assess the distortion of an adversarial example from its original input. The process of generating adversarial examples is formulated as follows:

$$\begin{aligned} &\min \|x' - x\| + c \cdot \max\{\max\{f_i(x') : i \neq t\} - f_t(x'), -\kappa\} \\ &\text{subject to } x' \in [0,1]^n \end{aligned}$$

It is important to mention that another widely recognized adversarial example attack, the Projected Gradient Descent (PGD) attack [83], is fundamentally similar to the BIM attack. We choose the latter for the sake of convenience. In most instances, the Carlini and Wagner (CW) attack surpasses the other three methods in terms of both effectiveness and size of distortion.

3.4 Analysis of Adversarial Attacks in IDS

In this section, we discuss the legitimacy of AE attacks in intrusion detection domain from the point of view of an attacker. Valid inputs to an IDS system consist of real network traffic flows within the problem space, which includes network packets with tangible meanings. Consequently, the generated adversarial examples should also reside within the same problem space as the IDS. To produce adversarial examples that can be mapped back to valid real

network events, we adapt existing feature-space adversarial example generation algorithms to problem-space algorithms.

3.4.1 Problem-Space AE Attack

In this section, we demonstrate the generation of adversarial examples for IDS in the problem space. The concept of problem-space attack stems from the need to generate physically meaningful, domain-specific attack instances, rather than merely modifying adversarial samples in feature space [114]. The problem space of an IDS encompasses all possible traffic instances in the form of network packet sequences. In contrast, the feature space of an IDS includes all possible feature vectors in the form of numerical entries representing packet length, packet inter-arrival time, and so on. Previous adversarial example generation algorithms [21, 66, 108] focused solely on image inputs, where the problem-space and feature-space adversarial examples are identical (i.e., a vector of pixels). Problem-space adversarial example attacks on network flow-based IDS have not yet been explored. Our work aims to address this gap.

Generating an adversarial example in the problem space involves two steps. First, we create a feature-space adversarial example x' from a clean input x. Second, we design a mapping function to project x' into the problem space, obtaining the final problem-space adversarial example z'. The corresponding representation of x in the problem space is denoted by z.

Conventional inverse mapping methods, as referenced in [29, 76], are not suitable for converting instances from feature-space to problem-space within our problem framework due to the mapping's non-invertible and non-differentiable nature. We adjust the AE generation processes by neutralizing perturbations on non-differentiable features to enable a differentiable mapping. We represent the i-th feature of $\boldsymbol{x}$ and $\boldsymbol{x'}$ as $\boldsymbol{x}[i]$ and $\boldsymbol{x'}[i]$,

respectively. We categorize features into two groups:$\{S_{diff},\ S_{non-diff}\}$. We mandate that $x'[i] = x[i]$ for $i \in S_{non-diff}$, resulting in $z'[i] = z[i]$ for $i \in S_{non-diff}$. Once the non-differentiable features are excluded, we map the differentiable features of x back to z. In an IDS, non-differentiable features include categorical aspects such as 'protocol type' and 'service type.' Suppressing perturbations on these non-differentiable features aids in obtaining a meaningful AE. Simultaneously, we require that an AE maintains its original purpose, meaning that an intrusion flow keeps its attacking ability while a benign traffic flow sustains its non-malicious functionality. We describe preserving the essential qualities and retaining the original function as maintaining the intrinsic property. We discuss methods to support the preservation of intrinsic property in the following section.

We note that an attack might lose its harmful properties if an attacker can freely alter features of a network traffic flow. Additionally, we observe that AEs created by widely-used adversarial generation techniques do not attempt to preserve potential data correlations between pairs of features. Such AEs can be easily filtered by establishing data correlations as fundamental filtering rules. We refer to these situations as a loss of the AE's intrinsic properties. To address this issue, we provide some general guidelines that involve adding constraints to the AE generation process. Our AE generation approach is more likely to maintain an instance's properties and validity compared to an unconstrained method. For a detailed examination of preserving network traffic flow properties in AE generation, readers are directed to [112, 143]. These guidelines can be applied to various problem domains, including but not limited to IDS.

1. Distinguish between functional features S_{func} and non-functional features $S_{non-func}$. Aim to avoid altering the values of functional features, which directly impact the instance's functionality. For example, in the IDS problem domain, 'port number' and 'network service used' are functional features. A straightforward method to identify

functional features is to consider features with categorical values as functional by default. For instance, 'protocol type', with three potential categorical values ('icmp', 'tcp', and 'udp'), is a functional feature. Keep in mind that domain expertise is required to determine the final set of functional features, as non-categorical features may also be functional.

2. Ensure that the value of a modified feature remains valid. For continuous features, the modified value should be within the acceptable range, while discrete features should stay within the set of valid values after modification.

3. Maintain a small perturbation magnitude for non-functional features, as large perturbations on these features may alter an instance's intrinsic properties.

4. Apply consistent modifications to correlated features, as it is unrealistic to assume that all features are independent. Independently adding perturbations to correlated features may jeopardize the validity of an instance.

These guidelines aim to preserve an instance's intrinsic properties and validity as much as possible. We refer to an AE that adheres to these guidelines as a problem-space AE. We attempt to maintain the properties of adversarially generated examples by imposing multiple constraints (i.e., guidelines). However, it is still possible for a problem-space AE to deviate from its desired properties or become invalid after perturbation. We impose the following additional constraints for problem-space AE generation compared to feature-space

AE generation:

$$\|\boldsymbol{x'} - \boldsymbol{x}\|_2 \leq L_2,$$
$$\|\boldsymbol{x'}[i] - \boldsymbol{x}[i]\| \leq p * R_i, \text{ if } i \in (S_{diff} \cap S_{non-func}),$$
$$\boldsymbol{x'}[i] = \boldsymbol{x}[i], \text{ if } i \in (S_{non-diff} \cup S_{func}),$$
$$\boldsymbol{x'}[\boldsymbol{i}] = f_{ji}(\boldsymbol{x'}[j]),\ j \in S^i_{corr},$$

where x denotes the original input and $\boldsymbol{x'}$ represents an AE generated from $\boldsymbol{x}$. L_2 is used to denote the maximum L_2 distance between the original input and the generated AE. We use R_i to represent the range of the i-th feature $x[i]$ and p to represent the maximum change ratio for a feature. S_{diff} corresponds to the set of differentiable features. Only the differentiable and non-functional features are eligible for modification. S^i_{corr} denotes indexes of features that are correlated to the i-th feature $x[i]$ and f_{ji} is the mapping function from feature j to feature i.

3.4.2 Properties of AE

First, we explore the concept of manifold learning to better present our AE categorization scheme. Manifold learning is based on the assumption that input data resides on or close to a low-dimensional manifold embedded within the ambient space [79]. For instance, a plane can be considered a manifold for a group of three-dimensional data points if they lie on a plane (a flat, 2-dimensional surface). Manifold learning involves automatically discovering the geometric and topological properties of a given manifold [155]. Most manifold learning techniques concentrate on data representation, as demonstrated in [13, 121, 139]. Let $\mathcal{M}$ represent a manifold model. Formally, we define the inference on an input x as 'manifold evaluation,' denoted by $\mathcal{M}(x)$. Similar to machine learning, the output of manifold

evaluation is a vector displaying the probabilities of x residing in each sub-manifold (i.e., each class).

Next, we return to the analysis of AE generated for IDS. The IDS model, $\mathcal{F}$, maps an input sample x to a confidence vector $y \in p_0, p_1$. The final predicted class is $c(x) = \arg\max \mathcal{F}(\theta, x)$. A classification of $c(x) = 1$ signifies that x is labeled as malicious, while $c(x) = 0$ indicates a benign classification. We denote an AE generated from x as $x' = x + \eta$. There are two attack scenarios: 1) $c(x) = 1$ (the true label of x is also 1) and the goal of the AE attack is $c(x') = 0$; and 2) $c(x) = 0$ (the true label of x is also 0) and the goal of the AE attack is $c(x') = 1$. Our defense does not rely on the value of $c(x)$, so all analyses for the first attack scenario apply to the second as well. For consistent illustration purposes, we use the first attack scenario in the following discussion (both scenarios are considered in our experiments). We assume that x' must maintain the intrinsic property of its true class, i.e., 'malicious'. In other words, even though x' appears closer to the 'benign' class in the IDS model's view, it does not fully conform to the 'benign' class properties. Otherwise, x' would not be an AE but rather a new instance of the 'benign' class, which would not have any attack impact.

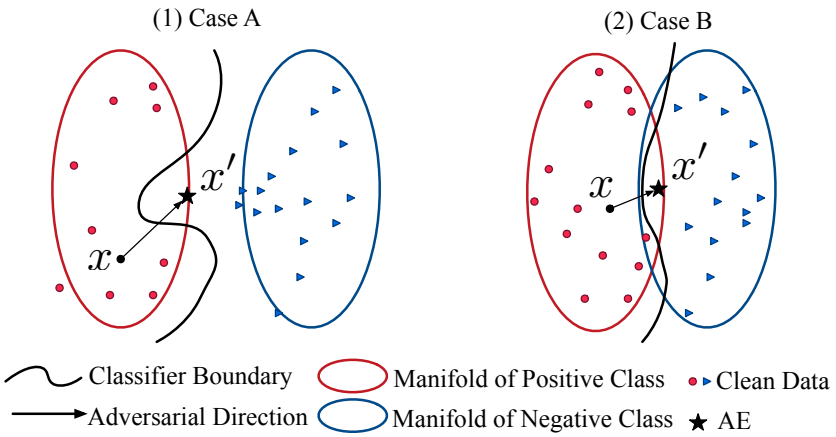


Figure 3.3: The illustration of AEs positions to the decision boundary in the 2-D view.

Assuming that AE must maintain the essential property of its true class (as supported by our experimental results), successful AEs can be divided into two categories:

- Case A: x' is close to the 'malicious' class manifold but far from the 'benign' class manifold. This often occurs when malicious and benign manifolds are entirely separable from each other.
- Case B: x' is close to both manifolds. This situation arises when both manifolds are near each other or even overlapping (i.e., they cannot be perfectly separated).

Figure 3.3 illustrates Case A and Case B for two-dimensional data. In practice, the dimension of a manifold depends on the dimension and distribution of input samples. It is noted that the two cases (i.e., Case A and Case B) are not mutually exclusive. They may coexist for the same IDS model at different segments of the decision boundary.

3.4.3 Assessing Adversarial Example Validity

We identify three scenarios that do not qualify as successful adversarial examples (AE) in our study. The first scenario pertains to original sample for generating AEs, while the other two focus on the reliability of the generated AEs. We describe these two scenarios as follows:

- If the original sample is misclassified by the Intrusion Detection System (IDS) model, there is no need to generate an AE for that sample.
- If the generated AE fails to deceive the IDS model into misclassifying it, the AE is not considered successful.
- If the generated AE loses its fundamental characteristics, it cannot be deemed a successful AE.

To provide further insight into the third scenario, we present the following example. Suppose we have an original traffic flow instance x, which the IDS model can correctly identify as

an intrusion. An attacker generates an AE x' for x. If x' is significantly distant from the "malicious" class manifold and near the "benign" class manifold, x' is not a successful AE since it violates the assumption that an AE must maintain the essential properties of its true class. The same argument applies if x' is far from both manifolds.

3.5 Detailed Design of MANDA System

In this section, we introduce MANDA, our proposed adversarial example detector for machine learning-based intrusion detection systems, and discuss the rationale behind each design decision. For detecting AE, we propose two rules within the MANDA system. The key insight for detecting adversarial examples is to recognize the discrepancy between true benign samples and adversarial examples. This understanding encourages us to examine the position of adversarial examples relative to the decision boundary of the IDS model and their placement within the traffic manifolds formed by training samples.

3.5.1 Overview of MANDA

Here we introduce our AE detection scheme, MANDA, which is based on the above AE categorization. We assume that the IDS model has high accuracy on clean data since it makes less sense to discuss detecting AEs already misclassified by the IDS model. A clean input needs to traverse the decision boundary of the IDS model to be an AE. As mentioned in Section 3.1, MANDA consists of two components, Manifold and DB. Manifold combines both x's classification and manifold evaluation results to detect AEs. If the two outputs of x are inconsistent, x is highly likely to be an AE. DB examines whether an input x is near the decision boundary of the IDS model to detect AE. Specifically, if we add small Gaussian

noise to x and the corresponding y (and thus $c(x)$) changes frequently, x is highly likely to be an AE. We want to emphasize that both Manifold and DB can be used as a stand-alone AE detection scheme while MANDA combines them together for better performance.

3.5.2 Detection by Manifold

In Case A, as depicted in Figure 3.3, an adversarial example is located within (or close to) the manifold of the "malicious" class but far from the "benign" class. Simultaneously, the IDS model classifies the AE as "benign". Consequently, the IDS model's output and the manifold evaluation results for the same input are inconsistent. In contrast, results for clean inputs tend to be consistent. Intuitively, we can leverage the inconsistency between the IDS model and manifold evaluation as a criterion for detecting AEs.

To capture the data manifold for the malicious and benign classes, we utilize a transductive learning model proposed by Zhou et al. in [178]. This learning method examines the inherent structure collectively revealed by a set of labeled and unlabeled data points. It ensures both local and global consistency of known data points. In other words, (1) points in close proximity are likely to share the same label, and (2) points within the same structure (commonly referred to as a manifold) are likely to have identical labels. The manifold evaluation model obtained from this learning method is smooth with respect to the intrinsic data structure. We employ this learning approach to obtain the manifold for each class in this study.

Detection Rule 1 We classify an input as an AE if there is an inconsistency between the IDS model's identification and the manifold evaluation.

In the implementation of the Detection Rule 1, we calculate a score $score_1$ to estimate the likelihood of an input being adversarially generated, as illustrated in Algorithm 2.

Algorithm 2 Score-Compute() for Detection Rules 1 & 2

Input: test data record $x \in \mathbb{R}^n$, IDS model $\mathcal{F}(\theta)$, manifold evaluator $\mathcal{M}$
Output: $score_1, score_2$

$p \leftarrow \mathcal{M}(x)$ # confidence vector of manifold evaluation.
$q \leftarrow \mathcal{F}(\theta, x)$ # confidence vector of IDS model.
$score_1 \leftarrow \|p\| + \|q\| - \|p+q\|$ # Detection Rule 1
for $i = 0$ to N do
    $x_i = x + \mathcal{N}(0, \sigma^2)$
    $p_i \leftarrow \mathcal{F}(\theta, x_i)$
end for
$score_2 \leftarrow \frac{1}{N}\sum_{i=1}^{N} \|p_i\| - \frac{1}{N}\left\|\sum_{i=1}^{N} p_i\right\|$ # Detection Rule 2
return $score_1, score_2$

Algorithm 3 Manifold Detector

Input: test data record $x \in \mathbb{R}^n$, IDS model $\mathcal{F}(\theta)$, manifold evaluator $\mathcal{M}$, threshold τ_1
Output: $isAdversarial \in \{False, True\}$

$score_1, \sim \leftarrow$ Score-Compute$(x, \mathcal{F}, \mathcal{M})$
if $(score_1 > \tau_1)$ then
    $isAdversarial \leftarrow True$
end if
return $isAdversarial$

Specifically, we compare the confidence vector p from the manifold evaluation with q from the IDS model and calculate the discrepancy between the two confidence vectors by

$$score_1 = \|p\| + \|q\| - \|p+q\| \tag{3.1}$$

If the two confidence vectors are identical, then the score is zero. And the score increases with the difference between the two vectors. We then compare $score_1$ to an optimal threshold τ_1 to determine if an input sample is an AE (see Algorithm 3). The threshold τ_1 is chosen based on the score statistics of clean data points. In the evaluation section, we select the 95th percentile of the clean data points' scores as τ_1.

3.5.3 Detection by DB

In contrast to Case A, the two manifolds of "malicious" and "benign" classes are not completely separable in Case B. In this scenario, Detection Rule 1 will not be effective, as the manifold evaluation is not accurate. To address this situation, we propose an alternative detection rule based on decision boundary analysis, which we refer to as DB. Before delving into the details of the DB detection rule, we first examine the characteristics of AEs in this case and present the following proposition:

PROPOSITION 1: In an intrusion detection system consisting of two classes– malicious and benign classes, if

(i) the two manifolds of 'malicious' and 'benign' classes are not fully separable but most instances are still distinguishable;

(ii) IDS classifier $\mathcal{F}$ is with optimized accuracy;

(iii) x is a clean malicious input, i.e., $c(x)=1$. x' is x's corresponding AE where $c(x')=0$ (or x is a clean benign input, i.e., $c(x)=0$ and $c(x')=1$); and

(iv) x' keeps the essential property of the 'malicious' class.

Then, x' is very close to the decision boundary of $\mathcal{F}$ with high probability.

In Case B, as depicted in Figure 3.3, an AE should be very close to the decision boundary according to Proposition 1. Consequently, we employ the proximity of an input to the decision boundary as a secondary detection rule for AE detection. Estimating the distance of an input to the decision boundary is not trivial, as the decision boundary is not easy to capture. We propose using the sensitivity of input to small perturbations as a method to evaluate the closeness of an AE to the decision boundary. It is important to note that this

Algorithm 4 DB Detector

Input: test data record $x \in \mathbb{R}^n$, IDS model $\mathcal{F}(\theta)$, threshold τ_2
Output: $isAdversarial \in \{False, True\}$

```
∼, score_2 ← Score-Compute(x, F, ∼)
if (score_2 > τ_2) then
    isAdversarial ← True
end if
return isAdversarial
```

detection rule may identify a clean input near the boundary as an AE. However, due to the curse of dimensionality, only a few correctly classified clean inputs are close to the boundary [53]. Our experimental results also support this hypothesis. On the other hand, this implies that the IDS model may not be confident in its predictions when an input data point is very close to the decision boundary. As a result, reporting such inputs to intrusion detection management can be beneficial for improved system performance.

DB aims to assess whether an input is a near-boundary example in high-dimensional space. We accomplish this by evaluating the uncertainty of the IDS model's output when the input is subjected to a small additive perturbation. For a near-boundary example, such a small perturbation might cause it to cross the decision boundary, resulting in highly unstable outputs from the IDS model when the input is perturbed. In contrast, a small perturbation on an input far from the boundary is unlikely to cause such a change. We calculate model uncertainty on an input with an additive Gaussian perturbation $\mathcal{N}(0, \sigma^2)$ in Algorithm 2. For an input x, the uncertainty of the output from the IDS model is evaluated as the variance of the confidence vector $\mathcal{F}(\theta, x_i)$ for input $x_i = x + \mathcal{N}(0, \sigma^2), (i \in \mathbb{N}, i \leq N)$:

$$score_2 = \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{F}(\theta, x_i)\| - \frac{1}{N}\left\|\sum_{i=1}^{N} \mathcal{F}(\theta, x_i)\right\|. \tag{3.2}$$

Similar to Manifold, DB employs an optimal threshold for $score_2$ to determine whether x is an AE or not. The DB method is illustrated in Algorithm 4. DB first computes $score_2$

using Eq. 3.2. Then, DB compares $score_2$ to an optimal threshold τ_2 to decide if an input sample is an AE. It is worth noting that the selection of the threshold τ_2 is based on the score distribution of clean data, aligning with the approach of calculating τ_1. We summarize the detection rule as follows.

Detection Rule 2 We classify an input with high model uncertainty on small perturbations as an AE.

The two detection rules can function independently, but both may produce some misclassifications since they are designed for specific scenarios. In the subsequent section, we aim to develop a comprehensive detection method that operates effectively for both cases by integrating the two rules.

3.5.4 Combination of Manifold and DB

MANDA is designed to optimally utilize both Manifold and DB for AE detection. There are two approaches to combine the two detection rules: an unsupervised approach and a supervised approach.

- Unsupervised Approach: In this case, we connect the two rules sequentially. We first use the Manifold detection rule to filter out suspicious AEs and directly report an alert. If the input is flagged as clean, we additionally apply the DB rule. If the DB rule classifies the input as an AE, we report an alert.

- Supervised Approach: We generate some AEs using the clean training dataset. We then mix these AEs with the clean training inputs and denote the combined inputs as X. Next, MANDA obtains $[score_1, score_2]$ for each input in X by executing Algorithm 2. We assign label 1 to $[score_1, score_2]$ if the input is an AE and label 0 if the input is

Algorithm 5 MANDA

Input: IDS model $\mathcal{F}(\theta)$, manifold evaluator $\mathcal{M}$, test data record $x_{test} \in \mathbb{R}^n$, training dataset X, AE flag $\mathrm{Y_{adv}}$
Output: $isAdversarial \in \{False, True\}$

```
if training then
    S1, S2 ← Score-Compute(X, F, M)
    model ← LogistcRegression(S1, S2, Y_adv)
else
    score1, score2 ← Score-Compute(x_test, F, M)
    isAdversarial ← model(score1, score2)
end if
return isAdversarial
```

clean. We then create a new dataset $([\mathrm{S_1}, \mathrm{S_2}], \mathrm{Y_{adv}})$, where $[\mathrm{S_1}, \mathrm{S_2}]$ represents the set of scores for each input. Finally, MANDA trains a logistic regression model on the new dataset and uses it for AE detection. Algorithm 5 illustrates the MANDA process.

3.6 Experimental Results

3.6.1 Datasets

The NSL-KDD dataset [138] has been used in AE attacks in intrusion detection systems [80]. NSL-KDD contains 125,973 training records and 22,544 testing records. Each record has 41 entries (in problem space) and is further processed into 121 numerical features as an input-space (feature-space) vector. The 41 features can be divided into four groups: Intrinsic Characteristics, Content Characteristics, Time-based Characteristics, and Host-based Characteristics. There are four categories of intrusion: DoS, Probing, Remote-to-Local (R2L), and User-to-Root (U2R), with each containing more attack sub-categories. The training set has 24 sub-categories of attacks, while the testing set has 38 sub-categories (14 of which are unseen in the training set). Our experiments focus on

evaluating an IDS model that discriminates intrusions from normal traffic. We trained a binary classifier that flags an input as benign or intrusion.

The CICIDS2017 dataset [124] includes benign traffic and 12 up-to-date attacks, resembling real-world Packet Capture (PCAPs) data. The flow-based dataset is extracted from PCAPs files using CICFlowMeter[2], based on the timestamp, source and destination IPs, source and destination ports, and protocols. After primary preprocessing (e.g., removing NaN), there are 2,416,775 data records. Each data record, corresponding to a traffic flow, contains 78 features and is assigned a label indicating the attack type or benign. We randomly split the dataset into a training set with 1,933,420 data records and a test set with 483,355 data records. Our experiments focus on evaluating an IDS model that discriminates intrusions from normal traffic and further identifies the attack types. We trained a multi-class classifier that classifies an input as benign or a specific type of intrusion.

The MNIST dataset [68] consists of handwritten digits from 0 to 9. Each image's corresponding digit serves as its label. Each class has 6,000 training samples and 1,000 test samples, resulting in a total of 60,000 training samples and 10,000 test samples for the entire MNIST dataset. All images have the same size of 28×28 and are in grey-level.

3.6.2 Experimental Settings and Evaluation Metrics

We implemented the problem-space attacks and MANDA in TensorFlow and conducted all the experiments on a server equipped with an Intel Core i7-8700K CPU @ 3.70GHz×12, a GeForce RTX 2080 Ti GPU, and Ubuntu 18.04.3 LTS.

For the NSL-KDD dataset and the CICIDS dataset, we build the IDS model using a multilayer perceptron (MLP) with one input layer, one hidden layer with 50 neurons, and one

[2] https://github.com/ahlashkari/CICFlowMeter

output layer. For completeness, we also implemented other models for IDS, including Logistic Regression (LGR), K-Nearest Neighbors (KNN), Naive Bayes classifier for multivariate Bernoulli (BNB), Decision Tree Classifier (DTC), and Support Vector Machine (SVM) using the scikit-learn library [111]. We implemented four AE attacks: FGSM, BIM, CW (the L_2-norm version), and JSMA (refer to Section 3.3.3). We adapted the first three to the problem space of IDS. We generated AEs on the test samples that the IDS model correctly classified in each experiment, excluding misclassified test samples. Next, we combined the successful AEs with an equal number of clean data points (randomly selected) to create a mixed dataset, on which we ran all detection algorithms. The benchmark for comparison is Artifact [37], the same as in [20, 53]. Artifact, proposed by Feinman et al. in [37], is one of the state-of-the-art AE detection schemes. Unlike MANDA, Artifact uses kernel density estimation (KDE) and Bayesian neural network uncertainty as two criteria for detecting AEs.

For the image dataset, the MNIST dataset, we used a convolutional neural network (CNN) instead of the MLP as the target model for AE attacks. The CNN model comprises 4 convolutional layers with ReLU activation, followed by 2 fully connected layers.

We use the following evaluation metrics for AE detection: True Positives (TPs), False Positives (FPs), True Negatives (TNs), and False Negatives (FNs):

- TP: a sample is an AE and detected as an AE.
- FP: a sample is a clean sample but detected as an AE.
- TN: a sample is a clean sample and detected as clean.
- FN: a sample is an AE but detected as clean.

We then calculate the True Positive Rate (TPR) and False Positive Rate (FPR) as follows:

Perturbation restriction (%)	FGSM		BIM		CW	
	Acc (%)	L_2	Acc (%)	L_2	Acc (%)	L_2
0	90.64	0	90.64	0	90.64	0
1.0	84.48	1.40	83.84	1.54	77.02	1.08
2.5	71.10	1.51	64.05	1.58	52.61	1.45
5.0	64.27	1.58	59.48	1.58	42.68	1.58
7.5	60.09	1.67	55.95	1.62	37.47	1.68
10.0	56.63	1.79	52.79	1.67	34.51	1.67
No restriction	2.42	2.57	0.08	1.53	0.00	0.96

[a]Perturbation is only applied to a subset of features.

Table 3.2: The accuracy of IDS model under AE attacks using the NSL-KDD Dataset.

$\text{TPR} = \frac{\#TPs}{\#TPs+\#FNs}, \text{FPR} = \frac{\#FPs}{\#TNs+\#FPs}$.

The Receiver Operating Characteristics (ROC) curve is created by plotting the TPR against the FPR at various threshold settings. The Area Under the Curve - Receiver Operating Characteristics (AUC-ROC) score is defined as the area under the ROC curve, and we use shortcut AUC to refer to it in the following discussions.

3.6.3 AE Attack on the NSL-KDD Dataset

We present the classification accuracy of the IDS model under FGSM, BIM, and CW attacks in TABLE 3.2. The accuracy degradation is further illustrated in Figure 3.4. The first row of TABLE 3.2 with zero perturbation restriction indicates that no perturbation is added to the input samples, and the accuracy of 90.64% represents the IDS accuracy in an attack-free scenario. We can draw two main conclusions from the experimental results:

- The larger the perturbation used in AE attacks (i.e., p), the more powerful the AE attacks become, and consequently, the lower the targeted IDS model's accuracy. Recall that p represents the maximum change ratio on each feature of the modifiable feature set S_{diff} (in Section 3.4.1). TABLE 3.2 demonstrates that the model accuracy drops from 90.64% with $p = 0$ to 34.51% with $p = 10\%$ under the CW attack (the strongest

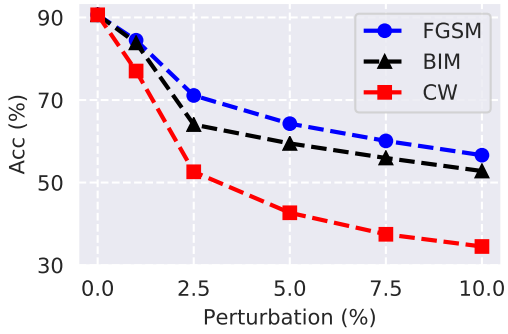


Figure 3.4: The accuracy degradation of the IDS model under adversarial attacks using the NSL-KDD dataset.

attack investigated).

- The success attack rate of feature-space attacks (i.e., the last row in TABLE 3.2) is higher than that of problem-space attacks (i.e., any row other than the last in TABLE 3.2) because the latter faces more restrictions in generating AEs (See Section 3.4.1). Problem-space AE attacks can still cause a significantly low accuracy of the targeted IDS model with larger p.

In the following sections, we will use $p = 5\%$ and evaluate the detection performance of our proposed approach under AE attacks.

Adversarial examples generated for one target model can transfer to other models, known as AE transferability [106]. The transferability makes AEs effective even in a black-box attack scenario. We investigate the transferability of AE among different IDS models. TABLE 3.3 presents the results of our problem-space adversarial examples generated from the MLP model and then applied to models including LGR, KNN, BNB, SVM, and DTC. The significant decrease in accuracy confirms that our problem-space AE generation scheme maintains the capability of adversarial examples to transfer to different models.

Table 3.3: AE transferability among different victim models.

Models	Acc (%)	Acc (%) after attack			
		FGSM	BIM	CW	Overall
LGR	89	15	70	52	45
KNN	91	33	38	51	40
BNB	87	84	86	26	65
SVM	89	22	26	25	24
DTC	84	79	79	73	77

Table 3.4: AE detection performance on the NSL-KDD dataset

Detection Method	FGSM			BIM			CW		
	TPR(%)		AUC	TPR(%)		AUC	TPR(%)		AUC
	FPR=5%	FPR=15%		FPR=5%	FPR=15%		FPR=5%	FPR=15%	
Manifold (Ours)	94.04	100.00	0.9792	98.41	99.98	0.9714	98.38	100.00	0.9805
DB (Ours)	17.27	53.57	0.8471	27.91	98.62	0.9340	71.00	97.91	0.9439
MANDA (Ours)	92.88	99.89	0.9765	98.04	100.00	0.9726	95.93	100.00	0.9851
Artifact [37]	12.60	96.63	0.8984	14.78	96.31	0.9023	28.07	97.66	0.9123

3.6.4 AE Detection on the NSL-KDD Dataset

Here, we present the performance of the proposed detection schemes against the generated adversarial examples, including MANDA with Manifold only, MANDA with DB only, and MANDA.

Figure 3.5 presents the ROC curves for detecting three AE attacks—FGSM, BIM, and CW attacks. It is evident that Manifold and MANDA achieve similar ROC results, outperforming both DB and Artifact. These results suggest that most adversarial examples can be successfully detected by the inconsistency between the IDS model and the manifold model. We further compute the AUC and TPR when FPR is fixed in TABLE 3.4 to better demonstrate the detection performance. From the table, we observe that the best AUC score under FGSM attack is achieved by Manifold. For BIM attack and CW attack, MANDA achieves 0.9726 and 0.9851 AUC and outperforms other detection methods. We also present the TPR by fixing FPR as 5% and 15% in TABLE 3.4. The proposed Manifold achieves the

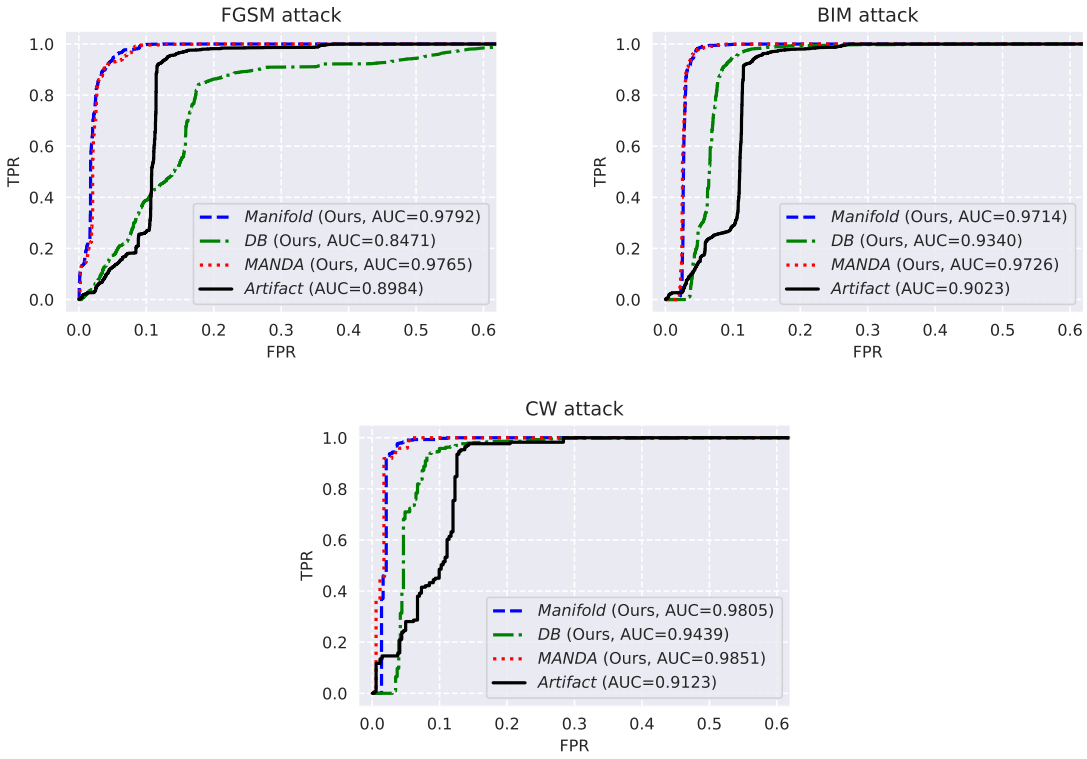


Figure 3.5: ROC curve of AE detection results using the NSL-KDD dataset.

highest TPR under FGSM attack when FPR=5% and 15%. And MANDA outperforms the other methods under CW attack. For the BIM attack, Manifold achieves the highest TPR when FPR=5%, while MANDA works best when FPR=15%.

In summary, Manifold and MANDA outperform other baselines in AE detection by achieving the highest AUC scores and TPR. In most cases, the performance of Manifold and MANDA are comparable, indicating that the Manifold component contributes the most to the final detection. The possible reason behind this is that the malicious manifold and benign manifold are well-separated in the NSL-KDD dataset, and Manifold detection is highly effective in this scenario. We anticipate that MANDA would outperform Manifold when the malicious manifold and benign manifold are not perfectly separable. In this scenario, as pointed out in Section 3.5.1, the DB component in MANDA will be able to detect those AEs that Manifold

fails to detect. As a result, MANDA can detect more AEs than Manifold.

We also evaluate the computation complexity of the proposed AE detection methods. In the detection phase, the three methods, Manifold, DB, and MANDA, has similar execution time. And the average detection time for one data record (i.e., a network traffic flow) is 0.26 milliseconds, making them excellent candidates for fast online detection.

3.6.5 AE Attack on the CICIDS Dataset

To demonstrate the efficacy of MANDA on multi-class IDS, we further evaluate MANDA on the CICIDS dataset. For the CICIDS dataset, we build a multi-class IDS model capable of identifying the attack type of an intrusion. In contrast to a binary-class IDS, the multi-class IDS model is more powerful as it can not only detect malicious incoming traffic flows but also recognize their attack types. However, this also makes AE detection for a multi-class IDS much more challenging, as each type of attack tends to hide its maliciousness in its own way. In order to conduct a meaningful attack, we only consider the attack scenario where an attacker tries to evade detection (i.e., appear benign) but not the scenario where an attacker pretends to be another attack type or a normal one pretends to be malicious. The accuracy of IDS on the CICIDS dataset without adversarial attacks is 98.2% in our evaluation.

TABLE 3.5 shows the Attack Success Rate (ASR) of FGSM, BIM, and CW on the IDS

Table 3.5: ASR using the CICIDS dataset.

Perturbation restriction (%)	FGSM	BIM	CW
1.0	72.8	87.8	100.0
2.5	83.6	90.0	100.0
5.0	85.4	90.3	100.0
7.5	86.0	90.4	100.0
10.0	86.6	92.7	100.0
None	99.7	99.7	100.0

[a]Perturbation is only applied to a subset of features.

model. A high ASR indicates low detection effectiveness. We omit the presentation of IDS accuracy under attacks since ASR and IDS accuracy are correlated to each other. And the correlation can be represented by

$$\text{ACC}a = (1 - \text{ASR}) * \text{ACC}n \tag{3.3}$$

where ACCa denotes the IDS accuracy under AE attacks and ACCn denotes the original accuracy of IDS without attack. For instance, with ASR = 90%, we can derive that the IDS model accuracy drops to $98.2\% * (1 - 90\%) = 9.8\%$.

One observation from comparing the performance of the NSL-KDD dataset and the CICIDS dataset is: the ASR for the CICIDS dataset surpasses the ASR for the NSL-KDD dataset. One potential explanation for this observation could be the different feature engineering techniques employed by the IDS models designed for each dataset. Furthermore, as mentioned earlier, generating AE for a multi-class model might be simpler than for a binary model. This observation that AE attacks effectively undermine two IDSs based on separate feature engineering approaches highlights the power of AE attacks in IDSes.

3.6.6 AE Detection on the CICIDS Dataset

In this section, we assess the performance of MANDA on the CICIDS2017 dataset and compare it with other baseline defense techniques. To facilitate a more comprehensive understanding of the evaluation results, we will first present the fundamentals of the multi-class ML-based IDS model.

In a multi-class problem, one can envision a decision boundary existing between each pair of classes. A decision boundary is a set of points where the a posteriori probabilities are equal, taking the form of a point, line, plane, hyperplane, solid, hypersolid, curved surface,

or curved hypersurface [69]. The definition of a decision boundary is as follows:

Definition 3.1. For a n-class model $\mathcal{F}$ with model parameter θ, a decision boundary between class i and j $(i, j \leq n)$ is defined as

$$D_{ij} : \{\mathrm{x} | \mathcal{F}(\theta, \mathrm{x})[i] = \mathcal{F}(\theta, \mathrm{x})[j]\}$$

where $\mathcal{F}(\theta, \mathrm{x})$ is the confidence vector, and $\mathcal{F}(\theta, \mathrm{x})[i]$ is the likelihood that the model predicts x as class i.

The inter-class distance between class i and j can be evaluated by taking the average Euclidean distance between a set of data points by c_m from class i and a set of data points denoted by c_n from class j

$$l_{ij} = \frac{1}{|c_m||c_n|} \sum_{\mathrm{x}_a \in c_m} \sum_{\mathrm{x}_b \in c_n} \|\mathrm{x}_a - \mathrm{x}_b\|_2.$$

The understanding of the decision boundary is fundamental to our DB detection methods.

We evaluate MANDA's performance and present the ROC curves for the proposed AE detection methods in identifying FGSM, BIM, and CW attacks in Figure 3.6. It is evident that Manifold and MANDA outshine Artifact on CICIDS. We also observe that the AE detection performance of DB on the CICIDS dataset does not match up to the other detection methods. This is attributed to the use of constant Gaussian noise with σ^2 variance in DB's current design. The constant noise magnitude is inadequate for gauging the proximity of data points to their respective decision boundaries since the inter-class distance varies among different classes.

To better understand the impact of the varying inter-class distances on the detection performance of the DB method, we present a toy example as follows. Suppose we have

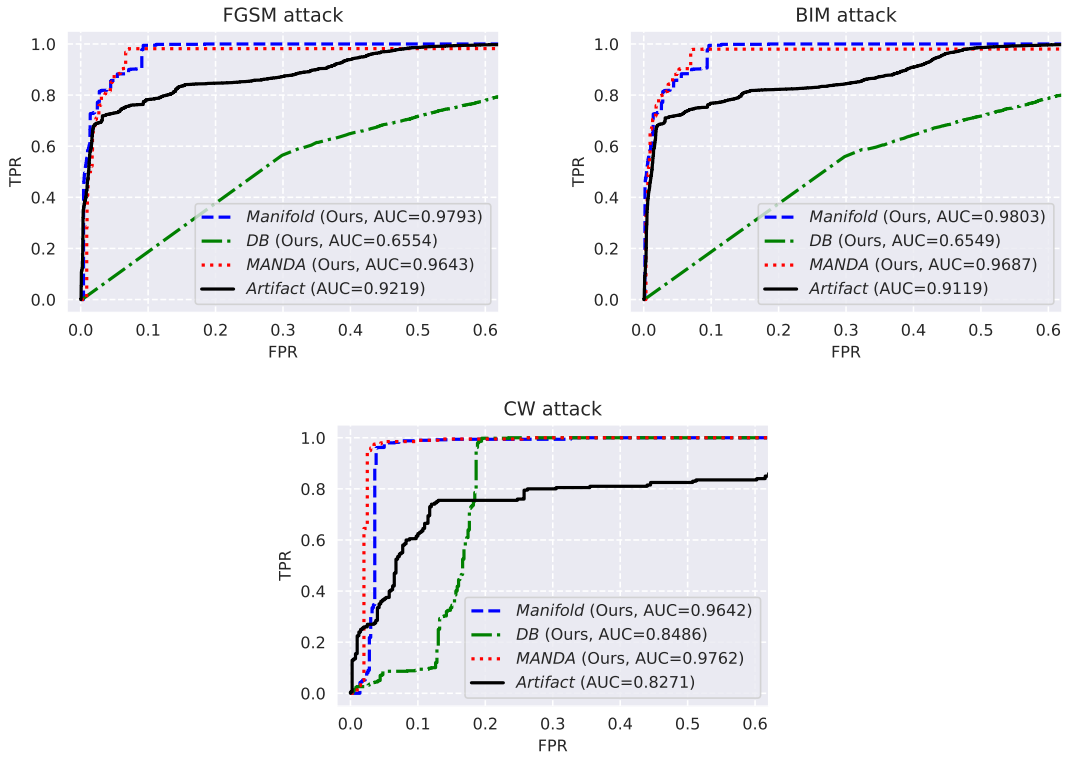


Figure 3.6: ROC curve of AE detection under AE attacks using the CICIDS dataset. (The victim IDS model is multi-class classifier)

two data points x_a and x_b. x_a is from Class 1. And the top two classes produced by a classifier are Class 1 and Class 2. The inter-class distance is $l_{12} = 1$. The L_2 norm distance from x_a to decision boundary D_{12} is 0.8. We would consider x_a is not close to D_{12} since the ratio 0.8/1 is large. Another data point x_b is from class 3. And the top two classes produced by a classifier are Class 3 and Class 4. The inter-class distance between Class 3 and Class 4 is $l_{34} = 100$, while the L_2 norm distance from x_b to decision boundary D_{34} is also 0.8. x_b is considered close to D_{34} since 0.8/100 is small.

From this toy example, the same distance (i.e., 0.8) to different decision boundaries will lead to different conclusions, either close to the decision boundary or far from the decision boundary. It causes a problem when employing DB. In the original design of DB, it utilizes a constant Gaussian noise with $\sigma = 2$ to evaluate the closeness of data points to decision

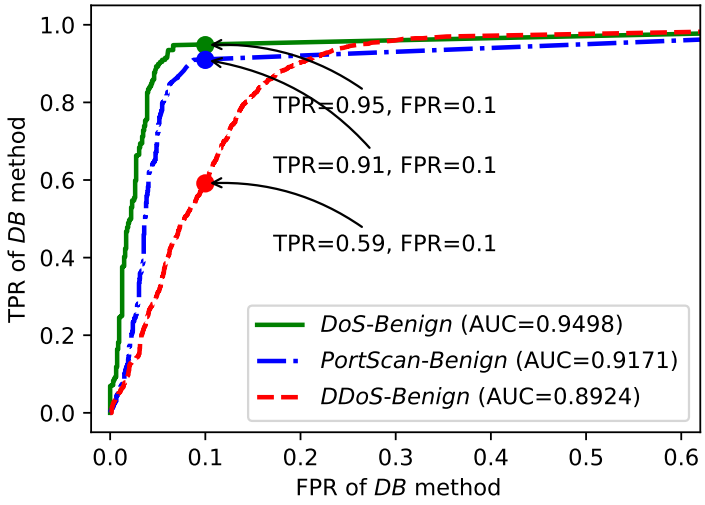


Figure 3.7: The ROC curve of the proposed DB detection method. The evaluated AEs are generated by the FGSM technique.

boundaries. So for the two data points with identical distance to the decision boundary, it is likely we get an incorrect result that that both x_a and x_b are close to decision boundaries. In this way, we made an FP on x_a. On the other hand, we can decrease the σ to 0.2 to avoid FP on x_a. However, we would make an FN on x_b in the new setting.

Based on the analysis of the toy example, we further verify our hypothesis by using empirical results. Specifically, we explored the performance of DB when only sampling two classes of data from the CICIDS dataset, one class is benign and the other class is a specific type of attack. In this setting, the noise level only depends on the inter-distance between the two selected classes. We conduct three samplings and achieved three class pairs: Dos and benign, PortScan and benign, and DDos and benign. We tune the optimal noise level for the three cases and present the performance of DB in Figure 3.7. Compared with Figure 3.6, we observe a significant performance improvement of the DB method.

The improvement of DB performance with noise size tuning implies that it is critical to customize the noise size to the non-identical inter-class distances if we decide to adapt the

Table 3.6: FPR of AE detection (TPR=0.95) using the CICIDS dataset.

Detection Method	FGSM	BIM	CW
Manifold	0.091	0.094	0.038
DB	0.828	0.827	0.186
MANDA	0.067	0.070	0.025
Artifact	0.419	0.439	0.653

DB method to the multi-class situation. This is a potential future work. It is worth noting that the Manifold is not sensitive to the inter-class distance so we can still achieve good detection performance with Manifold.

The detailed performance of MANDA is shown in TABLE 3.6. In this table, we compute the minimum FPR of different detection methods when they achieve 0.95 TPR. We observe that MANDA achieves 0.067 FPR under FGSM attack. To achieve the same TPR, other AE detection methods make higher FPR. For BIM attack and CW attack, MANDA achieves the smallest FPR among all detection methods. We notice that the FPR of DB is larger than all the other AE detection methods, indicating that it is unsuitable to be used alone. And the reason for this low detection rate is related to the constant noise size, which is discussed at the beginning of this section. Though there is space to improve the current DB methods, we are still able to improve the AE detection performance of Manifold by incorporating DB as shown in TABLE 3.6.

In sum, Manifold and MANDA outperform other AE detection methods by achieving both higher AUC scores and higher TPR when using the CICIDS dataset.

3.6.7 AE Detection on the MNIST Dataset

The evaluation of MANDA on the two network traffic datasets—the NSL-KDD dataset and the CICIDS dataset— demonstrates its effectiveness. Additionally, we assert that the proposed techniques can be applied to other domains, as they do not rely on specific

input data formats. To showcase MANDA's adaptability in different domains, we perform experiments using the MNIST image dataset.

It is important to note that a low-dimensional feature vector must be extracted from images, as the high dimensionality of the raw data makes it challenging to learn a useful manifold. To achieve this, we introduce a feature extractor that transforms the original image input (i.e., a pixel vector) into a low-dimensional feature vector. We build a convolutional auto-encoder as the feature extractor. The auto-encoder consists of three convolutional layers with 32, 64, and 64 filters in the encoder and three convolutional layers with 64, 64, and 32 filters in the decoder. The output of the auto-encoder is a reconstructed image. We train the auto-encoder by minimizing the reconstruction error. The trained encoder extracts low-dimensional features from the MNIST dataset images, with the output feature dimension being 64. In comparison to the original input size of 28x28, the size of the extracted high-level features is smaller (i.e., 64x1), which enhances both the efficiency and effectiveness of manifold learning.

Figure 3.8 displays the AE detection performance for four attack types, namely FGSM, BIM, JSMA, and CW attacks. We find that both MANDA and Manifold surpass other baselines by achieving the highest AUC scores. With a fixed FPR at 5%, MANDA attains TPRs of 0.89, 0.94, 0.90, and 0.99 under FGSM, BIM, JSMA, and CW attacks, respectively. In comparison to Artifact, MANDA improves TPR under FGSM, BIM, JSMA, and CW attacks by 0.36, 0.19, 0.10, and 0.31, respectively. These outcomes verify that MANDA is not only effective in network intrusion detection but also in identifying adversarial image manipulation. We plan to investigate more application scenarios in future work.

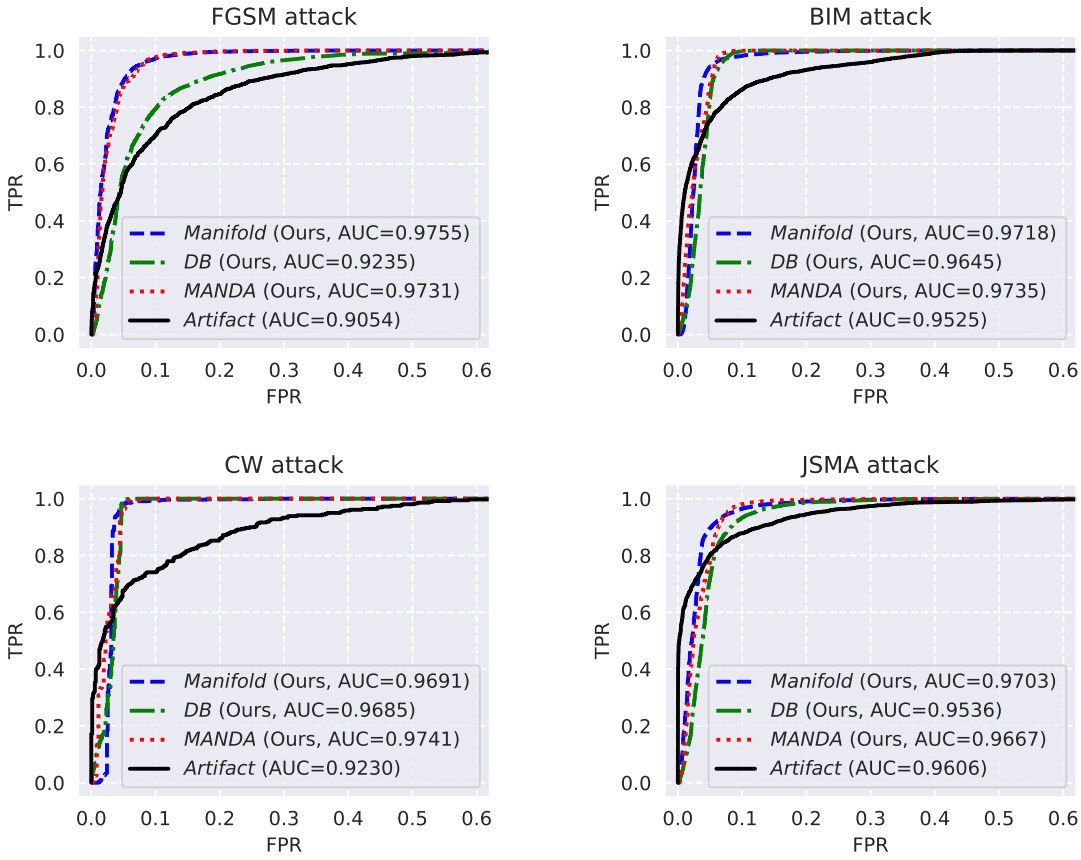
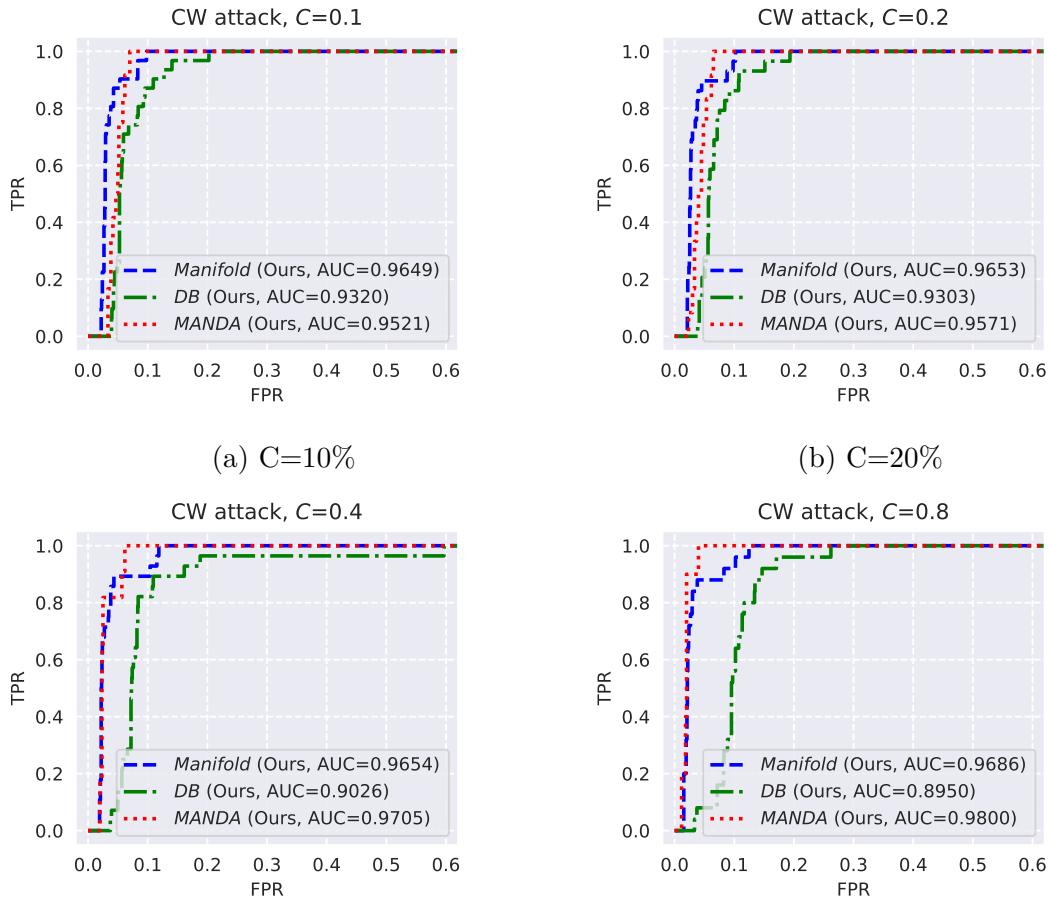


Figure 3.8: ROC curve of AE detection methods under AE attacks using the MNIST dataset.

3.6.8 Performance against Adaptive Attack

We further evaluate the effectiveness of MANDA in a more challenging threat model where an attacker is aware of the defense mechanism. Armed with this knowledge, the attacker can execute adaptive attacks to create adversarial examples to evade detection. We initiate an adaptive attack, and the strategy is to generate AEs that are far from the decision boundary. In theory, the decision-boundary-based detection component (DB) would struggle to detect such adaptive attacks, as it is not able to distinguish an AE far from the decision boundary from true clean data.

We implement the adaptive adversarial attack on top of the CW attack. We enable an



(a) C=10% (b) C=20%

(c) C=40% (d) C=80%

Figure 3.9: ROC curves of AE detection results under adaptive CW attacks using the NSL-KDD dataset.

adjustable confidence level of AEs in this attack. Specifically, we use the difference between confidence in the adversarial class and confidence in the ground truth class to represent the strength of AEs. And we denote this confidence difference as C. We show MANDA's performance against AEs with different Cs. Figure 3.9a 3.9b 3.9c 3.9d show the AE detection results when $C = 10\%, 20\%, 40\%,$ and 80% respectively. MANDA and its two components, i.e., DB and Manifold, are all included in the figures. We can see that the performance of DB degrades with the increase of AE confidence as expected.

In contrast, both the manifold-based detection method and MANDA remain effective in detecting these AEs with the increase of AE confidence. These results imply that the Manifold component is effective against such adaptive attacks. The reason is that the inconsistency between the victim model prediction and the manifold evaluation increases with C. We further reason the success of MANDA in detecting such adaptive attacks as follows. The generated adversarial example should retain its original properties, regardless of its confidence level. As a result, the manifold evaluation will likely predict the adversarial example to belong to its original class. Conversely, the target classifier will categorize it as a different class. This discrepancy between the manifold evaluation and classifier prediction allows the manifold-based detection component to identify the adversarial example successfully.

3.7 Lessons learned from AEs against IDS

An intriguing observation from our experiments is that the generation of adversarial examples in the IDS problem space exhibits certain common patterns, offering insights into evading IDS from an attacker's perspective.

We compare adversarial examples to their corresponding clean traffic flows and identify modification patterns. For each feature, there are two potential modifications: increasing or decreasing the feature's value. Various attack types necessitate distinct modification patterns to evade IDS. As an example, we examine the modification pattern for the DoS attack from the NSL-KDD dataset. We randomly choose 200 instances of DoS traffic flow and apply the adversarial example attack to the selected traffic flows. Out of the 41 features, we permit modification of only six features, as displayed in Figure 3.10. The definition of each feature is available in [27]. Among the 200 generated adversarial examples, 51 successfully evade IDS.

We summarize the likelihood of a feature being increased or decreased to achieve a successful evasion attack in Figure 3.10. Figure 3.10 reveals that an attacker can evade detection by extending the connection duration, reducing the number of connections per second, and so on. These findings offer guidance for designing evasion attacks and also inspire us to come up with more effective intrusion detectors.

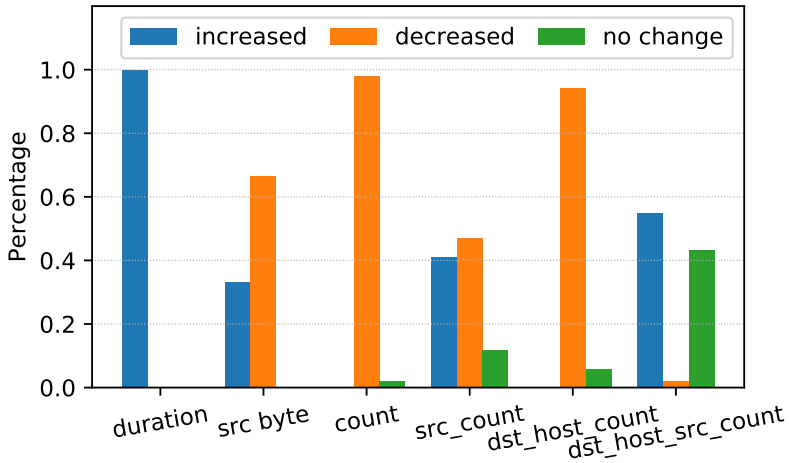


Figure 3.10: The percentage of AEs that a selected feature (e.g., duration, src_byte, etc) value is increased or decreased compared with the original input on which AEs are generated.

3.8 Conclusions

In this chapter, we investigate three recent adversarial example attacks against ML-based IDSs. The results confirm that problem-space adversarial example attacks effectively disrupt IDSs, as they allow malicious events to evade detection with high probability. We identify common characteristics of successful adversarial examples and use them to design an effective and accurate adversarial example detector, MANDA. The MANDA system features a unique design that leverages 1) inconsistencies between manifold evaluation and IDS model inference and 2) evaluates model uncertainty on small perturbations to distinguish adversarial examples from clean network traffic. Our evaluation of MANDA using the

NSL-KDD and CICIDS datasets reveals that it outperforms the state-of-the-art statistical test model (i.e., Artifact) by achieving higher AUC scores and higher true-positive rates with a 5% false-positive rate. MANDA also demonstrates relatively high detection effectiveness when tested using the MNIST dataset, suggesting that the detector may be applicable to other domains, such as computer vision.

Chapter 4

Robust Network Intrusion Detection via Contrastive Learning

(Copyright notice[1])

4.1 Introduction

The last decade has seen an exponential growth of the Internet of Things (IoT) devices. Having achieved the milestone of 12 billion connected devices in 2020, it is estimated that by 2025 there will be more than 30.9 billion IoT devices in the market[2]. IoT starts a new paradigm where billions of smart devices with embedded computational capability and Internet connectivity can automatically work with minimal human intervention. Due to low cost and versatility, IoT devices are being used in almost all sectors: healthcare, smart cities, agriculture, and transportation, to name a few [77, 103, 133].

However, such pervasiveness also increases the risk of data breaches and cyberattacks. In the past decade, we have seen increased attacks involving IoT devices and IoT systems. Many IoT devices have limited on-device resources such as computing power and memory,

[1]This chapter previously appeared as a part of a conference paper published in INFOCOM 2022. ©2022 IEEE. Reprinted, with permission, from Ning Wang, Yimin Chen, Yang Hu, Wenjing Lou, and Y. Thomas Hou, "FeCo: Boosting Intrusion Detection Capability in IoT Networks via Contrastive Learning," IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, pages 1409-1418, 2022 [150].

[2]https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/

which limits the amount and types of security mechanisms that can be implemented in them. Many IoT manufacturers are not security-savvy. In a rush to roll out new products, very often only minimal security features are included, not to mention providing ongoing support or software security updates. The default configuration of an IoT device usually remains in place if no one makes the effort to change it [99]. One notable attack on IoT is Mirai [8] which overwhelmed several high-profile targets with massive distributed denial-of-service (DDoS) attacks in late 2016. More than half a million devices were infected in a few months. Security patching is one possible remedy to security issues. However, many devices lack appropriate facilities for automated security updates, or there may be significant delays until device manufacturers provide them. Considering that IoT devices typically connect to the Internet through a local gateway, a more practical and effective idea to secure IoT devices and systems is to implement an IDS in the local gateway. An IDS continuously monitors incoming and outgoing data streams generated by diverse sources and analyzes them to detect cyber threats.

Ideally, a network IDS device should be placed at a data concentration point in the network for best performance. For instance, most often an IDS device is deployed behind the firewall at the gateway of an edge network. For an IDS in an IoT network, there are two main placement strategies: distributed and centralized. In a distributed placement [38, 102], a local gateway or edge router independently manages its own IDS for the local network. Due to the scarcity of local data, distributed IDS may suffer low accuracy. On the other hand, the rising concern of privacy poses great challenges to a centralized placement [5, 117]. Legal restrictions (e.g., HIPAA[3]) actually prohibit collecting and storing certain types of user data to a central server. In our pursuit to design an efficient, accurate, yet privacy-conscious IDS for IoT network, we resort to the Federated Learning (FL) framework

[3]https://www.hhs.gov/hipaa/index.html

[57, 87, 145]. FL enables local gateways to cooperatively contribute to the training of a global model by providing their local model parameter to a central server. Through an iterative learning process, the global model achieves good generalization by learning knowledge from a large number of IoT devices. However, due to device heterogeneity (in terms of communication protocols and co-existing technologies), the pattern of normal network traffic varies dramatically among different types of IoT devices. Learning a universal model across all different device types may render the IDS useless. To address this problem, we choose the device-type-specific design [99] that builds an IDS model for each type of IoT device, which tends to provide more accurate models.

In general, there exist two main approaches for intrusion detection: anomaly-based Intrusion Detection System (IDS) and signature-based IDS [33, 42, 56, 64]. An anomaly-based IDS learns a model that represents the normal behavior and generates an alarm when the deviation of an incoming instance from the normal behavior surpasses a security threshold. A signature-based IDS detects intrusions by comparing incoming traffic with the saved signatures in a database of known attacks. Due to the nature of their design, signature-based IDSs are not able to detect zero-day attacks. We focus on anomaly-based IDS in this chapter due to its superior capability of detecting novel attacks. With the great success of machine learning in pattern recognition, utilizing machine learning in anomaly-based IDSs is a significant trend in the last two decades. In machine-learning-based IDSs, it is critical to learn a normal profile. While being able to detect novel attacks, the machine-learning-based IDSs also suffer from a high false-positive rate (FPR) compared to the signature-based IDSs. A false positive occurs when an incoming benign traffic instance deviates from the learned profile and is misclassified as malicious. Due to the large variability of normal network traffic, it has been a real challenge for machine-learning-based IDSs to find stable notions of normality for such normal traffic.

Extracting the common properties of benign variations precisely and effectively is a key step in learning a stable normal profile. This chapter proposes a new way to achieve this—employing contrastive learning [141, 142, 161] to transform an original traffic instance into a new representation. Specifically, we build a feed-forward artificial neural network (ANN) that takes an original traffic instance as input and outputs a new representation. Our goal is to learn a new feature space where the benign representations lie in a small cluster while attack representations stay far from the benign cluster so that the differentiation of the two can be made easier. By minimizing the volume of a hyper-sphere that encloses the representations of the normal data, we can train an ANN model that is able to extract the common properties of benign variations more precisely, which leads to improved robustness of the normal profile and significantly reduced FPR. Furthermore, in order to reduce the computation overhead, we build a lightweight ANN with only two hidden layers for resource-constrained IoT devices.

Finally, we build FeCo, a Federated-Contrastive-learning framework, by incorporating FL into the contrastive-learning-based IDS to achieve accurate detection and preserve data privacy simultaneously. In the FeCo design, each local gateway manages a local IDS model and all gateways cooperatively work with a central server to boost local detection performance. To avoid the overfitting of an IDS model to irrelevant features, we propose a two-step feature selection scheme for pre-processing the input data. We remove less significant features and only retain essential information for detection. We extensively evaluate FeCo using a network traffic dataset (i.e., NSL-KDD dataset [138]) to demonstrate the effectiveness of FeCo in detecting intrusions. Our contributions are summarized as follows:

- We propose a novel method for building the "norm" in an anomaly-based IDS by learning new representations for network traffic based on contrastive learning. With the proposed new method, the learned representations of benign inputs lie only in a

small cluster, enabling FeCo to extract a stable template for benign inputs. Extensive evaluation results show that representation learning in FeCo significantly boosts its detection accuracy compared to previous works.

- We propose a two-step feature selection scheme to reduce the risk of overfitting. Our feature selection scheme exploits feature correlation and importance and extracts only the essential information for intrusion detection. Such a scheme also helps reduce computation complexity as a result of smaller input dimensionality, making FeCo more suitable for resource-constrained IoT devices.

- We extensively evaluate FeCo using the NSL-KDD dataset and the BaIoT dataset. By comparing FeCo with 11 baselines, we demonstrate the effectiveness of contrastive-learning-based IDS. On the NSL-KDD dataset, FeCo achieves an 8% accuracy improvement over the state-of-the-art. For zero-day attacks (i.e., attacks unseen by the training dataset), FeCo achieves a recall 8% to 42% higher than other baselines. FeCo is robust to non-IID (Independent and Identically Distributed) data by showing consistent accuracy in different data distributions. We also investigate FeCo on the convergence performance, scalability, and overhead to show that it is suitable for IoT systems.

- We implement FeCo on a Raspberry Pi device to demonstrate its applicability in resource-constrained IoT devices. We perform model optimization using the model weights quantization technique in order to save memory and storage space. The optimized lightweight model achieves comparable accuracy with a standard FeCo model.

4.2 Related Work

In this section, we discuss the placement methods of IDS in IoT systems and the design of machine learning-based IDS.

Table 4.1: Review of Machine-Learning-based Intrusion Detection System for IoT system.

paper	ML Mechanism	Detection Type	Dataset	Security Threats
Kitsune [91], 2018	an ensemble of Autoencoder	Type 1	self generated data on 9 IoT devices	Recon., Man-in-the-Middle, DoS, and Mirai Botnet.
EMFC [115], 2018	ELM-based semi supervised Fuzzy C-Means	others	NSL-KDD [138]	DoS, Probing, U2R, R2L
DIoT [99], 2019	Gated Recurrent Units	Type 1	self-generated data on 33 IoT devices	Mirai Botnet
CVAE [82], 2017, [169], 2019	conditional varational autoencoder	Type 1	NSL-KDD [138], UNSW-NB15 [93]	Exploits, Fuzzers, Recon., Analysis, Backdoor, Shellcode, Worms, etc.
TDTC [104], 2019	Naive Bayes and K Nearest Neighbor	Type 2, 3	NSL-KDD [138]	DoS, Probing, U2R, R2L
EIDS [94], 2019	Ensemble of Decision tree, Naive Bayes, and ANN	Type 1	UNSW-NB15 [93], NIMS[4]	Analysis, Backdoor, DoS, Exploit, Fuzzers, Generic, Recon., etc
AE-DNN[119], 2019	Autoencoder plus deep neural network (DNN)	Type 2	AWID dataset [60]	Flooding, impersonation, and injection
RF-SD-IoT[175], 2020	Random Forest	Type 2	self-generated data using Mininet-WiFi emulator [40]	SYN, ping flood, UDP port scan and UDP flood
IoTDefender [35], 2020	Convolutional neural network	Type 2	NSL-KDD [138], CICIDS2017 [124], Datasets [58][91]	PortScan, DDoS, ARP Spoofing, ARP MitM, Fuzzing, Mirai, etc.
SDRK [116], 2020	deep neural network with random sampling-K-means	Type 2	NSL-KDD [138]	DoS, Probing, U2R, R2L
Passban [33], 2020	Isolation Forest, Local outlier factor	Type 3	self generated data on 6 IoT devices	Port Scan, HTTP&SSH Brute Force, SYN flood
SENTINEL [26], 2021	Naive Bayes, Decision Tree, Logistic Regression, and Random Forest Classifier	Type 2	Self-generated data from 9 IoT devices	Network scan, Exfiltration, C&C keep alive, Black/grey hole
Ours	Contrastive-learning-based anomaly detection	Type 4	BaIoT dataset [89] and NSL-KDD [138]	DoS, Probing, U2R, R2L, Marai, and BashLite

IDS placement is a crucial design consideration in IoT systems compared to computer networks. There are three primary IDS placement strategies: distributed IDS placement

[38, 102], centralized IDS placement [117], and hybrid IDS placement [5]. Recently, with the advances of Federated Learning (FL) [87], FL-based IDSs [81, 92, 99] have gained increasing popularity. FL enables distributed placement to achieve better generalization performance by leveraging diverse training data sets from numerous IoT devices. Moreover, FL offers enhanced privacy preservation in IDSs compared to centralized placement. Nguyen et al. [99] were the first to employ FL in anomaly-based IDSs. In their design, a local gateway uses its local data to train the model and submits the model parameters to the cloud server. The cloud server then aggregates these local models into a global model. Since then, various studies [81, 92] have explored the use of the FL framework to enable decentralized edge devices to learn an anomaly detection model using only on-device data at each edge device.

In the field of anomaly-based IDSs, machine-learning techniques have been extensively explored in the literature. The most common strategy for detecting attacks involves monitoring network activity and reporting potential abnormal events—deviations from profiles of normality previously learned from benign traffic [30, 91, 99, 131]. A key component of a general anomaly-based IDS is a model representing legitimate traffic. In this section, we review anomaly-based IDSs, focusing on those in the domain of IoT systems.

DeepLog [30] employs long short-term memory to model system logs as natural language sequences and learns log patterns from normal execution. In the detection phase, it automatically detects anomalies when the ongoing log deviates from the learned pattern. Mirsky et al. [91] introduced an ensemble of multiple autoencoders to distinguish between normal and abnormal traffic patterns. The autoencoder reconstructs an input and computes the reconstruction error in terms of root mean squared errors (RMSE). An alarm is generated when the RMSE value exceeds a threshold. The study in [167] utilized a non-parametric density estimation method to learn and predict legitimate access patterns. [99] modeled network packets as symbols in a language, enabling the use of Gated Recurrent Units (GRU)

for anomaly detection. Specifically, the GRU model estimates the probability of the next symbol, and an alarm is raised if the occurrence probabilities of a sufficient number of packets fall below a detection threshold.

Besides the mechanisms discussed above, some other IDSs directly learn a binary or multi-class classifier for detecting intrusions. Yan et al. [168] applied SVM to detect botnets using the high-level features extracted from the command and control channel. TDTC [105] utilizes two tires of classification to improve intrusion detection performance. TDTC utilizes the Naive Bayes classifier to identify anomalous behavior in the first tire. In the second tire, a K-Nearest Neighbor model is used to further detect anomalies from those instances that are classified as normal in the first tire in order to reduce false negatives. Wang et al. [149] employed multi-layer perceptron (MLP) and manifold learning to detect evasive intrusions. ESFCM [115] integrates a Fuzzy C-Means with the Extreme Learning Machine (ELM) classifier to achieve efficient attack detection in IoT systems. Wang et al. [147] proposed a graphic model that stores known traffic patterns as a relational graph between patterns and their labels to detect DDoS attacks in the cloud computing scenario.

Most anomaly-based IDSs suffer from a high false-positive rate (FPR) compared to signature-based IDSs. To address this issue, we propose a contrastive-learning-based anomaly detection method. Like traditional anomaly-based IDSs, this method aims to learn the pattern of benign traffic. The crucial difference lies in maximizing the distance between normal patterns and intrusion traffic while minimizing the distance among normal instances during the pattern learning phase. By identifying network traffic that deviates from the pattern as an intrusion, our method can reduce the FPR while maintaining the capability to detect novel attacks.

We also summarize representative papers on machine-learning-based (ML-based) IDS for IoT systems in TABLE 4.1. We present the ML mechanisms, datasets for evaluation, and

security threats that IDS targets. We categorize the detection mechanisms into four different types:

- Type 1 mechanisms learn the pattern of the benign traffic and reject network traffic that deviates from the learned pattern. Only benign traffic is used for learning.

- Type 2 mechanisms learn a binary classifier between the benign traffic and intrusion traffic. Both benign data and intrusion data are used for learning.

- Type 3 mechanisms measure the distance (or local density) of a data point to the rest of the data (or the neighbors) to detect potential outliers.

- Type 4 mechanisms learn the pattern of the benign traffic. During pattern learning, it tries to extract the key features that make the normal patterns as distinguishable from the intrusion traffic data as possible. This type of detection then detects intrusions by measuring network traffic's deviation from the normal pattern. Both benign data and intrusion data are used for learning.

We have summarized both traditional ML methods (e.g., naive Bayes, decision tree, and random forest) and deep neural networks (e.g., gated recurrent unit, autoencoder, and convolutional neural network) for IDS. As shown in the table, the evaluation datasets include both self-generated data using IoT devices or emulation platforms and well-published datasets. We hope the list of published datasets will encourage more research on IDS. Due to space limitations, not all the reviewed related papers are shown in the table. Please refer to survey papers [2, 22] to get connected to ML-based IDSs.

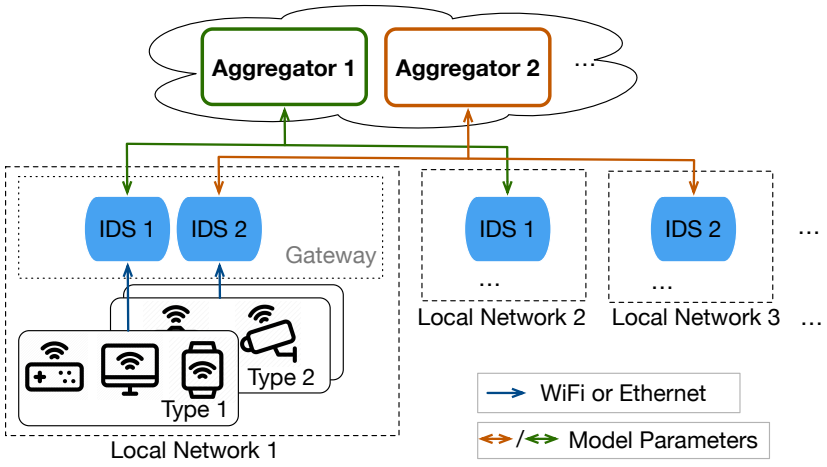


Figure 4.1: The system design of our device-type-specific IDS–FeCo. Local gateways maintain an IDS model for each type of IoT device. The detail of the IDS shown as a blue icon in this figure is depicted in Figure 4.3.

4.3 System Model and Threat Model

We assume an IoT network consisting of multiple types of IoT devices (e.g., Web cameras, voice assistants) connected via a local gateway. As illustrated in Figure 4.1, we propose the device-type-specific IDS design. For each type of device, several gateways collaboratively learn an IDS model through an FL framework. An FL system comprises two main components: local gateways (i.e., clients) and a model aggregator, which we describe below.

A local gateway $\mathcal{G}$ serves as a client within a federated learning (FL) system, managing intrusion detection systems (IDSs) to identify compromised IoT devices on a local network. If multiple device types are present in the network, $\mathcal{G}$ may handle several IDSs. Depending on its computational capability, $\mathcal{G}$ can opt to participate in the learning process, resulting in two operational modes: learning mode and consumer mode. In learning mode, $\mathcal{G}$ functions as an FL system client, whereas in consumer mode, $\mathcal{G}$ only requests an IDS model from a central server. Throughout this chapter, $\mathcal{G}$'s default mode is assumed to be the learning mode. $\mathcal{G}$

is also in charge of identifying device types when a new IoT device connects to the local network. Device-type-identification techniques have been well-researched in [85, 113, 171]. It is assumed that $\mathcal{G}$ can access the network traffic of IoT devices within the local network, consistent with previous research [91, 99].

A model aggregator $\mathcal{A}$ is tasked with aggregating parameter updates for an IDS model from multiple $\mathcal{G}s$ and distributing the updated IDS model back to the $\mathcal{G}s$. Typically, an $\mathcal{A}$ is operated by a major service provider such as Amazon, Google, or Microsoft.

The learning process between an aggregator $\mathcal{A}$ and $\mathcal{G}s$ can be explained as follows. One single aggregator is responsible for training an IDS model for one type of IoT device. In the beginning, $\mathcal{A}$ randomly initializes a global model Θ. Then an FL iteration starts with the selection of participants at $\mathcal{A}$. $\mathcal{A}$ selects a subset of local Gateways $\mathcal{G}s$ that maintain an IDS model for the targeted type of IoT device. It distributes the current global model Θ to the selected $\mathcal{G}s$ for initializing their local IDS models. Each selected local gateway $\mathcal{G}$ continues training its IDS model with data generated by local IoT devices. After several local training epochs, selected local gateways $\mathcal{G}s$ upload their model weights θ_i to $\mathcal{A}$. $\mathcal{A}$ aggregates received model weights following an aggregation rule such as FedSGD and FedAvg [87] and concludes one FL iteration. After multiple FL iterations, $\mathcal{A}$ can achieve a well-generalized model that can be used for intrusion detection at local $\mathcal{G}s$.

Threat Model: IoT devices are often susceptible to various network-based intrusions, including unauthorized access, address spoofing, false data injection, and network connectivity disruption. When compromised, these devices can be used to target other network components or services with attacks. In this chapter, our objective is to identify malicious network traffic, regardless of its purpose.

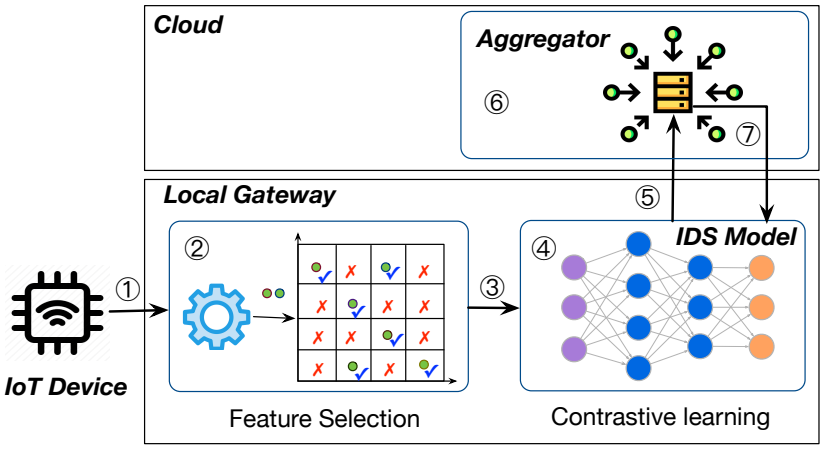


Figure 4.2: The workflow of one learning iteration of FeCo. The core of FeCo (i.e., Step 4) is further illustrated in Figure 4.3.

4.4 FeCo Design

4.4.1 Workflow of IDS

The workflow of the proposed FeCo is shown in Figure 4.2 which provides an overall view of its operations.

A local gateway $\mathcal{G}$ will request initial IDS model parameters according to the types of IoT devices and initializes its IDS model with the received parameters when joining the learning system. During a learning iteration, $\mathcal{G}$ (①) collects the raw network traffic data and (②) extracts significant features from the raw data. Then (③) the extracted features will be fed into the IDS model. $\mathcal{G}$ further (④) trains the local IDS model using the contrastive learning algorithm and the preprocessed data and (⑤) uploads the model parameter update to model aggregator $\mathcal{A}$. $\mathcal{A}$ (⑥) aggregates received model parameter updates from multiple local gateways and use them to update the global model. (⑦) The updated global model will be distributed to local gateways for another learning iteration. Steps ① - ⑦ repeat until the global model converges. We can see from Figure 4.2 that there are three

important components in FeCo including Feature Selection, Contrastive-learning-based IDS, and Federated Aggregation. We will introduce the three components in detail respectively in the rest of this section. The symbols used in the description and their meanings are listed in TABLE 4.2.

Table 4.2: Symbol definition.

Symbol	Definition
x_i	feature vector of a traffic flow, and $x_i \in \mathbb{R}^d$
y_i	label, $y_i \in \{0, 1\}$ where 0 for benign and 1 for intrusion
z_i	output feature representation of x_i, and $z_i \in \mathbb{R}^o$
θ	model parameter
f_θ	mapping function from x to z, $f_\theta : \mathbb{R}^d \to \mathbb{R}^o$
$\mathrm{v_i}$, $\mathrm{u_i}$	output feature representation of x_i if $y_i = 0$ and $y_i = 1$, respectively
N, M	benign records number and intrusion records number
$\bar{z}$	normal template, i.e., average of normalized zs
$S(x_i)$	the cosine similarity of x_i with template $\bar{z}$
p	percentage value, $0 \leq p \leq 100$
ρ	threshold for determining normal or intrusion

4.4.2 Feature Selection

In FeCo, We choose to perform feature selection to remove features that are explicitly demonstrated as irrelevant to intrusion detection. We propose a two-step feature selection scheme to select essential features for IDS. By removing redundant features, we simplify the model architecture and reduce the training time as well.

Redundant Feature Removal

Not all the statistical attributes contain unique information about an individual traffic flow. For example, a feature with zero variance exhibits a constant value in the dataset, and thus it contains no useful information for intrusion detection. Therefore, we first remove the

zero-variance features. Furthermore, a feature vector may contain redundant information if there exist multiple highly correlated features. If we use all the highly correlated features for model training, it is likely to cause overfitting because there exists an implicit emphasis on these correlated features. To reduce the risk of overfitting, we use the Spearman rank correlation coefficient [135] to quantify the correlations among the numerical features and then keep only one feature for each set of highly correlated features.

Feature Importance Ranking

In FeCo, we propose feature importance ranking to rank the importance of features. We employ different methods to measure the importance of categorical features (e.g., the protocol type and service type) and numerical features (e.g., source bytes) since they contribute to the final prediction differently. Specifically, we utilize mutual information (MI) between a feature and the output label to measure the importance of a categorical feature. A zero MI implies that the output label is independent of the target feature. The MI ranges from zero to one, and a higher MI means a higher significance. For numerical features, we employ analysis of variance (ANOVA) to evaluate feature importance. In ANOVA, the observed values of a feature are divided into two groups that are attributable to different values of the label. ANOVA measures whether or not the target feature is statistically different in these two groups. In practice, the ANOVA score is the ratio of the variance between the two groups to the variance within the same group. A larger ANOVA score means higher importance.

The analysis of the proposed feature selection methods and the impact of such data preprocessing on the performance of FeCo are evaluated and reported in Section 4.5.3.

4.4.3 Contrastive-learning-based IDS

Contrastive-learning-based IDS is the building block of FeCo. It is deployed for the training process at each $\mathcal{G}$. Contrastive learning is first proposed to improve recognition accuracy in the computer vision field. Our goal of deploying contrastive learning is to train a model that produces similar representations for all normal traffic instances and make the intrusion representations far from normal representations. We build a binary IDS model and formally introduce Contrastive-learning-based IDS below.

We assume each record in the training dataset consists of two fields: input feature vector $x_i \in \mathbb{R}^d$ and output label $y_i \in \{0, 1\}$ where 0 indicates a normal traffic flow and 1 indicates an intrusion. The goal of the contrastive learning algorithm is to learn a new representation (rather than a label) for each input instance. Specifically, contrastive learning trains an ANN model that takes $x_i \in \mathbb{R}^d$ as input and outputs a new representation $z_i \in \mathbb{R}^o$. The ANN model can be represented by a function $f_\theta : \mathbb{R}^d \to \mathbb{R}^o$ where θ denotes the model parameters. The ANN model of FeCo consists of four layers: the input layer, two hidden layers, and finally the output layer. The corresponding size of each layer is d, 128, 256, and o, respectively. The default value for o is 128.

For convenience, we use $\mathrm{v}_i = f_\theta(x_i)$ to denote the output of a benign input x_i and $\mathrm{u}_i = f_\theta(x_i)$ to denote the output of an intrusion input x_i. We assume the dataset contains N normal traffic flows and M intrusion traffic flows. For each pair of normal inputs, we can obtain representation v_i and representation v_j. Our goal is to maximize the similarity between v_i and v_j and minimize the similarity between v_i and $\mathrm{u}_m|_{m\in[M]}$ (we define $[M] := \{1, 2, ..., M\}$). The likelihood that v_i is close to v_j is represented by

$$l_{ij} = \frac{\exp(\mathrm{v}_i^T \mathrm{v}_j/\tau)}{\exp(\mathrm{v}_i^T \mathrm{v}_j/\tau) + \sum_{m=1}^{M} \exp(\mathrm{v}_i^T \mathrm{u}_m/\tau)}, \tag{4.1}$$

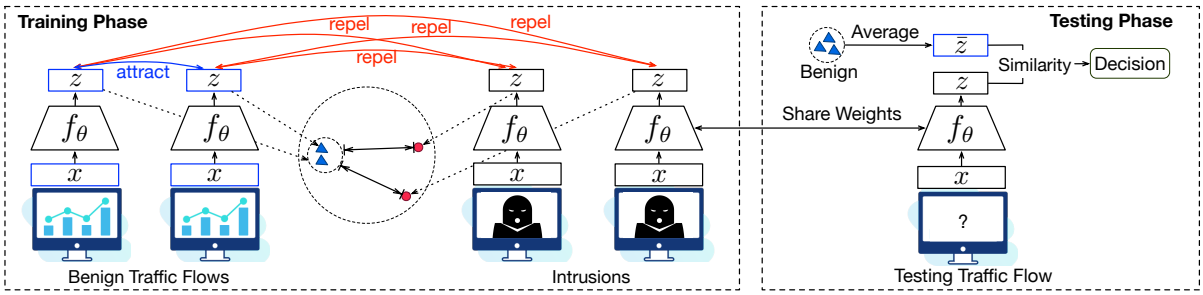


Figure 4.3: The training process and testing process of the contrastive-learning-based IDS.

where $0 < \tau < 1$ denotes the temperature coefficient, and such τ makes the likelihood distribution more peaked and narrow compared to the version without temperature coefficient (i.e., $\tau = 1$). It helps the algorithm to capture the similarity more efficiently, and it is a hyper-parameter to be tuned. We then define a loss function $\mathcal{L}_{ij}$ as the negative logarithm of the likelihood function.

$$\mathcal{L}_{ij} = -log(l_{ij}) \tag{4.2}$$

Similarly, we can obtain $\mathcal{L}_{ji}$. The overall loss function of the pair v_i and v_j is the summation of $\mathcal{L}_{ij}$ and $\mathcal{L}_{ji}$. We then sum up the loss functions for all pairs of normal vectors and obtain a loss function $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathcal{L}_{ij} + \mathcal{L}_{ji}. \tag{4.3}$$

We use a stochastic gradient descent (SGD) optimizer to minimize the loss function $\mathcal{L}$. By minimizing the loss function, we can achieve the goal of maximizing the similarity among normal representations and minimizing the similarity between normal representations and intrusion representations.

We intuitively interpret the learning process in Figure 4.3 as well. We can see that the normal zs (i.e., $\mathrm{v_i}$) attracts each other while they repel intrusion zs (i.e., $\mathrm{u_j}$) in the training

process. After several iterations, we can learn a model f_θ that outputs a new representation for each input instance. In the new feature space, the representations of benign traffic flows are expected to fall into a compact cluster, while those of intrusion traffic flows are far from such a cluster. In the testing phase, we collect the representations of benign traffic flows and use the average value of the normalized representations as the normal template $\bar{z}$:

$$\bar{z} = \frac{1}{\sum_i \mathbb{1}(y_i = 0)} \sum_i (\mathbb{1}(y_i = 0) \frac{f_\theta(x_i)}{\|f_\theta(x_i)\|_2}), \tag{4.4}$$

where $\mathbb{1}(.)$ is the indicator function, and $\mathbb{1}(y_i = 0)$ equals 1 if $y_i = 0$, $\|.\|_2$ denotes the L^2-norm function and $\frac{f_\theta(x_i)}{\|f_\theta(x_i)\|_2}$ represents a normalized representation. After obtaining the normal template $\bar{z}$, we utilize the cosine similarity estimator to measure the similarity $S(x_j^{test})$ between an upcoming traffic flow x_j^{test} and the normal template:

$$S(x_j^{test}) = \frac{\bar{z}^T f_\theta(x_j^{test})}{\|\bar{z}\| \times \|f_\theta(x_j^{test})\|}. \tag{4.5}$$

The similarity score $S(x_j^{test})$ ranges from 0 to 1. We need a threshold score $0 \leq \rho \leq 1$ for determining whether x_j^{test} is an anomaly or not. We obtain the threshold ρ by calculating the statistics of the scores of benign training data. Particularly, we first sort the scores of benign data in ascending order and obtain the sorted sequence S = $[S_1, S_2, ..., S_N]$. Then we select the p-th percentile of S as the threshold ρ. In FeCo, we first calculated the index r of the score to select:

$$r = \left\lfloor \frac{p}{100} * N \right\rfloor, \tag{4.6}$$

where $\lfloor . \rfloor$ denotes the round down function, and $0 \leq p \leq 100$ represents a percentage number. We obtain $\rho = S_r$ which is the r-th entry of S. We can manually select a value for p. We should select a small value for p (e.g., $p = 5$) as a larger p leads to a higher FPR. The final

decision $\hat{y}_j$ is made by

$$\hat{y}_j = \mathbb{1}(S(x_j^{test}) < \rho). \tag{4.7}$$

An input instance is predicted as an intrusion if its similarity score is smaller than threshold ρ.

4.4.4 Federated Aggregation

We build FeCo by incorporating the contrastive-learning-based IDS into the federated learning framework. In that case, each client participates in the FL process by providing its model parameter update. We utilize the FedAVG [87] algorithm to aggregate the updates from multiple clients. In time step t, the model aggregator $\mathcal{A}$ computes the global model parameter Θ_t by:

$$\Theta_t = \Theta_{t-1} + \sum_i c_i * (\theta_i - \Theta_{t-1}), \tag{4.8}$$

where θ_i is the local model parameters at client i and c_i is a weight coefficient. Particularly, we define c_i as the ratio of the size of the local training dataset at client i to the number of total training samples at all selected clients.

4.5 Experimental Results

4.5.1 Datasets and Experiment Settings

We implement FeCo in the PyTorch platform [109]. We ran all the experiments on a server equipped with an Intel Core i7-8700K CPU 3.70GHz×12, a GeForce RTX 2080 Ti GPU, and Ubuntu 18.04.3 LTS. We experiment with FeCo using two network traffic datasets including the NSL-KDD dataset and BaIoT dataset. The NSL-KDD dataset is a network traffic dataset

that is widely used for IoT scenarios [22, 105, 115]. We evaluate FoCo on the NSL-KDD dataset in order to provide comparisons with other IDSs [22, 105, 115]. The BaIoT dataset is dedicated to IoT devices.

The NSL-KDD dataset [138] includes benign traffic and four categories of intrusions, i.e., DoS, Probing, Remote-to-Local (R2L), and User-to-Root (U2R). Each category of intrusion contains several sub-classes as shown in TABLE 4.3. The whole dataset includes a training set and a testing set, and the training set contains 125,973 records while the test set 22,544 records. Note that some intrusion sub-classes exist only in the testing set, i.e., they are unseen in the training set (e.g., mscan, sqlattack), which makes it possible to evaluate FeCo against zero-day attacks. Each record consists of 41 attributes extracted from a traffic flow and a label indicating its category (i.e., Normal, DoS, Probing, R2L, or U2R). In practice, it is easy to extract attributes from network packets by using existing packet analyzers (e.g., WireShark[4]).

Table 4.3: Classes of intrusions and sub-classes in the NSL-KDD dataset.

Attacks	DoS (10)	Probe (6)	R2L (16)	U2R (7)
In both Training & Testing Set	back, land, Neptune, pod, smurf, teardrop	ipsweep, nmap, portsweep, satan	ftp_write, guesspasswd, imap, multihop, phf, warezmaster	bufferoverflow, loadmodule, perl, rootkit
Only in Testing Set	apache2, mailbomb, processtable, udpstorm	mscan, saint	httptunnel, named, worm sendmail, snmpgetattack, snmpguess, xlock, xsnoop,	ps, sqlattack, xterm
Only in Training Set			spy, warezclient	

The BaIoT dataset [89] contains the traffic data gathered from 9 commercial IoT devices authentically infected by Mirai and BASHLITE. Benign traffic collected at the normal running phase is also included. Both Mirai and BASHLITE is composed of five sub-class attacks: 'scan', 'ack', 'syn', 'udp' and 'udpplain' for Mirai; and 'scan', 'junk', 'udp', 'tcp', and 'combo' for BASHLITE. The whole dataset contains 6,506,674 malicious records and 556,932 benign records, and each record contains 115 attributes. The dataset is available

[4]https://www.wireshark.org/

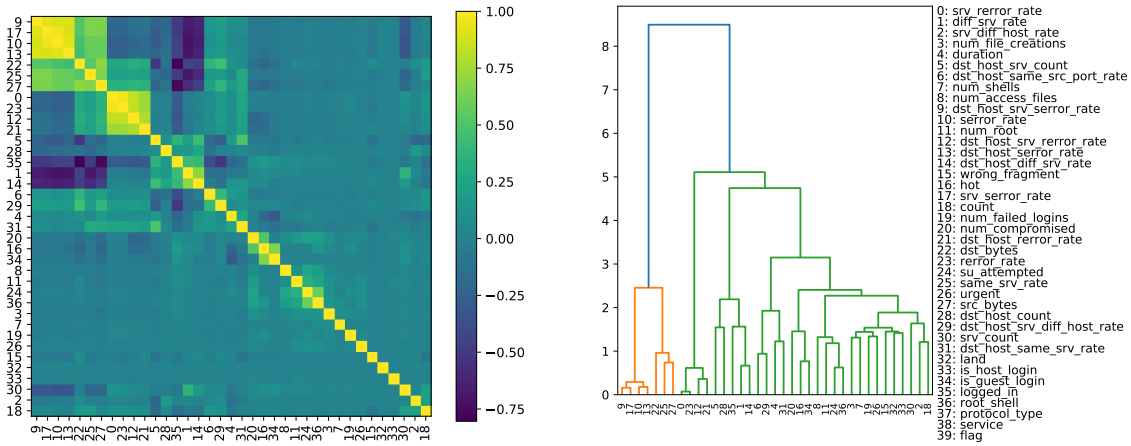
from https://archive.ics.uci.edu/ml/machine-learning-databases/00442/

Some papers [3, 49] reported detection accuracy as high as 99% on the NSL-KDD dataset. However, these works introduce a new splitting on the dataset. They either combine the training set and testing set into one then randomly split it into two sets for training and testing, or directly split the training set into two sets. Such arrangements can only demonstrate the effectiveness of IDS models in detecting known intrusions but not unseen intrusions. We train our IDS model with the training set and evaluate it using the testing set to explicitly show its performance in detecting unseen intrusions.

Other default settings of FeCo are shown as follows. In order to better simulate the distributed characteristics of a real FL system, we choose 50 clients which is relatively larger than 10 and 15 used in [81, 99]. We generate data for clients by splitting the whole dataset. Therefore, the number of clients and the size of local data would be inversely proportional to each other. By using different splitting strategies, we obtain both IID data and non-IID data (See Sec. 4.5.5 for detail). For data preprocessing, we first use our feature selection scheme to remove 10 attributes from the 41 attributes. Then we use the one-hot encoding method to map the remaining 31 attributes into 112 input features. The size of the input layer and the output layer of the ANN model in FeCo are $d = 112$ and $o = 128$, respectively. The machine learning baselines used for comparison are imported from scikit-learn [111]. In FeCo, each client uses an SGD optimizer with a learning rate of 0.001 for local training. The clients perform four epochs of local training before sending its model parameters to $\mathcal{A}$ in each FL round. We set the number of total FL rounds as 15.

4.5.2 Evaluation Metrics

For a binary detection problem, there are four important terms: True Positive (TP) means correctly detected as an intrusion; False Positive (FP) means incorrectly detected as an intrusion; True Negative (TN) means correctly detected as benign; False Negative (FN) means incorrectly detected as benign. We use N_* to represent the number of $* \in \{\text{TP, TN, FP, FN}\}$. We compute five evaluation metrics: accuracy $A = \frac{N_{TP}+N_{TN}}{N_{TP}+N_{FP}+N_{TN}+N_{FN}}$, recall $R = \frac{N_{TP}}{N_{TP}+N_{FN}}$, precision $P = \frac{N_{TP}}{N_{TP}+N_{FP}}$, F1 score $F = \frac{2\times P\times R}{P+R}$, and False Positive Rate (FPR) $fpr = \frac{N_{FP}}{N_{FP}+N_{TN}}$. We also provide the receiver operating characteristics (ROC) curve by plotting R against FPR at various threshold settings. The AUC score is defined as the area under the ROC curve.



(a) The heat map of the Spearman correlations. (b) The dendrogram of the correlation.

Figure 4.4: Feature correlations.

4.5.3 Feature Selection

Here we show the process of our two-step feature selection scheme using NSL-KDD as an example.

We first remove the zero-variance feature (i.e., 'num_outbound_cmds') from the dataset. Then we evaluate the feature correlation and show the heat map of the correlation matrix in Figure 4.4a. We further perform hierarchy clustering on the computed correlations among features, shown in Figure 4.4b. Focusing on the height at which any two objects are joined together, we can see the height of the link that joins 'feature 0' and 'feature 23' is the smallest, indicating that the two features are the most correlated. We can further obtain the second most correlated feature pair and the third most correlated feature pair. We randomly remove one feature from the three feature pairs as removing these features does not degrade the detection performance.

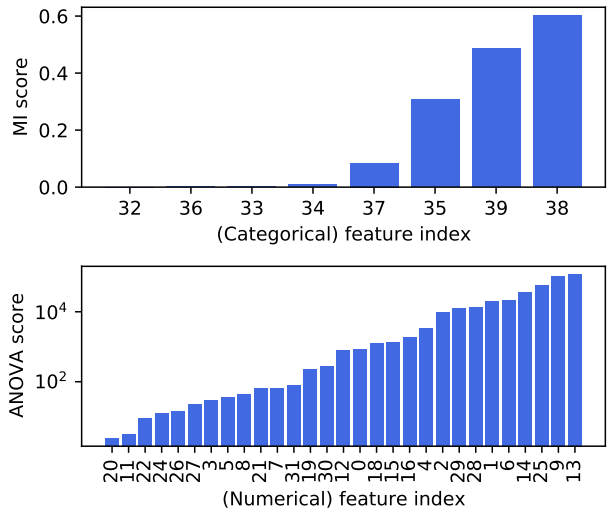


Figure 4.5: The ranking of MI scores and ANOVA scores.

Second, we evaluate importance scores for all features. We show the ranking of MI scores and ANOVA scores in Figure 4.5. The MI score of Feature 32, 33, and 36 (i.e., 'land', 'root_shell', 'is_host_login') is zero, implying that they are irrelevant to intrusion detection. Therefore, we remove the three features. Note that the ANOVA scores are shown on a logarithmic scale as the scores vary dramatically among different features. Unlike MI scores, ANOVA scores have no zero entries. We start by removing the features with the lowest ANOVA score. We

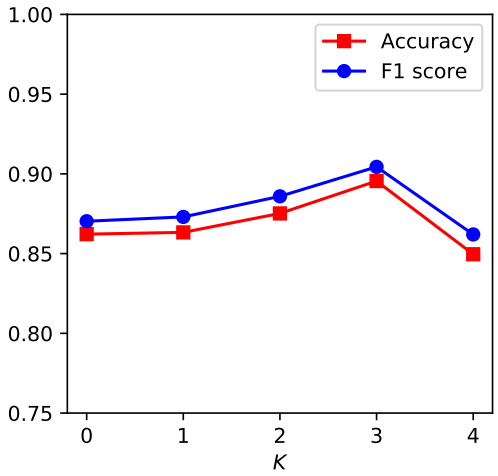


Figure 4.6: vary removed feature number K

vary the number K of features to remove. Intuitively, useful information may be discarded if K is too large, while redundant information may result in overfitting if K is too small. We tune $K \in \{0, 1, 2, 3, 4\}$ to show the impact of K on detection accuracy. Figure 4.6 shows the performance of FeCo with different K. We can see that both accuracy and the F1 score increase with $K \leq 3$, implying that removing features with low significance could boost the detection performance. We select $K = 3$ as the accuracy peaks at $K = 3$.

4.5.4 Performance on NSL-KDD in Centralized Setting

We first investigate the performance of FeCo under the centralized setting to focus on evaluating our contrastive-learning-based IDS. We evaluate the impacts of the global aggregation of FL in the next part. We compare the performance of FeCo to both state-of-the-art IDSs [104, 105, 115] and some other widely-used machine learning baselines including support vector machine (SVM), variational autoencoder (VAE), isolation forest (IsoForest), multilayer perceptron (MLP), logistic regression (LGR), Bernoulli naive Bayes (BNB), K-nearest neighbors (KNN), and decision tree classifier (DTC).

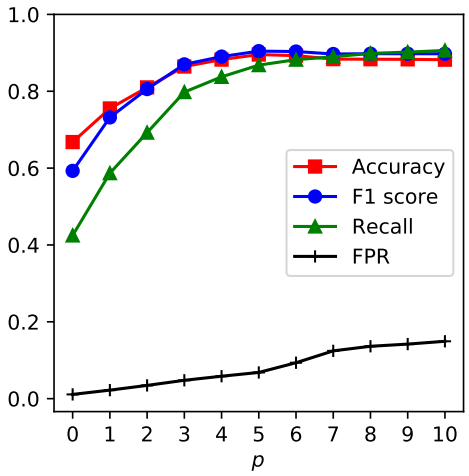


Figure 4.7: FeCo's performance by varying threshold p

As shown in Sec. 4.4.3, the threshold for intrusion detection is computed by manually selecting the quantile number p. Figure 4.7 shows the performance of FeCo with different p values. We can see that the recall increases with p and the FPR also increases as expected. The accuracy and F1 score first increase dramatically with the increase of p then decrease slowly. Given such results, it is intuitive that one can select the value for p based on the desired FPR. We should select a small p if we desire a low FPR. Ideally, FPR should be very close to the value of $p/100$ if the learned model generalized well on the testing set. In practice, we set $p = 5$, and get $FPR = 6.8\%$ in the testing set. This discrepancy may be due to novel attacks in the testing set. However, we believe that the small gap between $FPR = 6.8\%$ and $5/100$ confirms that FPR would approximate the value of $p/100$.

We show the detection performance of FeCo and other baselines in TABLE 4.4. We set $p = 5$ to obtain these results. FeCo achieves detection accuracy as high as 89.55%. From TABLE 4.4 we can see that FeCo outperforms other methods by achieving both the highest recall and the highest accuracy. Note that some entries of TABLE 4.4 are missing because they were not provided in the references. To demonstrate FeCo's capability in detecting

Table 4.4: Performance (%) of FeCo in Centralized Setting.

Method	Accuracy	Recall	Precision	F1 score	FPR
FeCo (ours)	89.55	86.80	94.39	90.44	6.82
ESFCM [115]	80.69	80.72	80.85	80.45	-
Two-Tier [104]	-	82.00	-	-	5.43
TDTC [105]	-	84.86	-	-	4.86
VAE	81.77	73.71	92.78	82.15	7.58
IsoForest	79.54	69.35	92.90	79.42	7.00
MLP	79.65	66.41	96.85	78.80	2.85
SVM	77.26	64.65	93.35	76.40	6.09
LGR	75.83	66.05	88.59	75.68	11.24
BNB	77.63	63.27	96.12	76.31	3.38
KNN	77.58	62.39	97.23	76.01	2.35
DTC	77.36	64.01	94.44	76.30	4.98

Table 4.5: Recall (%) of FeCo for Novel Testing Attacks.

FeCo	VAE	IsoForest	MLP	SVM	LGR	BNB	KNN	DTC
78.45	70.08	61.94	48.43	35.60	52.24	59.84	42.72	49.73

zero-day attacks, we split the testing attacks into known attacks and novel attacks (i.e., attacks unseen in the training data). We present the recall of FeCo for novel testing attacks in TABLE 4.5. We can see FeCo achieves a recall 8% to 42% higher than other machine learning baselines. SVM, BNB, KNN, DTC, and MLP achieve less than 50% recall, which indicates they are not suitable for detecting zero-attacks.

Among all the methods shown in TABLE 4.4, FeCo, IsoForest, and VAE share a similar detection strategy that detects intrusions by learning a model to characterize benign traffic. These methods compute a score for inputs using the learned model. The output labels are predicted by comparing the scores to a threshold. Consequently, a different threshold would lead to different detection accuracy. We plot the ROC curves of the three methods in Figure 4.8. Each point in one curve corresponds to the recall and FPR under one specific threshold. We can see that FeCo achieves the highest recall under the same FPR compared to IsoForest and VAE. Obviously, FeCo outperforms the other two methods in terms of the AUC score as well.

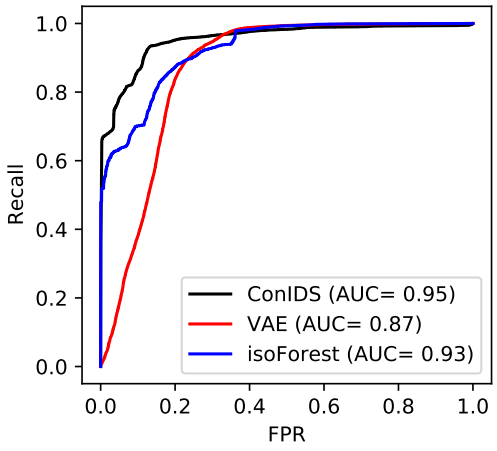


Figure 4.8: ROC curves under the centralized setting.

To better understand the performance of FeCo, we further look into the recall of FeCo for each category of intrusion attacks (i.e., DoS, Probing, R2L, and U2R). Intuitively, for one attack category, the recall means the proportion of records detected as intrusions, i.e., labeled with '1'. Formally, the recall of each attack category $i \in \{1, 2, 3, 4\}$ (1 for 'DoS', 2 for 'Probe', 3 for 'U2R', and 4 for 'R2L') is defined as

$$R(i) = \frac{\sum_j \mathbb{1}\left(\hat{y}_j = 1 \ \& \ y_j^{multi} = i\right)}{y_j^{multi} = i}, \tag{4.9}$$

where $\hat{y}_j$ denotes the prediction of FeCo on the j-th data record, and it takes a value of either 0 or 1 as FeCo is a binary IDS. Note that we have true labels of whether a testing record is 'Normal' or 'Intrusion' (i.e., $y_j \in \{0, 1\}$) and whether a testing record is 'Normal', 'DoS', 'Probe', 'U2R', or 'R2L' (i.e., $y_j^{multi} \in \{0, 1, 2, 3, 4\}$).

In TABLE 4.6, we show the confusion matrix including the attack-specific recall. Here we omit the recall of 'Normal' as normal traffic is not an attack. From TABLE 4.6, we can see that FeCo achieves higher recall on both the DoS attack and Probe attack, while the detection rate on R2L is as low as 0.51. To the best of our knowledge, all work with evaluations on

Table 4.6: Detection performance (%) in centralized setting

		Prediction		Total	Recall (%)
		Normal	Intrusion		
Ground Truth	Normal	9049	662	9711	-
	DoS	212	7246	7458	97.16
	Probe	95	2326	2421	96.08
	R2L	1352	1402	2754	50.91
	U2R	35	165	200	82.50

the NSL-KDD dataset show a low detection performance on R2L as well [104, 105]. There are two possible reasons behind the low detection rate on R2L: 1) the number of training instances belonging to R2L attack is as small as 995, and 2) there are many new attacks in the testing set that do not exist in the training set as shown in TABLE 4.3.

To investigate the impact of data distribution, we evaluate FeCo under the scenario when we only have partial data drawn from the whole data distribution. In particular, we sample the data belonging to one attack category (e.g., 'DoS') and the normal data (i.e., 'Normal'). In other words, each FeCo model in these experiments is trained from records of only one attack category and 'Normal' class. As a result, the detection performance of such FeCo models is expected to be lower than that of FeCo models trained from the whole NSL-KDD dataset. We evaluate the detection accuracy with the same testing dataset of NSL-KDD and show the results in Figure 4.9. We can see the detection accuracy with partial data is lower than that with the whole NSL-KDD dataset, which is as expected. The accuracy of the FeCo model trained with only 'R2L' and 'Normal' is as low as 50.8%. This observation indicates that self-learning may result in extremely low detection accuracy as the local data may not exhibit the overall data distribution.

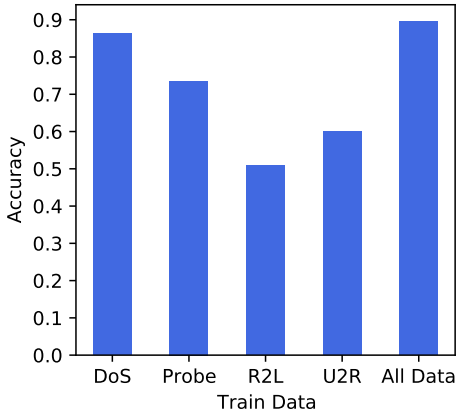


Figure 4.9: Accuracy when training data contains a single attack

4.5.5 Performance on NSL-KDD in Federated Setting

Here we focus on the performance of FeCo in the federated setting. To present a thorough comparison, we have three different learning frameworks: FL, self-learning, and centralized learning. In self-learning, the training process is done on the local device using the local data. Centralized learning refers to FeCo investigated in Sec. 4.5.4. Furthermore, data distribution is one of the biggest factors that impact FL performance. Therefore, we propose to explore three different data distributions, including one IID data distribution and two non-IID data distributions. For the IID data distribution, we randomly select a subset of each class in the whole training dataset as the local data of one client. In non-IID-1, we assume that each client only has $n \in \{1, 2, 3, 4\}$ out of the total four intrusion attack categories. In non-IID-2, each client has $n = 1$ intrusion category. For both non-IID-1 and non-IID-2, we assume all users have benign data, i.e., data from the 'Normal' class. non-IID-2 is a special case of non-IID-1 and training on non-IID-2 is more challenging than other settings due to fewer attack categories for training. Meanwhile, we use the same testing data in all settings for a fair comparison.

We show the detection performance of FeCo with combinations of the three learning types

Table 4.7: Performance (%) of FeCo in FL, self-learning, and centralized learning frameworks.

Learning Type	Data Distribution	Accuracy	Recall	Precision	F1 score	FPR	DoS	Probe	R2L	U2R
FL	IID	85.65	81.15	92.73	86.55	8.41	92.69	98.68	35.00	74.00
	non-IID-1	86.23	82.10	92.89	87.16	8.29	93.12	95.67	41.03	72.50
	non-IID-2	85.53	77.90	95.93	85.98	4.26	83.33	95.29	36.49	76.50
Self-learning	IID	81.91	74.53	92.21	82.43	8.33	87.66	92.14	24.33	63.07
	non-IID-1	82.62	75.69	92.39	83.21	8.23	90.48	87.37	25.63	71.81
	non-IID-2	67.66	46.51	93.89	62.21	4.39	54.20	47.22	24.14	59.25
Centralized Learning	-	89.55	86.80	94.39	90.44	6.82	97.16	96.08	50.91	82.50

and three data distributions in TABLE 4.7. In self-learning, the value of evaluation metrics is shown as an averaged value over all clients. Compared with self-learning, FL achieves higher accuracy, recall, and precision regardless of the data distribution. Furthermore, FL achieves similar detection performance under the three data distributions, implying that FeCo is able to obtain stable learning performance with different data distributions. Unlike FL, the performance of self-learning heavily depends on the data distribution. We can see that self-learning achieves as low as 67.66% average accuracy under non-IID-2 data distribution. One noticeable result is that the detection accuracy of FL is still lower than the centralized learning. In practice, centralized learning is difficult to deploy due to privacy concerns. We further show the box-plot of the self-learning accuracy of 50 clients in Figure 4.10 to further accommodate TABLE 4.7 (TABLE 4.7 only shows the average accuracy among clients). We can see that the variance of the accuracy of non-IID-2 is much larger than those of IID and non-IID-1.

We have another observation when comparing the performance of FeCo with different data distributions but the same learning framework. In the FL setting, the accuracy of FeCo on non-IID-1 is 86.23% which is higher than the 85.65% accuracy achieved on IID data. Similar phenomena occur in self-learning as well. The possible reason behind this is that: the IID data contain a relatively large number of attack sub-classes (i.e., 20) within the small local dataset. Therefore, it becomes difficult to learn a stable representation of benign

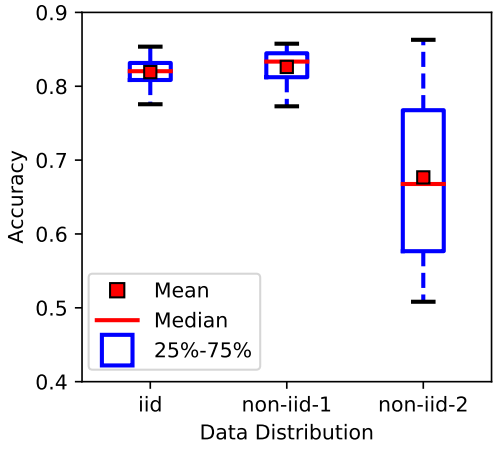


Figure 4.10: Accuracy in self-learning mode.

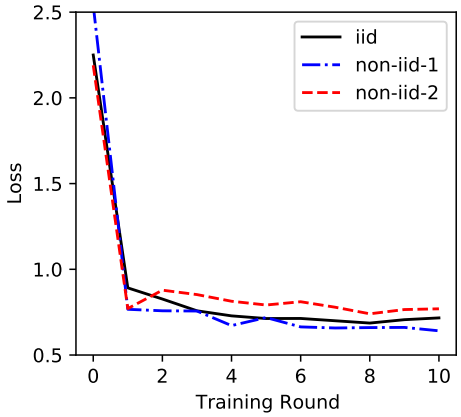


Figure 4.11: Convergence performance.

instances when contrastive learning iteratively pushes benign representations away from so many different attacks. On the contrary, in the non-IID-1 setting, each client possesses only a subset of attack classes thus a smaller number of attack sub-classes. Therefore, it is easier to learn a stable normal template. The performance of non-IID-2 is worse than IID possibly because overfitting occurs as the attack categories in the local data are too limited.

We study the convergence performance of FeCo by plotting the value of loss through the training process. As shown in Figure 4.11, the loss drops sharply at the beginning of the

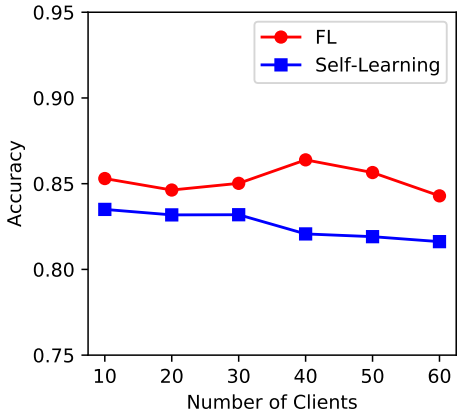


Figure 4.12: Accuracy VS. client number.

training process, decreases slowly after certain epochs, and finally stays stable. We can see that the loss of FeCo reaches the stable state fast under the three data distributions, implying that FeCo converges after a small number of learning rounds.

We also explore the scalability of FeCo by varying the number of clients. We show the accuracy of FeCo with different numbers of clients in Figure 4.12. We can see that the accuracy of self-learning decreases monotonically with the increase in the number of clients. This is because the size of the local data also decreases with the number of clients as they are inversely proportional to each other. However, the accuracy of FL does not show a decreasing trend. This observation indicates that FeCo scales well with the number of clients in the FL mode.

We show the overhead of FeCo in Figure 4.13. Specifically, we present the per-client running time and the total running time of all clients for one FL round. Note that an FL round means that the selected clients finish training their local models and upload the model parameters one time. We evaluate the running time with the number of clients ranging from 10 to 60. Note that a larger number of clients means a smaller dataset at the local client as discussed above. We can see that the per-client running time decreases with the increase

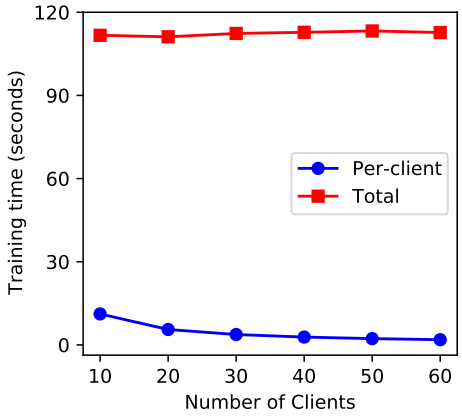


Figure 4.13: Running time

in the number of clients while the total running time stays approximately the same. The per-client running time is as low as 1.878 seconds when the local data contains 2000 records, indicating that FeCo requires relatively little computation resources. We plan to implement FeCo in a gateway device in our future work to demonstrate that FeCo is affordable even in a local gateway with low computation capacity.

4.5.6 FeCo performance on BaIoT dataset

We further evaluate FeCo using a real-world IoT dataset. The BaIoT dataset includes nine commercial IoT devices as shown in TABLE 4.8. We train a model for each IoT device and evaluate the performance separately.

In order to test the capability of FeCo to detect zero-day attacks, we design our data split method as follows. The BaIoT dataset is composed of ten classes of malicious data: five classes from the Mirai attack family and five classes from the BASHLITE attack family. We randomly select three from the five Mirai attacks as the training data. The remaining two Mirai attacks and all five BASHLITE attacks are used for testing. There are no Mirai attack records for two of the nine devices (the Ennio doorbell and the Samsung SNH 1011

Table 4.8: Performance of FeCo on the BaIoT dataset.

Index	Devices Make and Model	Device Type	Accuracy (%)	Recall (%)	Precision (%)	F1 (%)	FPR (%)
1	Danmini	Doorbell	99.98	99.99	100.00	99.99	0.16
2	Ennio	Doorbell	99.95	99.99	99.96	99.97	0.77
3	Ecobee	Thermostat	99.98	99.98	100.00	99.99	0.13
4	Phillips B120N/10	Baby monitor	99.94	99.97	99.97	99.87	0.37
5	Provision PT-737E	Security camera	99.96	99.98	99.98	99.98	0.50
6	Provision PT-838	Security camera	99.96	99.98	99.98	99.98	0.34
7	SimpleHome XCS7-1002-WHT	Security camera	99.97	99.98	99.99	99.98	0.37
8	SimpleHome XCS7-1003-WHT	Security camera	99.98	99.98	100.00	99.99	0.31
9	Samsung SNH 1011 N	Webcam	99.95	99.95	99.99	99.97	0.07

N webcam). Therefore, we randomly select three from the five BASHLITE attacks as the training data, and the remaining two BASHLITE attacks are used for testing. The benign traffic data is split into 70% for training and 30% for testing.

We show the performance of FeCo on the BaIoT dataset in TABLE 4.8. The BaIoT data is imbalanced: the number of malicious data records (6,506,674) is about ten times of benign data records (556,932). Therefore, we present recall and FPR in addition to detection accuracy to better exhibit the detection performance. FeCo achieves recall as high as 99.99% with a small FPR of 0.16%. TABLE 4.8 demonstrates that FeCo is effective in detecting malicious traffic for various types of IoT devices. Furthermore, the high overall recall indicates that the IDS model trained on partial intrusion attacks effectively detects unseen attacks. Other baselines in [89] achieve similar recall but higher FPR. The FPR of Autoencoder, isolation forest, SVM, and FeCo is shown in Figure 4.14. We can see IsolationForest is unstable in detecting intrusions among different types of IoT devices as it results in a large FPR on Device 4 and Device 5. FeCo outperforms all the evaluated baselines by achieving a much smaller FPR.

We further explore how FeCo detects intrusions. As discussed in Sec. 4.4.3, FeCo computes the cosine similarity of an input with the normal template and flags the input as intrusion if the calculated similarity score is less than a pre-defined threshold. We show the histogram of

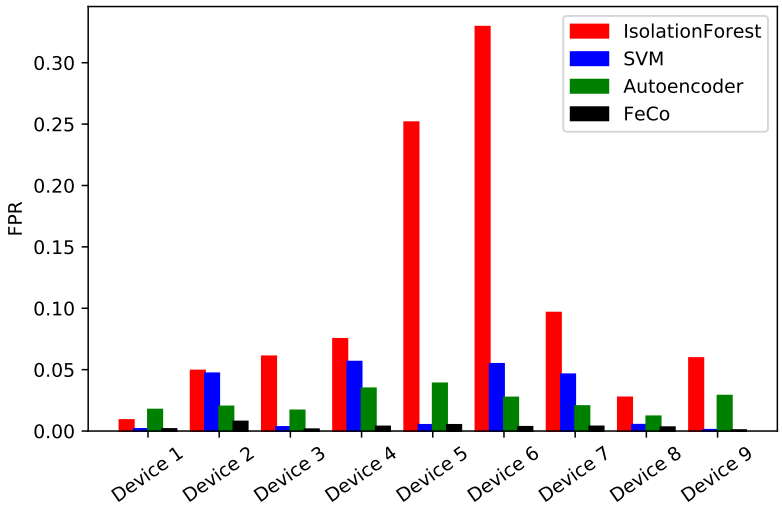
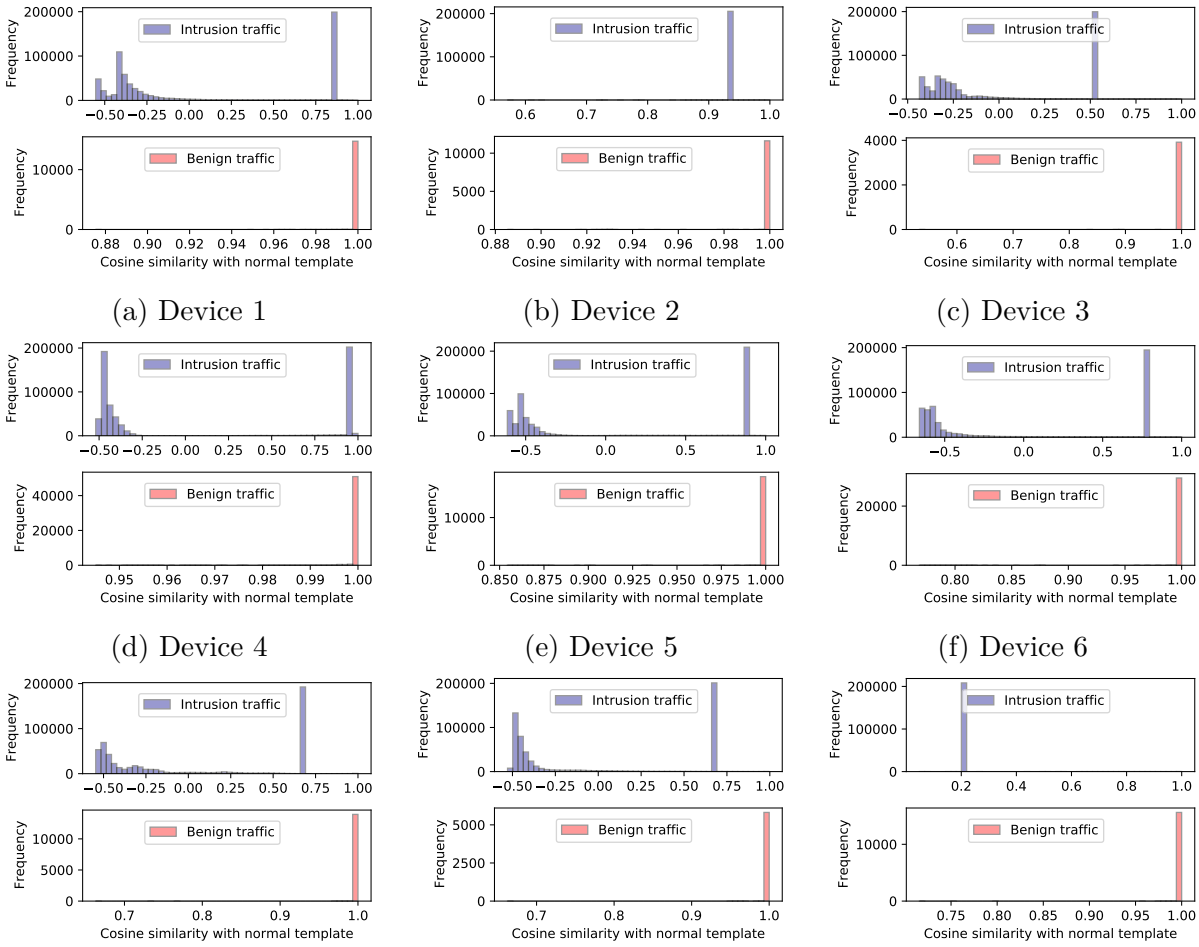


Figure 4.14: FPR when the value of Recall is fixed to 0.9998.

the similarity scores of test data points in Figure 4.15. To give a clear view of the similarity score distribution, we analyze the scores of intrusion records and scores of benign records separately. As shown in Figure 4.15a, two sub-figures are given: the top sub-figure showing the similarity score distribution of the intrusion traffic and the bottom sub-figure showing the similarity score distribution of benign traffic. We can see that similarity scores of intrusion traffic are distributed from -0.56 to 0.88 and has two peaks at around -0.4 and 0.88. The instances that form the left peak and its vicinity represent the Mirai attack, and the instances form the right peak represents the BASHLITE attack. For benign traffic, the similarity score gathers at a very narrow range that is very close to the value of 1.00. The gap between the largest score of intrusion traffic and the smallest value of benign score demonstrates FeCo's capability to detect intrusions. The similarity score distribution of the other eight devices is shown in Figure 4.15b to Figure 4.15i. Refer to TABLE 4.8 for mapping a device number in the caption of Figure 4.15 to a real device. We can also see only one peak in the similarity score distribution of the intrusion traffic in Device 2 and Device 9. The observation is because the two devices are only infected by the BASHLITE attack but not the Mirai attack.



(a) Device 1 (b) Device 2 (c) Device 3

(d) Device 4 (e) Device 5 (f) Device 6

(g) Device 7 (h) Device 8 (i) Device 9

Figure 4.15: Evaluations of FeCo on BAIoT dataset.

4.6 Computation Optimization

We further optimize FeCo to make it compatible with computationally constrained IoT devices. In this section, we optimize the FeCo by reducing the size of the neural network model used by FeCo. We instrument the optimized FeCo system to a Raspberry Pi to demonstrate its applicability in computation-constrained IoT devices.

4.6.1 Model Weights Quantization

In order to reduce the model size, we perform model weight quantization. With a smaller model size, we can achieve the following advantages. A smaller model will occupy less storage space on an IoT device. In this way, an IoT device can save more storage space for its original operation data. Secondly, a smaller model will use less RAM in the run-time, which saves more memory for other applications of an IoT device. Moreover, quantization will also simplify the calculations involved in the inference, resulting in decreasing the amount of time of a single inference operation (i.e., latency). However, model weight quantization also potentially sacrifices the model's accuracy. There exists a trade-off between model accuracy and model size. We employ post-training quantization that quantizes the model parameters after a model is trained. There are two weights quantization methods:

- Int-8 quantization: The 32-bit float weights of a trained model are quantized to an 8-bit integer. The model size will be 4 times smaller.
- Float-16 quantization: The 32-bit float weights of a trained model are quantized to 16-bit float. The model size is expected to be 2 times smaller.

We perform model quantization on an open-source framework, i.e., the TensorFlow Lite [5].

[5] https://www.tensorflow.org/lite

Model quantization is a post-training process, meaning that the model training stays the same as what is depicted in Sec. 4.4. The operational flow is: firstly, training a model on a local gateway; optimizing the trained model to a smaller size by weight quantization; implementing the optimized model to a Raspberry Pi device. The first two steps can be completed by either a local gateway or a server. We will focus on the third step as we aim to analyze the performance of FeCo when it is implemented on resource-constrained devices (i.e., most IoT devices).

4.6.2 Implementation and Evaluation

We prototype our computation-optimized FeCo on a Raspberry Pi 4 equipped with 1.5 GHz quad-core A72 64-bit ARMV8 CPU, 4GB RAM, 32GB SIM card memory, and Raspbian OS. This experiment aims to check the feasibility of deploying FeCo on resource-constrained IoT devices. The metrics we evaluate include the testing phase performance and the inference latency of various quantized models.

Table 4.9 demonstrates the performance of FeCo on a Raspberry Pi. In the table, we compare the performance among three model optimization methods: No quantization (Float-32), Float-16 quantization, and Int8 quantization. We witness a significant reduction in the model storage size with model quantization methods. Compared to No quantization, the model storage space is reduced by about half when applying Float-16 quantization. And the storage space is reduced by about 3/4 when applying Int-8 quantization. Further, the inference time is also reduced by applying model quantization. Int-8 quantization saves 0.016 ms for inference on one data record compared to No quantization, which speeds up the inference by 11%. Float-16 quantization has nearly the same inference time as No quantization. This is because the ARM processor does not support 16-bit computation,

Table 4.9: Efficiency and performance of FeCo implemented on a Raspberry Pi device.

Quantization method	Model size	Inference time	Accuracy (%)	Recall (%)	Precision (%)	F1 (%)	FPR (%)
No quantization (Float-32)	254kB	0.152ms	99.96	99.97	100	99.98	0.33
Float-16 quantization	129kB	0.151ms	99.96	99.97	100	99.98	0.33
Int-8 quantization	67kB	0.136ms	99.96	99.97	100	99.98	0.33

and the model converted to 16-bit will be upsampled to float 32 before the inference. Most importantly, the Int-8 quantization and Float-16 quantization suffer from no decay in detection performance compared to the Non-quantized model. Therefore, we recommend Int-8 quantization on the CPU-based platform (e.g., the ARM processor). And both Int-8 quantization and Float-16 quantization will be good choices for GPU-based platform as GPU support both 8-bit computation and 16-bit computation. In sum, Table 4.9 demonstrates the optimized model can save storage space and speed up the inference without degrading the detection performance.

4.7 Discussions

Device type identification is challenging and critical for managing IoT networks [85, 113, 171], and it is also a significant part of FeCo design. The goal of device type identification in this work is to cluster IoT devices together based on their security requirements and vulnerability. It is demonstrated that devices from a given manufacturer running the same version of firmware will have the same vulnerabilities [85]. We borrow their idea to map devices to an abstract device type for which the system has learned a specific set of policies. In the proposed FeCo system, we assume IDS service providers (i.e., the IDS learner on the cloud side) will initialize an IDS model in a cloud server only if they identify a type of device on the market. The service provider then shares the policy set with local gateways. Local gateways perform device type identification with the knowledge shared by the cloud server.

Local gateways can decide to participate in the learning or not when their devices match one or more known device types. The devices unknown by the IDS service providers won't contribute to the learning. Further, the addition of a new device type relies on the IDS service provider to recognize the new type on the market and initialize a new intrusion detection task in the cloud.

System scalability is another critical problem. Here we discussed the scalability at local gateways and the cloud server. At the local gateway, the computation overhead is related to both the number of devices and the number of device types. The number of IDS models maintained at a local gateway will linearly increase with the number of device types. The computation is reasonable since both the size of the IDS model and the size of the local network is not too large. Further, within one IDS model, a local gateway can choose the amount of data for model training based on its computation capability. The server's communication and computation overhead is determined by the number of participating local gateways. As the design of federated learning (FL), the cloud server can choose the number of local gateways to participate in a learning iteration. Therefore, even if a large number of local gateways volunteer to participate in FL, the cloud server can still select a small portion of them in each learning iteration to ensure the task is within its communication and computation capability.

4.8 Conclusion

In this chapter, we propose FeCo, a machine-learning-based IDS for IoT networks. FeCo incorporates contrastive learning into the federated learning framework to support distributed intrusion detection while preserving user data privacy. More importantly, FeCo features a novel detection method based on network traffic representation learning through

contrastive learning. While learning for the network traffic representation, FeCo tries to maximize the distance between benign and malicious samples and minimize the distance among benign samples. This effectively enables FeCo to achieve better detection accuracy than other baselines as FeCo obtains a more stable normal profile of network traffic. In order to avoid overfitting, we further propose a two-step feature selection scheme to remove redundant features before learning. The feature selection scheme also decreases computation complexity, making FeCo more suitable for resource-constrained IoT devices. We demonstrate the high effectiveness of contrastive learning in IDS through extensive evaluations with the NSL-KDD dataset and BaIoT dataset. We perform model optimization by leveraging the model weight quantization method. We implement and evaluate the optimized FeCo model for a low-end device (i.e., Raspberry Pi) to demonstrate the efficiency and effectiveness of FeCo in IoT devices.

Chapter 5

Privacy-Preserving Federated Meta-Learning

(Copyright notice[1])

5.1 Introduction

Deep learning (DL) has enjoyed great success in many sectors of society, mainly due to the big leap in computing hardware capability and availability of massive amounts of data. However, the increased concentration of data needed by DL has raised widespread concerns over data privacy, leading to data privacy regulations such as General Data Protection Regulation (GDPR[2]) in European Union and California Consumer Privacy Act (CCPA[3]) in the US. In many cases, it is often impossible to move the data to a central location due to legal restrictions, such as those imposed by Health Insurance Portability and Accountability Act (HIPAA[4]). Therefore, more and more data are now stored distributively at edge nodes or end devices close to their sources rather than at a central location.

Federated learning (FL) [62, 72, 87] has emerged as a new paradigm to enable collaborative

[1]This chapter previously appeared as a part of a conference paper published in ACSAC 2022. ©2022 Copyright held by the owner/author(s). Reprinted, with permission, from Ning Wang, Yang Xiao, Yimin Chen, Ning Zhang, Wenjing Lou, and Y. Thomas Hou, "Squeezing More Utility via Adaptive Clipping on Differentially Private Gradients in Federated Meta-Learning" In Proceedings of the 38th Annual Computer Security Applications Conference (ACSAC '22), pages 647–657, 2022 [153].

[2]https://gdpr.eu/

[3]https://oag.ca.gov/privacy/ccpa

[4]https://www.hhs.gov/hipaa/index.html

training over distributed private data. In FL, participants jointly train a global model without sharing their private data. FL outputs a common model for all the users and does not customize the model for each user. This is an important missing feature for practical deployment in distribution learning scenarios, especially given the heterogeneity of the underlying data distribution and learning task for various users. In light of this gap, federated meta-learning [23, 34, 78, 173] has emerged as one powerful AI framework for enabling fast model adaptation amid collaborative training, with prime use cases in IoT and mobile computing scenarios. The meta-learning component [39, 101] enables a cloud server to extract common knowledge from distributed data owners with different training data and local tasks. The common knowledge, in the form of a global meta-model, can be quickly customized to a new client (called "model consumer") with a few new data samples.

Despite its prospect of making the best of two worlds, the federated meta-learning framework is still prone to inference-based privacy attacks on individual clients' data, including the membership inference attack [98, 174] and model inversion [157]. An adversary can recover the private training data [154, 157] or sensitive partial information [98] by leveraging a specific inference model. Meanwhile, prior wisdom alludes that Differential privacy (DP) [31] can be used to provide rigorous privacy guarantee to FL algorithms by adding noise to gradients update in a controlled manner [43, 88, 158]. Privacy protection faces complications in the federated meta-learning framework compared to a traditional ML. On the one hand, the adoption of DP into FL may lead to a significant decrease in model utility. Taking the popular FedAvg [87] algorithm as an example, naively adopting DP into FedAvg may cause 2-10 times training loss than the original model [160]. On the other hand, the privacy notion is further complicated due to the hierarchical nature of federated meta-learning frameworks. There are two types of models across system participants: base models at clients and the meta-model at the central server. It is unclear which gradients contain what level of privacy

and are exposed to whom. [70] directly applies DP to the federated meta-learning framework and thus results in a large accuracy sacrifice. We aim to improve the model's accuracy while satisfying a minimum privacy protection requirement.

In this chapter, we address the above privacy leakage problem by substantiating a new differentially private federated meta-learning architecture, namely DP-FedMeta. Catering to the hierarchical learning framework of federated meta-learning, DP-FedMeta features two DP mechanisms, namely DP-AGR (AGregation Rule) and DP-AGRLR (AGregation Rule with Local Randomness), for achieving two practical privacy protection notions for clients' training data against the curious central server and curious local clients, respectively. DP mechanisms in ML generally entail bounding each user's model update contribution by clipping its gradient to some constant value. However, in meta-learning tasks, it can be hard to obtain a priori knowledge of the clipping threshold across tasks and learning settings. To achieve a better trade-off between privacy and accuracy, building on the intuition that the current gradient norm is predictive with the knowledge of data geometry in earlier iterations, we propose an adaptive clipping mechanism to dynamically achieve a minimized clipping threshold while preserving most of the gradient information. We emphasize that naive adaption of adaptive clipping for federated meta-learning, such as determining the clipping threshold based on the current batch of gradients, would leak clients' private information since such clipping threshold is computed with the true gradients. To avoid further privacy leakage in adjusting the clipping threshold, we utilize the historical differentially private aggregated gradients (i.e., a by-product of differentially private gradient aggregation of the previous training set) instead of the true online gradients. As a result, our adaptive clipping mechanism retains the same level of privacy protection while boosting the system's overall accuracy. Putting the above designs together, DP-FedMeta attains a significantly lower privacy budget and higher model accuracy simultaneously compared to the state-of-the-art

DP solution for federated meta-learning [70] which is a direct application of DP-based meta-learning to the federated setting.

The major contributions of this chapter are summarized as follows:

- We introduce a differentially private federated meta-learning architecture, dubbed DP-FedMeta, to enable collaborative model training by heterogeneous users with fast model adaptation and rigorous privacy guarantee. Catering to two practical trust levels on the central server, DP-FedMeta features two variants of DP mechanisms, DP-AGR and DP-AGRLR, respectively.

- For both DP-AGR and DP-AGRLR, we propose a novel adaptive gradient clipping method that maximally preserves model accuracy while guaranteeing a fixed level of privacy. This method feeds on past differentially private gradients (instead of the sensitive original gradients), which essentially helps achieve a significantly lower privacy budget ($\epsilon = 1.5$ or 2.5) than the state-of-the-art DP federated meta-learning work ($\epsilon = 9.5$) [70] while maintaining reasonable model accuracy.

- We analyze the impact of client/task sampling rate on accumulated privacy loss for the whole training process of DP-FedMeta. We derive an upper bound of the final accumulated privacy loss given the sampling rate and noise level (Section 5.3.5). An important insight is that privacy loss depends on the sampling rate, rather than the number of training tasks sampled.

- We thoroughly evaluate DP-FedMeta's performance on three well-known datasets, i.e., Omniglot, CIFAR-FS, and mini-ImageNet. We evaluate the impact of various factors including gradient clipping, noise multiplier, and user sample size on model accuracy and privacy for both DP-AGR and DP-AGRLR methods. Under a small $(1.5, 10^{-6})$-DP

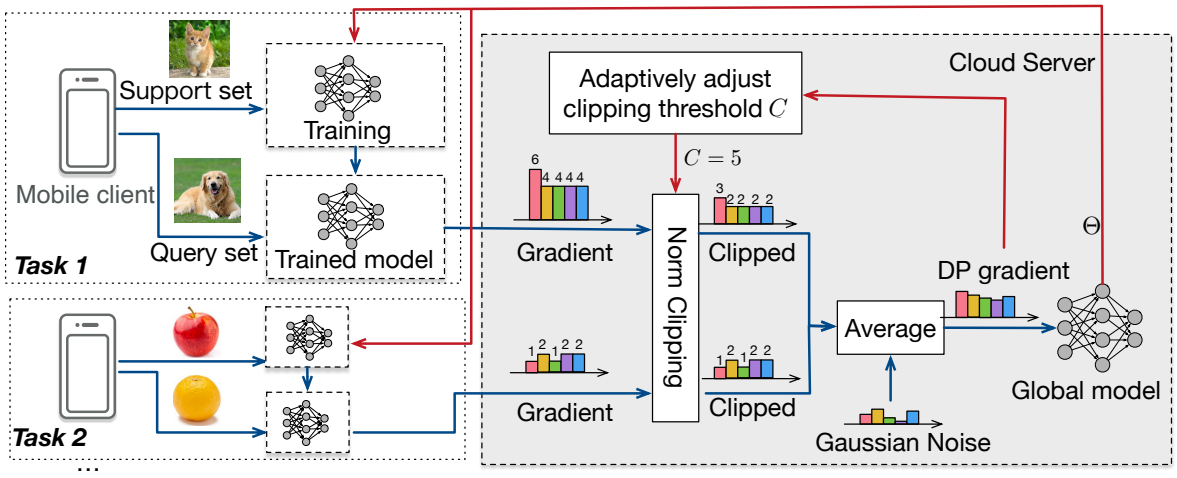


Figure 5.1: System design of DP-FedMeta.

budget, DP-AGR achieves as high as 96.8% accuracy and DP-AGRLR achieves 89.7% accuracy (the accuracy of [70] is 75%) for 5-way 5-shot learning on Omniglot.

5.2 DP-FedMeta Overview

5.2.1 System Architecture and Vision

The vision of DP-FedMeta is to enable distributed mobile clients to perform federated meta-learning with rigorous privacy guarantees on client data. Figure 5.1 illustrates the basic workflow of the DP-FedMeta architecture. There are three types of participants in our system:

Mobile Clients contribute to the meta-training process with their private data. A mobile client first initializes its neural network θ with the current meta-model Θ. It then trains the base-model θ with its local data. Due to the energy constraint, a mobile client can only perform one to several steps of gradient descent. It then computes model gradients and submits them to the central server. In later sections, the mobile client is also referred to as

client for convenience.

Central Server's primary role is processing and aggregating the model gradients from mobile clients. At the system onset, it randomly initializes the meta-model Θ. In each meta-training round, it first distributes Θ to a group of mobile clients, collects gradients from them, and applies gradient clipping with a clipping threshold. The central server then sums over all clipped gradients added with Gaussian noise. The noisy gradient is used to update the meta-model. It outputs the final meta-model at the end of the meta-training round.

Model Consumers are some of the mobile clients who only consume meta-models from the central server without reporting local gradients to it.

We assume mobile clients have limited storage space and are energy-constrained. Therefore, there are only a small amount of data can be stored in a mobile device, and the mobile device does not support a computationally-intensive process, e.g., training a deep model from scratch independently. Clients participate in an FL system to cooperatively train a model initialization parameter. Each mobile client (including mobile consumers) has its own training task. We assume that the training tasks for different clients are not necessarily the same. One example of a task is to differentiate dogs from cats, and another task is to differentiate apples from pears. It can be a general case in practice since different clients may have different application scenarios. The overall goal of our system is not to train a model that works for only one task but to train a well-generalized model for various tasks. The traditional FL can not deal with this problem as traditional FL is for a scenario where clients' training tasks are the same. We propose to employ meta-learning to learn a parameter initialization and customize it to different tasks of clients. Meta-learning [14, 39, 148] learns common knowledge across a large number of tasks which is an optimized starting point for various new tasks as it can fast adapt to unseen tasks. We build DP-FedMeta based on a meta-learning algorithm (i.e., MAML[39]), and our design also applies to other meta-learning

algorithms. Please refer to Section 5.3.2 for a more detailed discussion on MAML.

5.2.2 Threat Model and Challenges

In FL, both the central server and clients can be curious about clients' privacy. We assume there are two levels of trust on the central server: trusted and honest-but-curious. A trusted central server strictly follows the aggregation procedure and is not inquisitive about clients' training data. An honest-but-curious central server also follows the aggregation procedure but may sniff updates from clients to reveal the information of their training data. In practice, some reputable organizations, such as publicly owned and government-backed institutions, can be assumed to be trusted. For other commercial entities, the central server is assumed honest-but-curious. Furthermore, we also assume clients (including model consumers) are honest but curious. Recent literature demonstrates that an adversary [157] who has access to the model and output label can reconstruct the training data. Further, an adversary [98, 174] can differentiate whether one data record is in the training dataset or not by accessing only the model. We apply DP to FL to protect individual clients' training data privacy.

We aim to maximize the model's accuracy while satisfying a minimum privacy protection requirement. However, adopting DP into FL may lower model utility significantly, e.g., naively incorporating DP into FedAvg may cause 2-10 times training loss than the original model [160]. The hierarchy gradients of federated meta-learning bring further challenges in applying DP to the learning process. We will detail the challenges and our solutions in what follows.

5.2.3 Applying DP to Federated Learning Clients

The objective of DP is to enable the utilization of information on a population while hiding individual information. Mathematically, DP is defined as follows:

Definition 5.1 (Differential Privacy). A mechanism $\mathcal{M}$: $D \rightarrow R$ is (ϵ, δ)-differentially private if for any subset of outputs $S \subseteq R$ and for any two adjacent databases $d, d' \in D$, $\mathcal{M}$ satisfies that:

$$\Pr[\mathcal{M}(d) \in S] \leq e^{\epsilon} \Pr[\mathcal{M}(d') \in S] + \delta. \tag{5.1}$$

d and d' differ in at most a single entry, i.e., $\|d - d'\|_1 \leq 1$.

Considering $\delta \rightarrow 0$ and $\epsilon \rightarrow 0$, $\frac{\Pr[\mathcal{M}(d) \in S]}{\Pr[\mathcal{M}(d') \in S]} \leq e^{\epsilon} \approx 1$ indicates that one cannot distinguish between d and d' by observing $\mathcal{M}(d)$ and $\mathcal{M}(d')$.

When applying DP to an ML model, we need to add noise to gradients before updating the model in the training process. Gradient clipping and noise adding are two critical procedures that impact the privacy-utility trade-off.

5.2.4 Why Adaptive Clipping?

The trade-off between privacy and utility is a significant challenge to a federated learning system. It is not the actual noise size, but the ratio of noise over the individual gradient impacts the privacy protection level. We call the ratio noise multiplier denoted by z. A larger z will better conserve privacy but may significantly harm the model's accuracy. A more negligible z helps maintain good model utility at each learning step while the privacy budget will be used up quickly. In this way, the utility of the final model may also be low since we may need to terminate the training process early because of depleting privacy budget. We need first select an appropriate z. The real noise level is also related to the

gradient clipping threshold besides z. We figured out there is still space to squeeze the privacy budget after z is selected. We can perform the gradient clipping more smartly. The reason why adaptive clipping is more favorable than constant clipping is explained in the following.

DP is designed to hide the existence of every individual data so that the noise should be z times the most significant gradient. Therefore, the largest gradient determines the final noise size. To maintain model utility, we should avoid large noise sizes. To this end, we clip the gradients using a pre-defined clipping threshold. A small clipping threshold will result in a small noise size but lose the original gradient's information. We should minimize the clipping threshold while preserving most of the gradient information. It is not trivial to decide the clipping threshold since the gradient norm varies across tasks and training processes. In response, we propose an adaptive clipping mechanism to achieve this goal (see Sec. 5.3.1).

5.2.5 Two Levels of Privacy Protection

A critical challenge arises from the hierarchical architecture of the federated meta-learning framework, mainly attributed to the meta-learning component. There are two ML models across the participants: base models at clients and the meta-model at the central server, which consume different types of gradient updates. Based on the threat model, we need to clearly define which gradients contain what level of privacy and are exposed to whom. Accordingly, we assume two trust levels on the central server. For a trusted central server, only the meta-model Θ will be exposed to adversaries. For an honest-but-curious central server, both Θ and base model θ will be exposed to adversaries. To incorporate DP mechanisms, we define two levels of privacy protection for the two adversary models accordingly:

1) User-level DP: Releasing Θ will at no point compromise information regarding any specific task client.

2) Two-fold DP: Releasing Θ will at no point compromise either any specific data records or any client. Meanwhile, uploading θ_i will at no point reveal the existence of any specific data records used during training to the central server. Here "two-fold" refers to both user-level and record-level DP.

Two-fold DP is more strict than the user-level DP. For DP-FedMeta, both the user-level DP and Two-fold DP can preserve the client-level privacy from honest-but-curious clients or model consumers, meaning that the model consumers who receive the intermediate/final meta-model can not differentiate whether another user participates in the meta-training process or not. When faced with an honest-but-curious central server, two-fold DP additionally incorporates DP at the record level. Record-level DP means that no one can reveal the information of a client's individual data record by seeing the client's update. Compared to previous DP works [43, 70, 88], which either protect record-level privacy or user-level privacy, we are the first to preserve the two different levels of privacy simultaneously to the best of our knowledge.

5.3 Detailed System Design with Differential Privacy

This section presents our methods to achieve DP in DP-FedMeta. We detail our schemes, i.e., DP-AGR (Differentially Private AGregation Rule) and DP-AGRLR (Differentially Private AGregation Rule with Local Randomness), for achieving user-level DP and two-fold DP respectively. We first introduce adaptive clipping since it is a critical component for both algorithms.

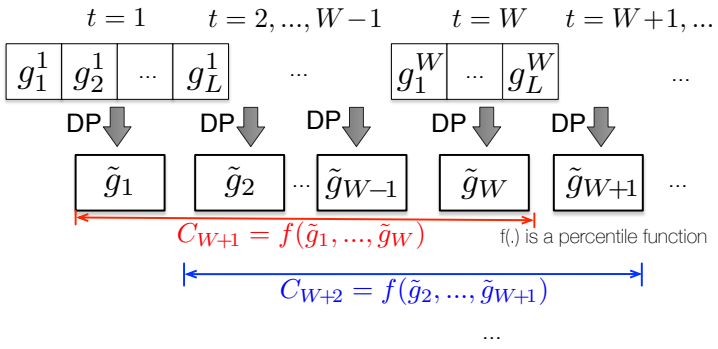


Figure 5.2: Illustration of proposed adaptive clipping method. $\tilde{g}_t$ denotes the differentially private version gradient at time step t. The central server calculates adaptive clipping threshold C_t using DP gradients in a window of size W.

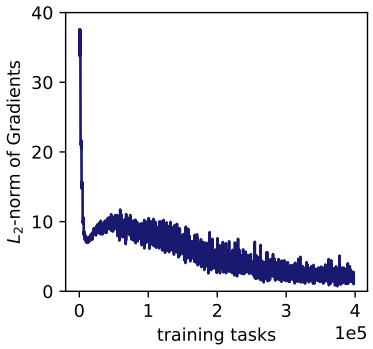


Figure 5.3: The L_2-norm of gradients through the training process.

5.3.1 Adaptive Gradient Clipping Method

The goal of the adaptive gradient clipping method is to minimize the clipping threshold while preserving most of the gradient information. The method is based on the L_2-norm. One observation in our experiments is that gradient norms decrease significantly over the course of the training process, as shown in Figure 5.3. Therefore, an optimized clipping threshold for the early training stage will be too large for the later training stage.

In our adaptive clipping method, we use a sliding window with size W and step size ΔW to obtain a runtime clipping threshold C, illustrated in Figure 5.2. $\Delta W = 1$ in our

implementations. We introduce the concept of time step t, and t increases by one once one training iteration is completed. We assume there are L clients sampled to participate in FL in each iteration. Thus, L clients' gradients will be sent to the central server at the end of each learning iteration. Each entry g_i^t in the first row of boxes in Figure 5.2 is the L_2-norm of gradient from the i-th client of the L sampled clients at the time step t. As shown in the second row of boxes in Figure 5.2, we will obtain a noisy aggregation $\tilde{g}$ of the current L gradients. Notably, the noisy aggregation step is the key procedure of DP-AGR, not a design for the adaptive clipping method. The adaptive clipping method takes advantage of differentially private versions of gradients for calculating a dynamic clipping threshold without incurring additional privacy loss. Specifically, the adaptive clipping threshold at time step $t+1$ is computed with a sequence of differentially private versions of gradients before $t+1$ (i.e., $\tilde{g}_{t-W+1}, \tilde{g}_{t-W+2}, ..., \tilde{g}_t$) by

$$C_{t+1} := f(\{\tilde{g}_{t-W+1}, ..., \tilde{g}_t\}, k) \tag{5.2}$$

where $f(S, k)$ represents the k-th percentile of a sequence S and time step $t > W$. We use the constant clipping method for the first W time steps in the training process since the number of collected differentially private gradients is less than the window size. Another observation in our experiments is that gradient norms in DP-AGR exhibit sharp spikes due to the additive perturbation during the training process. The adaptive clipping threshold C_t (in DP-AGR) fluctuates when we directly apply the aforementioned window-sliding mechanism. We solve this issue by adding a smoothing scheme, i.e., updating C_t only when it gets no larger than the previous C_{t-1}.

5.3.2 Instantiating Meta-learning with MAML

In practice, the data generated by different clients may follow different distributions, e.g., one client has apple and orange images while another client has cat and dog images. In this chapter, we propose to employ meta-learning [14, 39, 148] to learn a parameter initialization. Meta-learning learns common knowledge across a large number of tasks which is an optimized starting point for various new tasks as it can fast adapt to unseen tasks.

We leverage a widely accepted meta-learning paradigm, model-agnostic meta-learning (MAML) [39], to learn the common knowledge of multiple clients. In MAML, there are multiple tasks. Each task makes use of two datasets: support set for local model training and query set for meta-model training. The meta-learner maintains a global meta-model while a task-learner learns a base model for its tasks. Meta-model and base-model are neural networks with the same architecture but different weights, which can be represented by a mapping function f: from input $x \in \mathbb{R}^{w*h*c}$ to output $y \in \mathbb{R}^N$ (N is the number of classes). We use shortcuts f_Θ and f_θ to differentiate the two models. The training phase of MAML starts by initializing the base model as the current meta-model Θ, proceeds to train on the support set, and obtains a trained base model for each task. The training process of a base model for task $\mathcal{T}_i$ is:

$$\theta_i \leftarrow \Theta - \eta_1 \nabla_{\theta_i} \sum_{(\boldsymbol{x},y)\in\mathcal{D}^s_{train,i}} \mathcal{L}(f_{\theta_i}(\boldsymbol{x}), y), \tag{5.3}$$

where $\mathcal{D}^s_{train,i}$ is the support set, η_1 is the learning rate, $\mathcal{L}(f_{\theta_i}(\boldsymbol{x}), y)$ is the loss of θ_i, and ∇ is the differential operator for calculating the gradient. After learning L base-models for selected tasks $\mathcal{T}_s = \{\mathcal{T}_1, ..., \mathcal{T}_L\}$), the meta-model Θ is updated using the summation of loss

from multiple training tasks' query set as follows:

$$\Theta \leftarrow \Theta - \frac{\eta_2}{L} \nabla_\Theta \sum_{i \in \mathcal{T}_s} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}^q_{train,i}} \mathcal{L}(f_{\theta_i}(\boldsymbol{x}), y), \tag{5.4}$$

where η_2 is the learning rate for meta-model. MAML uses losses from multiple tasks to update Θ, thus it obtains across-task knowledge. In the implementation, MAML has two loops. Eq. (5.3) occurs in the inner loop while Eq. (5.4) in the outer loop. We can achieve a trained meta-model Θ which can be used to initialize an ML model for a client with a new task. We can achieve high accuracy with only a few pieces of data since the initialization contains common knowledge.

Our algorithm is completed by both the clients and the central server as a traditional FL algorithm. In the beginning, L clients are selected to participate in the training process by the central server. To take advantage of meta-learning, the selected clients randomly split their local data into two sets with equal size: a support set $\mathcal{D}^s_i$ and a query set $\mathcal{D}^q_i$. Each client trains its local model using its support set following Eq (5.3). Then each client further calculates a gradient of the trained model using the query set and sends the gradient to the central server. Instead of aggregating the received gradients as the traditional FL system, the central server first performs gradient clipping and noise adding. The user-level gradient of Client i is:

$$g_i = \frac{1}{|\mathcal{D}^q_i|} \nabla_\Theta \sum_{(\boldsymbol{x}, y) \in \mathcal{D}^q_i} \mathcal{L}(f_{\theta_i}(\boldsymbol{x}), y) \tag{5.5}$$

The gradient of the i-th client will be clipped as:

$$\bar{g}_i = \text{Clip}(g_i, C) \tag{5.6}$$

where Clip(a, C) denotes clip a by C. And the clipping threshold C is calculated using

Eq. (5.2) of Sec. 5.3.1. We use L to denote the group of clients in one learning iteration. The aggregated gradient is:

$$\tilde{g} = \sum_{i=0}^{L} \bar{g}_i + \mathcal{N}(0, z^2 C^2 \mathrm{I}), \tag{5.7}$$

where $\mathcal{N}(0, z^2 C^2 \mathrm{I})$ is the additive Gaussian noise. The perturbed summation $\tilde{g}$ is then used to update the meta-model:

$$\Theta \leftarrow \Theta - \eta_2 \tilde{g} \tag{5.8}$$

In summary, DP-AGR works as follows:

- Client Side: A local client splits its own dataset as support set $\mathcal{D}^s$ and query set $\mathcal{D}^q$. The local client initializes its local model as the received global model Θ and trains the model using $\mathcal{D}^s$. Finally, it computes local gradient g with $\mathcal{D}^q$ and submits g to the central server (See Algorithm 6).

- Central Server Side: The central server performs gradient clipping for a group of randomly selected clients using the current clipping threshold in each round. The clipping threshold C is updated automatically at each round using our adaptive clipping method. The central server adds noise to the summation of all clipped gradients $\sum_i \hat{g}_i + \mathcal{N}(0, z^2 C^2 \mathrm{I})$ (See Algorithm 7). We calculate the average $\bar{g}$ of the summation to update the global model Θ (we assume the data amount in every client is the same. Otherwise, we will compute the weighted average taking into account the data amount).

- The Final Output our algorithm is a differentially private model Θ. Compared to conventional FL, our global model Θ is not a ready-to-use model but an initialization parameter that has fast adaptation capability. Model consumers can customize Θ to

Algorithm 6 DP-AGR (Client Side)

Input: Current global model Θ, local data $\mathcal{D}$
Output: gradient g

Function g =Base-Model-Train(Θ, $\mathcal{D}^s, \mathcal{D}^q$):
Initialize base-model: $\theta \leftarrow \Theta$;
Split local data $\mathcal{D}^s, \mathcal{D}^q \leftarrow \mathcal{D}$
Update base-model: $\theta \leftarrow \theta - \eta_1 \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^s)$;
Gradient: $g \leftarrow \nabla_\Theta \mathcal{L}(\theta, \mathcal{D}^q)$.

Algorithm 7 Central Server Aggregation in DP-AGR and DP-AGRLR

Input: Clients set $\mathcal{T}$, noise multiplier z, user sample size L, Learning round T, (ϵ, δ)
Output: meta-model Θ

Initialize t=0;
while (ϵ, δ)-DP not exhausted and $t <= T$ do
  Randomly Sample L clients $\mathcal{T}_s \leftarrow$ sample$(\mathcal{T}, L)$;
  for $i \in \mathcal{T}_s$ do
    $g_i \leftarrow$ Base-Model-Train(Θ, $\mathcal{D}_i^s, \mathcal{D}_i^q$) #by clients
    $C \leftarrow$ adaptive-clipping$(\tilde{g}_{t-W}, ..., \tilde{g}_{t-1})$
    Clip gradient: $\hat{g}_i \leftarrow g_i * \min(1, \frac{C}{\|g_i\|})$
  end for
  $\tilde{g}_t \leftarrow \frac{1}{L}\left(\sum_i \hat{g}_i + \mathcal{N}(0, z^2 C^2 \mathrm{I})\right)$
  Update meta-model $\Theta \leftarrow \Theta - \eta_2 \tilde{g}$;
  $(\epsilon, \delta) \leftarrow$ Compute-RDP$(z, L, t, *arg)$;
  t+=1;
end while

their task with only a few local data points.

5.3.3 DP-AGR for User-level DP

To conserve the privacy budget, we enforce that each client is selected up to once. We refer to this enforcement as a one-pass meta-training process. Model accuracy is preserved even with the one-pass learning process thanks to the meta-learning framework and a large number of clients (the typical number of local devices participating in an FL system can be millions, as shown in [57]).

5.3.4 DP-AGRLR for Two-fold DP

Compared to DP-AGR, DP-AGRLR additionally protects the data privacy of clients from the honest-but-curious central server. To this end, a client does not report its true gradients but noisy ones. Note that the process at the central server side of DP-AGRLR remains the same as DP-AGR. The local client first bounds the impact of an individual record $\sum_{(\boldsymbol{x}_j,y_j)\in\mathcal{D}_i^q}$ by clipping the record-level gradient with threshold C.

$$\bar{g_j} = \text{Clip}(\nabla_\Theta \mathcal{L}(f_\theta(\boldsymbol{x}_j), y_j)), C) \tag{5.9}$$

where C is the clipping threshold. This clipping threshold is determined by the local client using our proposed adaptive clipping method. The local client then computes a Gaussian noise and adds the noise to the true gradients.

$$\tilde{g} = \sum_{j=0}^{|D_i^q|} \bar{g_j} + \mathcal{N}(0, z^2C^2\text{I}), \tag{5.10}$$

The noisy gradient $\tilde{g}$ is sent to the central server. DP-AGRLR can hide the existence of an individual record against the curious central server. Two-fold DP targets a much higher level of privacy protection, thus inevitably sacrificing more model accuracy. The algorithm is shown in Algorithm 8.

5.3.5 Privacy-Utility Trade-off

In DP-AGR we use one-pass training over clients (i.e., no client is used more than once) to boost privacy. In learning round, we sample the clients with $q = \frac{L}{N_{client}}$. The overall privacy

Algorithm 8 DP-AGRLR (Client Side)

Input: Current global model Θ, local data $\mathcal{D}$, DP parameter (ϵ_0, δ_0), C_0, z_0
Output: gradient g

Function g = Base-Model-Train(Θ, $\mathcal{D}^s, \mathcal{D}^q$):
Initialize base-model: $\theta \leftarrow \Theta$
Split local data $\mathcal{D}^s, \mathcal{D}^q \leftarrow \mathcal{D}$
$z_0 \leftarrow$ compute_noise$(\epsilon_0, \delta_0, *args)$
for $(x_i, y_i) \in \mathcal{D}^s$ do
    record- level gradient: $g_i \leftarrow \nabla_\theta \mathcal{L}(\theta, x_i)$
    clip gradient: $\hat{g}_i \leftarrow g_i * \min(1, \frac{C_0}{\|g_i\|})$
end for
$\tilde{g} \leftarrow \frac{1}{|\mathcal{D}^s|} (\sum_i \hat{g}_i + \mathcal{N}(0, (z_0 C_0)^2 \mathrm{I}))$
update base-model: $\theta \leftarrow \theta - \eta_1 \tilde{g}$;
for $(x_i, y_i) \in \mathcal{D}^q$ do
    record-level gradient: $g_i \leftarrow \nabla_\theta \mathcal{L}(\theta, x_i)$
    clip gradient: $\hat{g}_i \leftarrow g_i * \min(1, \frac{C_0}{\|g_i\|})$
end for
$g \leftarrow \frac{1}{|\mathcal{D}^q|} (\sum_i \hat{g}_i + \mathcal{N}(0, (z_0 C_0)^2 \mathrm{I}))$.

loss of DP-AGR can be represented as:

$$\alpha(\lambda) \leq \frac{q\lambda(\lambda+1)}{(1-q)z^2} + O(q^2\lambda^3/z^3). \tag{5.11}$$

where z is the noise multiplier and λ is moments number. The privacy loss $\alpha(\lambda)$ is related to q but not the learning rounds. Therefore, we can boost accuracy by adding more clients and keep the same level of privacy protection if we keep q constant. A smaller privacy loss indicates better privacy protection. The privacy parameter (ϵ, δ) is directly related to the accumulated privacy loss by equation $\delta = \min_\lambda \exp(\alpha(\lambda) - \lambda\epsilon)$. We can then use this equation to convert the moments bound $\alpha(\lambda)$ to the (ϵ, δ) guarantee.

The derivation of Eq. (5.11) is based on the moments accountant [1] that offers the state-of-the-art estimation of privacy loss of Gaussian mechanisms. For a traditional DL system, each record contributes to the training process multiple times through multiple

epochs. Different from traditional DL, a client contributes to the training system only once in meta-learning. We will highlight the impact of such a difference on privacy performance in the following.

For moments accountant [1], the λ-th moments of privacy loss of one SGD step satisfies that,

$$\alpha_{\mathcal{M}}(\lambda) \leq \frac{q^2\lambda(\lambda+1)}{(1-q)z^2} + O(\frac{q^3\lambda^3}{z^3}), \tag{5.12}$$

where mechanism $\mathcal{M} : \bar{g} = \sum_{i\in L} g_i + \mathcal{N}(0, z^2C^2I)$ represents gradients summation and perturbation, L denotes a batch, and q is the sampling probability. $O(\cdot)$ is the Bachmann–Landau notation that describes how closely a finite series approximates a given function in the case of an asymptotic expansion. The inequality holds if $z \geq 1$ and $q \leq \frac{1}{16z}$. In statistics, 'moment' of a distribution is the quantitative measure related to the shape of the distribution (e.g., the first moment is the expected value and the second central moment is the variance).

By applying composability theory of DP [1], the accumulated privacy loss after T updates are:

$$\alpha(\lambda) \leq \sum_j \alpha_{\mathcal{M}_j}(\lambda) \leq T\left(\frac{q^2\lambda(\lambda+1)}{(1-q)z^2} + O(\frac{q^3\lambda^3}{z^3})\right). \tag{5.13}$$

For DL, sampling probability is $q = \frac{L}{N_{train}}$, the number of updates is $T = N_{epoch} * \frac{N_{train}}{L}$, where N_{train} is the number of data points in the training dataset and L is the batch size. Therefore, the above equation can be rewritten as

$$\alpha(\lambda) \leq \frac{N_{epoch} * q\lambda(\lambda+1)}{(1-q)z^2} + O(N_{epoch} * \frac{q^2\lambda^3}{z^3}). \tag{5.14}$$

Different from traditional DL, for DP-AGR, $q = \frac{L}{N_{client}}$ and $T = \frac{N_{client}}{L} = \frac{1}{q}$ is the total

number of learning rounds where L denotes the number of clients for one learning round and N_{client} the total number of clients. Considering that we use one-pass training over clients in DP-AGR, the inequality in Eq. (5.13) can be rewritten as Eq. (5.11).

Eq. (5.14) and Eq. (5.11) have different implications. According to Eq. (5.14), if we increase N_{epoch}, $\alpha_{\mathcal{M}}(\lambda)$ also increases meaning privacy protection is worse. Therefore, more training iterations for DL increase privacy loss. On the contrary, according to Eq. (5.11), the privacy loss will not change with adding more clients if we keep q as constant.

5.4 Privacy Analysis

We would like to show that adaptive clipping in DP-AGR and DP-AGRLR does not result in additional privacy loss. Recall that DP-AGR works in the scenario that the central server is trusted while clients are honest-but-curious. DP-AGRLR is designed for a stronger adversary model in which both clients and the central server are honest-but-curious. For convenience, we denote the above-mentioned adversary models as Adversary Model 1 and Adversary Model 2, respectively.

For Adversary Model 1, privacy protection is applied by the central server for individual clients. The central server applies DP in the cloud to train a meta-model. To improve performance while preserving the data privacy of individual clients, our proposed adaptive clipping method (see Sec. 5.3.1) operates on the deferentially private version of gradients, and thus is a post-processing step from the privacy perspective. More concisely, the adaptive clipping threshold C_t at time step t is computed from the differentially private version of gradients $\tilde{g}_{t-W}, ...\tilde{g}_{t-1}$ as:

$$C_t = \mathrm{P}(\{\tilde{g}_{t-W}, \tilde{g}_{t-W+1}, ..., \tilde{g}_{t-1}\}, k), \tag{5.15}$$

where $\tilde{g}_t$ denotes the differentially private version of gradient at time step t, W represents the size of the sliding window, and $\mathrm{P}(S, k)$ represents the k-th percentile of a sequence S. All the gradients $\tilde{g}_{t-W}, \tilde{g}_{t-W+1}, ..., \tilde{g}_{t-1}$ are differentially private. According to the post-processing rule [31], further computation on differentially private data will not incur additional privacy loss. Therefore, the calculation of adaptive clipping threshold C_t does not lead to additional information leakage.

Similarly, adaptive clipping does not compromise data privacy in Adversary Model 2. In summary, the adaptive clipping method in DP-AGR and DP-AGRLR does not harm the target DP level. The privacy loss of DP-AGR and DP-AGRLR comes solely from the accumulated privacy loss of updating the meta-model.

5.5 Implementation and Experimental Settings

We implemented DP-FedMeta in PyTorch. We ran all our experiments on a server equipped with a 3.3 GHz Intel Core i9-9820X CPU, three GeForce RTX 2080 Ti GPUs, and Ubuntu 18.04.3 LTS. In DP-FedMeta, we trained a VGG-Net [128] using Adam optimizer with a learning rate of 0.01 for outer loop. For inner-loop training, we used the SGD optimizer with a learning rate of 0.1. The optimizer setting is adopted from the benchmark meta-learning algorithm [9]. Our code is available at https://github.com/ning-wang1/DPFedMeta.

Table 5.1: Datasets details.

Dataset	$N_c \times N_s$	Image size	Train:Val:Test
Omniglot	1623×20	28 ×28 × 1	71: 3:26
CIFAR-FS	100×600	32×32 × 3	64:16:20
Mini-ImageNet	100×600	84×84 × 3	64:16:20

For the experiment, we simulated federated meta-learning using data from three popular datasets, including Omniglot [67], CIFAR-FS [14], and Mini-ImageNet [146]. Table 5.1

provides the detail of the datasets including the number of classes, number of samples per class, image size, the splitting ratio among the meta-training set, validation set, and meta-testing set. N_c represents the number of classes and N_s denotes the number of samples per class in the corresponding dataset.

We simulated 400,000 mobile clients and 600 model consumers, assuming each client is with one learning task. The number of selected mobile clients for each learning round was set to 1,600. We followed the well-received practice in [146] to distribute the whole dataset to mobile clients. Firstly, we separated the total classes into two non-overlapping sets: one set for mobile clients and the other for model consumers. Second, each mobile client/model consumer was assigned a set of 30 labeled examples from 5 classes in the corresponding set. Different mobile clients were allowed to have overlapping data. N-way K-shot means we test a model initialization with $K * N$ data points evenly extracted from N classes. Model consumers will first initialize its model with the trained meta-model from the central server. Then it continues training the model with $K * N$ data points and then evaluate model accuracy. The average testing accuracy overall model consumers will be the final reported accuracy.

We estimate the privacy loss in the form of (ϵ, δ) at each learning round using the RDP moments accountant (https://github.com/tensorflow/privacy), and terminate the training process if the privacy budget is used up. Unless otherwise mentioned, the default settings are: the DP scenario is user-level, the few-shot learning case is 5-way 1-shot on Omniglot, clip percentile $k = 90$, noise multiplier $z = 1$, and user sample size $L = 1600$, clip window $W = 10, \Delta W = 1$. We run each experiment 3 times and obtain the average meta-testing accuracy.

5.6 Evaluations

we implement two algorithms, DP-AGR and DP-AGRLR, to accommodate different trust levels of the central server. Both DP-AGR and DP-AGRLR output a model initialization parameter Θ at the end of meta-training. In this section, we mainly answer three questions as follows:

- Q1: 'How is the performance of our adaptive clipping method compared to constant clipping and other adaptive clipping baselines?'
- Q2: 'How to achieve a good trade-off between accuracy and privacy? Specifically, how to improve model accuracy under a fixed ϵ? '
- Q3: 'Will differentially private meta-learning maintain reasonable accuracy? Specifically, will it achieve higher accuracy than other differentially private initialization methods (e.g., DP transfer learning) or random initialization?'

5.6.1 Adaptive Clipping

In this part, we will keep all other factors the same and only evaluate the performance of different clipping methods. We compare DP-AGR to the constant clipping method wildly used in [1, 7, 43, 158] which is to use the median norm of unclipped gradients as the clipping threshold. DP-AGR uses an adaptive clipping method with one configurable parameter k. As shown in Figure 5.4, the adaptive clipping method outperforms the constant clipping method. We also present an experimental comparison between our adaptive clipping method and one other adaptive clipping method (i.e., the AQC method [7]). Figure 5.4 shows that our adaptive clipping method outperforms the AQC method by accuracy, indicating that DP-AGR achieves a better trade-off between accuracy and privacy.

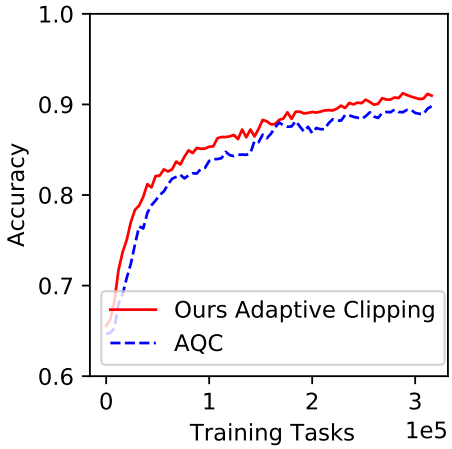


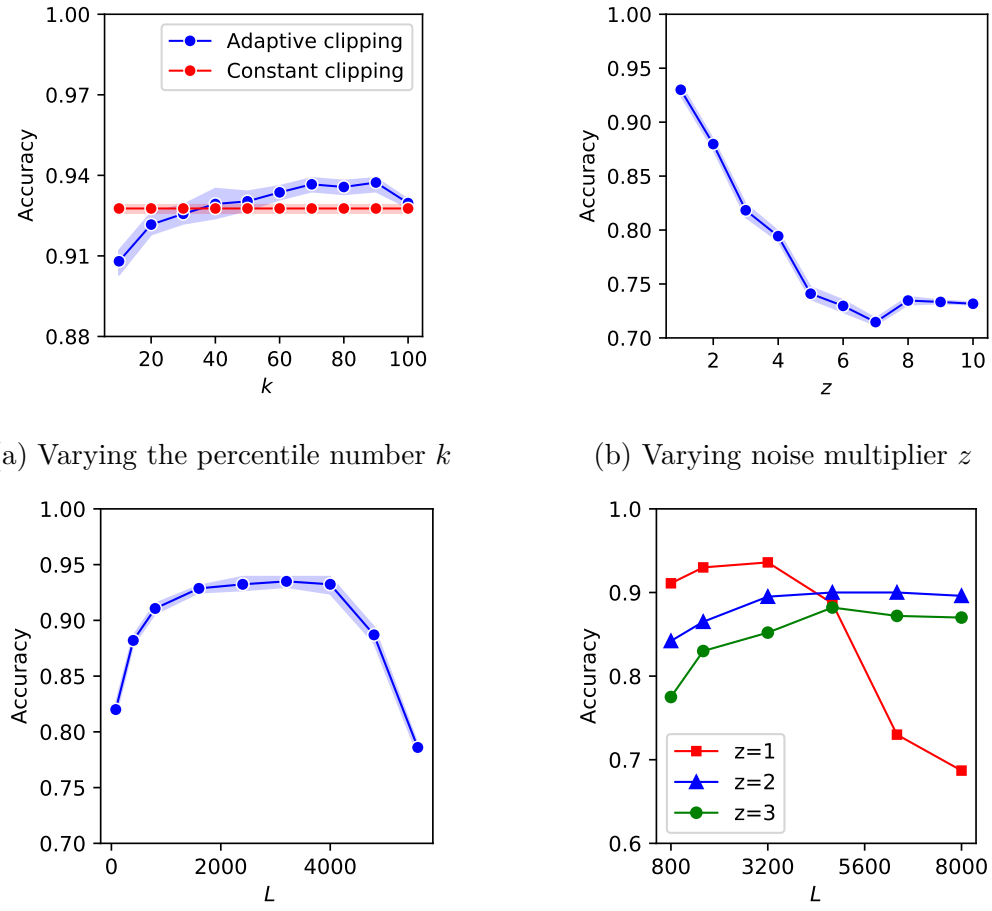
Figure 5.4: Test Accuracy. Our Adaptive Clipping Method vs. AQC [7] and constant clipping with 5-way 1-shot learning using the Ominiglot dataset.

Our adaptive clipping method has one configurable parameter k. As shown in Figure 5.5a, the adaptive clipping method changes when k varies. DP-AGR achieves the highest testing accuracy 93.9% at $k = 90$. With a proper value of $k \geq 20$ (not too small), we can expect a better model accuracy with adaptive clipping than with constant clipping. Our adaptive clipping method achieves a better trade-off between accuracy and privacy.

5.6.2 Trade-off between Accuracy and Privacy

To achieve a better trade-off between accuracy and privacy of DP-AGR, we tune other parameters, e.g., noise level and user sampling size. Specifically, we fix the privacy budget to see how to improve the model accuracy.

Noise Multiplier Figure 5.5b shows that DP-AGR achieves 93.9% accuracy at $z=1$, and the accuracy drops with the increase of z and tends to be stable at around 73%. We should start from a small z and increase z only when you use up the privacy budget before training converges.



(a) Varying the percentile number k

(b) Varying noise multiplier z

(c) Varying lot size L

(d) Varying L and z

Figure 5.5: Impacts of parameter settings on test accuracy when privacy budget is fixed.

User Sample Size We fixed the privacy budget to evaluate the accuracy with various user sample size L ranging from 100 to 5,600. As shown in Figure 5.5c, DP-AGR achieves the best accuracy when $1600 \leq L \leq 4000$. Testing accuracy drops fast when L is larger than 4,000. We looked into the experiment and found that the training process was early terminated because of depleting the DP budget when $L > 4000$.

Combination of noise multiplier and user sample size We find that the optimal L strongly depends on the noise multiplier z. We demonstrate the above finding by Figure 5.5d, in which we show the testing accuracy of different combinations of L and z. For $z = 1$, accuracy

reaches the peak at $L=1,600$ and drops quickly as $L\geq 4,800$ because of draining the privacy budget. For $z=2$, accuracy peaks at $L=3,200$ and stays stable with larger L. For $z=3$, accuracy peaks at $L=4,800$ and stays stable with larger L. Based on such observations, we conclude that the best L is proportional to z and should be tuned accordingly if z changes.

Guidelines for k, z, and L From the above results, we share our general guidelines to work with DP. First, we recommend to start from a small noise multiplier z (e.g., 1) and increase z only when you can not guarantee convergence before using up the privacy budget. Second, we recommend starting with a relatively large L especially when z is large. A good start is $L=\frac{N_{task}}{16*z}$ as $L<\frac{N_{task}}{16*z}$ by moments accountant. We can decrease L only when you can not guarantee convergence before using up the privacy budget. Compared with the non-private training, we need apply a larger learning rate since the training rounds are limited because of privacy concerns. Finally, as privacy parameter ϵ is only determined by z and L, we can adjust other parameters, such as k, to boost the model accuracy. We explore various k values in multiple datasets and find that the accuracy peaks at different k values in different datasets. However, the adaptive clipping method outperforms the fixed clipping when $k \geq 50$ across all three datasets, implying that the median is a good initial choice.

Meta-learning is a model initialization method that outputs an initialized meta-model for clients. This partly resembles transfer learning in that one model is used for initializing the training process for another model. In this experiment, we evaluate the model utility (i.e., accuracy) of DP-AGR and DP-AGRLR compared with other model initialization methods, including Random, MAML, transfer learning [28], and DP-transfer learning. The first three methods are with no privacy protection. DP-transfer learning is the transfer learning method incorporated DP.

We use the CIFAR-FS dataset (which contains 100 classes) as an example to show the comparisons. We first divide the dataset into two separate sets: Initialization training

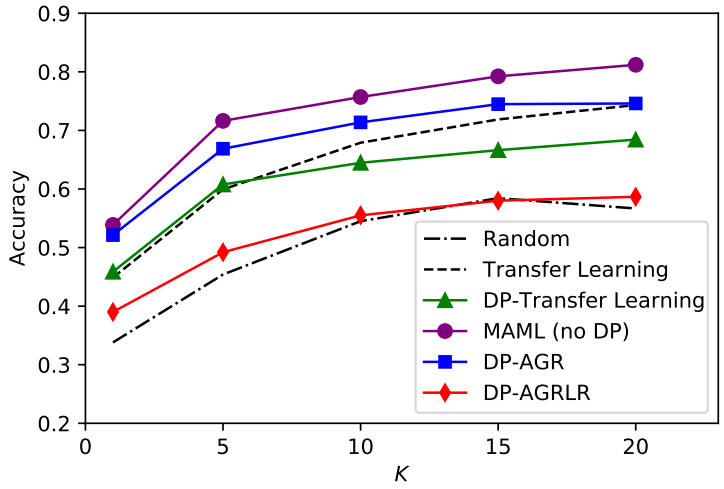


Figure 5.6: Test accuracy with different initializations using CIFAR-FS dataset. The x-axis K denotes the number of shots in the Initialization testing phase.

set (i.e., $\mathcal{D}_{train}$) consists of 80 classes and Initialization testing set ($\mathcal{D}_{test}$) consists of 20 classes. We perform both meta-training and DP meta-training to respectively obtain a meta-model Θ_{maml}, Θ_{dp-agr} and $\Theta_{dp-agrlr}$. We also conduct a transfer learning process and a DP transfer learning process [28] on the same set to achieve a trained model Θ_{trans} and $\Theta_{trans-dp}$ respectively. In the Initialization testing phase, the test set $\mathcal{D}_{test}$ is organized as tasks and each task contains a support set and a query set. We generate 600 tasks from $\mathcal{D}_{test}$. For each task, we use the trained initialization parameter to initialize its neuron network, train 30 epochs on its support set, and obtain testing accuracy on its corresponding query set. The average testing accuracy on all 600 tasks is used as the ultimate evaluation metric.

Figure 5.6 show the results of different initialization methods on CIFAR-FS. We can see that MAML achieves the best testing accuracy among all initialization methods as expected. The proposed DP-AGR outperforms all other initialization methods including transfer learning and DP-transfer learning by achieving higher accuracy. The Random initialization is, without doubt, the worst among the five methods. We also see that when K is large enough, the accuracy of transfer learning approaches that of DP-AGR.

Table 5.2: Meta-testing accuracy (%).

Dataset	N-way K-shot	Random initial	Non-private	Private Algorithm		
				DP-AGR	DP-AGRLR	GBML [70]
Omniglot	5-way 1-shot	49.2	99.4	93.9	72.4	44.6
	5-way 5-shot	61.0	99.8	96.8	89.7	75.0
CIFAR-FS	5-way 1-shot	33.8	61.0	47.1	39.0	32.2
	5-way 5-shot	45.4	78.6	58.2	49.2	48.6
Mini-ImageNet	5-way 1-shot	23.3	51.7	37.3	27.7	26.1
	5-way 5-shot	24.2	65.3	48.8	33.2	38.0

5.6.3 Comparison with Other Model Initialization Methods

Besides the transfer learning method, we also compare DP-AGR and DP-AGRLR with a state-of-the-art differentially private meta-learning solution, GBML [70]). In Table 5.2, GBML is the state-of-the-art differentially private meta-learning algorithm proposed in [70], Random initial denotes that the learning begins with the random initialization, and Non-private denotes the MAML algorithm without privacy consideration. We can see that DP-AGR outperforms GBML by achieving higher accuracy in all six learning tasks. DP-AGRLR achieves higher accuracy than GBML on both the Omniglot dataset and CIFAR-FS dataset while showing lower accuracy on the Mini-ImageNet dataset.

In Table 5.2, both DP-AGR and DP-AGRLR are with $\epsilon = 1.5$ in task level DP. The record-level DP parameter of DP-AGRLR is $\epsilon = 2.5$. The baseline GBML is with $\epsilon = 9.5$. A smaller ϵ is better for privacy protection. We can see that both DP-AGR and DP-AGRLR achieve higher accuracy than GBML on both the Omniglot dataset and the CIFAR-FS dataset. Further, DP-AGR is better in model accuracy than DP-AGRLR while DP-AGRLR provides much more strict privacy protection. Noted that GBML has a different DP notion from DP-AGR and DP-AGRLR, and its DP notion is more strict than DP-AGR but less strict than DP-AGRLR. The practical privacy leakage relies on both the value of DP parameters and the DP notion itself. For example, DP-AGRLR provides better privacy protection since it has a more strict DP notion and a smaller ϵ than GBML. On the other

Table 5.3: Per-task computation time.

Dataset	MAML	DP-AGR	DP-AGRLR
Omniglot	39.9ms	54.7ms	0.52s
CIFAR-F	68.7ms	81.3ms	1.06s
Mini-ImageNet	112.3ms	102.7ms	1.17s

hand, it is not clear whether DP-AGR provides better privacy protection than GBML since its DP notion is less strict but has a smaller ϵ than GBML. We will consider evaluating the resilience of different DP algorithms against inference attacks to test their practical privacy protection in future work.

5.6.4 Computation Time

We present the per-task computation overhead of DP-AGR and DP-AGRLR in Table 5.3. Noted that each task contains $N * K$ data records for N-way K-shot learning. We measure the training time of 5-way 1-shot learning on three different datasets. We can see that the per-task computation time of DP-AGR is only slightly longer than that of MAML. On our server, the training time of a typical DP-AGR algorithm on the Omniglot dataset with 400,000 tasks is around 6 hours. DP-AGR achieves comparable computational performance with the original non-private MAML algorithm. Compared to DP-AGR, DP-AGRLR is more time-consuming due to the need for computing per-record gradients at the local device. Although the DP-AGRLR is more time-consuming, it is scalable since the time-intensive per-record gradient computation is done at the distributed local devices of clients rather than the central server and the number of records within an individual device is small. Therefore, we believe DP-AGR and DP-AGRLR are affordable for a wide range of learning applications.

5.7 Related Work

After Song et al. [132] formulated the first differentially private stochastic gradient descent (DP-SGD) algorithm, Abadi et al. [1] proposed a powerful tool (moments accountant) for tight privacy analysis and control in DP-SGD. Pathak et al. [110] were the first to study the DP framework in a multiparty setting. This work provided a privacy-preserving protocol to compose a differentially private aggregate classifier using classifiers trained locally by separate mutually untrusted parties. Shokri et al. [126] investigated DP-SGD for distributed datasets and proposed a practical system that enables multiple parties to learn an accurate and private neural network model jointly. More recently, federated learning (FL) emerged as a promising framework since its introduction by Google researchers [62, 87] with its initial goal to learn from data stored in users' smartphones or tablets. Though the data is stored locally and never exchanged among clients, it has been shown that clients' data is still susceptible to inference attacks [98, 157]. [43, 88] explored the client-level privacy in FL and focused on balancing the trade-off between privacy and utility, assuming the central server is trusted. In order to save more privacy budget, Andrew et al. [7] proposed adaptive quantile clipping (AQC) to make an estimate of a targeted quantile of the distribution of unclipped gradient norms.

Building on the recent advances in meta-learning [39, 101, 148] and FL, a significant body of work has been devoted to federated meta-learning [23, 34, 78, 173]. Despite the compelling applications of federated meta-learning in many domains, its privacy problem remains less understood. Li et al. [70] are the first to study the privacy problem in meta-learning. It directly applies a DP-enabled meta-learning algorithm to the federated setting, which is the current state of the art and also the most relevant work to ours. In this chapter, we explored two notions of DP, including user-level DP and two-fold DP, which are different from the notions of DP studied in [70]. Compared to [70], we achieve relatively high model accuracy

with a much lower DP budget.

5.8 Conclusion

We develop DP-FedMeta, a differentially private federated meta-learning architecture that protects clients from inference-based data privacy attacks. To deal with different requirements on privacy levels pertaining to the trust on the central server, we customize two DP mechanisms for DP-FedMeta, DP-AGR and DP-AGRLR. DP-AGR protects the participating information of an individual client against curious clients and model consumers. DP-AGRLR provides two-fold privacy protection from curious clients (including model consumers) and an honest-but
-curious central server. We design an adaptive gradient clipping method and a one-pass training process to conserve the privacy budget. Our adaptive gradient clipping shows superior performance over both constant gradient clipping and another adaptive one. Extensive evaluation of multiple datasets demonstrates that our DP-AGR outperforms the state-of-the-art differentially private federated meta-learning solution by achieving a much lower privacy budget without sacrificing additional accuracy. Further, we provide useful insights and heuristic guidelines on how to set various parameters, such as gradient clipping threshold, noise multiplier, and lot size, which are applicable when incorporating DP in different types of deep learning problems.

Chapter 6

Conclusions and Future Work

6.1 Summary and conclusion remarks

In this dissertation, we studied the trustworthiness of machine learning systems in adversarial environments. We have proposed to protect machine learning systems from data manipulations at different lifetime cycles, covering training-phase poisoning attack and defense, testing-phase adversarial example generation, and detection. We are dedicated to figuring out unexplored vulnerabilities in machine learning systems, developing defense mechanisms accordingly, and improving the effectiveness of the existing defense. Further, we have studied the data privacy leakage problems in machine learning systems with the aim of a good trade-off between privacy and accuracy. The proposed privacy-preserving machine learning system has maintained relatively high model utility while addressing data providers' privacy concerns. We also improved the robustness of machine learning-based anomaly detection. The detail of the research accomplishment is listed below.

In Chapter 2, we tackle the model poisoning attack in federated learning systems using latent space representations. We analyze the limitation of the current Byzantine-resilient aggregation rules (BRARs). We proposed FLARE that explores the penultimate layer representation (PLR) as a new feature to detect malicious models. FLARE is demonstrated to defend successfully against various model poisoning methods. While FLARE achieves great detection performance for independent and identically distributed (IID) data, it suffers

from performance degradation for non-iid data scenarios, which is a common shortcoming of most of the current detection methods. More research on the non-iid scenario needs to be done.

In Chapter 3, we address the adversarial examples in machine learning by detecting these maliciously crafted perturbations. We exploit unique features we observe while trying to categorize AE attacks from the viewpoint of machine learning models and data manifolds. We further analyze the common inherent properties of AEs and introduce a novel AE detection method MANDA by combining two building blocks (i.e., Manifold and DB). MANDA generalizes well to different types of data and various attacks and significantly improved the detection rate against various powerful AE attacks. Manifold learning is a key component in the proposed detection mechanism, and it faces great challenges when the input data dimensionality is relatively high. It is critical to improve manifold learning effectiveness for high-dimensional data.

In Chapter 4, we improve the robustness of anomaly detection methods when facing up with training data variance. We propose a novel method for building the "norm" in an anomaly-based IDS by learning new representations for network traffic based on contrastive learning. With the proposed new method, the learned representations of benign inputs lie only in a small cluster, enabling FeCo to extract a stable template for benign inputs. Extensive evaluation results show that representation learning in FeCo significantly boosts its detection accuracy compared to previous works. The two-step feature selection scheme in FeCo not only reduces the risk of overfitting and also reduces computation complexity as a result of smaller input dimensionality, making FeCo more suitable for resource-constrained IoT devices.

In Chapter 5, we propose DP-FedMeta to protect data privacy in federated learning while maintaining the model accuracy by applying differential privacy. The framework preserves

user privacy with rigorous privacy guarantees and enables heterogeneous users for fast model adaptation. DP-FedMeta caters to two practical trust levels on the central server by featuring two variants of DP mechanisms. The proposed adaptive gradient clipping method in DP-FedMeta maximally preserves model accuracy while guaranteeing a target level of privacy.

6.2 Future research direction

The objective of this dissertation is to defend against adversarial attacks, protect data privacy, and improve the robustness of anomaly detection methods. I hope the study introduced in this dissertation will contribute to a more secure and trustworthy machine-learning ecosystem. Meanwhile, we acknowledge that there are still various security issues in the machine learning system to analyze and resolve. I list a few as future research directions.

As FL has emerged as a popular distributed AI framework for intelligent IoT applications, such as smart healthcare and transportation, the non-IID data property in IoT systems has posed great challenges for malicious client detection. Due to the high variance among benign clients originating from non-IID data, benign models are indistinguishable from poisoned/malicious models. Most existing model poisoning detection methods dedicated to IID data scenarios will either make high false positives or high false negatives, rendering the defense useless. To improve model poisoning detection accuracy in the non-IID scenario, I plan to design a data-distribution-aware detection mechanism in future work.

The current work focuses on supervised learning while lacking a discussion on reinforcement learning systems. Reinforcement learning (RL) has been adopted as the automation design in CPS, e.g., intelligent transportation and smart grid, because of its capability to learn, adapt, and work independently in a dynamic environment. Adversarial example attacks

discovered in image classification have been found effective in reinforcement learning systems that take image observations as inputs. However, its effectiveness for autonomous systems with other types of observations (e.g., the radio-resource allocation with RF signal as observation) is under-investigated. Securing automation systems from adversarial attacks is more challenging due to the open environment. I plan to investigate the effectiveness of adversarial attacks in such domains and propose corresponding defenses.

Besides data privacy discussed in Chapter 5, model privacy is attracting more attention than ever as machine learning models (e.g., ChatGPT) that consume enormous training data and computation power are deemed intellectual property. ML-as-a-service systems have been deployed with publicly accessible query interfaces. A novel threat against such confidential ML models emerges where attackers can steal the model via a reasonable number of queries. The confidentiality of an ML model itself is as important as the training data privacy. I plan to design detection methods to protect ML models from stealing in the future.

To build trustworthiness over machine learning systems, besides dealing with adversarial attacks which is the focus of this dissertation, another direction is to improve the interpretation of the predictions. Lacking explainability reduces the confidence of a practitioner to use deep neural networks, especially for safety-critical applications, e.g., malware detection. Explainable ML is a promising direction and complements the current work on trustworthy ML.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 16), pages 308–318. ACM, 2016.

[2] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. IEEE Communications Surveys & Tutorials, 22(3): 1646–1685, 2020.

[3] OY Al-Jarrah, Amna Siddiqui, M Elsalamouny, Paul D Yoo, Sami Muhaidat, and Kwangjo Kim. Machine-learning-based feature selection techniques for large-scale network intrusion detection. In 2014 IEEE 34th international conference on distributed computing systems workshops (ICDCSW), pages 177–181. IEEE, 2014.

[4] Riyad Alshammari and A Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, ip addresses and payload inspection? Computer networks, 55 (6):1326–1350, 2011.

[5] João P Amaral, Luís M Oliveira, Joel JPC Rodrigues, Guangjie Han, and Lei Shu. Policy and network-based intrusion detection system for ipv6-enabled wireless sensor networks. In 2014 IEEE international conference on communications (ICC), pages 1796–1801. IEEE, 2014.

[6] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. Baffle: Backdoor detection via feedback-based federated learning. In 2021 IEEE 41st

International Conference on Distributed Computing Systems (ICDCS), pages 852–863. IEEE, 2021.

[7] Galen Andrew, Om Thakkar, H Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. arXiv preprint arXiv:1905.03871, 2019.

[8] Manos Antonakakis, Tim April, Michael Bailey, et al. Understanding the mirai botnet. In 26th {USENIX} security symposium ({USENIX} Security 17), pages 1093–1110, 2017.

[9] Antreas Antoniou et al. How to train your MAML. In International Conference on Learning Representations (ICLR 19), 2019.

[10] Giovanni Apruzzese, Michele Colajanni, and Mirco Marchetti. Evaluating the effectiveness of adversarial attacks against botnet detectors. In 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), pages 1–8. IEEE, 2019.

[11] Sana Awan, Bo Luo, and Fengjun Li. Contra: Defending against poisoning attacks in federated learning. In European Symposium on Research in Computer Security, pages 455–475. Springer, 2021.

[12] 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. PMLR, 2020.

[13] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In Advances in neural information processing systems (NeurIPS), pages 585–591, 2002.

[14] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In International Conference on Learning Representations (ICLR 19), 2019.

[15] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In International Conference on Machine Learning, pages 634–643. PMLR, 2019.

[16] 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.

[17] James D Burton. Cisco security professional's guide to secure intrusion detection systems. Syngress Publ., 2003.

[18] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. Network and Distributed Systems Security Symposium NDSS, 2021.

[19] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Provably secure federated learning against malicious clients. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 6885–6893, 2021.

[20] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In Proc.10th ACM Workshop on Artificial Intelligence and Security, pages 3–14, 2017.

[21] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural

networks. In 2017 IEEE Symposium on Security and Privacy (SP 17), pages 39–57. IEEE, 2017.

[22] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, and Parvez Faruki. Network intrusion detection for iot security based on learning techniques. IEEE Communications Surveys & Tutorials, 21(3):2671–2701, 2019.

[23] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. arXiv preprint arXiv:1802.07876, 2018.

[24] Yudong Chen, Lili Su, and Jiaming Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 1(2):1–25, 2017.

[25] Eric Conrad, Seth Misenar, et al. Eleventh Hour CISSP®: Study Guide. Syngress, 2016.

[26] Adrien Cosson, Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Patrick McDaniel, and A Selcuk Uluagac. Sentinel: A robust intrusion detection system for iot networks using kernel-level system information. In Proceedings of the International Conference on Internet-of-Things Design and Implementation, pages 53–66, 2021.

[27] L Dhanabal and SP Shantharajah. A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. Int. Journal of Advanced Research in Computer and Communication Engineering, 4(6):446–452, 2015.

[28] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In International conference on machine learning (ICML 14), pages 647–655, 2014.

[29] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. arXiv preprint arXiv:1605.09782, 2016.

[30] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1285–1298, 2017.

[31] Cynthia Dwork et al. The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science, 9(3–4):211–407, 2014.

[32] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Louis Alexandre Rouault. The hidden vulnerability of distributed learning in byzantium. In International Conference on Machine Learning, 2018.

[33] Mojtaba Eskandari, Zaffar Haider Janjua, Massimo Vecchio, and Fabio Antonelli. Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. IEEE Internet of Things Journal, 7(8):6882–6897, 2020.

[34] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In Advances in Neural Information Processing Systems(NeurPIS 20), pages 3557–3568, 2020.

[35] Yulin Fan, Yang Li, Mengqi Zhan, Huajun Cui, and Yan Zhang. Iotdefender: A federated transfer learning intrusion detection framework for 5g iot. In 2020 IEEE 14th International Conference on Big Data Science and Engineering (BigDataSE), pages 88–95. IEEE, 2020.

[36] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning

attacks to byzantine-robust federated learning. In 29th {USENIX} Security Symposium ({USENIX} Security 20), pages 1605–1622, 2020.

[37] Reuben Feinman, Ryan R Curtin, et al. Detecting adversarial samples from artifacts. arXiv preprint arXiv:1703.00410, 2017.

[38] Aidin Ferdowsi and Walid Saad. Generative adversarial networks for distributed intrusion detection in the internet of things. In 2019 IEEE Global Communications Conference (GLOBECOM), pages 1–6. IEEE, 2019.

[39] Chelsea Finn et al. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the 34th International Conference on Machine Learning (ICML 17), pages 1126–1135, 2017.

[40] Ramon Fontes, Samira Afzal, Samuel Brito, Mateus Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. 11 2015. doi: 10.1109/CNSM.2015.7367387.

[41] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. arXiv preprint arXiv:1808.04866, 2018.

[42] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, et al. Anomaly-based network intrusion detection: Techniques, systems and challenges. computers & security, 28(1-2):18–28, 2009.

[43] Robin C Geyer et al. Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557, 2017.

[44] Zhitao Gong, Wenlu Wang, et al. Adversarial and clean data are not twins. arXiv preprint arXiv:1704.04960, 2017.

[45] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.

[46] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. The Journal of Machine Learning Research, 13(1):723–773, 2012.

[47] Kathrin Grosse, Praveen Manoharan, et al. On the (statistical) detection of adversarial examples. arXiv preprint arXiv:1702.06280, 2017.

[48] Rachid Guerraoui, Sébastien Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In International Conference on Machine Learning, pages 3521–3530. PMLR, 2018.

[49] Vajiheh Hajisalem and Shahram Babaie. A hybrid intrusion detection system based on abc-afs algorithm for misuse and anomaly detection. Computer Networks, 136:37–50, 2018.

[50] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604, 2018.

[51] Dan Hendrycks and Kevin Gimpel. Early methods for detecting adversarial images. In Int. Conf. on Learning Representations (ICLR), 2017.

[52] Pengfei Hu, Hongxing Li, et al. Dynamic defense strategy against advanced persistent threat with insiders. In 2015 IEEE Conference on Computer Communications (INFOCOM), pages 747–755. IEEE, 2015.

[53] Shengyuan Hu, Tao Yu, et al. A new defense against adversarial images: Turning a weakness into a strength. In Advances in Neural Information Processing Systems (NeurIPS), pages 1635–1646, 2019.

[54] Peter J Huber. Robust statistics, volume 523. John Wiley & Sons, 2004.

[55] Melvin Johnson, Mike Schuster, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. Trans. of the Assoc. for Computational Linguistics, 5:339–351, 2017.

[56] VVRPV Jyothsna, Rama Prasad, and K Munivara Prasad. A review of anomaly based intrusion detection systems. International Journal of Computer Applications, 28(7): 26–35, 2011.

[57] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. Foundations and Trends® in Machine Learning, 14(1), 2021. ISSN 1935-8237. doi: 10.1561/2200000083. URL http://dx.doi.org/10.1561/2200000083.

[58] Hyunjae Kang, Dong Hyun Ahn, Gyung Min Lee, Jeong Do Yoo, Kyung Ho Park, and Huy Kang Kim. Iot network intrusion dataset, 2019. URL https://dx.doi.org/10.21227/q70p-q449.

[59] Jakob Nikolas Kather, Cleo-Aron Weis, Francesco Bianconi, Susanne M Melchers, Lothar R Schad, Timo Gaiser, Alexander Marx, and Frank Gerrit Zöllner. Multi-class texture analysis in colorectal cancer histology. Scientific reports, 6:27988, 2016.

[60] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Stefanos Gritzalis.

Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset. IEEE Communications Surveys & Tutorials, 18(1):184–208, 2016.

[61] Constantinos Kolias, Georgios Kambourakis, and et al. Ddos in the iot: Mirai and other botnets. Computer, 50(7):80–84, 2017.

[62] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527, 2016.

[63] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492, 2016.

[64] Okan Kopuklu, Jiapeng Zheng, Hang Xu, and Gerhard Rigoll. Driver anomaly detection: A dataset and contrastive learning approach. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 91–100, 2021.

[65] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[66] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533, 2016.

[67] Brenden M Lake et al. Human-level concept learning through probabilistic program induction. Science, 350(6266):1332–1338, 2015.

[68] 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.

[69] Chulhee Lee and David A Landgrebe. Feature extraction based on decision boundaries. IEEE Trans. on Pattern Analysis and Machine Intelligence, 15(4):388–400, 1993.

[70] Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. Differentially private meta-learning. In International Conference on Learning Representations, 2020. URL https://openreview.net/forum?id=rJgqMRVYvr.

[71] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. arXiv preprint arXiv:2002.00211, 2020.

[72] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine, 37(3): 50–60, 2020.

[73] Xin Li and Fuxin Li. Adversarial examples detection in deep networks with convolutional filter statistics. In Proc. IEEE Int. Conf. on Computer Vision (ICCV), pages 5764–5772, 2017.

[74] Bin Liang, Hongcheng Li, et al. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. IEEE Trans. on Dependable and Secure Computing, 2018.

[75] Hung-Jen Liao, Chun-Hung Richard Lin, et al. Intrusion detection system: A comprehensive review. Journal of Network and Computer Applications, 36(1):16–24, 2013.

[76] Shijun Liao and Yinlong Zhao. On the method of directly defining inverse mapping for nonlinear differential equations. Numerical Algorithms, 72(4):989–1020, 2016.

[77] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey

on internet of things: Architecture, enabling technologies, security and privacy, and applications. IEEE internet of things journal, 4(5):1125–1142, 2017.

[78] Sen Lin et al. A collaborative learning framework via federated meta-learning. In 2020 International Conference on Distributed Computing Systems (ICDCS), 2020.

[79] Tong Lin and Hongbin Zha. Riemannian manifold learning. IEEE Trans. on Pattern Analysis and Machine Intelligence, 30(5):796–809, 2008.

[80] Zilong Lin, Yong Shi, et al. Idsgan: Generative adversarial networks for attack generation against intrusion detection. arXiv preprint arXiv:1809.02077, 2018.

[81] Yi Liu, Sahil Garg, Jiangtian Nie, Yang Zhang, Zehui Xiong, Jiawen Kang, and M Shamim Hossain. Deep anomaly detection for time-series data in industrial iot: a communication-efficient on-device federated learning approach. IEEE Internet of Things Journal, 8(8):6348–6358, 2020.

[82] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot. Sensors, 17(9):1967, 2017.

[83] 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.

[84] Yunlong Mao, Xinyu Yuan, Xinyang Zhao, and Sheng Zhong. Romoa: Robust model aggregation for the resistance of federated learning to model poisoning attacks. In European Symposium on Research in Computer Security, pages 476–496. Springer, 2021.

[85] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and N Asokan. Audi: Toward autonomous iot device-type identification using periodic communication. IEEE Journal on Selected Areas in Communications, 37(6):1402–1412, 2019.

[86] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. ACM Trans. on Information and System Security (TISSEC), 3(4):262–294, 2000.

[87] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics (AISTATS 17), pages 1273–1282, 2017.

[88] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In International Conference on Learning Representations (ICLR 18), 2018.

[89] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Computing, 17(3):12–22, 2018.

[90] Jan Hendrik Metzen, Tim Genewein, et al. On detecting adversarial perturbations. In Int. Conf. on Learning Representations (ICLR), 2017.

[91] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089, 2018.

[92] Viraaji Mothukuri, Prachi Khare, Reza M Parizi, Seyedamin Pouriyeh, Ali

Dehghantanha, and Gautam Srivastava. Federated learning-based anomaly detection for iot security attacks. IEEE Internet of Things Journal, 2021.

[93] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In 2015 military communications and information systems conference (MilCIS), pages 1–6. IEEE, 2015.

[94] Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. IEEE Internet of Things Journal, 6(3):4815–4830, 2018.

[95] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In Advances in Neural Information Processing Systems, pages 4694–4703, 2019.

[96] Aamir Mustafa, Salman H Khan, et al. Image super-resolution as a defense against adversarial attacks. IEEE Trans. on Image Processing, 29:1711–1724, 2019.

[97] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating dnn-based traffic analysis systems in real-time with blind adversarial perturbations. In 30th {USENIX} Security Symposium ({USENIX} Security 21), 2021.

[98] Milad Nasr et al. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In 2019 IEEE Symposium on Security and Privacy (SP 19), pages 739–753. IEEE, 2019.

[99] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, et al. Dïot: A federated self-learning anomaly detection system for iot. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pages 756–767. IEEE, 2019.

[100] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning attacks on federated learning-based iot intrusion detection system. In Proc. Workshop Decentralized IoT Syst. Secur.(DISS), pages 1–7, 2020.

[101] Alex Nichol et al. Reptile: a scalable metalearning algorithm. arXiv preprint arXiv:1803.02999, 2:2, 2018.

[102] Doohwan Oh, Deokho Kim, and Won Woo Ro. A malicious pattern detection engine for embedded security systems in the internet of things. Sensors, 14(12):24188–24211, 2014.

[103] Babatunji Omoniwa, Riaz Hussain, Muhammad Awais Javed, Safdar Hussain Bouk, and Shahzad A Malik. Fog/edge computing-based iot (feciot): Architecture, applications, and research issues. IEEE Internet of Things Journal, 6(3):4118–4149, 2018.

[104] Hamed Haddad Pajouh, GholamHossein Dastghaibyfard, and Sattar Hashemi. Two-tier network anomaly detection model: a machine learning approach. Journal of Intelligent Information Systems, 48(1):61–74, 2017.

[105] Hamed Haddad Pajouh, Reza Javidan, Raouf Khayami, Ali Dehghantanha, and Kim-Kwang Raymond Choo. A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks. IEEE Transactions on Emerging Topics in Computing, 7(02):314–323, 2019.

[106] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277, 2016.

[107] 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. arXiv preprint arXiv:1602.02697, 1(2):3, 2016.

[108] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages 372–387. IEEE, 2016.

[109] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32:8026–8037, 2019.

[110] Manas Pathak et al. Multiparty differential privacy via aggregation of locally trained classifiers. In Advances in Neural Information Processing Systems(NeurPIS 10), pages 1876–1884, 2010.

[111] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[112] Xiao Peng, Weiqing Huang, and Zhixin Shi. Adversarial attack against dos intrusion detection: An improved boundary-based method. In 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), pages 1288–1295. IEEE, 2019.

[113] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis. In 2020 IEEE European Symposium on Security and Privacy (EuroS&P), pages 474–489. IEEE, 2020.

[114] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro.

Intriguing properties of adversarial ml attacks in the problem space. In 2020 IEEE Symposium on Security and Privacy (SP), pages 1332–1349. IEEE, 2020.

[115] Shailendra Rathore and Jong Hyuk Park. Semi-supervised learning based distributed attack detection framework for iot. Applied Soft Computing, 72:79–89, 2018.

[116] Nagarathna Ravi and S Mercy Shalinie. Semisupervised-learning-based security to detect and mitigate intrusions in iot network. IEEE Internet of Things Journal, 7(11): 11041–11052, 2020.

[117] Shahid Raza, Linus Wallgren, and Thiemo Voigt. Svelte: Real-time intrusion detection in the internet of things. Ad hoc networks, 11(8):2661–2674, 2013.

[118] Mengye Ren, Andrei Pokrovsky, et al. Sbnet: Sparse blocks network for fast inference. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition(CVPR), pages 8711–8720, 2018.

[119] Shahadate Rezvy, Yuan Luo, Miltos Petridis, Aboubaker Lasebae, and Tahmina Zebin. An efficient deep learning model for intrusion classification and prediction in 5g and iot networks. In 2019 53rd Annual Conference on information sciences and systems (CISS), pages 1–6. IEEE, 2019.

[120] Maria Rigaki and Sebastian Garcia. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In 2018 IEEE Security and Privacy Workshops (SPW), pages 70–75. IEEE, 2018.

[121] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. science, 290(5500):2323–2326, 2000.

[122] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust

and communication-efficient federated learning from non-iid data. IEEE transactions on neural networks and learning systems, 31(9):3400–3413, 2019.

[123] Soroosh Shafieezadeh-Abadeh, Daniel Kuhn, and Peyman Mohajerin Esfahani. Regularization via mass transportation. Journal of Machine Learning Research, 20 (103):1–68, 2019.

[124] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In ICISSP, pages 108–116, 2018.

[125] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending against poisoning attacks in collaborative deep learning systems. In Proceedings of the 32nd Annual Conference on Computer Security Applications, pages 508–519, 2016.

[126] Reza Shokri et al. Privacy-preserving deep learning. In Proceedings of the 22nd ACM SIGSAC conference on computer and communications security (CCS 15), pages 1310–1321. ACM, 2015.

[127] Dule Shu, Nandi O Leslie, Charles A Kamhoua, and Conrad S Tucker. Generative adversarial attacks against intrusion detection systems using active learning. In Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning, pages 1–6, 2020.

[128] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[129] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In International Conference on Learning Representations, 2015.

[130] Jinhyun So, Başak Güler, and A Salman Avestimehr. Byzantine-resilient secure federated learning. IEEE Journal on Selected Areas in Communications, 2020.

[131] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy, pages 305–316. IEEE, 2010.

[132] Shuang Song et al. Stochastic gradient descent with differentially private updates. In 2013 IEEE Global Conference on Signal and Information Processing (GlobalIPS 13), pages 245–248. IEEE, 2013.

[133] Tianyi Song, Ruinian Li, Bo Mei, Jiguo Yu, Xiaoshuang Xing, and Xiuzhen Cheng. A privacy preserving communication protocol for iot applications in smart homes. IEEE Internet of Things Journal, 4(6):1844–1852, 2017.

[134] Yang Song, Taesup Kim, et al. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In Int. Conf. on Learning Representations (ICLR), 2018.

[135] Charles Spearman. The proof and measurement of association between two things. 1961.

[136] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? arXiv preprint arXiv:1911.07963, 2019.

[137] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[138] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis

of the kdd cup 99 data set. In IEEE symp. on computational intelligence for security and defense applications (CISDA), pages 1–6. IEEE, 2009.

[139] Joshua B Tenenbaum, Vin De Silva, et al. A global geometric framework for nonlinear dimensionality reduction. science, 290(5500):2319–2323, 2000.

[140] Shixin Tian, Guolei Yang, et al. Detecting adversarial examples through image transformation. In 32ed AAAI Conf. on Artificial Intelligence, 2018.

[141] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16, pages 776–794. Springer, 2020.

[142] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? arXiv preprint arXiv:2005.10243, 2020.

[143] Yunzhe Tian, Yingdi Wang, Endong Tong, Wenjia Niu, Liang Chang, Qi Alfred Chen, Gang Li, and Jiqiang Liu. Exploring data correlation between feature pairs for generating constraint-based adversarial examples. In 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), pages 430–437. IEEE, 2020.

[144] Florian Tramèr, Alexey Kurakin, et al. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204, 2017.

[145] Nguyen H Tran, Wei Bao, Albert Zomaya, Minh NH Nguyen, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pages 1387–1395. IEEE, 2019.

[146] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In Advances in neural information processing systems (NeurIPS 16), pages 3630–3638, 2016.

[147] Bing Wang, Yao Zheng, Wenjing Lou, and Y Thomas Hou. Ddos attack protection in the era of cloud computing and software-defined networking. Computer Networks, 81: 308–319, 2015.

[148] Lingxiao Wang, Qi Cai, Zhuoran Yang, and Zhaoran Wang. On the global optimality of model-agnostic meta-learning. In International Conference on Machine Learning, pages 9837–9846. PMLR, 2020.

[149] Ning Wang, Yimin Chen, Yang Hu, Wenjing Lou, and Y Thomas Hou. Manda: On adversarial example detection for network intrusion detection system. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications, pages 1–10. IEEE, 2021.

[150] Ning Wang, Yimin Chen, Yang Hu, Wenjing Lou, and Y Thomas Hou. Feco: Boosting intrusion detection capability in iot networks via contrastive learning. In IEEE INFOCOM 2022-IEEE Conference on Computer Communications. IEEE, 2022.

[151] Ning Wang, Yimin Chen, Yang Xiao, Yang Hu, Wenjing Lou, and Thomas Hou. Manda: On adversarial example detection for network intrusion detection system. IEEE Transactions on Dependable and Secure Computing, 2022.

[152] 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.

[153] Ning Wang, Yang Xiao, Yimin Chen, Ning Zhang, Wenjing Lou, and Y Thomas Hou. Squeezing more utility via adaptive clipping on differentially private gradients in federated meta-learning. In Proceedings of the 38th Annual Computer Security Applications Conference, pages 647–657, 2022.

[154] Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning your identity and disease from research papers: information leaks in genome wide association study. In Proceedings of the 16th ACM conference on Computer and communications security, pages 534–544. ACM, 2009.

[155] Ruiping Wang and Xilin Chen. Manifold discriminant analysis. In IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pages 429–436. IEEE, 2009.

[156] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. IEEE Journal on Selected Areas in Communications, 37(6):1205–1221, 2019.

[157] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In IEEE Conf. on Computer Communications (INFOCOM), pages 2512–2520. IEEE, 2019.

[158] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. IEEE Trans. on Information Forensics and Security, 15:3454–3469, 2020.

[159] Di Wu, Binxing Fang, Junnan Wang, Qixu Liu, and Xiang Cui. Evading machine

learning botnet detection models via deep reinforcement learning. In 2019 IEEE Int. Conf. on Communications (ICC), pages 1–6. IEEE, 2019.

[160] Nan Wu, Farhad Farokhi, David Smith, and Mohamed Ali Kaafar. The value of collaboration in convex machine learning with differential privacy. In 2020 IEEE Symposium on Security and Privacy (SP), pages 304–317. IEEE, 2020.

[161] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3733–3742, 2018.

[162] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.

[163] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In International Conference on Learning Representations, 2020.

[164] Wayne Xiong, Lingfeng Wu, et al. The microsoft 2017 conversational speech recognition system. In IEEE int. conf. on acoustics, speech and signal processing (ICASSP), pages 5934–5938. IEEE, 2018.

[165] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In Proceedings of the 2016 network and distributed systems symposium, volume 10, 2016.

[166] Weilin Xu, David Evans, et al. Feature squeezing: Detecting adversarial examples in deep neural networks. Proc. Network and Distributed System Security Symposium, 2018. doi: 10.14722/ndss.2018.23198. URL http://dx.doi.org/10.14722/ndss.2018.23198.

[167] Qiben Yan, Ming Li, Feng Chen, Tingting Jiang, Wenjing Lou, Y Thomas Hou, and Chang-Tien Lu. Specmonitor: Toward efficient passive traffic monitoring for cognitive radio networks. IEEE Transactions on Wireless Communications, 13(10):5893–5905, 2014.

[168] Qiben Yan, Yao Zheng, Tingting Jiang, Wenjing Lou, and Y Thomas Hou. Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis. In 2015 IEEE Conference on Computer Communications (INFOCOM), pages 316–324. IEEE, 2015.

[169] Yanqing Yang, Kangfeng Zheng, Chunhua Wu, and Yixian Yang. Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. Sensors, 19(11):2528, 2019.

[170] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In International Conference on Machine Learning, pages 5650–5659. PMLR, 2018.

[171] Lingjing Yu, Bo Luo, Jun Ma, Zhaoyu Zhou, and Qingyun Liu. You are what you broadcast: Identification of mobile and iot devices from (public) wifi. In 29th {USENIX} Security Symposium ({USENIX} Security 20), pages 55–72, 2020.

[172] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In Proceedings of the 2014 ACM conference on SIGCOMM, pages 371–372, 2014.

[173] Sheng Yue, Ju Ren, Jiang Xin, Sen Lin, and Junshan Zhang. Inexact-admm based federated meta-learning for fast and continual edge learning. In Proceedings of the 22ed International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, pages 91–100, 2021.

[174] Jingwen Zhang, Jiale Zhang, Junjun Chen, and Shui Yu. Gan enhanced membership inference: A passive local attack in federated learning. In ICC 2020-2020 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2020.

[175] Yuntong Zhang, Jingye Xu, Zhiwei Wang, Rong Geng, Kim-Kwang Raymond Choo, Jesús Arturo Pérez-Díaz, and Dakai Zhu. Efficient and intelligent attack detection in software defined iot networks. In 2020 IEEE International Conference on Embedded Software and Systems (ICESS), pages 1–9. IEEE, 2020.

[176] Ying Zhao, Junjun Chen, Jiale Zhang, Di Wu, Jian Teng, and Shui Yu. Pdgan: A novel poisoning defense method in federated learning using generative adversarial network. In International Conference on Algorithms and Architectures for Parallel Processing, pages 595–609. Springer, 2019.

[177] Zhihao Zheng and Pengyu Hong. Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks. In Advances in Neural Information Processing Systems (NeurIPS), pages 7913–7922, 2018.

[178] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In Advances in neural information processing systems, pages 321–328, 2004.