PROTECTING MODELS AND DATA IN FEDERATED AND CENTRALIZED LEARNING

Najeeb Moharram Salim Jebreel

UNIVERSITAT
ROVIRA i VIRGILI

Protecting Models and Data in Federated and Centralized Learning

---

Author:
Najeeb JEBREEL

DOCTORAL THESIS
2023

Najeeb JEBREEL

Protecting Models and Data in Federated and Centralized Learning

DOCTORAL THESIS

Supervisors:
Dr. Josep DOMINGO-FERRER
Dr. David SÁNCHEZ

Department of Computer Engineering and Mathematics

UNIVERSITAT ROVIRA i VIRGILI

April, 2023

UNIVERSITAT ROVIRA i VIRGILI

WE STATE that the present study, entitled “Protecting Models and Data in Federated and Centralized Learning”, presented by Najeeb Moharram Jebreel for the award of the degree of Doctor, has been carried out under our supervision at the Department of Computer Engineering and Mathematics of this university.

Tarragona, March 27 2023

Doctoral Thesis Supervisor/s

DOMINGO FERRER JOSEP - 33890313C
Digitally signed by DOMINGO FERRER JOSEP - 33890313C
Date: 2023.03.28 14:47:36 +02'00'

Josep Domingo-Ferrer

David Sánchez Ruenes - DNI 47763566H (TCAT)
Firmado digitalmente por David Sánchez Ruenes - DNI 47763566H (TCAT)
Fecha: 2023.03.28 15:59:23 +02'00'

David Sánchez Ruenes

Abstract

Federated learning (FL) enables learning a global machine learning (ML) model from data distributed among a set of participating peers, under the coordination of a server that acts as a model manager. This makes it possible: a) to reduce the training computation cost at the server side, because model updates are locally computed by participating peers on their respective data, b) to train more accurate models due to learning from rich, joint training data, and c) to improve privacy by not sharing the peers' local private data with others. Despite these benefits, FL is vulnerable to various security and privacy attacks due to its distributed nature. Poisoned updates sent by malicious peers may compromise the global model's availability or integrity, whereas good updates sent by honest peers may reveal their private local information. As FL becomes more prevalent in real-world applications, safeguarding its models against these attacks becomes crucial. Existing defenses against poisoning attacks impose high computation overhead on the server and/or are limited by assumptions on the peers' data distribution or are ill-suited to high-dimensional models. On the other side, countering privacy attacks via update distortion damages accuracy, whereas doing so via update aggregation damages security because it does not allow the server to filter out individual poisoned updates.

Additionally, FL shares some security vulnerabilities with centralized learning. On the one hand, both paradigms are vulnerable to backdoor attacks (BA), which are difficult to detect despite their dangerous security effects. On the other hand, both in centralized and federated learning, valuable models obtained after intensive training might be stolen or misused, thereby affecting their owners' intellectual property.

Motivated by this, in this thesis we propose three defenses against poisoning attacks and one comprehensive defense against both poisoning and privacy attacks in the FL paradigm. The defenses are: i) a

method which analyzes the biases of the last layer to neutralize Byzantine poisoning attacks efficiently; ii) LFighter, which dynamically extracts and clusters the relevant gradients of the label-flipping (LF) attack from the last layer to counter the attack; iii) FL-Defender, which extracts squeezed discriminative features from the peers' last-layer gradients and uses them to mitigate the impact of the targeted poisoning attacks; and iv) fragmented federated learning (FFL), which addresses the security-privacy-accuracy conflict by extending cooperation between peers in FL systems to preserve their privacy without renouncing robust and accurate aggregation of their updates to the global model.

In addition to those four defenses, we propose two more defenses against backdoor and model stealing attacks that can be adopted both in federated and centralized learning. These defenses are: v) a method to protect against BAs based on layer-wise feature analysis; and vi) KeyNet, which employs multi-task learning to embed a watermark in deep neural networks and detect stolen models.

Experimental results on real data sets demonstrate the effectiveness of the proposed defenses in making ML more secure and private.

Resum

L'aprenentatge federat (AF) permet d'aprendre un model global d'aprenentatge automàtic a partir de dades distribuïdes entre un conjunt de participants, sota la coordinació d'un servidor que actua com a gestor del model. Això fa possible: a) reduir el cost computacional de l'entrenament pel cantó del servidor, perquè les actualitzacions del model són calculades pels participants per a llurs dades respectives, b) entrenar models més acurats gràcies a l'aprenentatge de dades conjuntes més riques, i c) millorar la privadesa perquè els participants no comparteixen amb ningú més llurs respectives dades privades. Malgrat aquests avantatges, el caràcter distribuït de l'AF el fa vulnerable a diversos atacs contra la seguretat i contra la privadesa. En efecte, participants maliciosos poden perjudicar la disponibilitat o la integritat del model enviant actualitzacions enverinades, alhora que les actualitzacions correctes enviades per participants honrats poden revelar-ne les dades locals privades. A mesura que l'AF guanya popularitat en aplicacions reals, protegir-ne els models contra aquests atacs esdevé crucial. Les defenses existents contra atacs d'enverinament causen un gran sobrecost computacional al servidor o són poc efectives, perquè necessiten fer suposicions sobre la distribució de les dades dels participants o no funcionen bé per a models de moltes dimensions. D'altra banda, contrarestar atacs a la privadesa a base de distorsionar les actualitzacions perjudica la precisió del model, mentre que fer-ho a base d'agregar actualitzacions en perjudica la seguretat perquè no permet al servidor de detectar i desempallegar-se d'actualitzacions individuals enverinades.

A més, l'AF comparteix algunes vulnerabilitats de seguretat amb l'aprenentatge centralitzat. D'una banda, tots dos paradigmes són vulnerables als anomenats "atacs per la porta del darrere" (APDs), que són difícils de detectar malgrat el perill que suposen per a la seguretat. D'altra banda, tant en aprenentatge centralitzat com federat, és possible

que models que han costat molt d'entrenar siguin robats o mal utilitzats, cosa que afectaria la propietat intel·lectual dels qui els han entrenats.

Per tot això, en aquesta tesi proposem tres defenses contra atacs d'enverinament, així com una defensa completa contra atacs d'enverinament i també contra atacs a la privadesa en AF. Aquestes defenses són: i) un mètode que analitza els biaixos de la darrera capa del model per neutralitzar atacs bizantins d'enverinament de manera eficient; ii) LFighter, que extreu dinàmicament i agrega els gradients rellevants per als atacs de canvi d'etiqueta de la darrera capa per tal de contrarestar-los; iii) FL-Defender, que extreu característiques discriminants condensades a partir dels gradients de darrera capa dels participants i els fa servir per mitigar l'impacte d'atacs d'enverinament dirigits; iv) aprenentatge federat fragmentat, que tracta el conflicte entre seguretat, privadesa i precisió estenent la cooperació entre participants en sistemes AF per preservarne la privadesa sense renunciar a agregar-ne les actualitzacions de manera robusta i precisa.

A més de les quatre defenses anteriors, en proposem dues més contra atacs per la porta del darrere i contra atacs de robatori de model que poder fer-se servir tant en aprenentatge federat com centralitzat. Aquestes dues defenses addicionals són: v) un mètode per protegir-se contra APDs basat en l'anàlisi de característiques per capes; vi) KeyNet, que fa servir aprenentatge multi-tasca per encastar una marca d'aigua en una xarxa neuronal profunda i detectar-ne el robatori si es produeix.

Els nostres resultats experimentals amb conjunts de dades reals demostren l'efectivitat de les defenses proposades per fer l'aprenentatge automàtic més segur i més privat.

Resumen

El aprendizaje federado (AF) permite aprender un modelo global automático a partir de datos distribuidos entre un conjunto de pares participantes, bajo la coordinación de un servidor que hace las veces de gestor del modelo. Esto hace posible: a) reducir el coste computacional del entrenamiento del lado del servidor, pues las actualizaciones del modelo son calculadas por los participantes para sus datos respectivos, b) entrenar modelos más precisos gracias al aprendizaje sobre datos conjuntos más ricos, y c) mejorar la privacidad porque los participantes no comparten con nadie más sus respectivos datos privados. A pesar de estas ventajas, la naturaleza distribuida del AF lo hace vulnerable a varios ataques contra la seguridad y la privacidad. En efecto, participantes maliciosos pueden perjudicar la disponibilidad y/o la integridad del modelo enviando actualizaciones envenenadas, a la vez que las actualizaciones correctas enviadas por participantes honrados pueden revelar sus datos locales privados. A medida que el AF se va imponiendo en aplicaciones reales, proteger sus modelos contra estos ataques se hace imprescindible. Las defensas existentes contra ataques de envenenamiento causan un gran sobrecoste computacional al servidor o son poco efectivas, porque necesitan suposiciones sobre la distribución de los datos de los participantes o no funcionan bien para modelos de muchas dimensiones. Por otro lado, contrarrestar ataques a la privacidad a base de distorsionar las actualizaciones daña la precisión del modelo, mientras que hacerlo a base de agregar actualizaciones daña su seguridad porque impide al servidor detectar y desechar actualizaciones individuales envenenadas.

Además, el AF comparte algunas vulnerabilidades de seguridad con el aprendizaje centralizado. Por una parte, ambos paradigmas són vulnerables a los llamados "ataques por la puerta de atrás" (APAs), que son difíciles de detectar a pesar de lo peligrosos que son para la seguridad. Por otra parte, tanto en aprendizaje centralizado como federado, es

posible que modelos que han costado mucho de entrenar sean robados o mal utilizados, cosa que afectaría la propiedad intelectual de quien los ha entrenado.

Por todo lo dicho, en esta tesis proponemos tres defensas contra ataques de envenenamiento, así como una defensa completa contra ataques de envenenamiento y también contra ataques a la privacidad en AF. Estas defensas son: i) un método que analiza los sesgos en la última capa del modelo para neutralizar ataques bizantinos de envenenamiento de forma eficiente; ii) LFighter, que extrae dinámicamente y agrega los gradientes relevantes para los ataques de cambio de etiqueta de la última capa para contrarrestarlos; iii) FL-Defender, que extrae características discriminantes a partir de los gradientes de última capa de los participantes y los usa para mitigar el impacto de ataques de envenenamiento dirigidos; y iv) aprendizaje federado fragmentado, que se ocupa del conflicto entre seguridad, privacidad y precisión mediante la cooperación entre participantes en sistemas de AF para preservar su privacidad sin renunciar a agregar sus actualizaciones del modelo global de forma robusta y precisa.

Además de la cuatro defensas anteriores, proponemos dos más contra ataques por la puerta de atrás y ataques de robo del modelo, que son útiles tanto en aprendizaje federado como centralizado. Estas dos defensas adicionales son: v) un método para protegerse contra APAs basado en el análisis de características por capas; y vi) KeyNet, que usa aprendizaje multitarea para empotrar una marca de agua en una red neuronal profunda y detectar así su robo en caso de producirse.

Nuestros resultados experimentales con conjuntos de datos reales demuestran la efectividad de las defensas propuestas para hacer el aprendizaje automático más seguro y privado.

Acknowledgements

I would like to express my gratitude to my supervisors, Prof. Josep Domingo-Ferrer and Prof. David Sánchez, for their unwavering support and guidance during the development of this thesis. Their valuable insights and constructive feedback have greatly enhanced my research skills and helped me to push the boundaries of my academic pursuits. I am also grateful to Dr. Alberto Blanco-Justicia for his instrumental contributions to this thesis. Many thanks to all members of the CRISES group, especially Rami Haffar and Jesús Manjón, as well as my friends for their constant support. Finally, I am grateful to Dr. Mohammed Jabreel for everything he has done for me during my stay and study in Catalunya.

This research was funded by the PhD Program of the "Secretaria d'Universitats i Recerca del Departament de Recerca i Universitats de la Generalitat de Catalunya" (2020 FI_B 00760); by the Government of Catalonia (ICREA Acadèmia Prizes to J.Domingo-Ferrer and D. Sánchez); by MCIN/AEI/ 10.13039/501100011033 and "ERDF A way of making Europe" under grant PID2021-123637NB-I00 "CURLING"; by the Spanish Government (projects RTI2018-095094-B-C21 "Consent" and TIN2016-80250-R "Sec-MCloud"); and by the European Commission under H2020 projects.

Contents

List of Figures

List of Tables

I would like to dedicate this thesis to the soul of my father.

To my mother.

To my wife and children.
And to my brother and sisters.

Chapter 1

Introduction

Federated Learning (FL, McMahan et al., 2017a) has emerged as a promising paradigm for training machine learning (ML) models using decentralized data. It enables a set of participating peers to train a ML model, usually a deep neural network (DNN), collaboratively without sharing their local data with a central server. DNNs are preferred to other ML algorithms because they can solve complex tasks more accurately, including computer vision, speech recognition, natural language processing, or stock market analysis Deng and Yu, 2014; LeCun, Bengio, and Hinton, 2015; Dargan et al., 2019. Peers in FL include organizations, banks, hospitals, and edge devices like smartphones. The FL training process involves peers fine-tuning a global model received from the server on their local data to compute local model updates that they upload to the server, which aggregates them to obtain an updated global model. This process is iterated until a high-quality global model is developed.

FL offers several advantages over traditional centralized machine learning: i) the server distributes the training computational load, which is significant for large-scale ML, across the peers' devices (*e.g.*, smartphones) (Bonawitz et al., 2019), ii) the peers and the server obtain more accurate models due to learning from rich, joint training data, and iii) privacy improves by not sharing the peers' local data with a central server. This latter advantage makes FL a suitable option for scenarios

dealing with personal data, such as facial recognition (Xu et al., 2017), voice assistants (Bhowmick et al., 2018), healthcare (Brisimi et al., 2018), next-word prediction (Hard et al., 2018), intrusion detection in IoT networks (Mothukuri et al., 2022) and location-based services (Huang, Tong, and Feng, 2022), or in case data collection and processing are restricted due to privacy protection laws such as the GDPR (European Commission, 2016)

1.1 Motivation

Despite these advantages, FL is vulnerable to various security and privacy attacks (Kairouz et al., 2019; Mothukuri et al., 2021).

Regarding security, FL is vulnerable to poisoning attacks (Blanco-Justicia et al., 2021; Lyu et al., 2022). Since the server has no control over the behavior of the participating peers, any of them may deviate from the prescribed training protocol to attack the global model by conducting either untargeted poisoning attacks (Blanchard et al., 2017; Wu et al., 2020b) or targeted poisoning attacks (Biggio, Nelson, and Laskov, 2012; Fung, Yoon, and Beschastnikh, 2020; Bagdasaryan et al., 2020). In the former type of attacks, the attacker aims to degrade the model's overall performance, whereas in the latter, he aims to cause the global model to misclassify some attacker-chosen inputs into an attacker-chosen class. Furthermore, poisoning attacks can be performed in two ways: model poisoning (Blanchard et al., 2017; Wu et al., 2020b; Bagdasaryan et al., 2020) or data poisoning (Biggio, Nelson, and Laskov, 2012; Fung, Yoon, and Beschastnikh, 2020; Tolpegin et al., 2020). In model poisoning, the attackers maliciously manipulate their local model parameters before sending them to the server. In data poisoning, they inject fabricated or falsified data samples into their training data before local model training. Both attacks result in poisoned updates being uploaded to the server in order to prevent the global model from converging or to bias

it. Figure 1.1 illustrates the federated learning process under poisoning attacks.

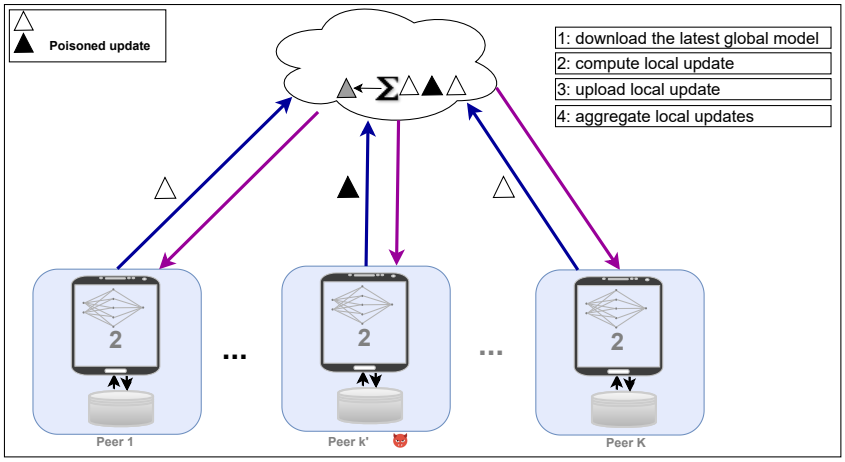


FIGURE 1.1: Illustration of the federated learning process under poisoning attacks. When the server aggregates good updates (uploaded by honest peers) and poisoned updates (uploaded by attackers), the new global model gets poisoned.

Regarding privacy, several works (McMahan et al., 2017b; Nasr, Shokri, and Houmansadr, 2019; Melis et al., 2019) have demonstrated that a *semi-honest* server can analyze individual updates to infer sensitive information on a peer's local training data. Recent powerful privacy attacks show that it is possible to reconstruct the original training data by inverting the gradients of updates (Zhu and Han, 2020; Zhao, Mopuri, and Bilen, 2020; Geiping et al., 2020). Figure 1.2 shows an example of a reconstructed image by the *semi-honest* server from the local gradients of a participating peer in FL.

As FL becomes more prevalent in real-world applications, safeguarding its models against poisoning and privacy attacks becomes crucial.

Several defenses against poisoning attacks have been proposed, which

FIGURE 1.2: Reconstruction of an input images from the local gradients of a participating peer. Left: Original input images. Right: Reconstructed image from original gradients.

we discuss in Chapter 3. Most of these defenses are effective against untargeted poisoning attacks, but they impose a high computational cost on the server to filter out poisoned updates (Blanchard et al., 2017; Chen, Su, and Xu, 2017; Yin et al., 2018). Moreover, they often become less effective or even fail against targeted poisoning attacks such as *label-flipping* attacks (LFs) or *backdoor* attacks (BAs). Finally, as we discuss in Chapter 3, they present a number of limitations: they are either impractical (Nelson et al., 2008; Jagielski et al., 2018), may degrade the performance of the aggregated model on the main task (Bagdasaryan et al., 2020; Wang et al., 2020a) or assume specific distributions of local training data (Shen, Tople, and Saxena, 2016; Blanchard et al., 2017; Tolpegin et al., 2020; Fung, Yoon, and Beschastnikh, 2020; Awan, Luo, and Li, 2021; Chen, Su, and Xu, 2017; Yin et al., 2018).

On the other hand, some solutions have been proposed to prevent the server from analyzing individual updates or linking them to their originators. These involve well-known privacy-enabling methods: differential privacy (DP) (Dwork, Roth, et al., 2014), homomorphic encryption (HE) (Gentry, 2009) and secure multiparty computation (SMC) (Yao, 1982). DP-based methods protect the peers' data by injecting random noise into the parameters of updates at the cost of sacrificing the accuracy of the global model (Domingo-Ferrer, Sánchez, and Blanco-Justicia, 2021; Blanco-Justicia et al., 2023). HE- and SMC-based methods securely

aggregate the updates of peers before sending them to the server. Yet, HE and SMC have a high computational cost and they prevent the server from inspecting individual updates, which makes approaches based on these techniques vulnerable to poisoning attacks. Note that countermeasures against poisoning attacks require direct access by the server to the individual updates in order to detect and/or filter out those that are poisoned (Yin et al., 2018; Blanchard et al., 2017; Fung, Yoon, and Beschastnikh, 2018).

Therefore, simultaneously achieving security, privacy and accuracy is a tough challenge for FL (Li et al., 2020; Kairouz et al., 2019; Blanco-Justicia et al., 2021).

Additionally, FL shares some security vulnerabilities with centralized learning (CL). In particular, both paradigms are susceptible to BAs (Gu et al., 2019) and *model stealing* attacks (Tramèr et al., 2016; Wang and Gong, 2018; Zhang et al., 2018).

In FL, malicious peers have the ability to conduct targeted poisoning attack, including BAs (Bagdasaryan et al., 2020). In CL, users who cannot afford training DNNs due to the required massive training data and computational resources may opt to outsource training to a third-party (*e.g.*, a cloud service) or leverage pre-trained DNNs. Unfortunately, losing control over training facilitates BAs against DNNs. In BAs, the adversary poisons a few training samples to cause the DNN to misclassify samples containing pre-defined trigger patterns into an adversary-specified target class. Nevertheless, the attacked models behave normally on benign samples, which makes the attack hard to detect.

On the other side, building highly accurate DNNs is costly for its owners in both paradigms. In CL, model owners such as technology companies devote significant computational resources to process vast amounts of proprietary training data, whose collection and labeling also imply a significant effort. For example, a conversational model from Google Brain contains 2.6 billion parameters and takes about one month

to train on 2048 TPU cores (Adiwardana et al., 2020). Besides, designing the architecture of a DL model and choosing its hyper-parameters requires substantial ML experience and many preliminary tests. In FL, a group of organizations can jointly build a high-fidelity model by training it on their big local data and using their own computational resources. For example, during the Covid crisis, 20 different health institutes around the world trained a global DNN model to more accurately predict how much oxygen COVID-19 patients will need based on their vital signs, lab tests and chest X-rays (Dayan et al., 2021). Thus, it is unsurprising that owners seek compensation for the incurred costs by reaping profits from commercial exploitation. They may monetize their models in ML as a Service (MLaaS) platforms (Ribeiro, Grolinger, and Capretz, 2015) or license them for a financial return to their customers for a specific period of time (Yao et al., 2017). Unfortunately, the high value of pre-trained DL models attracts attackers who would like to avoid those costs and steal the models to use them illegally. For example, a user may leak a pre-trained model to an unauthorized party or continue to use it after the license period has expired. Furthermore, if a model is offered as MLaaS, many model theft techniques are available to steal it based on its predictions (Tramèr et al., 2016; Wang and Gong, 2018). Due to the competitive nature of the technology market, a stolen or misused model is clearly detrimental to its owner on both economic and competitive terms.

Since model theft cannot be prevented in advance, legitimate owners need a robust and reliable way to prove their ownership of DL models in order to protect their intellectual property (IP).

1.2 Contributions

The contributions of this thesis are six defenses to safeguard models and data in federated and centralized learning. Among these defenses, there are three that address poisoning attacks in FL; one that comprehensively

tackles both poisoning and privacy attacks in FL; one that targets backdoor attacks during inference in CL (which can also be utilized in FL); and a DNN watermarking framework to tackle model stealing in CL, which can also be extended to FL. Specifically:

- We propose a novel method to detect and eliminate Byzantine attacks in FL with low computational overhead on the server side. It achieves this by examining only the biases of the last layer of updates from peers, and filtering out poisoned ones during the global model aggregation.
- We conduct an in-depth analysis of the label-flipping attack behavior in FL. As a result, we find that gradients connected to the source and target class neurons in the output layer serve as good discriminative features for attack detection. We then propose *LFighter*, a novel defense that extracts the potential source and target class gradients from local updates, clusters them, and filters out potential bad updates before model aggregation. The proposed defense stays robust under different data distributions and model sizes.
- We propose *FL-Defender*, a method that can mitigate the targeted attacks against FL regardless of the model dimensionality or the distribution of the peers' data. First, we use the peers' last-layer gradients to engineer more robust discriminative features that capture the attack behavior and discard redundant information. Specifically, we compute the peer-wise angle similarity for the peers' last-layer gradients and then compress the computed similarity vectors using principal component analysis (PCA) (Wold, Esbensen, and Geladi, 1987) to reduce redundant information. Finally, we penalize the peers' updates at the model aggregation phase based on their angular deviation from the centroid of the compressed similarity vectors.

- We propose the fragmented federated learning (*FFL*) framework to solve the security-privacy-accuracy conflict in FL. FFL introduces a lightweight protocol that enables peers to privately exchange and mix random fragments of their updates, thereby making it difficult for the server to link the updates to their originators or recover the complete original updates, which prevents the semi-honest server from conducting successful privacy attacks. The protocol also allows the server to correctly aggregate the mixed updates into a global model. To defend against poisoning attacks, we integrate a novel reputation-based defense mechanism in the FFL design that uses the quality of the updates and fragments exchanged to assign global and local reputations to peers. The server then selects peers for training and adaptively aggregates their mixed updates based on their global reputations, whereas honest peers are incentivized not to exchange fragments with peers having low local reputations.

- We propose a novel backdoor detection method at inference time that locates the critical layer in DNNs where the separation between poisoned and benign samples is the greatest. We then filter incoming suspicious samples by using feature differences at that layer. This method detects poisoned samples at inference time to train a DNN on CL and can also be used for the final global model trained on FL.

- We propose a novel digital watermarking framework called *KeyNet* that meets most of the requirements of an effective DNN watermarking framework. The framework uses three components: the watermark (WM) carrier set, the owner information, and a marked model along with its private model to embed and verify WM information. The framework can also be used to fingerprint different unique copies of a DL model for different users in the system. It can also be extended to the FL paradigm.

- We demonstrate the effectiveness of our defenses through an extensive experimental evaluation on standard data sets with different attack configurations and training settings. In particular, our defense against Byzantine attacks achieves similar attack detection performance as the best baseline method while significantly reducing the detection and aggregation runtime at the server compared to the other defenses. LFighter stays robust to the LF attack regardless of the peers' data distribution or model size and significantly outperforms the other state-of-the-art defenses in all evaluation metrics. FL-Defender achieves better performance than the other defenses at retaining the accuracy of the global model on the main task, reducing the attack success rate, and causing minimal computational overhead on the server. FFL can effectively counter privacy and security attacks while maintaining the global model's accuracy and imposing affordable communication cost and computation overhead on the participating parties. Our dedicated defense against backdoor attacks effectively identifies critical layers for different DNNs poisoned with a set of representative state-of-the-art backdoor attacks and, compared to several state-of-the-art defenses against BAs, it distinguishes better between poisoned and benign samples at the inference time. Finally, *KeyNet* successfully embeds WMs with reliable detection accuracy while preserving the accuracy of the original task.

1.3 Outline

The remainder of this thesis organized in the following chapters.

- **Chapter 2, Background**: We provide background on deep neural networks, federated learning, and principal components analysis (PCA). Also, we describe the data sets and computational resources used in the thesis experiments.

- **Chapter 3, Attacks and defenses in FL**: We provide an overview of the poisoning and privacy attacks against federated learning. Also, we survey state-of-the-art defense mechanisms against those attacks and discuss their limitations.
- **Chapter 4, Efficient detection of Byzantine attacks in FL**: We present our efficient method to detect Byzantine attacks in FL using the biases of the last layer of DNNs. We evaluate the performance of our method on real data sets and compare it with several state-of-the-art countermeasures against Byzantine attacks.
- **Chapter 5, Defending against the label-flipping attack in FL**: We analyze the LF attack and introduce LFighter, a novel method to effectively and efficiently defend against the LF attack. We evaluate the effectiveness of our defense against the LF attack and compare it with several state-of-the-art defenses.
- **Chapter 6, Combating targeted attacks in FL**: We analyze the behavior of targeted attacks on FL, and we present FL-Defender to mitigate the impact of the targeted poisoning attacks. We evaluate its effectiveness against the attacks and compare it with several state-of-the-art defenses against targeted attacks.
- **Chapter 7, Enhanced security and privacy via fragmented FL**: We present the design of the fragmented federated learning (FFL) framework and extensively evaluate its accuracy, privacy, security, computation overhead, and communication cost. We compare its performance against untargeted and targeted attacks with several state-of-the-art countermeasures.
- **Chapter 8, Defending against backdoor attacks by layer-wise feature analysis**: We conduct a layer-wise feature analysis of poisoned and benign samples from the target class. We then propose a simple yet effective method to filter poisoned samples by analyzing the feature differences between suspicious and benign

samples at the critical layers identified by the previous analysis. Finally, we conduct extensive experiments on two benchmark data sets and compare our method with several state-of-the-art countermeasures to confirm its effectiveness.

- **Chapter 9, Watermarking deep learning models**: We present KeyNet, our proposed watermarking framework for DNNs, which meets most of the desirable requirements for an effective watermarking framework. We conduct extensive experiments using different DNN model architectures to evaluate the proposed framework.
- **Chapter 10, Conclusions and future work:** We summarize the main contributions of this thesis, list the achieved scientific publications, and present some lines of future research.

Chapter 2

Background

2.1 Deep neural networks

Deep neural networks (DNNs) are a class of artificial neural networks that contain multiple hidden layers between the input and output layers. The hidden layers allow DNNs to learn more complex and sophisticated representations of the data they are trained on. This property leads to improved performance across a wide range of tasks, including computer vision, natural language processing (NLP), speech recognition, recommendation systems, and game playing.

Mathematically, a DNN is a function $f(x)$, obtained by composing L functions $f^l, l \in [1, L]$, that maps an input x to a predicted output $\hat{y}$. Each f^l is a layer that is parametrized by a weight matrix w^l, a bias vector b^l, and an activation function σ^l. f^l takes as input the output of the previous layer f^{l-1}. The output of f^l on an input x is computed as $f^l(x) = \sigma^l(w^l \cdot x + b^l)$. Therefore, a DNN can be formulated as

$$f(x) = \sigma^L(w^L \cdot \sigma^{L-1}(w^{L-1} \dots \sigma^1(w^1 \cdot x + b^1) \cdots + b^{L-1}) + b^L).$$

DNN-based classifiers consist of a feature extraction part and a classification part (Krizhevsky, Sutskever, and Hinton, 2017; Minaee et al., 2021). The classification part takes the extracted latent features and

makes the final classification decision. It usually consists of one or more fully connected layers where the output layer contains $|\mathcal{C}|$ neurons, with $\mathcal{C}$ being the set of all possible class values. The output layer's vector $o \in \mathbb{R}^{|\mathcal{C}|}$ is usually fed to the softmax function that transforms it into a vector p of probabilities, which are called the confidence scores.

The goal of training a DNN model is to find the set of parameters (w, b) that minimize a differentiable loss function $\mathcal{L}$ with respect to the ground-truth labels. In classification problems, $\mathcal{L}$ is usually the cross-entropy loss.

In this thesis, we use predictive DNNs as $|\mathcal{C}|$-class classifiers, where the final predicted label $\hat{y}$ is taken to be the index of the highest confidence score in p. Also, activations of intermediate layers are analyzed for detecting input samples poisoned by backdoor attacks. Additionally, the terms DNNs and deep learning (DL) models are used interchangeably in this thesis.

2.2 Federated learning

In the typical FL (a.k.a. horizontal FL), K peers and an aggregator server A collaboratively build a global model W. In each training iteration $t \in [1, T]$, the server randomly selects a subset of peers S of size $m = C \cdot K \geq 1$ where C is the fraction of peers that are selected in iteration t. After that, the server distributes the current global model W^t to all peers in S. Besides W^t, the server sends a set of hyper-parameters to be used to train the local models, which includes the number of local epochs E, the local batch size BS, and the learning rate η. After receiving W^t, each peer $k \in S$ divides her local data D_k into batches of size BS and performs E optimization steps on D_k to compute her update W_k^{t+1}, which she uploads to the server. Typically, the SGD optimizer is used to perform the optimization step. The federated averaging algorithm (*FedAvg*, McMahan et al., 2017a) is usually employed to perform the aggregation, and it

is defined as

$$W^{t+1} = \sum_{k=1}^{K} \frac{d_k}{d} W_k^{t+1}, \tag{2.1}$$

where d_k is the number of data points locally held by worker k, and d is the total number of data points locally held by the K workers, that is, $d = \sum_{k=1}^{K} d_k$.

Note that FedAvg is the standard way to aggregate updates in FL and is not meant to counter security attacks.

Types of FL. Federated Learning is not limited to the horizontal FL framework. Several other types of FL frameworks have been developed to handle different scenarios (Mammen, 2021):

- Horizontal federated learning (HFL): This is used when each peer has a data set with the same feature space but different sample instances. A classic use case is the Google Keyboard app, where participating mobile phones have different training data but the same features.

- Vertical federated learning (VFL): This is used when each peer has a data set with different features but from the same sample instances. For example, two organizations with data about the same group of people but with different feature sets can use Vertical FL to build a shared ML model.

- Federated transfer learning (FTL): This is similar to traditional ML, where we want to add a new feature to a pre-trained model. An example of this is extending Vertical FL to include more sample instances that are not present in all collaborating organizations.

- Cross-silo federated learning: This is a type of FL where participating peers are large distributed entities (*e.g.*, hospitals, banks, and companies) that have abundant local data and computational resources, and are available for all rounds. The training data can be in horizontal or vertical FL format.

- Cross-device federated learning: This is another type of FL where peers are small distributed entities (*e.g.*, smartphones, wearables, and edge devices) that have limited local data and computational resources. In this type, the number of peers is large, and they are not available for all rounds. Usually, the training data are in horizontal FL format.

In this thesis, we focus on horizontal FL, which is the most used type of FL in the literature.

2.3 Principal components analysis

Principal Components Analysis (PCA) (Wold, Esbensen, and Geladi, 1987) is a statistical technique used for dimensionality reduction in data analysis. It works by identifying the underlying structure or patterns in the data and projecting it onto a new set of orthogonal axes, called principal components, that capture most of the variance in the data. The first principal component is the axis that accounts for the largest amount of variance in the data, the second principal component is the axis that accounts for the second-largest amount of variance, and so on. By selecting a subset of the principal components that capture most of the variance in the data, we can reduce the dimensionality of the data and simplify the analysis. Mathematically, PCA can be defined as follows:

Given a matrix X of n observations and p variables, PCA aims to find a new set of k orthogonal axes, where $k < p$, that maximize the variance of the data projected onto the axes. This can be achieved by computing the eigenvectors and eigenvalues of the covariance matrix of X, and selecting the top k eigenvectors with the largest eigenvalues as the principal components. The data can then be projected onto the principal components by multiplying X times the matrix of eigenvectors. The resulting transformed data will have k dimensions and will capture most of the variance in the original data.

TABLE 2.1: Data sets used in the experiments of the thesis

Task	Data set	# Examples	# Training examples	# Test examples
Image classification	MNIST	70K	60K	10K
	CIFAR10	60K	50K	10K
	GTSRB	51,839	39,209	12,630
	Fashion-MNIST	70K	60K	10K
	STL10	13K	5K	8K
Sentiment analysis	IMDB	50K	40K	10K
Tabular classification	Adult	48,842	39,074	9,768

In this thesis, we use PCA to analyze and visualize the good and poisoned updates sent by honest and malicious peers.

2.4 Data sets and computational resources

In this thesis, we use seven data sets related to the following three ML tasks: tabular data classification, image classification and sentiment analysis. Table 2.1 summarizes the data sets we use.

Image classification. We use several data sets for this task:

- MNIST. It contains $70K$ grayscale images of handwritten digits ranging from 0 to 9 with a size of $28 \times 28 \times 1$ pixels (LeCun et al., 1999). It is split into a training set of $60K$ examples and a testing set of $10K$ examples.

- CIFAR10. It is made up of $60K$ color images belonging to 10 different classes (Krizhevsky, 2009). The images have a size of $32 \times 32 \times 3$ pixels and are split into a training set of $50K$ examples and a testing set of $10K$ examples.

- GTSRB. The German Traffic Sign Recognition Benchmark consists of about $51,839$ colored images of 43 different classes of traffic signs, with a size of $32 \times 32 \times 3$ pixels (Stallkamp et al., 2011). The data set is split into a training set of $39,209$ examples and a testing set of $12,630$ examples.

- Fashion-MNIST. It contains $70K$ grayscale images of fashion items such as shoes, bags, and shirts (Xiao, Rasul, and Vollgraf, 2017). The images have a size of $28 \times 28 \times 1$ pixels and are divided into a training set of $60K$ examples and a testing set of 10 examples. The number of classes in this data set is 10.

- STL10. This is a data set of $13K$ colored images belonging to 10 different classes, with a size of $96 \times 96 \times 3$ pixels (Coates, Ng, and Lee, 2011). It is split into a training set of $5K$ examples and a testing set of $8K$ examples.

Sentiment analysis. We use the IMDB Large Movie Review data set (Maas et al., 2011) for this binary sentiment classification task. The data set is a collection of $50K$ movie reviews and their corresponding sentiment binary labels (either positive or negative). We divided the data set into $40K$ training examples and $10K$ testing examples.

Tabular data classification. We use the Adult tabular data set that contains $48,842$ records of census income information with 14 numerical and categorical attributes (Kohavi et al., 1996). The class label is the attribute *income* that classifies records into either $>$ 50K \$ or $\leq$ 50K \$. We used 80% of the data as training data and the remaining 20% as validation data.

In the FL setting, we distribute the training examples among participating peers while we train the centralized models on the full training set examples. In both settings, we use the test examples to evaluate the performance of the trained models.

For all experiments in this thesis, we use the PyTorch framework (Paszke et al., 2019) to implement the experiments on an AMD Ryzen 5 3600 6-core CPU with 32 GB RAM, an NVIDIA GTX 1660 GPU, and Windows 10 OS.

Chapter 3

State of the art in poisoning and privacy attacks and defenses in FL

While federated learning offers several advantages over centralized learning, it remains vulnerable to poisoning and privacy attacks due to its decentralized approach. In fact, the distributed nature of federated learning can exacerbate these attacks compared to traditional centralized learning. This chapter provides an overview of the poisoning and privacy attacks that may occur in federated learning. Also, it surveys state-of-the-art defense mechanisms against those attacks and discusses their limitations.

3.1 Poisoning attacks against FL

Federated learning FL is vulnerable to poisoning attacks, which aim to manipulate the training process by injecting malicious data or model updates. Poisoning attacks against FL systems can be broadly categorized into two types: untargeted (Blanchard et al., 2017; Wu et al., 2020b; Fang et al., 2020) and targeted (Biggio, Nelson, and Laskov, 2012; Chen

et al., 2017; Bhagoji et al., 2019; Fung, Yoon, and Beschastnikh, 2020; Bagdasaryan et al., 2020). Both targeted and untargeted poisoning attacks can be carried out during the training phase of FL, either on the local data or on the local model. Data poisoning attacks involve the injection of manipulated samples into the training data set, which can result in the model being trained on biased or misleading data. On the other hand, model poisoning attacks involve manipulating the model parameters, either directly or indirectly, during the local model training process.

3.1.1 Untargeted attacks

Untargeted poisoning attacks aim to compromise the availability of the global model without any specific goal or objective. Among these attacks, the Byzantine attack is particularly harmful as it can upload malicious gradients to the server that can cause the entire global model to fail (Blanchard et al., 2017; Damaskinos et al., 2019; Lamport, Shostak, and Pease, 2019; Xie, Koyejo, and Gupta, 2020). Blanchard et al., 2017 have demonstrated that if there is no defense in the FL, a single Byzantine peer can completely control the FL aggregation process. Xie, Koyejo, and Gupta, 2020 have shown that consistent small changes to many parameters by a Byzantine peer can perturb the model's convergence. They achieve this by first using the local data of Byzantine peers to estimate the mean and standard deviation of the distribution, and then by analyzing the range in which changes to the parameters will not be detected by the defense. By choosing the maxima of this range, they can avert the convergence of the model.

3.1.2 Targeted attacks

Targeted poisoning attacks aim at making the global model misclassify a set of chosen samples to an attacker-chosen target class while minimizing the impact on the model performance on the main task. Typically,

targeted attacks are stealthier than untargeted attacks. Two common examples of targeted poisoning attack are the label-flipping attack (Biggio, Nelson, and Laskov, 2012; Fung, Yoon, and Beschastnikh, 2020) and the backdoor attack (Gu, Dolan-Gavitt, and Garg, 2017; Bhagoji et al., 2019; Bagdasaryan et al., 2020).

- **Label-flipping attack.** In the LF attack (Biggio, Nelson, and Laskov, 2012; Fung, Yoon, and Beschastnikh, 2020; Tolpegin et al., 2020), the attackers poison their local training data by flipping the labels of training examples of a source class c_{src} to a target class $c_{target} \in \mathcal{C}$ while keeping the input data features unchanged. Attackers poison their local data set D_k as follows: for all examples in D_k whose class label is c_{src}, they change their class label to c_{target}. After poisoning their training data, attackers train their local models using the same hyper-parameters, loss function, optimization algorithm, and model architecture sent by the server. Figure 3.1 shows an example of the LF attack. In the example, the attacker poisons his training data by flipping labels of the sample with the source class label "0" to the target class label "1", and trains his local model on the poisoned data. Accordingly, the local update is poisoned, which will classify test samples with the true class "0" into the target class label "1". When poisoned updates are aggregated with other good ones, as shown in Fig. 1.1, the obtained global model is expected to be poisoned and thus exhibit similar targeted behavior on poisoned samples at inference time.

- **Backdoor attack.** In the BA (Gu, Dolan-Gavitt, and Garg, 2017; Bagdasaryan et al., 2020), the attacker poisons his training data by embedding a specific pattern (*i.e.*, backdoor trigger) into training samples with specific features, and assigns them a target class label of his choice. The pattern acts as a trigger for the global model to output the desired target label for the poisoned samples. Fig. 3.2 illustrates an example of the backdoor attack. In the example, the

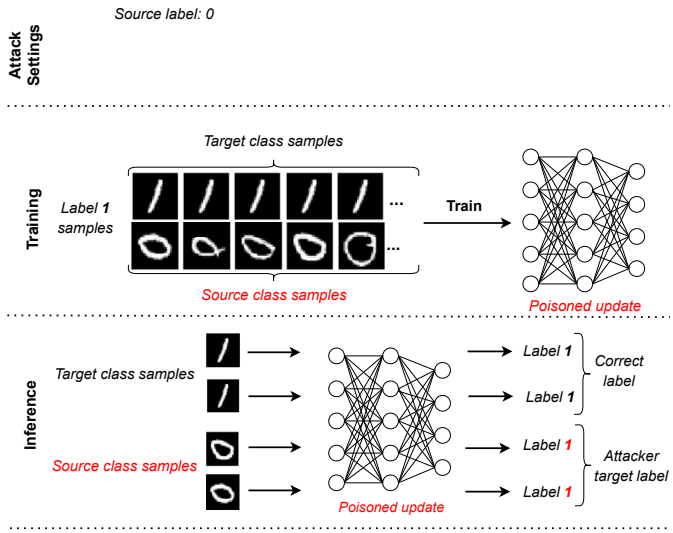


FIGURE 3.1: Example of label-flipping (LF) attack.

trigger is a white square on the bottom right corner, and the target label is "0". Some benign training samples are stamped with the trigger pattern, and their labels are changed to the attacker-specified target label. Accordingly, the local update is poisoned, which will classify test samples containing backdoor triggers as the target label while correctly predicting the label for the benign samples. When poisoned updates are aggregated with other good ones, as shown in Fig. 1.1, the obtained global model is expected to be poisoned and thus exhibit similar targeted behavior on poisoned samples at inference time.

3.2 Privacy attacks against FL

FL prevents private data sharing, but exchanging local updates can still leak sensitive information about the peers' data to attackers (McMahan

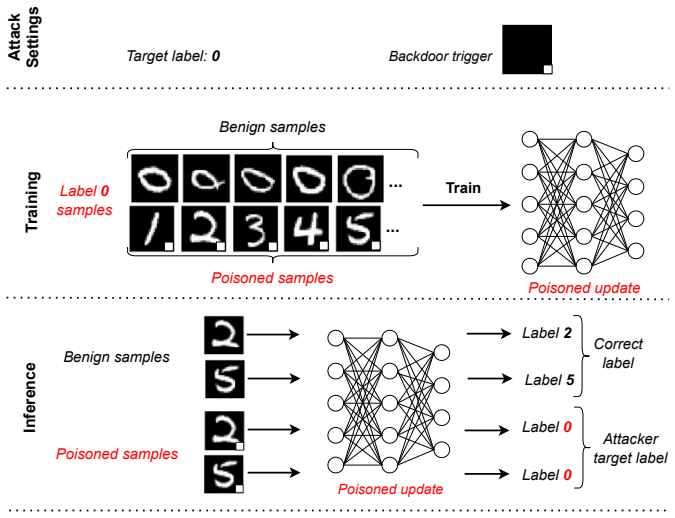


FIGURE 3.2: Illustration of the backdoor attack (BA).

et al., 2017b; Nasr, Shokri, and Houmansadr, 2019; Melis et al., 2019; Zhao, Mopuri, and Bilen, 2020; Geiping et al., 2020). Local gradients or consecutive snapshots of FL model parameters can reveal unintended training data features to adversaries, as DL models remember more features than needed for the main task (Zhang et al., 2021a). Peers' local updates are derived from their private training data, and the learnt model represents high-level statistics of the data set it was trained on. Therefore, those updates can reveal private information such as class representatives, membership, and properties of the local training data. Attackers can infer labels from shared gradients and even recover training samples without prior knowledge of the data (Zhu and Han, 2020). Next, we briefly explain the privacy attacks against FL based on the type of sensitive information that attackers want to obtain.

3.2.1 Membership inference attacks

Membership inference attacks (MIAs) aim to determine if a particular data point was used to train a model (Melis et al., 2019; Nasr, Shokri, and Houmansadr, 2019). For example, inferring whether a specific patient record was used to train a classifier associated with a certain disease. FL makes it possible for attackers to infer if a specific sample belongs to the private training data of a specific peer or any peer. MIAs in FL can be either passive or active. In the former, the attacker observes the updated model parameters and performs inference without tampering with the FL model training protocol (Melis et al., 2019). In the latter, the attacker modifies the FL learning process to boost the attack performance (Nasr, Shokri, and Houmansadr, 2019). For example, in the gradient ascent attack (Nasr, Shokri, and Houmansadr, 2019), the attacker runs gradient ascent on a target data sample, and observes whether its increased loss can be drastically reduced in the next communication round to determine if the sample is likely to belong to the training set.

3.2.2 Property inference attacks

These attacks try to infer specific properties about the training data (Ganju et al., 2018; Melis et al., 2019). Examples of properties that an adversary may attempt to infer include sensitive personal information, demographic information, or other features of the training data. The attackers are assumed to have access to auxiliary training data correctly labeled with the target property.

3.2.3 Distribution estimation attacks

These attacks aim to obtain examples from the same distribution of the peers' training data. One such attack is the GAN attack developed by Hitaj, Ateniese, and Perez-Cruz, 2017, where a malicious peer trains a GAN to generate prototypical samples of the targeted private training

data to infer class representatives. However, the attack assumes that the entire training corpus for a given class comes from a single peer and requires a lot of computational resources, which makes it impractical for real-world FL scenarios.

3.2.4 Reconstruction attacks

Reconstruction attacks are the most ambitious in that they attempt to extract the original training data from a peer's local update (Zhu and Han, 2020; Zhao, Mopuri, and Bilen, 2020; Geiping et al., 2020). The deep leakage from gradient (DLG) attack (Zhu and Han, 2020) proposes an optimization algorithm to extract both training inputs and labels. It can recover the raw images and texts used to train a DL model. The improved deep leakage from gradient (iDLG) (Zhao, Mopuri, and Bilen, 2020) is an analytical approach that uses shared gradients to extract labels by exploring the correlation between the labels and the signs of the gradients. iDLG can be applied to attack any differentiable model trained with cross-entropy loss and one-hot labels, which is common in classification tasks. Geiping et al., 2020 analyze the effects of the architecture and parameters on the difficulty of image reconstruction and show that even averaging gradients over several iterations or images does not protect user privacy in computer vision applications of FL. To mount any of these attacks, the attacker (typically the semi-honest server) needs to access individual local updates.

3.3 Defenses against poisoning attacks in FL

The defenses proposed in the literature to counter poisoning attacks are based on one of the following principles:

- *Evaluation metrics*. Approaches under this type exclude or penalize a local update if it has a negative impact on an evaluation metric of the global model, *e.g.* its accuracy. Specifically, Nelson et al.,

2008; Jagielski et al., 2018 use a validation data set on the server to compute the loss on a designated metric caused by each local update. Then, updates that negatively impact on the metric are excluded from the global model aggregation. However, realistic validation data require server knowledge of the distribution of the peers' data, which conflicts with the FL premise whereby the server does not see the peers' data.

- *Clustering updates*. Approaches under this type cluster updates into two groups, where the smaller group is considered potentially malicious and, therefore, discarded in the model learning process. Auror (Shen, Tople, and Saxena, 2016), multi-Krum (MKrum) (Blanchard et al., 2017), and Domingo-Ferrer et al., 2020 assume that the peers' data are IID, which results in high false positive and false negative rates when the data are non-IID (Awan, Luo, and Li, 2021). Moreover, they may require previous knowledge about the characteristics of the training data distribution (Shen, Tople, and Saxena, 2016) or the number of expected attackers in the system (Blanchard et al., 2017). Also, they impose a high computational cost on the server because they require analyzing the whole update parameters to filter out potential poisoned updates.

- *Peers' behavior*. This approach assumes that malicious peers behave similarly, meaning that their updates will be more similar to each other than those of honest peers. Consequently, updates are penalized based on their similarity. For example, FoolsGold (FGold) (Fung, Yoon, and Beschastnikh, 2020) and CONTRA (Awan, Luo, and Li, 2021) limit the contribution of potential attackers with similar updates by reducing their learning rates or preventing them from being selected. However, they also tend to incorrectly penalize good updates that are similar, which results in substantial drops in the model performance (Nguyen et al., 2021; Li et al., 2021a).

- *Update aggregation*. This approach uses robust update aggregation methods that are not affected by outliers at the coordinate level, such as the median (Yin et al., 2018), the trimmed mean (Tmean) (Yin et al., 2018) or the repeated median (RMedian) (Siegel, 1982). In this way, bad updates will have little to no influence on the global model after aggregation. Although these methods achieve good performance with updates resulting from IID data for small DL models, their performance deteriorates when updates result from non-IID data, because they discard most of the information in the model aggregation. Moreover, their estimation error scales up with the size of the model in a square-root manner (Chang et al., 2019). Furthermore, Tmean (Yin et al., 2018) requires explicit knowledge about the fraction of attackers in the system. Also (and especially for RMedian (Siegel, 1982)), they involve a high computational cost on the server.

- *Differential privacy (DP).* Methods under the DP approach (Bagdasaryan et al., 2020; Sun et al., 2019) clip individual update parameters to a maximum threshold and add random noise to the parameters to reduce the impact of potentially poisoned updates on the aggregated global model. However, there is a trade-off between the attack mitigation brought by the added noise and the performance of the aggregated model on the main task (Bagdasaryan et al., 2020; Wang et al., 2020a). Also, applying DP to FL incurs a significant computation overhead (Blanco-Justicia et al., 2023). FLAME (Nguyen et al., 2022) combines clustering, weight clipping, and DP to address the utility-robustness trade-off of previous DP-based defenses. First, the HDBSCAN dynamic clustering method (Campello, Moulavi, and Sander, 2013) is used to exclude updates with a large poisoning impact. Second, the remaining updates are clipped before aggregating them to a new global model to reduce the impact of scaled poisoned updates. Finally, a

bounded amount of noise is added to the global model to eliminate the impact of stealthy poisoned updates.

Several works focus on analyzing specific parts of the updates to defend against poisoning attacks. FGold (Fung, Yoon, and Beschastnikh, 2020) and CONTRA (Awan, Luo, and Li, 2021) analyze the output layer's weights to counter data poisoning attacks, but they suffer from the shortcomings mentioned above. Tolpegin et al., 2020 uses PCA to analyze the weights associated with *the possibly attacked source class*, and excludes potential bad updates that differ from the majority of updates in those weights. However, the method either needs explicit knowledge about the possibly attacked source class, or it performs a brute-force search to find it; moreover, it is only evaluated with simple DL models.

These methods share the shortcomings of (i) making assumptions on the distributions of peers' data and (ii) not providing deep analytical or empirical evidence of why focusing on specific parts of the updates contributes towards defending against the LF attack.

3.4 Defenses against privacy attacks in FL

On the privacy side, several works have been proposed to prevent the server from seeing individual updates. Bonawitz et al., 2017; So, Güler, and Avestimehr, 2021 use secret sharing to hide individual updates. Aono et al., 2017; Hardy et al., 2017; Zhang et al., 2020 encrypt the local updates and use homomorphic encryption to compute the global model from the encrypted updates. However, both approaches are vulnerable to poisoning attacks because they hinder the analysis of individual updates. Other works adopt differential privacy (DP) (McMahan et al., 2017b; Bhowmick et al., 2018; Geyer, Klein, and Nabi, 2017), which adds noise to local updates before sending them to the server. DP is practical but it only offers strong privacy guarantees for small values of ϵ

that, due to the noise they add, significantly hamper the accuracy of the global model (Domingo-Ferrer, Sánchez, and Blanco-Justicia, 2021; Blanco-Justicia et al., 2023).

3.5 Defenses against both poisoning and privacy attacks in FL

Recently, the literature has witnessed a growing interest in achieving FL that is both privacy-preserving and secure.

Naseri, Hayes, and De Cristofaro, 2020 uses DP to address both privacy and robustness against backdoor attacks. However, it does not deal with the trade-off between privacy and accuracy, and it does not consider poisoning attacks different from backdoor attacks.

PEFL (Liu et al., 2021) tries to address both privacy and security, but assumes there are two *non-colluding* servers that collaborate to filter out malicious updates while preventing each other from seeing individual updates. Moreover, PEFL builds on linear homomorphic encryption and a packing technique, and it involves exchanging the encrypted updates in four interacting protocols between the two servers for filtering and aggregation, which causes high communication and computation overheads.

ShieldFL (Ma et al., 2022b) also assumes two *non-colluding* servers, and uses a two-trapdoor HE scheme based on the Paillier cryptosystem to achieve both secure and privacy-preserving FL. In ShieldFL, the two servers execute three interactive protocols to compute the cosine similarity of the local updates in ciphertext. They then use the computed cosine similarities to filter out potential poisoned updates. Unfortunately, this also imposes significant computation and communication costs on the participating entities.

BREA (So, Güler, and Avestimehr, 2020) proposes a single-server framework where each peer secret-shares her local update with all the

other peers in the system. Also, each peer locally computes the peer-wise Euclidean distances to the shares of all peers, and then sends the computed distances to the server, which uses them to filter out potential poisoned updates. The server then selects the potential good updates and asks the peers to locally aggregate the selected good shares and upload the aggregates to the server. Finally, the server reconstructs the global model from the received aggregates and sends it to the peers in the next training round. Although this work has the advantage of being single-server, it imposes high computation and communication overheads on the participating parties.

Ma et al., 2022a integrate local DP on the peer's side with an intermediate shuffler between the peers and the aggregator server to achieve privacy. Besides, they use a Byzantine-robust stochastic aggregation algorithm at the server side to achieve security. However, this work is subject to the inevitable trade-off between privacy and accuracy due to the use of DP. In addition, the shuffler is a third party, and hence its honesty is not guaranteed.

SAFELearning (Zhang et al., 2021c) is based on the work of Bonawitz et al., 2017 to support backdoor detection and privacy-preserving aggregation simultaneously. To this end, it randomly divides peers into subgroups, securely aggregates a sub-model for each subgroup and filters out malicious sub-models instead of individual models. However, this approach faces a trade-off of another kind: the smaller the number of peers in a subgroup, the less privacy for these peers; conversely, the larger the number of peers, the easier it is for the attacker to hide her malicious model amid honest ones. The paper proposes to disclose part of the parameters of the aggregated sub-models, leading to another privacy/security trade-off.

Domingo-Ferrer et al., 2022 proposes a co-utile FL to solve the accuracy-privacy-security conflict. The proposed solution preserves the global model accuracy and allows defending against security attacks, but its privacy is limited to only breaking the link between local updates and

their originators. Although this provides a level of privacy protection, a semi-honest server still has direct access to original unlinked updates. Hence, it can use them to perform several privacy attacks, such as membership inference attacks and reconstruction attacks.

Chen et al., 2020; Zhang et al., 2021b employ trusted execution environments (TEEs) on the peers (for local training) and on the servers (for secure aggregation) in order to achieve accurate and privacy-preserving FL. However, the limited memory size of current TEEs makes them impractical for large DL models or large-scale FL systems. Besides, the authors of those papers do not consider data poisoning attacks such as label-flipping attacks, and assume trust in the manufacturers of the TEEs, which seems too strong an assumption.

3.6 Summary

Federated learning offers a promising solution for training ML models on decentralized data. However, it is vulnerable to a variety of poisoning and privacy attacks.

Existing defenses against poisoning attacks impose a high computational cost on the server, adopt specific assumptions about the peers' data distribution or the fraction of malicious peers, or are ill-suited for high-dimensional models.

On the other side, existing defenses against privacy attacks have different trade-offs between accuracy, privacy, and security, and impose significant computation and communication overheads on the participating entities.

Chapter 4

Efficient detection of Byzantine attacks in federated learning using last-layer biases

The computational overhead at the server side can become unaffordable as the model size and the number of peers involved in FL increase if *all* the model parameters are analyzed to detect Byzantine attacks.

In this chapter, we propose a new efficient method to detect Byzantine attacks in FL with a low computational overhead on the server side. The method analyzes only the biases of the last layer of deep learning models to detect and eliminate poisoned updates at each training round. We test our method on two data sets (MNIST and CIFAR-10) using two different DL model architectures and compare it with three state-of-the-art methods. Our results show that the proposed approach achieves similar attack detection performance as the best baseline (multi-Krum, Blanchard et al., 2017), while significantly reducing the runtime required for update verification and aggregation at each training round.

The contributions in this chapter have been published in Jebreel et al., 2020.

The chapter is structured as follows. Section 4.1 introduces preliminary notions and formalizes the threat model being considered. Section 4.2 details our method to defend against Byzantine attacks. Section 4.3 describes and reports the results of our experiments. Finally, Section 4.4 gathers conclusions.

4.1 Preliminaries

4.1.1 Bias in neural networks

The biological neuron is known to fire only when its processed input exceeds a certain threshold value. The same mechanism is present in artificial neural networks (Leshno et al., 1993), where the bias (a.k.a. threshold) is used to control the triggering of the activation function, i.e, it is used to tune the output along with the weighted sum of the inputs to the neuron, as shown in Figure 4.1. Thus, the bias helps the model in a way that it can best fit the given data.

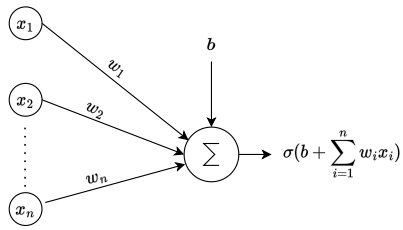
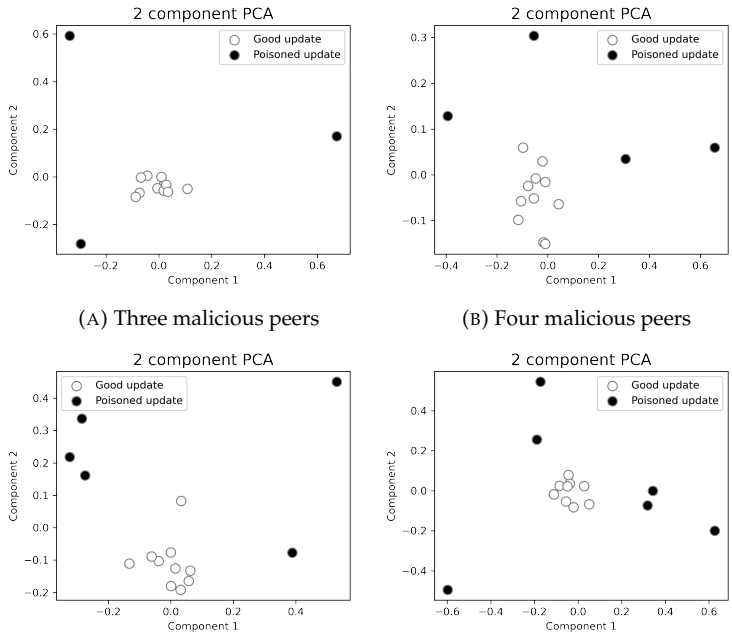


FIGURE 4.1: Illustration of an artificial neuron, where x_1 to x_n are inputs, w_1 to w_n are weights, b is a bias, and the application of an activation function σ to the weighted sum produces the neuron's output.

Accordingly, biases for models sharing the same architecture and trained by several peers are assumed to be similar. This assumption

is based on the similarity of the distribution of the local private data of each peer, and on the identical structure of the model and the hyper-parameters used during the training to optimize the model parameters. In our work, we take advantage of the similarity of the last-layer biases of the honest peers' neural networks, which we use to differentiate malicious peers from honest ones. This is illustrated in Figure 4.2, which shows how the biases for honest peers are close together, while the biases of malicious peers providing random updates follow a random and uncorrelated pattern.



(A) Three malicious peers (B) Four malicious peers

(C) Five malicious peers (D) Six malicious peers

FIGURE 4.2: Graphical representation of last-layer biases

We assume that, if there is a set $\{W_1, W_2, \ldots, W_K\}$ of peers' model updates, there is a last layer L in each W_k that contains $|\mathcal{C}|$ neurons and

a bias vector $b_k^L \in \mathbb{R}^{|\mathcal{C}|}$. Then, let B^L be the set that contains all the last-layer bias vectors $b_1^L, b_2^L, \ldots, b_K^L$ of the K peers' updates.

4.1.2 Geometric median

For a given set of m points $x_1, x_2, ..., x_m$ with $x_i \in \mathbb{R}^n$, the geometric median is defined as

$$\underset{y \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - y\|_2 , \tag{4.1}$$

where the value of the argument y is the point that minimizes the sum of Euclidean distances from the x_i's to y. The geometric median does not change if up to half of the sample data are arbitrarily corrupted (Lopuhaa, 1989). Therefore, it provides a robust way for identifying the biases of honest peers, because the geometric median will be closer to the majority who are honest.

4.1.3 Threat model

We consider a number of Byzantine peers $K' < \frac{K}{2} - 1$, which is consistent with the Byzantine resilience property. The property states that a method is considered to be Byzantine-resilient if the inequality $(2K' + 2) < K$ is satisfied (Blanchard et al., 2017). We assume the attackers to perform the random Gaussian noise poisoning attack (Li et al., 2019b; Wu et al., 2020b) to degrade the global model's performance and prevent it from converging. Furthermore, we assume the FL server to be honest and not compromised, and the attackers to have no control over the aggregator or the honest peers.

4.2 Efficient detection of Byzantine attacks

We consider the typical FL scenario, described in Section 2.2, where there is one server who uses the *FedAvg* aggregation method (McMahan

et al., 2017a) to coordinate the training process of K peers. The peers cooperatively train a shared global model W by using their local data. A fraction of such peers may be malicious. Honest peers compute their updates correctly and send the result to the server. However, malicious peers may deviate from the prescribed protocols and send random updates aiming at degrading the global model.

The key idea of our approach is that, at each global training round, the server assigns a score to each participating peer and uses that score to consider or not the peer's update at the aggregation phase. Unlike state-of-art methods, we do not analyze the entire peers' updates. Instead, we focus only on the last-layer biases of peers' updates to filter out malicious updates.

Algorithm 1 formalizes the methodology we propose. The algorithm has a set of hyper-parameters K, C, BS, E, η and τ, where τ is the confidence value used to detect malicious peers (see Section 2.2 about the meaning of the other hyperparameters).

At the first global training round, the server starts a federated learning task by initializing the global model W^0. Then it selects a random set S of m peers, where $m = max(C \cdot K, 1)$, transfers W^t to each peer in S, and asks them to train the model (locally) using the defined hyper-parameters. Each peer trains the model using her private local data and transfers the updates W_k^{t+1} to the server, who aggregates updates to create a new global model for the next training round W^{t+1}.

The server then decides whether a peer is honest or malicious by using a scoring function. The scoring function **GetScores** receives the set of last-layer biases B^L and computes its geometric median $GeoMed$. After that, it finds the set of distances $Dist$ between the $GeoMed$ and each b_k^L. Then, the server sorts $Dist$ and computes $Q1, Q3, IQR = Q3 - Q1$. Next, by using the hyper-parameter τ, the scoring function assigns 1 to a peer k if $dist_k \leq Q3 + \tau \times IQR$; else 0.

Once the server has obtained the set *Scores*, it performs the aggregation phase of updates W_k^{t+1} for each $k \in S$ such that $score_k = 1$. We propose replacing Equation (2.1) in *FedAvg* by $W^{t+1} = \sum_{k \in S, score_k=1} \frac{d_k}{sd} W_k^{t+1}$, where $sd = \sum_{k \in S, score_k=1} d_k$. Note that, since $\sum_{k \in S, score_k=1} \frac{d_d}{sd} = 1$, the convergence of the proposed averaging procedure at the server side is guaranteed as long as *FedAvg* converges.

4.3 Experiments

In this section, we empirically evaluate the effectiveness of our method and compare it with several state-of-the-art methods.

4.3.1 Experimental setup

Experiments have been carried out on two public data sets: MNIST and CIFAR-10. For both data sets, we divided the training data set into 100 equally sized shards, which were distributed among 100 simulated peers.

For MNIST, we used a small DL model with 21,840 trainable parameters. The earlier layers were convolutional layers that extracted the features and passed them to the last two fully connected layers: fully connected layer 1 (FC1), and fully connected layer 2 (FC2). FC1 contained 50 neurons with 1 bias for each neuron, while FC2 contained 10 neurons with 1 bias for each neuron. The output of FC2 was fed to a SoftMax layer that introduced the final probabilities of the 10 classes. For CIFAR-10, we used a large DL model that contained 14,728,266 trainable parameters in total. The earlier layers extracted the features and they had only one fully connected layer (FC1) that contained 10 neurons with 1 bias for each neuron. The output of the fully connected layer was fed to a softmax layer that introduced the final probabilities of the 10 classes.

We trained the small DL model for 30 global rounds. Each peer trained the model locally using the stochastic gradient descent (SGD)

with 5 local epochs, local batch size = 20, learning rate = 0.001, and momentum = 0.9. We trained the large DL model for 60 global rounds. Each peer trained the model locally using the stochastic gradient descent (SGD) with 5 local epochs, local batch size = 50, learning rate = 0.01 and momentum = 0.9.

For both models, we used the precision as a performance metric. Precision is the amount of true positives TP divided by the sum of true positives and false positives FP, that is $TP/(TP + FP)$.

In the experiments the server selected a fraction $C = 15\%$ of peers at each round. We limited the number of malicious updates per round between 1 to 6 to keep $m' < m/2 - 1$.

4.3.2 Experimental evaluation

Figure 4.3 shows the evolution of the classification precision achieved by the small DL model at each round when no detection is performed. The dashed line shows the precision when all the peers acted honestly. The global model achieves a precision around 96% after 30 rounds. In addition to the all-honest case, we considered different scenarios where a fraction of the peers were malicious. In our experiments, malicious peers trained their model honestly and, after that, they added some random noise to each parameter in their model from a Gaussian distribution with mean 0 and standard deviation 0.5 for the small DL model, and with mean of 0 and standard deviation 0.2 for the large DL model. It is noticeable that the precision of the model decreases as the number of malicious updates per round increases.

We next considered training with detection of malicious peers. We chose τ to be -0.5 in this case, that is, we consider each peer with a distance greater than the median to be malicious. We compared the results of our method with three state-of-art methods: Median, Krum, and Multi-Krum. Figure 4.4 shows the precision of the small DL model when subjected to up to 6 random updates per round. We can see that

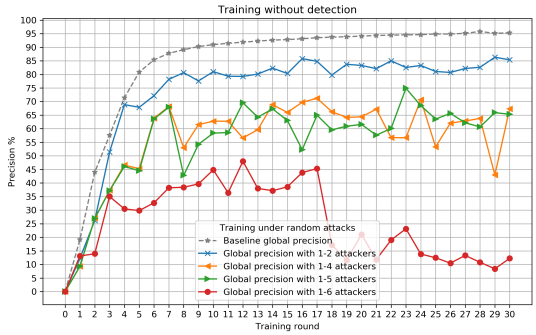


FIGURE 4.3: Evolution of the small DL model precision with MNIST under random attacks when no detection is performed

both Median and Krum have similar performance, and that both Multi-Krum and our method outperform them and obtain virtually identical results. The reason behind this identical performance is that our method takes the most honest set of updates, like Multi-Krum.

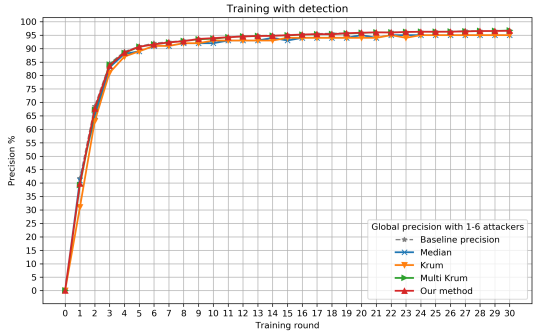


FIGURE 4.4: Evolution of the small DL model precision with MNIST under random attacks with detection

In Figure 4.5 we show the performance of our method with the large DL model and CIFAR-10 w.r.t. the values of τ. The dashed line shows the baseline precision when all peers acted honestly, which is around 84% after 60 rounds. With $\tau = 1$ our method filtered out most of the malicious peers when they were far enough from the majority. However, we can see that between training rounds 22 and 31 some malicious peers were able to escape detection, and that their effect on the model precision was devastating. With $\tau = 0.5$ most malicious peers were eliminated and with $\tau = -0.5$, all of them were eliminated.

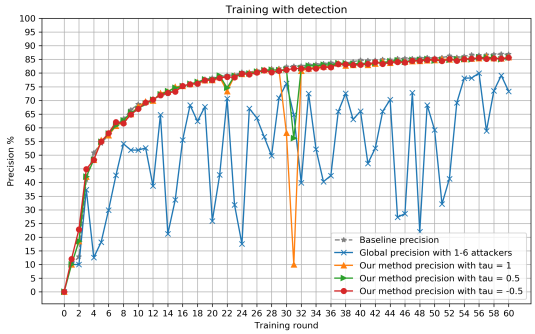


FIGURE 4.5: Evolution of the large DL model precision with CIFAR-10 under random attacks with detection

Even more relevant, Figure 4.6 shows that, in addition to our method offering detection as good as state-of-the-art methods, it is also significantly more *computationally efficient* than its counterparts. The reason is that for each peer our method only focuses on the last-layer parameters.

4.4 Conclusions

We have shown that, by focusing on a specific aspect of the learning model (biases), we can efficiently detect Byzantine attacks in FL. As

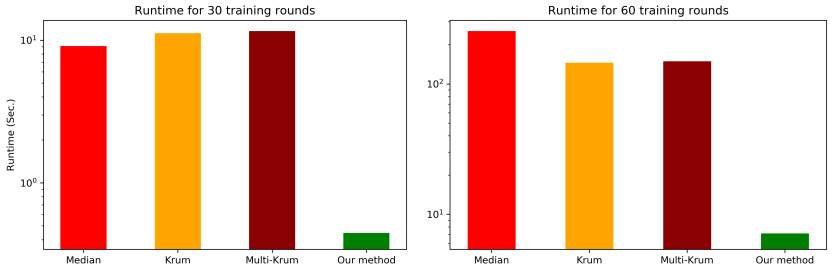


FIGURE 4.6: Runtime (in seconds, log scale) for each method: small DL model on the left and large DL model on the right.

shown in the experiments, our method achieves state-of-the-art accuracy while significantly decreasing the runtime of the attack detection at the server side. The scalability of our method is particularly useful when using large DL models with millions of parameters and peers. Although our results are promising, choosing the value of τ is challenging. The increase or the decrease of the value of τ must depend on the proportion of the malicious peers in the system. The higher the expected number of malicious peers the smaller the value of τ.

Protocol 1: Byzantine-attack robust aggregation of updates using last layer biases scores.

Input: K, C, BS, E, η, τ
initialize W^0
for *each round* $t = 0, 1, \ldots$ **do**
  $m \leftarrow \max([C \cdot K], 1)$
  S = random set of m peers.
  for *each peer* $k \in S$ ***in parallel*** **do**
    W_k^{t+1} = `peerUpdate`(k, W^t)
  end
  $Scores$ = `GetScores`(S)
  $W^{t+1} = \sum_{k \in S, score_k=1} \frac{d_k}{sd} W_k^{t+1}$ where $sd = \sum_{k \in S, score_k=1} d_k$
end
Function *peerUpdate*(k, W^t)
  $W \leftarrow W^t$
  for *each local epoch* $i = 1, 2, \ldots E$ **do**
    for *each batch* β *of size* BS **do**
      $W \leftarrow W - \eta \nabla L(W, \beta)$
    end
  end
  return W
end
Function *GetScores(S)*
  Let B^L be the set of biases of the last layers for each peer $k \in S$
  $GeoMed = GeometricMedian(B^L)$
  $Dist$ = the distances between the $GeoMed$ and each $b_k^L \in B^L$
  Compute $Q1, Q3$ of $Dist$
  $IQR = Q3 - Q1$
  $Scores$ = [ ]
  for *each* $k \in S$ **do**
    if $dist_k \leq Q3 + \tau \times IQR$ **then**
      Add(1, Scores)
    else
      Add(0, Scores)
    end
  end
  return Scores
end

Chapter 5

LFighter: defending against the label-flipping attack in federated learning

Although it is easy to perform, the *label-flipping attack* has a significantly negative impact on the source class accuracy and, sometimes, on the overall model accuracy (Tolpegin et al., 2020). Moreover, the attack's impact increases as the number of attackers and their flipped examples increase (Steinhardt, Koh, and Liang, 2017; Tolpegin et al., 2020). Furthermore, as the dimensionality of the model increases and the local data of peers become more diverse, detecting the attack becomes even more difficult.

In this chapter, we present LFighter, a novel defense against the LF attack that is effective regardless of the peers' data distribution or the model dimensionality. Specifically, we make the following contributions:

- We conduct in-depth conceptual and empirical analyses of the attack behavior, and we find a useful pattern that helps to better discriminate between the attackers' poisoned updates and the honest peers' good updates. Specifically, we find that the contradictory

objectives of attackers and honest peers on the source class examples are reflected in the parameters' gradients connected to the source and target class neurons in the output layer, making those gradients good discriminative features for attack detection. Moreover, we observe that those features stay robust under different data distributions and model sizes.

- We propose a novel defense that dynamically extracts the potential source and target classes' gradients from the peers' local updates, clusters those gradients, and analyzes the resulting clusters to filter out potentially poisoned updates before model aggregation.
- We demonstrate the effectiveness of our defense against the LF attack through an extensive empirical analysis on three data sets with different deep learning model sizes, peers' local data distributions, and ratios of attackers. In addition, we compare our approach with several state-of-the-art defenses and show its superiority at simultaneously delivering low test error, high overall accuracy, high source class accuracy, low attack success rate, and stability of the source class accuracy.

The contributions in this chapter have recently been submitted to a top journal.

This chapter is organized as follows. Section 5.1 introduces the threat model being considered and the assumptions on the local data distribution. Section 5.2 provides theoretical and empirical analyses of the label-flipping attack, and details the experimental setup. Section 5.3 presents the methodology of the proposed defense. Section 5.4 experimentally evaluates the robustness and performance of LFighter, and compares it with several state-of-the-art methods. Conclusions are gathered in Section 5.5.

5.1 Assumptions and threat model

Assumptions on training data distribution. Since the local data of the peers can come from heterogeneous sources (Bonawitz et al., 2019; Wang et al., 2019), they may be either identically distributed (IID) or non-IID. In the IID setting, each peer holds local data representing the whole distribution, which makes the locally computed gradient an unbiased estimator of the mean of all the peers' gradients. The IID setting requires each peer to have examples of all the classes in a similar proportion as the other peers. In the non-IID setting, the distributions of the peers' local data sets can be different in terms of the classes represented in the data and/or the number of examples each peer has of each class. We assume that the peers' training data distributions may range from non-IID to pure IID.

Threat model. We consider an attacker or a coalition of K' attackers, with $K' < K/2$ (see Section 5.3 for justification). The K' attackers perform the LF attack by flipping their training examples labeled c_{src} to a chosen target class c_{target} before training their local models. Furthermore, we assume the aggregator to be honest and non-compromised and the attacker(s) to have no control over the aggregator or the honest peers. The attackers' goal is to make the global model classify the examples belonging to c_{src} as c_{target} at test time and to degrade as much as possible the performance of the global model on the source class examples.

5.2 Analysis of the label-flipping attack

The effectiveness of any defense against the LF attack depends on its ability to distinguish good updates sent by honest peers from bad updates sent by attackers. In this section, we conduct comprehensive theoretical and empirical analyses of the attack behavior to find a discriminative pattern that better differentiates good updates from bad ones.

5.2.1 Theoretical analysis of the LF attack

To understand the behavior of the LF attack from an analytical perspective, let us consider a classification task where each local model is trained with the *cross-entropy* loss over one-hot encoded labels. First, the vector o of the output layer neurons (*i.e.*, the logits) is fed into the *softmax* function to compute the vector p of probabilities as

$$p_k = \frac{e^{o_k}}{\sum_{j=1}^{|\mathcal{C}|} e^{o_j}}, \quad k = 1, \ldots, |\mathcal{C}|.$$

Then, the loss is computed as

$$\mathcal{L}(y, p) = -\sum_{k=1}^{|\mathcal{C}|} y_k \log(p_k),$$

where $y = (y_1, y_2, \ldots, y_{|\mathcal{C}|})$ is the corresponding one-hot encoded true label and p_k denotes the confidence score predicted for the k^{th} class. After that, the gradient of the loss w.r.t. the output o_i of the i^{th} neuron (*i.e.*, the i^{th} neuron error) in the output layer is computed as

$$\delta_i = \frac{\partial \mathcal{L}(y, p)}{\partial o_i}$$

$$= -\sum_{j=1}^{|\mathcal{C}|} \frac{\partial \mathcal{L}(y, p)}{\partial p_j} \frac{\partial p_j}{\partial o_i} = -\frac{\partial \mathcal{L}(y, p)}{\partial p_i} \frac{\partial p_i}{\partial o_i} - \sum_{j \neq i} \frac{\partial \mathcal{L}(y, p)}{\partial p_j} \frac{\partial p_j}{\partial o_i} = p_i - y_i. \quad (5.1)$$

Note that δ_i will always be in the interval $[0, 1]$ when $y_i = 0$ (for the wrong class neuron), while it will always be in the interval $[-1, 0]$ when $y_i = 1$ (for the true class neuron).

The gradient ∇b_i^L w.r.t. the bias b_i^L of the i^{th} neuron in the output layer can be written as

$$\nabla b_i^L = \frac{\partial \mathcal{L}(y, p)}{\partial b_i^L} = \frac{\partial \mathcal{L}(y, p)}{\partial o_i} \frac{\partial o_i}{\partial b_i^L} = \delta_i \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}, \quad (5.2)$$

where a^{L-1} is the activation output of the previous layer $L-1$.

Likewise, the gradient ∇w_i^L w.r.t. the weights vector w_i^L connected to the i^{th} neuron in the output layer is

$$\nabla w_i^L = \frac{\partial \mathcal{L}(y,p)}{\partial w_i^L} = \frac{\partial \mathcal{L}(y,p)}{\partial o_i}\frac{\partial o_i}{\partial w_i^L} = \delta_i a^{L-1}\frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}. \tag{5.3}$$

From Equations (5.2) and (5.3), we can notice that δ_i directly and highly impacts on the gradients of its connected parameters. For example, for the ReLU activation function, which is widely used in DL models, we get

$$\nabla b_i^L = \begin{cases} \delta_i, & \text{if } (w_i^L \cdot a^{L-1} + b_i^L) > 0; \\ 0, & \text{otherwise;} \end{cases}$$

and

$$\nabla w_i^L = \begin{cases} \delta_i a^{L-1}, & \text{if } (w_i^L \cdot a^{L-1} + b_i^L) > 0; \\ 0, & \text{otherwise.} \end{cases}$$

The objective of the attackers is to minimize $p_{c_{src}}$ and maximize $p_{c_{target}}$ for their c_{src} examples, whereas the objective of honest peers is exactly the opposite. We notice from Expressions (5.1), (5.2), and (5.3) that these contradicting objectives will be reflected on the gradients of the parameters connected to the *relevant* source and target output neurons. For convenience, in this work, we use the term *relevant neurons' gradients* instead of the gradients of the parameters connected to the source and target output neurons. Also, we use the term *non-relevant neurons' gradients* instead of the gradients of the parameters connected to the neurons different from source and target output neurons. As a result, as the training evolves, the magnitudes of the relevant neurons' gradients are expected to be larger than those of the non-relevant and non-contradicting neurons. Also, the angle between the relevant neurons' gradients for an honest peer and an attacker is expected to be larger than those of the

non-relevant neurons' gradients. That is because the error of the non-relevant neurons will diminish as the global model training evolves, especially when it starts converging because honest and malicious participants share the same training objectives for non-targeted classes. On the other hand, the relevant neurons' errors will stay large during model training because of the contradicting objectives. Therefore, the relevant neurons' gradients are expected to carry a more valuable and discriminative pattern for attack detection than the whole model gradients or the output layer gradients, which carry less relevant information for the attack.

5.2.2 Experimental setup

This section describes the used data sets and models, and the training and attack settings.

Data distribution and training. We defined the following benchmarks by distributing the data from the data sets described in Chapter 2 among the participating peers in the following way:

- MNIST-CNN-IID. We randomly and uniformly divided the MNIST training data among 100 participating peers to generate IID data. The CNN model was trained for 200 iterations. In each iteration, the FL server asked the peers to train their models for 3 local epochs and a local batch size of 64. The participants used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with a learning rate = 0.001 and momentum = 0.9 to train their models.

- MNIST-CNN-non-IID. We adopted a Dirichlet distribution (Minka, 2000) with a hyperparameter $\alpha = 1$ to generate *non-IID* data for 100 participating peers. The training settings and hyper-parameters were the same as for MNIST-CNN-IID.

- CIFAR10-ResNet18-IID. We randomly and uniformly divided the CIFAR10 training data among 20 participating peers to generate IID data. The ResNet18 model was trained during 100 iterations. In each iteration, the FL server asked the 20 peers to train the model for 3 local epochs and a local batch size 32. The peers used the cross-entropy loss function and the SGD optimizer with a learning rate = 0.01 and momentum = 0.9.
- CIFAR10-ResNet18-non-IID. We adopted a Dirichlet distribution (Minka, 2000) with a hyperparameter $\alpha = 1$ to generate *non-IID* data for 20 participating peers. The training settings were the same as for the CIFAR10-ResNet18-IID benchmark.
- CIFAR10-ShuffleNetV2-IID. We adopted the same training data distribution and training settings for ShuffleNetV2 as in CIFAR10-ResNet18-IID. The only difference was the learning rate = 0.001.
- IMDB-BiLSTM. We randomly and uniformly split the $40K$ training examples among 20 peers. The BiLSTM model was trained during 50 iterations. In each iteration, the FL server asked the 20 peers to train the model for 1 local epoch and a local batch size of 32. The peers used the binary cross-entropy with logit loss function and the *Adam* optimizer with learning rate = 0.001.

Attack scenarios. With the MNIST benchmarks, the attackers flipped the examples with the source class 7 to the target class 1. With CIFAR10 benchmarks, the attackers flipped the examples with the label *Dog* to *Cat* before training their local models, whereas for IMDB, the attackers flipped the examples with the label *positive* to *negative*.

5.2.3 Empirical analysis of the LF attack

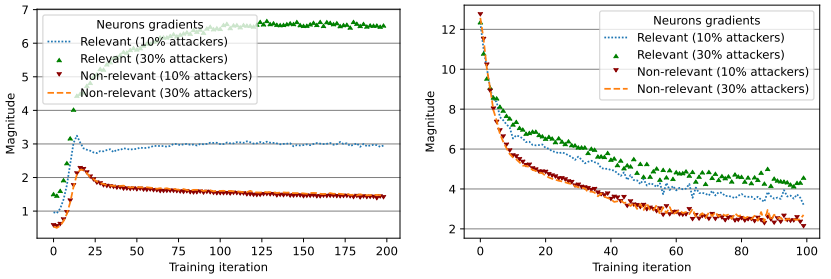
To empirically validate the analytical findings discussed in Section 5.2.1, we performed the following experiment with the CIFAR10-ResNet18-IID benchmark. First, a chosen peer trained her local model on her data

honestly, which yielded a good update. Then, the same peer flipped the labels of the source class *Cat* to the target class *Dog* and then trained her local model on the poisoned training data, which yielded a bad update. After that, we computed the magnitudes of and the angle between i) the whole updates, ii) the output layer gradients, iii) the relevant gradients related to c_{src} and c_{target}. Table 5.1 shows the obtained results, which confirm our analytical findings. It is clear that both whole gradients had approximately the same magnitude, and the angle between them was close to zero. On the other hand, the difference between the output layer gradients was large and even more significant in the case of the relevant neurons' gradients. As for non-relevant neurons, their gradients' magnitude and angle were not significantly affected because, in such neurons, there was no contradiction between the objectives of the good and the bad updates.

TABLE 5.1: Comparison of the magnitudes and the angle of the gradients of a good and a bad update for the whole update, the output layer parameters, the parameters of the relevant source and target neurons, and the parameters of the non-relevant neurons.

Gradients		Whole	Output layer	Relevant neurons	Non-relevant neurons
Magnitude	Good	351123	72.94	23.38	55.30
	Bad	351107	100.23	64.43	65.95
Angle		0.41	69.19	115	18

To underscore this point and check how the gradients of the non-relevant neurons vanish as the training evolves while the gradients of the relevant neurons remain larger, we show the gradients' magnitudes during training in Figure 5.1. The magnitudes of those gradients for the MNIST-CNN-IID and the CIFAR10-ResNet18-non-IID benchmarks are shown for ratios of attackers 10% and 30%. We can see that although the attackers' ratio and the data distribution had an impact on the magnitudes of those gradients, the gradients' magnitudes for the relevant source and target class neurons always remained larger.



(A) MNIST-CNN-IID (B) CIFAR10-ResNet18-non-IID

FIGURE 5.1: Gradient magnitudes during training for relevant and non-relevant neurons

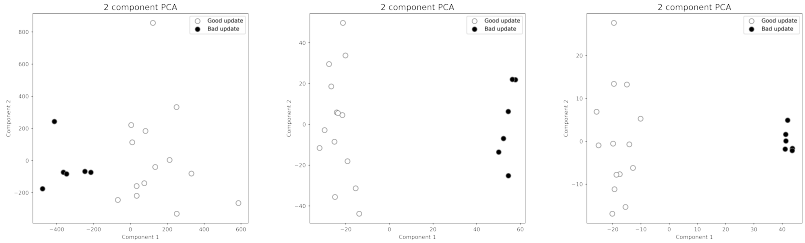
We also studied the impact of the model dimensionality and the local data distribution on differentiating between good updates and bad updates. We used exploratory analysis to visualize the local gradients resulting from simulating an FL scenario under the LF attack (with a ratio of attackers = 30%) for some of the benchmarks defined in Section 5.2.2. Besides the whole gradients, we visualized the output layer's gradients and the relevant neurons' gradients. We used Principal Component Analysis (PCA) on the selected gradients and plotted the first two principal components. We next report what we observed.

1) **Impact of the model dimensionality**. Figures 5.2 and 5.3 show the gradients of whole local updates, the gradients corresponding to the output layers, and the relevant gradients corresponding to the source and target neurons from the CIFAR10-ShuffleNetV2-IID and CIFAR10-ResNet18-IID benchmarks.

The figures show that when the model size is small (ShuffleNetV2), good and bad updates can be separated, whichever set of gradients is considered. On the other hand, when the model size is large (ResNet18), the attack's influence does not seem to be enough to distinguish good updates from bad ones if using whole updates' gradients; yet, the gradients of the output layer or those of the relevant neurons still allow for

a crisp differentiation between good and bad updates.

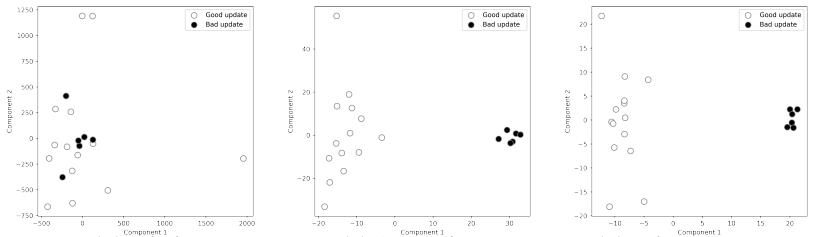
In fact, several factors make it challenging to detect LF attacks by analyzing an entire high-dimensional update. First, the computed errors for the neurons in a certain layer depend on all the errors for the neurons in the subsequent layers and their connected weights (Rumelhart, Hinton, and Williams, 1986). Thus, as the model size gets larger, the impact of the attack is mixed with that of the non-relevant neurons. Second, the early layers of DL models usually extract common features that are not class-specific (Nasr, Shokri, and Houmansadr, 2019). Finally, in general, most parameters in DL models are redundant (Denil et al., 2013). These factors cause the magnitudes of the whole gradients of good and bad updates and the angles between them to be similar, thereby making DL models with large dimensions an ideal environment for a successful label-flipping attack.



(A) Whole (B) Output layer (C) Relevant neurons

FIGURE 5.2: CIFAR10-ShuffleNetV2-IID benchmark gradients

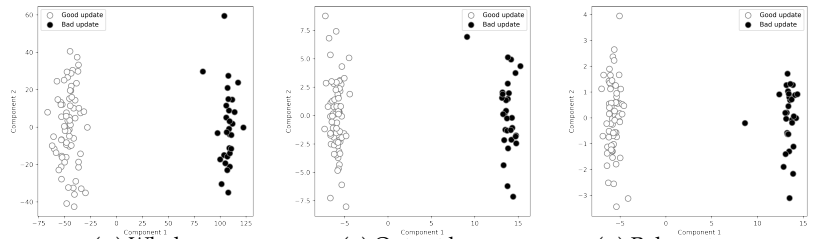
2) **Impact of the data distribution**. Figures 5.4 and 5.5 show the gradients of the whole local updates, the gradients corresponding to the output layers, and the relevant gradients corresponding to the source and target neurons from the MNIST-CNN-IID and MNIST-CNN-non-IID benchmarks. Figure 5.5 shows that, despite the model used for the MNIST-non-IID benchmark being small, distinguishing between good and bad updates was harder than in the IID setting shown in Figure 5.4.



(A) Whole (B) Output layer (C) Relevant neurons

FIGURE 5.3: CIFAR10-IID benchmark gradients

It also shows that using the relevant neurons' gradients provided the best separation compared to whole update gradients or output layer gradients.

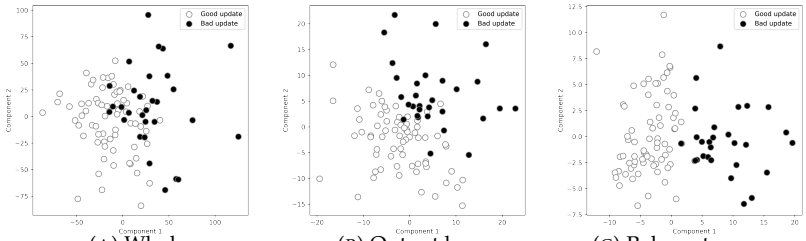


(A) Whole (B) Output layer (C) Relevant neurons

FIGURE 5.4: MNIST-CNN-IID benchmark gradients

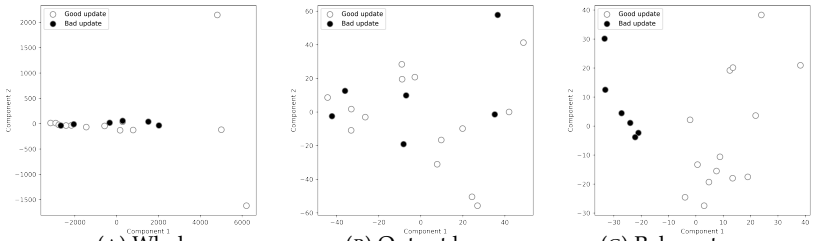
Figure 5.6 shows that the combined impact of model size and the data distribution in the CIFAR-ResNet18-non-IID benchmark made it very challenging to separate bad updates from good ones using the whole update gradients or even using the output layer gradients. On the other hand, the relevant neurons' gradients gave a clearer separation.

From the previous results, we can observe that analyzing the gradients of the parameters connected to the source and target class neurons led to better discrimination between good updates and bad ones for both



(A) Whole (B) Output layer (C) Relevant neurons

FIGURE 5.5: MNIST-CNN-non-IID benchmark gradients



(A) Whole (B) Output layer (C) Relevant neurons

FIGURE 5.6: CIFAR10-ResNet18-non-IID benchmark gradients

the IID and the non-IID settings. We can also observe that, in general, those gradients formed two clusters: one for the good updates and another for the bad ones. Moreover, the attackers' gradients were more similar among them, which caused their clusters to be denser than the honest peers' clusters.

Based on the analyses and observations presented so far, we conclude that *an effective defense against the label-flipping attack needs to consider the following aspects:*

- Only the gradients of the parameters connected to the source and target class neurons in the output layer must be extracted and analyzed.
- The extracted gradients need to be separated into two clusters that are compared to identify which of them contains the bad updates.
- A cluster with more similar gradients is more likely to be bad.

5.3 Design of LFighter

In this section, we present our proposed defense against the label-flipping attack in federated learning systems by considering the observations and conclusions discussed in the previous section.

Unlike other defenses, our proposal does not require a prior assumption on the peers' data distribution, is not affected by model dimensionality, and does not require prior knowledge about the proportion of attackers.

We formalize our method in Algorithm 2. The aggregator server A starts a federated learning task by selecting a random set S of m peers, initializes the global model W^0, and sends it to the m selected peers. Then, each peer $k \in S$ locally trains W^t on her data D_k and sends her local update W_k^{t+1} back to A. Once A receives the m local updates, it computes their corresponding gradients as $\{\nabla W_k^t = (W^t - W_k^{t+1})/\eta | k \in$

$S\}$. After that, A separates the gradients connected to the output layer neurons to obtain the set $\{\nabla_k^{L,t} | k \in S\}$.

Protocol 2: Defending against the label-flipping attack

Input: K, C, BS, E, η, T
Output: W^T, the global model after T training iterations

A initializes W^0
for *each iteration* $t \in [0, T-1]$ **do**
  $m \leftarrow \max(C \cdot K, 3)$
  $S \leftarrow$ random set of m peers
  A sends W^t to all peers in S
  for *each peer* $k \in S$ ***in parallel*** **do**
    $W_k^{t+1} \leftarrow$ `PEER_UPDATE`(k, W^t) `// A sends` W^t `to each peer` k `who trains` W^t `using her data` D_k `locally, and sends her local update` W_k^{t+1} `back to the aggregator`
  end
  Let $\{\nabla_k^{L,t} | k \in S\}$ be the peers' output layer gradients at iteration t
  for *each peer* $k \in S$ **do**
    for *each neuron* $i \in [1, |\mathcal{C}|]$ **do**
      Let $||\nabla_{i,k}^{L,t}||$ be the magnitude of the gradients connected to the output layer neuron i of the peer k at iteration t
    end
  end
  Let $||\nabla_{i,S}^{L,t}|| = \sum_{k \in S} ||\nabla_{i,k}^{L,t}||$ `// Neuron-wise magnitude aggregation`
  Let $imax_{1,S}, imax_{1,S}$ be the neurons with the highest two magnitudes in $(||\nabla_{1,S}^{L,t}||, .., ||\nabla_{i,S}^{L,t}||, .., ||\nabla_{|\mathcal{C}|,S}^{L,t}||)$
  `// Identifying potential source and target classes`
  $bad\_peers \leftarrow$ `FILTER`$(\{\nabla_k^{L,t} | k \in S\}, imax_{1,S}, imax_{1,S})$
  A aggregates $W^{t+1} \leftarrow$ `FedAvg`$(\{W_k^{t+1} | k \notin bad\_peers\})$.
end

Identifying the potential source and target classes. After separating the gradients of the output layer, we need to identify the potential

source and target classes, which is key to our defense. As we have shown in the previous section, the magnitudes of the gradients connected to the source and target class neurons for the attackers and honest peers are expected to be larger than the magnitudes of the other non-relevant classes. Thus, we can dynamically identify the potential source and target class neurons by analyzing the magnitudes of the gradients connected to the output layer neurons. To do so, for each peer $k \in S$, we compute the neuron-wise magnitude of each output layer neuron's gradients $||\nabla_{i,k}^{L,t}||$. After computing the output layer neuron magnitudes for all peers in S, we aggregate their neuron-wise gradient magnitudes into the vector $(||\nabla_{1,S}^{L,t}||, .., ||\nabla_{i,S}^{L,t}||, .., ||\nabla_{|\mathcal{C}|,S}^{L,t}||)$. We then identify the potential source and target class neurons $imax_{1,S}$ and $imax_{2,S}$ as the two neurons with the highest two magnitudes in the aggregated vector.

Filtering bad updates. We filter out bad updates by using the `FILTER` procedure detailed in Procedure 1. First, we extract the gradients connected to the identified potential source and target classes $imax_{1,S}$ and $imax_{2,S}$ from the output layer gradients of each peer. Then, we use the k-means (Hartigan and Wong, 1979) method with $k = 2$ to group the extracted gradients into two clusters cl_1 and cl_2. Once the two clusters are formed, we need to decide which of them contains the potentially bad updates. To make this critical decision, we consider two factors: the size and the density of clusters. Specifically, we tag the smaller and/or denser cluster as potentially bad. This makes sense because, when the two clusters have similar densities, the smaller one is probably the bad one. On the other hand, if the two clusters are close in size, the denser and more homogeneous cluster is probably the bad one. This is because the higher similarity between the attackers' relevant gradients makes their cluster usually denser, as discussed in the previous section. To compute the density of a cluster, we compute the pairwise angle θ_{ij} between each pair of gradient vectors i and j in the cluster. Then, for each gradient vector i in the cluster, we find $\theta_{max,i}$ as the maximum pairwise

angle for that vector. That is because, no matter how far apart two attackers' gradients are, they will be closer to each other due to the larger similarity of their directions compared to that of two honest peers. After that, we compute the average of the maximum pairwise angles for the cluster to obtain the inverse density value *dns* of the cluster. In this way, the denser the cluster, the lower *dns* will be. After computing dns_1 and dns_2 for cl_1 and cl_2, we compute $score_1$ and $score_2$ by re-weighting the computed clusters' inverse densities proportionally to their sizes. If both clusters have similar inverse densities, the smaller cluster will probably have the lowest score or, if they have similar sizes, the denser cluster will probably have the lowest score. Finally, we use $score_1$ and $score_2$ to decide which cluster contains the potentially bad updates. We compute the set *bad_peers* as the peers in the cluster with the minimum score.

Aggregating potentially good updates. After identifying the potentially bad peers, the server A computes $\texttt{FedAvg}(\{W_k^{t+1} | k \notin bad\_peers\})$ to obtain the updated global model W^{t+1}.

5.4 Experimental evaluation

We evaluated the robustness of LFighter against the LF attack scenarios described in Section 5.2.2 and compared it with several countermeasures discussed in Chapter 3, including the standard FedAvg (McMahan et al., 2017a) aggregation method (not meant to counter poisoning attacks), the median (Yin et al., 2018), the trimmed mean (TMean) (Yin et al., 2018), multi-Krum (MKrum) (Blanchard et al., 2017), FoolsGold (FGold) (Fung, Yoon, and Beschastnikh, 2020), Tolp (Tolpegin et al., 2020) and FLAME (Nguyen et al., 2022). We used the MNIST-CNN-non-IID, the CIFAR10-ResNet-non-IID and the IMDB-BiLSTM benchmarks with the local data distribution and training settings described in 5.2.2. We report the average results of the last 10 training iterations to ensure a fair comparison among defenses.

Procedure 1: Filtering potentially bad updates

```
FILTER({∇_k^{L,t} | k ∈ S}, imax_{1,S}, imax_{1,S}):
    data ← {∇_{i,k}^{L,t} | (k ∈ S, i ∈ {imax_{1,S}, imax_{1,S}})}
    cl_1, cl_2 ← kmeans(data, num_clusters = 2)
    // Computing cluster inverse densities
    dns_1 ← CLUSTER_INVERSE_DENSITY(cl_1)
    dns_2 ← CLUSTER_INVERSE_DENSITY(cl_2)
    // Re-weighting clusters inverse densities
    score_1 = |cl_1|/(|cl_1|+|cl_2|) * dns_1
    score_2 = |cl_2|/(|cl_1|+|cl_2|) * dns_2
    if score_1 < score_2 then
        bad_peers ← {k | k ∈ cl_1}
    else
        bad_peers ← {k | k ∈ cl_2}
    return bad_peers
CLUSTER_INVERSE_DENSITY({∇_i}_{i=1}^n):
    for each ∇_i do
        for each ∇_j do
            Let θ_ij be the angle between ∇_i and ∇_j
        Let θ_{max,i} = max_j(θ_i)
    dns = (1/n) Σ_i θ_{max,i}
    return dns
```

Our code and data are available for reproducibility purposes at https://github.com/NajeebJebreel/LFighter.

Evaluation metrics. We used the following evaluation metrics on the test set examples for each benchmark to assess the impact of the LF attack on the learned model and the performance of the proposed method w.r.t. the state-of-the-art methods:

- *Test error (TE)*. Error resulting from the loss function used in training. The lower TE, the better.
- *Overall accuracy (All-Acc)*. Number of correct predictions divided by the total number of predictions for all the examples. The greater

All-Acc, the better.

- *Source class accuracy (Src-Acc)*. Number of the source class examples correctly predicted divided by the total number of the source class examples. The greater Src-Acc, the better.
- *Attack success rate (ASR)*. Proportion of the source class examples incorrectly classified as the target class. The lower ASR, the better.
- *Coefficient of variation (CV)*. Ratio of the standard deviation σ to the mean μ, that is, $CV = \frac{\sigma}{\mu}$. The lower CV, the better.

While TE, All-Acc, Src-Acc, and ASR are used in previous works to evaluate robustness against poisoning attacks (Blanchard et al., 2017; Tolpegin et al., 2020; Fung, Yoon, and Beschastnikh, 2020), we also use the CV metric to assess the stability of Src-Acc during training. We justify our choice of this metric in Section 5.4. An effective defense needs to simultaneously perform well in terms of TE, All-Acc, Src-Acc, ASR, and CV.

Robustness to the LF attack. Here, we present the results with a 40% ratio of attackers that, for $m = 20$ peers, corresponds to $m' = (m/2) - 2$ attackers. This is the theoretical upper bound of the number of attackers MKrum (Blanchard et al., 2017) can defend against. Table 5.2 shows the obtained results with the three used benchmarks.

With the MNIST-CNN-non-IID benchmark, we can see that Tolp, LFighter, MKrum and FLAME effectively defended against the attack with comparable results for all metrics. The median and trimmed mean achieved good performance in this benchmark due to the small variability of the MNIST data set, which made each local update of an honest peer an unbiased estimator of the mean of all the good local updates. It can also be seen that FGold failed to counter the attack and achieved poor performance for all the metrics. Another interesting note is that, despite not being meant to mitigate poisoning attacks, FedAvg achieved

a good performance in this benchmark. That was also observed in Shejwalkar et al., 2021, where the authors argued that, in some cases, FedAvg is more robust against poisoning attacks than many of the state-of-the-art countermeasures.

TABLE 5.2: Performance of methods under 40% attacker ratio. Best score is in bold. Second best score is underlined. NA denotes no attack is performed.

Metric	Benchmark	*FedAvg (NA)*	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	LFighter
TE	MNIST-CNN-non-IID	*0.13*	0.19	0.23	0.23	0.18	0.28	0.16	0.15	**0.13**
All-Acc%		*96.21*	95.11	93.15	93.11	94.87	86.85	96.24	**96.36**	96.30
Src-Acc%		*94.84*	83.66	81.13	80.74	91.44	0.00	**94.25**	92.04	93.68
ASR%		*0.49*	9.44	7.68	7.98	1.56	88.52	**0.58**	1.26	0.68
TE	CIFAR10-ResNet18-non-IID	*0.85*	**0.93**	0.96	0.96	1.33	0.95	1.03	1.07	0.98
All-Acc%		*75.81*	**73.32**	72.77	72.83	65.51	72.95	71.43	68.42	72.82
Src-Acc%		*65.48*	24.22	23.53	21.62	11.05	29.92	14.52	11.33	**56.98**
ASR%		*14.90*	52.93	53.74	55.61	71.62	46.00	65.23	65.21	**12.10**
TE	IMDB-BiLSTM	*0.28*	1.13	0.96	0.98	8.87	0.35	0.35	**0.31**	0.35
All-Acc%		*88.59*	56.09	59.26	58.75	49.93	**87.72**	87.51	86.08	87.57
Src-Acc%		*86.03*	12.43	18.87	17.84	0.00	**85.75**	85.42	82.71	85.44
ASR%		*13.97*	87.57	81.13	82.16	100.00	**14.25**	14.58	17.29	14.56

With the CIFAR10-ResNet18-non-IID benchmark, we see that all methods, except LFighter, performed poorly due to the combined impact of the data distribution and the model dimensionality on the differentiation between the good and bad updates. On the other hand, thanks to the rich discriminative pattern we used to distinguish between good and bad updates, LFighter was robust against the attack and largely outperformed the others in the source class accuracy and the attack success rate. Since LFighter considered only the source and target class neuron gradients –the gradients relevant to the attack– and excluded the non-relevant gradients, it was able to differentiate between the good and bad ones successfully.

With the IMDB-BiLSTN benchmark, FGold, LFighter, Tolp, and FLAME effectively defended against the attack, and largely outperformed the other methods for all the metrics. FGold and Tolp performed well in this benchmark because the number of classes in the output layer was only two; hence, all the parameters' gradients in the output layer were relevant to the attack.

To summarize, LFighter effectively defended against the label-flipping attack, outperforming several state-of-the-art defenses in the CIFAR10-ResNet18-non-IID benchmark, where the local data are non-IID and the model size is large.

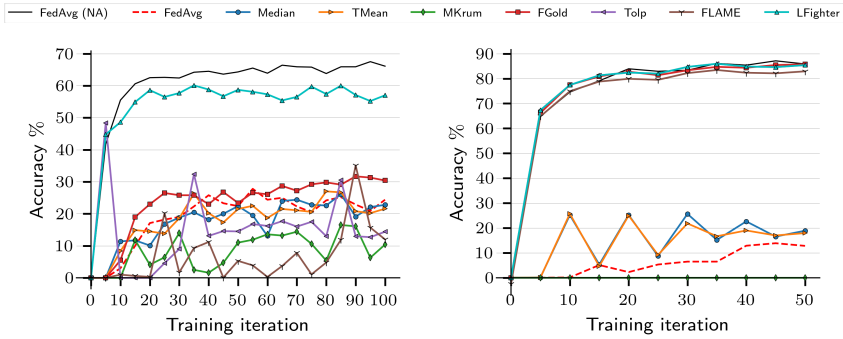
Accuracy stability. The stability of the global model convergence (and its accuracy in particular) during training is a problem in FL, especially when training data are non-IID (Li et al., 2018; Karimireddy et al., 2020). Furthermore, with an LF attack targeting a particular source class, the evolution of the accuracy of the source class becomes more unstable. Since an updated global model may be used after some intermediate training rounds (as in Hard et al., 2018), this may entail degradation of the accuracy of the source class at inference time. Keeping the accuracy stable is needed to prevent such consequences. In the following, we study this aspect by using the CV metric to measure the stability of the source class accuracy. Table 5.3 shows the CV of the accuracy of the source class in the used benchmarks for the different defense mechanisms. We can see that LFighter outperformed the other methods in most cases and achieved stability very close to that of FedAvg when the attackers' ratio was 0% (*i.e.*, absence of attack).

TABLE 5.3: Coefficient of variation (CV) of the source class accuracy during training for the considered benchmarks with 40% attacker ratio. NaN value in the table resulted from zero values of the source class accuracy in all the training rounds. The best figure in each column is shown in boldface.

Benchmark	*FedAvg (NA)*	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	LFighter
MNIST-CNN-non-IID	*0.081*	0.275	0.433	0.426	0.437	1.35	0.098	0.103	**0.097**
CIFAR10-ResNet18-non-IID	*0.142*	0.281	0.278	0.277	0.45	0.261	0.507	0.582	**0.152**
IMDB-BiLSTM	*0.102*	0.352	0.266	0.267	NaN	0.095	**0.094**	0.098	**0.094**

To provide a clearer picture of the effectiveness of our defense, Figure 5.7 shows the evolution of the accuracy of the source class as training progresses when the attacker ratio was 40% in the CIFAR10-ResNet18-non-IID and IMDB-BilSTM benchmarks. It is clear from the figure that

the accuracy achieved by our defense was the most similar to the accuracy of FedAvg when no attack was performed.

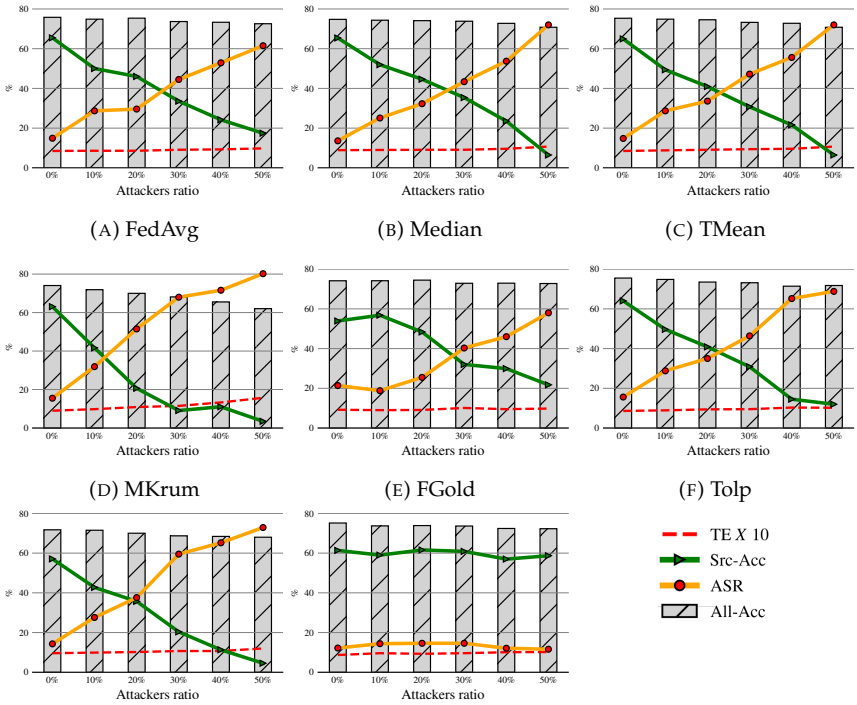


(A) CIFAR10-ResNet18-non-IID (B) IMDB-BiLSTM

FIGURE 5.7: Evolution of the source class accuracy with 40% attackers ratio

Impact of attackers' ratio. We studied the impact of the ratio of attackers on the performance of all the methods using the CIFAR10-ResNet18-non-IID benchmark and the following range of ratios: {0%, 10%, 20%, 30%, 40%, 50%}. Figure 5.8 reports the obtained results. Note that we scaled the TE by 10 to make it noticeable. It is clear that the performance of all methods, except LFighter, significantly degraded as the ratio of attackers in the system increased. In contrast, LFighter successfully excluded the bad updates from the global model aggregation. This shows that our method is robust against the attack regardless of the ratio of attackers.

Runtime. Figure 5.9 shows the runtime in seconds of each method (on the server side) for one training iteration. Excluding FedAvg, which just averages updates, our results show that FGold had the lowest runtime in all cases. On the other hand, the runtime of our method was similar to that of Tolp, which ranked second after FGold. Given the effectiveness of our method in countering the LF attack, the runtime incurred by our method can be viewed as very reasonable compared to



(A) FedAvg (B) Median (C) TMean

(D) MKrum (E) FGold (F) Tolp

(G) FLAME (H) LFighter

FIGURE 5.8: Robustness against the label-flipping attack with the CIFAR10-Mild benchmark

other methods.

Robustness to multi-label LF attack. In addition to the single-label LF attack considered so far, we also evaluated the performance of LFighter and the other methods under the multi-label LF attack (Fang et al., 2020) scenario. In this attack, each attacker flips the label of each training example from class c_i to class $c_{(i+1) \mod |\mathcal{C}|}$. For this experiment, we used the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers equal to 40%.

The results in Table 5.4 show that our method was also robust to this

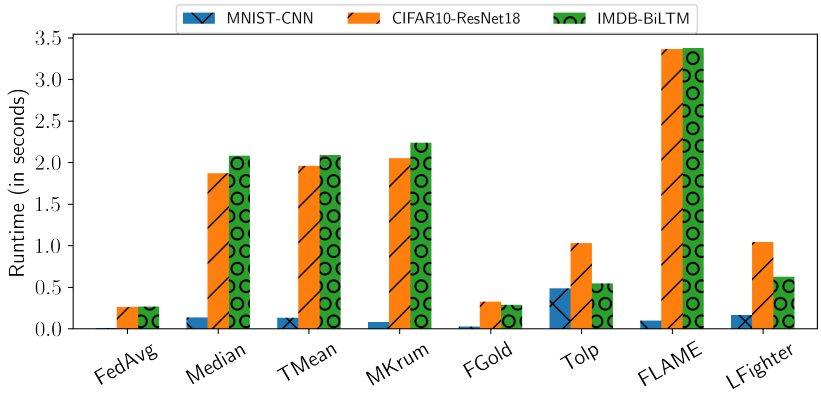


FIGURE 5.9: Runtime overhead in seconds.

attack and mitigated its influence on both test error and overall accuracy. Moreover, it outperformed all the other methods at providing the lowest test error and the highest overall accuracy. Note that this attack has a higher negative impact on the overall performance of the global model compared to the single-label LF attack considered in the rest of the chapter.

TABLE 5.4: Robustness to multi-LF attack. Best score is in bold. Second best score is underlined. NA denotes no attack is performed.

Metric	*FedAvg (NA)*	FedAvg	Median	Tmean	MKrum	FGold	Tolp	FLAME	LFighter
TE	*0.85*	1.47	1.17	1.14	1.54	1.16	1.09	1.19	**0.94**
All-Acc%	*75.81*	62.61	62.72	64.30	48.88	63.03	66.72	61.81	**72.32**

Impact of feature similarity between the source and the target classes. We report here the impact of the feature similarity between the source and the target class samples on the performance of both the LF attack and our defense. We used the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers equal to 40%. The following three attack scenarios were implemented:

1. Source and target classes with high feature similarity: the attackers flipped the examples with the label *Dog* to *Cat*.
2. Source and target classes with moderate feature similarity: the attackers flipped the examples with the label *Truck* to *Car*.
3. Source and target classes with high feature dissimilarity: the attackers flipped the examples with the label *Frog* to *Plane*.

Results reported in Table 6.6 reveal that the higher the feature similarity between the two classes, the more effective the attack is. This is because, when the features of the two classes are more similar, their decision boundaries overlap more, which makes the attacker's task easier. This also explains why ASR was about 15% in the *Dog-Cat* scenario in the absence of attacks. Nevertheless, our defense was effective in all scenarios and achieved a similar ASR to FedAvg when no attack was present.

TABLE 5.5: Impact of feature similarity between the source and target class

Metric	Method	Source-Target		
		Dog-Cat	Truck-Car	Frog-Plane
All-Acc%	*FedAvg (NA)*	*75.81*	*75.81*	*75.81*
	FedAvg	73.32	73.78	75.21
	LFighter	72.82	72.90	74.62
Src-Acc%	*FedAvg (NA)*	*65.48*	*81.62*	*87.40*
	FedAvg	24.22	56.75	75.10
	LFighter	56.98	78.73	81.80
ASR%	*FedAvg (NA)*	*14.90*	*8.41*	*0.50*
	FedAvg	52.93	29.90	6.10
	LFighter	12.10	8.50	0.70

Impact of the local data distribution. We studied the impact of the local data distribution on the performance of our defense using the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers = 40%. We generated local data for 20 peers with different degrees of non-IIDness

using different values of the Dirichlet distribution parameter $\alpha \in \{0.5,0.7,1,5,10,1000\}$. Figure 5.10 shows the per quantity and per class data distribution of the first 10 peers. We selected just 10 random peers to make it easier to read the figure. It can be seen that lower values of α correspond to higher non-IIDness.

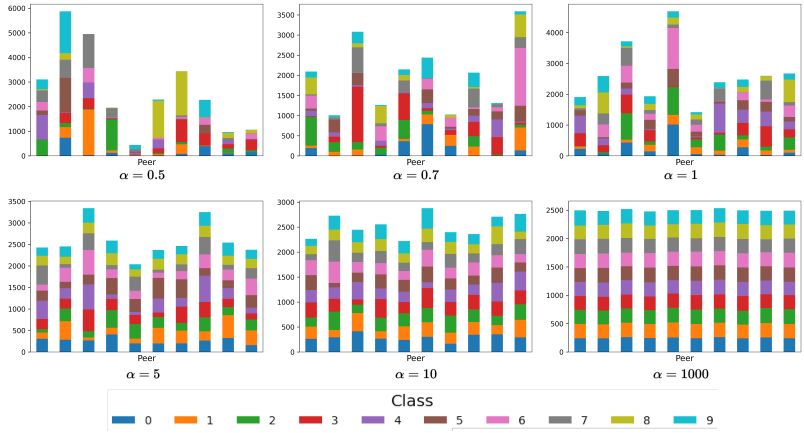


FIGURE 5.10: Local data distributions of 10 peers generated using the Dirichlet distribution with different α values using the CIFAR10 training data.

Table 5.6 shows that high non-IIDness of the peers' local data negatively impacted on the overall accuracy of the final global model. In fact, the non-IIDness of the local data of peers is a challenge to the performance and convergence of FL, as it has been observed in several FL studies such as Zhao et al., 2018. It can also be seen that higher non-IIDness (lower α) of the local data distribution corresponds to higher LF ASRs. This is because the larger divergence in local updates caused by high non-IIDness makes it more challenging to differentiate between good and poisoned updates. Nevertheless, it appears that our defense was able to effectively mitigate the impact of the attack without significantly reducing accuracy on the main task.

TABLE 5.6: Impact of the non-IIDness of the local data

Metric	Method	α					
		0.5	0.7	1	5	10	1K
All-Acc%	*FedAvg (NA)*	*74.22*	*74.57*	*75.81*	*76.41*	*76.44*	*77.87*
	FedAvg	73.06	72.56	73.32	73.06	74.72	75.23
	LFighter	72.07	72.34	72.82	74.03	74.71	75.31
Src-Acc%	*FedAvg (NA)*	*60.61*	*61.72*	*65.48*	*64.81*	*65.23*	*67.13*
	FedAvg	21.32	23.63	24.22	21.32	24.22	24.44
	LFighter	54.14	56.83	56.98	59.24	61.32	63.15
ASR%	*FedAvg (NA)*	*10.91*	*12.54*	*14.92*	*16.63*	*15.32*	*14.57*
	FedAvg	57.30	54.62	52.93	57.33	54.14	54.18
	LFighter	14.81	14.62	12.10	12.11	11.93	11.97

Impact of the local learning rate. The learning rate is one of the most important hyper-parameters affecting the performance of DL models in both centralized and federated learning. Choosing an appropriate value for the learning rate depends on several factors, such as model dimensionality and training data complexity (Roy, 1994). We used the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers = 40% and different learning rates. Table 5.7 shows that larger learning rate values (up to 0.05) led to better global model performance. We can also note that, with larger learning rates, which resulted in better global model performance, our method defended better against the attack.

Impact of the local batch size. The batch size determines how many training examples the model sees at once during training. Larger batch sizes can provide computational speed-ups but usually lead to poor generalization. The reason for this is not fully understood in the field (Keskar et al., 2016). We studied the impact of the local batch size on the performance of our defense using the CIFAR10-ResNet18-non-IID benchmark with a ratio of attackers = 40% and different batch sizes. Table 5.8 shows that smaller batch sizes correspond to better global model performance in the used benchmark. Also, as long as the global performance was good, our method defended effectively against the attack.

TABLE 5.7: Impact of the learning rate

Metric	Method	Learning rate				
		0.0001	0.001	0.01	0.05	0.1
All-Acc%	*FedAvg (NA)*	*55.24*	*64.75*	*75.81*	*76.77*	*74.94*
	FedAvg	54.23	63.31	73.32	73.81	71.56
	LFighter	53.79	62.81	72.82	73.82	72.84
Src-Acc%	*FedAvg (NA)*	*46.72*	*56.13*	*65.48*	*64.52*	*64.72*
	FedAvg	17.93	19.91	24.22	19.64	8.03
	LFighter	39.74	49.33	56.98	60.65	60.01
ASR%	*FedAvg (NA)*	*19.41*	*17.14*	*14.92*	*13.12*	*11.41*
	FedAvg	47.42	49.42	52.93	58.70	69.15
	LFighter	21.31	19.04	12.10	9.53	10.40

TABLE 5.8: Impact of the local batch size

Metric	Method	Local batch size			
		16	32	64	128
All-Acc%	*FedAvg (NA)*	*77.03*	*75.81*	*72.58*	*65.70*
	FedAvg	74.76	73.32	69.41	64.35
	LFighter	74.60	72.82	70.42	66.46
Src-Acc%	*FedAvg (NA)*	*64.70*	*65.48*	*60.03*	*56.93*
	FedAvg	25.70	24.22	19.32	20.11
	LFighter	60.72	56.98	56.43	47.76
ASR%	*FedAvg (NA)*	*14.10*	*14.92*	*18.81*	*17.61*
	FedAvg	51.70	52.93	56.80	53.84
	LFighter	12.21	12.10	12.81	19.10

5.5 Conclusions

We have conducted a comprehensive analysis of the label-flipping attack behavior. From it, we have observed that the contradictory objectives of attackers and honest peers turn the parameter gradients connected to the source and target class neurons into robust discriminative features to detect the attack. Accordingly, we have presented LFighter, a novel defense that dynamically extracts those gradients and uses them as input features to an adapted clustering method in order to detect attackers.

The empirical results we report show that our defense is effective and simultaneously achieves better test error, overall accuracy, source class accuracy, and attack success rate than related works.

Moreover, unlike the related works, our defense was still robust to significantly large ratios of attackers.

Chapter 6

FL-Defender: combating targeted attacks in federated learning

Targeted attacks are more serious than untargeted poisoning attacks, given their stealthy nature and severe security implications (Steinhardt, Koh, and Liang, 2017; Fung, Yoon, and Beschastnikh, 2020; Tolpegin et al., 2020; Awan, Luo, and Li, 2021; Jebreel et al., 2022b).

The existing defenses against these attacks (discussed in Chapter 3) have several limitations, two of the most significant ones being that they are constrained by the distribution of data among peers and/or the dimensionality of the model.

In this chapter, we analyze the behavior of targeted attacks on FL, and then we propose a robust defense against them based on that behavior analysis. In particular, our contributions are as follows:

- We first analyze the label-flipping attack and the backdoor attack on FL of deep-learning models to understand how these attacks behave. We find that an unbalanced distribution of the peers' local data and a high dimensionality of the DL model make the detection of attacks quite challenging. Moreover, we observe that the

attack-related last-layer neurons exhibit a different behavior from the attack-unrelated last-layer neurons, which makes the last-layer gradients useful features for detecting such targeted attacks.

- We propose *FL-Defender*, a method that can mitigate the attacks regardless of the model dimensionality or the distribution of the peers' data. First, we use the peers' last-layer gradients to engineer more robust discriminative features that capture the attack behavior and discard redundant information. Specifically, we compute the peer-wise angle similarity for the peers' last-layer gradients and then compress the computed similarity vectors using principal component analysis (PCA) to reduce redundant information. Finally, we penalize the peers' updates at the model aggregation stage based on their angular deviation from the centroid of the compressed similarity vectors.

- Experimental results on three data sets with different DL model sizes and peer data distributions demonstrate the effectiveness of our approach at defending against the attacks. Compared with several state-of-the-art defenses, *FL-Defender* achieves better performance at retaining the accuracy of the global model on the main task, reducing the attack success rate, and causing minimal computational overhead on the server.

The contributions in this chapter have been published in Jebreel and Domingo-Ferrer, 2023.

This chapter is organized as follows. Section 6.1 introduces the threat model being considered and the assumptions on the local data distribution. Section 6.2 analyzes the behavior of label-flipping and backdoor attacks, and shows the robustness of the engineered features. Section 6.3 presents the methodology of the proposed defense. Section 6.4 details the experimental setup and reports the obtained results. Finally, conclusions are gathered in Section 6.5.

6.1 Threat model and local data distribution

Threat model. We consider a number of attackers $K' \leq K/5$, that is, no more than 20% of the K peers in the system. For example, with millions of users (Davenport and Davenport, 2018) in Gboard (Hard et al., 2018), controlling just a small percentage of user devices requires the attacker(s) to compromise a large number of devices, which demands huge effort and resources and is therefore impractical. Furthermore, we assume the FL server to be honest and not compromised, and the attackers to have no control over the aggregator or the honest peers.

Attacker's goals. The attacker's goal for the LF attack is to cause the learned global model to classify the source class examples into the target class at test time, while maintaining the benign model performance for the non-source class examples. The attacker's goal for the BA is to fool the global model into falsely predicting the attacker's chosen class for any target example carrying the backdoor pattern, while maintaining the benign model performance for non-poisoned examples.

Defender's goal. The goal of the defender is to obtain a non-poisoned and accurate global model by excluding poisoned updates and considering good updates in the model aggregation.

Assumptions on training data distribution. Since the local data sets of the peers may come from heterogeneous sources (Bonawitz et al., 2019; Wang et al., 2019), they may be either identically distributed (IID) or non-IID. In the IID setting, each peer holds local data representing the whole distribution. In the non-IID setting, the distributions of the peers' local data sets can be different in terms of the classes represented in the data and/or the number of samples each peer holds for each class. Consequently, each peer may have local data with i) all the classes being present in a similar proportion as in the other peers' local data (IID setting), ii) some classes being present in a different proportion (non-IID setting).

6.2 Analyzing targeted attacks against FL

This section is key in our work. We study the behavior of label-flipping and backdoor attacks to find robust discriminative features that can detect such attacks.

Let us consider an FL classification task where each local model is trained with the cross-entropy loss over one-hot encoded labels as follows. First, the activation vector o of the last-layer neurons (a.k.a. logits) is fed into the softmax function to compute the vector p of probabilities as follows:

$$p_k = \frac{e^{o_k}}{\sum_{j=1}^{|\mathcal{C}|} e^{o_j}}, \quad k = 1, \ldots, |\mathcal{C}|. \tag{6.1}$$

Then, the loss is computed as

$$\mathcal{L}(y, p) = -\sum_{k=1}^{|\mathcal{C}|} y_k \log p_k, \tag{6.2}$$

where $y = (y_1, y_2, \ldots, y_{|\mathcal{C}|})$ is the corresponding one-hot encoded true label and p_k denotes the confidence score predicted for the k^{th} class. After that, the gradient of the loss w.r.t. the output o_i of the i^{th} neuron (a.k.a the i^{th} neuron error) in the output layer is computed as

$$\delta_i = \frac{\partial \mathcal{L}(y, p)}{\partial o_i} = p_i - y_i. \tag{6.3}$$

The gradient ∇b_i^L w.r.t. the bias b_i^L connected to the i^{th} neuron in the output layer can be written as

$$\nabla b_i^L = \delta_i \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}, \tag{6.4}$$

where a^{L-1} is the activation output of the previous layer. Likewise, the gradient ∇w_i^L w.r.t. the weights vector w_i^L connected to the i^{th} neuron in

the output layer is

$$\nabla w_i^L = \delta_i a^{L-1} \frac{\partial \sigma^L}{\partial (w_i^L \cdot a^{L-1} + b_i^L)}. \tag{6.5}$$

From Equations (6.4) and (6.5), we can notice that δ_i directly and highly impacts on the gradients of the output layer's weights and biases.

Behavior of label-flipping attacks. In FL, an LF attacker always tries to minimize $p_{c_{src}}$ for any sample in his training data, including samples that belong to class c_{src}. On the other side, he always tries to maximize $p_{c_{target}}$ for samples that belong to c_{src} during model training. Since this goes in the opposite direction of the objective of honest peers for examples in c_{src}, the attack will entail substantial alteration of $\delta_{c_{src}}$ and $\delta_{c_{target}}$, as it can be seen from Expression (6.3). In turn, from Expressions (6.4) and (6.5), it follows that altering $\delta_{c_{src}}$ and $\delta_{c_{target}}$ directly alters the biases and weights corresponding to the output neurons of c_{src} and c_{target} during the training of the attacker's local model. Hence, the impact of the attack can be expected to show in the gradients of the last-layer neurons corresponding to c_{src} and c_{target}. However, the last layer is likely to contain other neurons unrelated to the attack where both the attacker and the honest peers share the same objectives, which makes the attack harder to spot. Considering all layers is still worse, because in the layers different from the last one the impact of the attack will be even less perceptible (as it will be mixed with more unrelated parameters). Moreover, there are two other factors that increase the difficulty of detecting the attack by analyzing the update as a whole: i) the early layers usually extract common features that are not class-specific (Nasr, Shokri, and Houmansadr, 2019) and ii) in general, most parameters in DL models are redundant (Denil et al., 2013). That causes the magnitudes and angles of the bad and good updates to be similar, which makes models with large dimensionality an ideal environment for a successful label-flipping attack.

Behavior of backdoor attacks. Backdoor attacks might be viewed as a particular case of label-flipping attacks because the attacker flips the label of a training sample when it contains a specific feature or pattern, whereas he retains the correct label when the sample does not contain the pattern (benign sample). However, since the global model will correctly learn from a majority of honest peers and, in the clean samples, from the attackers as well, the received global model will probably overlook the backdoor pattern and assign the correct classes to the poisoned sample, especially in the early training iterations. This will prompt the attackers to try to minimize $p_{c_{src}}$ and maximize $p_{c_{target}}$ for the poisoned samples, which can be expected to stand out in the magnitudes and the directions of the gradients contributed by the attackers. On the other hand, since the attackers also try to maximize $p_{c_{src}}$ for their benign samples, the impact of the backdoor attacks is expected to be stealthier compared to that of LF attacks even when looking at the last-layer gradients.

Engineering more robust features to detect attacks. From the above analysis, it is clear that focusing on analyzing last-layer gradients is more helpful to detect targeted attacks than analyzing all layers. Nevertheless, the presence of a large number of attack-unrelated gradients in the last layer may still render attack detection difficult. Getting rid of those redundant and unrelated gradient features could help obtain more robust discriminatory features for targeted attacks. Since the impact of the targeted attacks is directly reflected in the directions of gradients of attack-related neurons, comparing the difference in directions between the gradients of a good update and a poisoned update can be expected to better capture the attack's behavior. If we look at the angular similarity of the peers' last-layer gradients, the similarity values between good and poisoned updates are expected to display unique characteristics in the computed similarity matrix. PCA can be used to capture those unique characteristics from the matrix and reduce redundant features.

Empirical analysis. To empirically validate our previous conceptual discussion, we used 20 local updates resulting from simulating an

FL scenario under the LF and BA attacks with each of the CIFAR10-IID and CIFAR10-non-IID benchmarks, where 4 updates (that is, 20%) were poisoned. In these two benchmarks, the ResNet18 (He et al., 2016) architecture, which contains about $11M$ parameters, was used. In addition, training data were randomly and uniformly distributed among peers in CIFAR-IID, while we adopted a Dirichlet distribution (Minka, 2000) with $\alpha = 1$ to generate non-IID data for the 20 peers in CIFAR10-non-IID. The details of the experimental setup are given in Section 6.4.1.

Then, for each benchmark and attack scenario, we computed the following:

- The first two principal components (PCs) of the all-layer gradients for each local update. Next, we computed the centroid (CTR) of the first two PCs for the 20 local updates. After that, we computed the angle between CTR and every pair of PCs for each update.
- The centroid (CTR) of the last-layer gradients for the 20 local updates and the angle between CTR and each last-layer gradient for each update.
- The first two PCs of the last-layer gradients for each local update. Then, we computed the centroid (CTR) of the first two PCs of the 20 last-layer gradients. After that, we computed the angle between CTR and every pair of PCs for each last-layer gradient.
- The cosine similarity for the 20 peers' last-layer gradients. Then, we computed the first two PCs of the computed similarity matrix. After that, we computed the centroid (CTR) of the first two PCs of the 20 similarity vectors. Finally, we computed the angle between CTR and every pair of PCs of each similarity vector.

Once the above computations were completed, we visualized the magnitude of each input, and the angle between the input and its corresponding centroid.

Fig. 6.1 shows the visualized vectors for the CIFAR-IID benchmark. For the LF attack, we can see that analyzing the first two PCs of the all-layer gradients (All) led to poisoned updates and good updates with very similar magnitudes and angular deviation from the centroid, which made it quite challenging to tell them apart. The same applies to analyzing the last-layer gradients (Last). On the other hand, analyzing the first two PCs of the last-layer gradients (Last-PCA) led to an apparent separation between good and bad updates. This also applied to analyzing the engineered features (Engineered). For the BA, the results were similar with the difference that our engineered features allowed better separation than Last-PCA. This confirms our conceptual discussion that redundant and attack-unrelated gradients make the attacks stealthier.

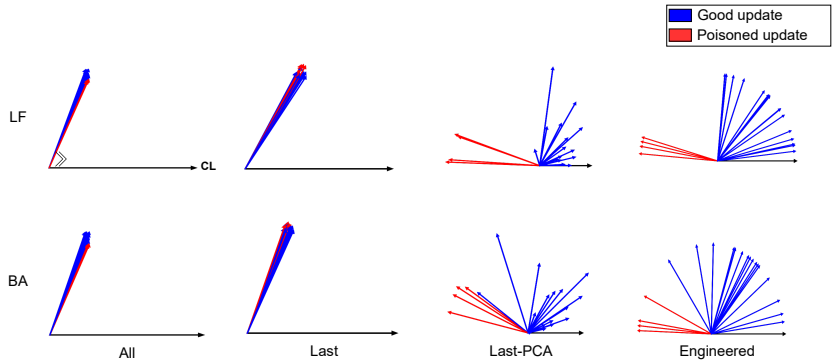


FIGURE 6.1: Deviation of CIFAR10-IID gradient features from the centroid

Fig. 6.2 shows the visualized vectors for the CIFAR-non-IID benchmark, where the data were non-IID among peers. For the LF attack, we can see that only our engineered features provided robust discrimination between good and bad updates. For the BA, even if our engineered features allowed better separation, it was challenging to tell updates apart. This is because of the impact of the non-IIDness and also

because BAs are stealthier than LF attacks. This again confirms our intuitions and shows that our engineered features are more useful than the alternatives to detect targeted attacks.

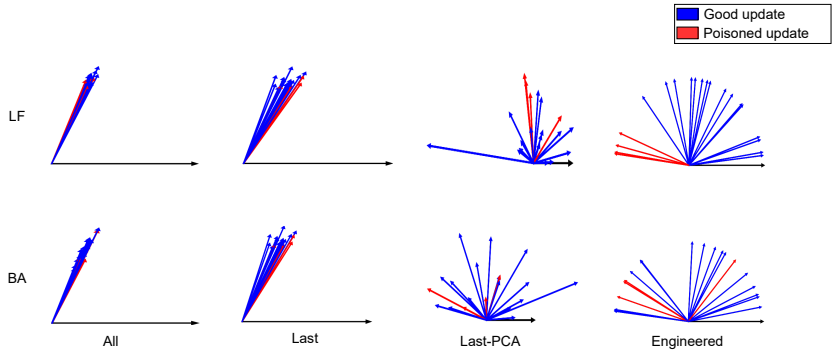


FIGURE 6.2: Deviation of CIFAR10-non-IID gradient features from the centroid

We were able to conclude from these analytical and empirical explorations that a high model dimensionality and the distribution of the peers' training data highly impact on the ability to discriminate between good updates and updates poisoned by targeted attacks. Fortunately, it also became evident that we can use the last-layer gradients to engineer more robust discriminative features for attack detection.

6.3 FL-Defender design

In this section, we present the design of FL-Defender, our proposed defense against FL targeted poisoning attacks. Our aims are: 1) to prevent attackers from achieving their goals by mitigating the impact of their poisoned updates on the global model, 2) to maintain the global model performance on the main task, 3) to stay robust against the attacks regardless of the model size or the peers' data distribution, and 4) to avoid substantially increasing the computational cost at the server.

According to the lessons learned in the previous section, our defense first extracts the last-layer gradients of the peers' local updates, computes the the peer-wise cosine similarity, and then compresses the computed similarity vectors using PCA to reduce redundant information and extract more robust features. After that, it computes the centroid of the compressed similarity vectors and computes the cosine similarity values between the centroid and each compressed vector. The similarity values are accumulated during training and used, after re-scaling them, to re-weight the peers' updates in the global model aggregation.

We formalize our method in Algorithm 2. The aggregator server A starts a federated learning task by initializing the global model W^0 and the history vector H^0 that is used to accumulate the similarities between the directions of the peers' engineered features and their centroid.

Then, in every training iteration t, A selects a random subset S of m peers and sends the current global model W^t to the m selected peers. Each peer $k \in S$ locally trains W^t on her data D_k and sends her local update W_k^{t+1} back to A. Once A receives the m local updates, it separates the gradients of the last layers to obtain the set $\{\nabla L_k^{t+1} | k \in S\}$.

Cosine similarity. In Procedure 1, we compute the cosine similarities among gradients of the last layers to capture the discrepancy between the gradients from the honest peers and those from the attackers. This discrepancy is caused by their contradicting objectives. The cosine similarity between two gradients ∇_i and ∇_j is defined as

$$cs(\nabla_i, \nabla_j) = \cos\varphi = \frac{\nabla_i \cdot \nabla_j}{||\nabla_i|| \cdot ||\nabla_j||}. \tag{6.6}$$

This way, if ∇_i and ∇_j lie in the same direction, their cosine similarity will be 1, and their similarity value will decrease as their directions differ more. The cosine similarity measures the angular similarity among gradients and is more robust than the Euclidean distance because, even if the attackers scale their model update gradients to avoid detection, they need to keep their directions to achieve their objectives.

Procedure 2: FL-Defender: Combating targeted attacks in FL

Input: $K, C, BS, E, \eta, T, \tau$
Output: W^T, the global model after T training rounds

A initializes W^0, $H^0 = \{H_k^0 = 0\}_{k=1}^K$;
for *each round* $t \in [0, T-1]$ **do**
  $m \leftarrow \max(C \cdot K, 1)$;
  $S \leftarrow$ random set of m peers;
  A sends W^t to all peers in S;
  for *each peer* $k \in S$ ***in parallel*** **do**
    $W_k^{t+1} \leftarrow$`peer_UPDATE`(k, W^t); `// A sends` W^t `to each peer` k `who trains` W^t `using her data` D_k `locally, and sends her local update` W_k^{t+1} `back to the aggregator.`
  end
  Let $\{\nabla_1, \ldots, \nabla_i, \ldots, \nabla_m\}$ be the gradients of the last layers of $\{W_k^{t+1} | k \in S\}$;
  $\{\varsigma_1, \ldots, \varsigma_i, \ldots, \varsigma_m\} \leftarrow$`COMPUTE_SIMILARITY`$(\{\nabla_1, \ldots, \nabla_i, \ldots, \nabla_m\}, \tau)$;
  for *each peer* $k \in S$ **do**
    $\varsigma_k^t \leftarrow$ `Assign`$(\{\varsigma_1, \ldots, \varsigma_i, \ldots, \varsigma_m\})$; `// Assign the computed similarities to their corresponding peers.`
    $H_k^t = H_k^{t-1} + \frac{t+1}{T} \times \varsigma_k^t$; `// Accumulate the weighted similarities of the peer to the centroid.`
  end
  $Q1 \leftarrow$ `FIRST_QUARTILE`(H^t);
  $\gamma = H^t - Q1$; `// Subtract Q1 from every entry in` H^t`. Since attackers are expected to be below Q1, this will make their trust values in` γ `negative.`
  for *each peer* $k \in (1, \ldots, K)$ **do**
    $\gamma_k \leftarrow \max(\gamma_k, 0)$; `// Attacker trusts are brought to 0 to neutralize them in the aggregation.`
  end
  $\gamma = \gamma / max_k(\gamma)$; `// Normalize trust in peers updates to 0-1 range.`
  A aggregates $W^{t+1} \leftarrow \frac{1}{\sum_{k \in S} \gamma_k} \sum_{k \in S} \gamma_k W_k^{t+1}$.
end

Procedure 1: Compute peers' compressed gradients similarity to their centroid

```
COMPUTE_SIMILARITY({∇_1, ..., ∇_i, ..., ∇_m}, τ):
    {cs_{1,1}, ..., cs_{i,j}, ..., cs_{m,m}} ←COSINE_SIMILARITY({∇_1, ..., ∇_i, ..., ∇_m})
        // ∀i, j ∈ (1, ..., m), cs_{i,j} is the cosine similarity
        between ∇_i and ∇_j.
    Let P be the number of the first PCs that explain at least τ of
     the variance in the computed similarity matrix.
    {(p_i^1, ..., p_i^P)}_{i=1}^m ←PCA({cs_{1,1}, ..., cs_{m,m}}, n_pcs = P)
        // (p_i^1, ..., p_i^P) are the first P PCs of
        {cs_{i,1}, ..., cs_{i,m}}.
    CTR ←Median({(p_i^1, ..., p_i^P)}_{i=1}^m) // CTR is the
        component-wise centroid of the compressed
        similarity vectors.
    Let ς_i be the cosine similarity between (p_i^1, ..., p_i^P) and CTR;
    return (ς_1, ..., ς_i, ..., ς_m).
```

Compressing similarity vectors. Since the number of peers is expected to be large in FL, we use principal components analysis to compress the computed similarity matrix and reduce the attackers' chance to hide their impact on the high-dimensional similarity matrix. PCA returns a compact representation of a high-dimensional input $X \in \mathbb{R}^N$ by projecting it onto a subspace $\mathbb{R}^P$ of dimension $P < N$ so that the unique characteristics of the input are reduced to that subspace. P is the number of used principal components and is the primary hyper-parameter PCA needs to compress its input data. In less noisy and non-complex data, small values of P (*e.g.*, 2 or 3) are enough to capture most of the variance in the input data. In our case, however, the distribution of the similarity matrix is expected to become more complex as the non-IIDness of the local data and the model dimension increase. Hence, we use a number P of principal components that explains at least a threshold τ of the variance of the computed matrix.

Similarity between compressed vectors and their centroid. After

compressing the peers' similarity vectors into their first P PCs, we aggregate the latter component-wise using the median to obtain their centroid $CTR \in \mathbb{R}^P$. Since we assume the majority of the peers ($\geq$ 80%) to be honest, the centroid is expected to fall into the heart of a majority of good components. After that, we compute the cosine similarity between the centroid and each peer's pair of PCs. Good components are expected to have a similar direction to the centroid and thus have similarity values close to one. On the other hand, poisoned components are expected to be farther from the centroid with values closer to -1.

Update history and compute trust scores. We use the similarity values with the centroid to update the similarity history vector H for the selected m peers. This guarantees that, as the training evolves, the closest peers to the centroid are assigned larger similarity values than the farthest peers. Since we assume the centroid falls amid the honest peers, honest peers have larger accumulated similarity values than attackers. Note that the attackers may exploit this point to subvert our defense by launching attacks in the last training rounds after behaving honestly and accumulating high similarity values in the previous rounds. To counter this strategy, in every training round t we scale the computed similarities by $(t+1)/T$ before updating H. This also forces the peers to refrain from behaving maliciously as the training evolves, because they will quickly lose the high similarity values they accumulated earlier. We demonstrate the effectiveness of such scaling in Section 6.4.2. After updating the similarity history vector H, we compute the first quartile $Q1$ for the values in H and then subtract it from the similarity value accumulated by every peer. That is, we shift every similarity value in H to the left by $Q1$ and we assign the shifted similarities to the peers' trust scores vector γ. Since the accumulated similarities of attackers are low, they are likely to be below $Q1$ and hence they become negative after the shift. After that, we set negative trust scores in γ to 0, in order to neutralize attackers when using trust scores as weights in the final aggregation (see below). Finally, we normalize the scores in γ to be in the range $[0, 1]$

by dividing them by their maximum value.

Re-weighting and aggregating updates. In the final step of Algorithm 2, the server uses the trust scores in γ to re-weight the corresponding local updates and aggregates the global model using the re-weighted local updates. Note that, since $\frac{1}{\sum_{k \in S} \gamma_k} \sum \gamma_k = 1$, the convergence of the proposed aggregation procedure at the server side is guaranteed as long as *FedAvg* converges.

6.4 Experiments

In this section we compare the performance of our method with that of several state-of-the-art countermeasures against poisoning attacks. For reproducibility, our code and data are available at https://github.com/anonymized30/FL-Defender.

6.4.1 Experimental setup

Data distribution and training. We defined the following benchmarks by distributing the data from the data sets described in Chapter 2 among the participating peers in the following way:

- MNIST-non-IID. We adopted a Dirichlet distribution (Minka, 2000) with a hyper-parameter $\alpha = 1$ to generate non-IID data for 20 participating peers. We used a two-layer convolutional neural network (CNN) with two fully connected layers (number of model parameters $\approx$ 22K). The CNN model was trained during 200 iterations. In each iteration, the FL server asked the peers to train their models for 3 local epochs with a local batch size 64. The participants used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with learning rate = 0.01 and momentum = 0.9 to train their models.

- CIFAR10-IID. We randomly and uniformly divided the CIFAR10 training data among 20 peers. We used the ResNet18 CNN model (He et al., 2016) with one fully connected layer (number of parameters $\approx$ 11M). The ResNet18 model was trained during 100 iterations. In each iteration, the FL server asked the 20 peers to train the model for 3 local epochs with a local batch size 32. The peers used the cross-entropy loss function and the SGD optimizer with learning rate = 0.01 and momentum = 0.9.

- CIFAR10-non-IID. We took a Dirichlet distribution with a hyper-parameter α = 1 to generate non-IID data for 20 participating peers. The training settings were the same as in CIFAR10-IID.

- IMDB. We randomly and uniformly split the $40K$ training examples among 20 peers. We used a Bidirectional Long/Short-Term Memory (BiLSTM) model with an embedding layer that maps each word to a 100-dimensional vector. The model ends with a fully connected layer followed by a sigmoid function to produce the final predicted sentiment for an input review (number of parameters $\approx$ 12M). The BiLSTM model was trained during 50 iterations. In each iteration, the FL server asked the 20 peers to train the model for 1 local epoch with a local batch size 32. The peers used the binary cross-entropy with logit loss function and the *Adam* optimizer with learning rate = 0.001.

Attack settings. i) Label-flipping attacks. In the MNIST experiments, the attackers flipped the examples with the label 9 to 4 before training their local models, while they flipped the examples with the label *Dog* to *Cat* in the CIFAR10 experiments. For IMDB, the attackers flipped the examples with the label *positive* to *negative*. ii) Backdoor attacks. In MNIST, the attackers embedded a 3×3 square with white pixels in the bottom-right corner of the examples belonging to class *9* and changed their label to class *0*. In CIFAR10, the attackers embedded the same pattern in the examples belonging to class *Car* and changed their label to class *Plane*

before training their local models. In all the experiments, the number of attackers K' ranged in $\{0, 2, 4\}$, which corresponds to ratios of attackers in $\{0\%, 10\%, 20\%\}$, respectively.

Defense settings. For the method in Tolpegin et al., 2020, we looped through all the last-layer neurons and considered the neuron that best separates its gradients into two clusters as the potential source class. Then, we identified the smaller group as potentially poisoned and excluded it from model aggregation. For FLAME (Nguyen et al., 2022), we used $\epsilon = 3705$ for image classification benchmarks and $\epsilon = 4191$ for the NLP benchmark, as suggested in the original paper. For our method, we used a number P of first principal components that capture at least $\tau = 90\%$ of the variance.

Evaluation metrics. We used the following evaluation metrics on the test set examples to assess the impact of the attacks on the learned model and the performance of the proposed method w.r.t. the state of the art:

- *Test error (TE)*. This is the error resulting from the loss functions used in training. The lower the test error, the more robust the method is against the attack.
- *Overall accuracy (All-Acc)*. This is the number of correct predictions divided by the total number of predictions.
- *Source class accuracy (Src-Acc)*. We evaluated the accuracy for the subset of test examples belonging to the source class. Note that one may achieve a good overall accuracy while degrading the accuracy of the source class.
- *Attack success rate (ASR)*. This is the proportion of targeted examples (with the source label or the backdoor pattern) that are incorrectly classified into the label desired by the attacker.

An effective defense against the attacks needs to retain the benign performance of the global model on the main task while reducing ASR.

6.4.2 Experimental evaluation

We evaluated the robustness of our defense against LF and BAs and compared it with several countermeasures discussed in Chapter 3: median (Yin et al., 2018), trimmed mean (TMean) (Yin et al., 2018), multi-Krum (MKrum) (Blanchard et al., 2017), FoolsGold (FGold) (Fung, Yoon, and Beschastnikh, 2020), Tolp (Tolpegin et al., 2020) and FLAME (Nguyen et al., 2022). We also compared with the standard FedAvg (McMahan et al., 2017a) aggregation method (that is not meant to counter security attacks). We report the average results of the last 10 training rounds to ensure a fair comparison among methods.

Robustness against label-flipping attacks. Table 6.1 shows the performance of defenses under the LF attack in the MNIST-non-IID benchmark. Due to the small size of the model, all methods stayed robust against the attack for all attacker ratios considered. Although FedAvg is not meant to mitigate poisoning attacks, it achieved a good performance in this benchmark. That was also observed in Shejwalkar et al., 2021, where the authors argued that, in some cases, FedAvg could achieve good performance against poisoning attacks.

TABLE 6.1: Robustness against the label-flipping attack in the MNIST-non-IID benchmark. $K'\%$ denotes the ratio of attackers. Boldfaced values are the best results among all defenses. Underlined values are the second-best results.

Method→ $K'\%\downarrow$	Metric↓	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	Ours
0	TE	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**
	All-Acc%	98.94	99.01	98.94	**99.04**	98.98	**99.02**	98.98	98.99
	Src-Acc%	97.82	98.12	97.82	98.32	98.02	98.32	98.41	**98.71**
	ASR%	0.40	0.40	0.40	**0.30**	0.50	**0.30**	0.40	**0.30**
10	TE	0.04	0.04	**0.03**	0.04	**0.03**	**0.03**	0.04	0.04
	All-Acc%	98.92	98.96	**98.99**	98.95	98.92	98.96	98.97	98.86
	Src-Acc%	96.73	98.02	97.62	98.02	97.72	98.12	97.72	**98.61**
	ASR%	1.29	0.50	0.69	**0.40**	0.50	**0.40**	0.69	**0.40**
20	TE	0.06	**0.04**	**0.04**	**0.04**	**0.04**	0.05	0.05	**0.04**
	All-Acc%	98.77	98.95	98.88	98.88	98.94	98.89	98.86	**98.97**
	Src-Acc%	95.04	97.03	96.53	97.92	98.02	96.43	96.23	**98.81**
	ASR%	2.97	1.49	1.78	**0.30**	0.50	1.78	1.88	**0.30**

Table 6.2 shows the performance of defenses under the LF attack in the CIFAR10-IID benchmark. When no attack is performed, all methods achieved comparable performance to FedAvg for the test error and the overall accuracy. However, in the presence of attacks, only FoolsGold, Tolp and our method performed well regarding the source class accuracy and attack success rate. MKrum achieved the worst performance because it considers all layers, which caused a lot of false positives and false negatives. Note that the theoretical upper bound on the number of attackers MKrum (Blanchard et al., 2017) can resist is $K' = (K/2) - 2$, which corresponds to $K' = 8$ in our setting. The performance of the rest of the methods (FedAvg, Median, TMean and FLAME) was diminished with regard to the protection of the source class, even though the data distribution were IID. The reason was the large model size.

TABLE 6.2: Robustness against the label-flipping attack in the CIFAR10-IID benchmark. K'% denotes the ratio of attackers. Boldfaced values are the best results among all defenses. Underlined values are the second-best results.

K'% ↓	Method→ / Metric↓	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	Ours
0	TE	**0.80**	**0.80**	0.81	0.83	**0.80**	0.82	**0.80**	**0.80**
	All-Acc%	**76.93**	76.35	76.91	76.83	77.24	76.37	76.57	76.65
	Src-Acc%	63.6	65.30	66.50	65.70	**67.60**	65.41	64.25	65.53
	ASR%	15.60	14.40	13.50	13.40	**12.70**	16.68	13.13	12.85
10	TE	0.80	**0.78**	0.79	0.84	0.84	0.86	0.80	0.80
	All-Acc%	76.96	76.50	76.47	**77.04**	76.87	76.14	76.48	76.49
	Src-Acc%	55.20	55.80	57.90	53.90	**65.60**	65.10	56.98	65.22
	ASR%	23.40	23.00	20.60	24.50	**15.00**	15.81	19.01	15.63
20	TE	0.85	**0.81**	**0.81**	0.95	0.84	0.94	0.83	**0.81**
	All-Acc%	75.27	76.31	75.76	75.21	**76.42**	74.88	75.67	75.61
	Src-Acc%	44.00	54.20	47.70	38.90	63.00	**64.00**	42.52	63.85
	ASR%	33.60	25.30	31.60	37.70	16.40	16.37	32.04	**15.11**

Table 6.3 shows the performance of defenses under the LF attack in the CIFAR10-non-IID benchmark. Looking at the results, we can see the influence of the data distribution and the model size on the performance of the methods. However, thanks to the robust engineered features, our method preserved the global model performance while preventing the

attackers from performing successful label-flipping attacks. Note that FoolsGold and Tolp achieved limited robustness in this benchmark compared to what they did in CIFAR10-IID.

TABLE 6.3: Robustness against the label-flipping attack in the CIFAR10-non-IID benchmark. $K'\%$ denotes the ratio of attackers. Boldfaced values are the best results among all defenses. Underlined values are the second-best results.

Method→ $K'\%\downarrow$	Metric↓	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	Ours
0	TE	**0.85**	0.95	0.89	0.90	0.98	1.16	**0.85**	**0.85**
	All-Acc%	**75.88**	74.40	74.86	74.18	73.93	75.17	75.37	75.25
	Src-Acc%	**66.70**	66.10	66.10	65.90	52.90	62.10	61.66	65.01
	ASR%	14.90	13.60	14.80	15.50	21.40	17.00	14.91	13.42
10	TE	0.93	0.92	0.97	1.04	0.92	1.19	**0.87**	0.88
	All-Acc%	74.96	74.12	74.62	71.60	74.08	74.44	**75.14**	74.96
	Src-Acc%	48.40	51.70	50.40	39.10	58.40	49.72	45.73	**64.60**
	ASR%	28.70	25.10	28.70	31.90	18.80	29.67	30.01	**13.33**
20	TE	0.92	0.91	0.96	1.08	0.93	1.19	**0.89**	0.90
	All-Acc%	**75.22**	74.28	74.18	69.84	74.51	73.41	73.51	74.16
	Src-Acc%	44.20	48.80	42.60	21.50	51.10	42.93	32.33	**65.41**
	ASR%	29.60	32.30	33.60	51.50	25.50	34.5	44.92	**13.24**

Table 6.4 shows the performance of defenses under the LF attack in the IMDB benchmark. We can see FGold, Tolp, FLAME and our defense achieved similar high robustness against the attacks. Moreover, they outperformed the other methods by a large margin at providing adequate and simultaneous protection for all the metrics. FGold performed well in this benchmark because it is its ideal setting: updates for honest peers were somewhat different due to the different reviews they gave, while updates for attackers became very close to each other because they shared the same objective. Tolp and FLAME performed well because the task was binary classification, and there was no redundant information from other classes.

Robustness against backdoor attacks. Fig. 6.3 shows the results for the backdoor attacks. We employed as a baseline FedAvg when no attacks were performed. We can see that, in general, all methods achieved similar overall accuracy to the baseline. This is because a BA does not

TABLE 6.4: Robustness against the label-flipping attack in the IMDB benchmark. K'% denotes the ratio of attackers. Boldfaced values are the best results among all defenses. Underlined values are the second-best results.

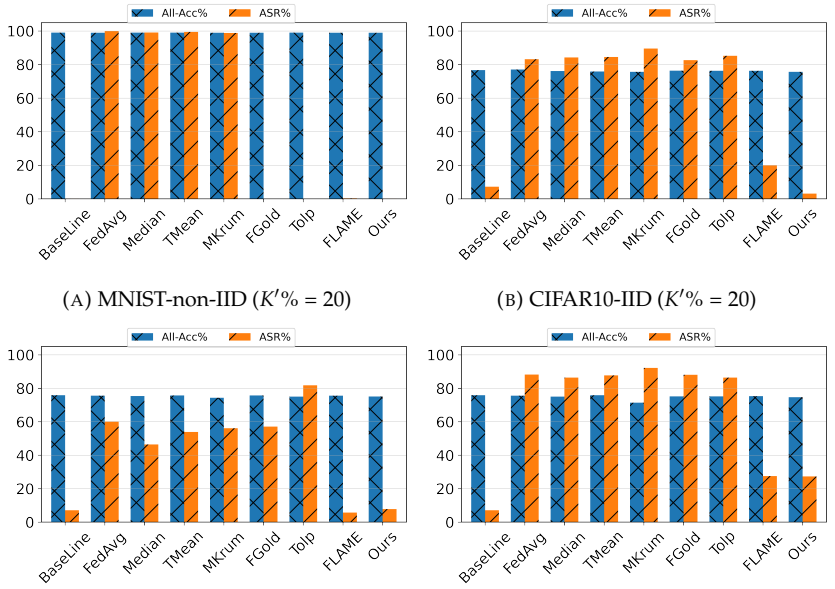
Method→ / K'% ↓	Metric↓	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	Ours
0	TE	0.28	**0.27**	0.28	0.28	0.28	0.28	0.28	0.28
	All-Acc%	88.55	88.75	88.55	88.61	88.73	**88.88**	88.77	88.65
	Src-Acc%	85.88	86.12	85.88	86.12	86.16	86.28	**86.48**	86.22
	ASR%	14.12	13.88	14.12	13.88	13.84	13.72	13.52	13.78
10	TE	0.40	0.34	0.37	0.45	**0.29**	**0.29**	**0.29**	0.30
	All-Acc%	81.52	84.21	82.94	79.57	**88.66**	88.42	88.48	88.32
	Src-Acc%	66.07	72.74	69.52	61.65	**86.44**	85.92	86.38	86.24
	ASR%	33.93	27.26	30.48	38.35	**13.56**	14.08	13.62	13.76
20	TE	0.63	0.49	0.53	0.85	**0.30**	**0.30**	**0.30**	0.31
	All-Acc%	72.50	77.11	75.97	64.9	**88.45**	88.24	88.22	88.18
	Src-Acc%	46.04	56.26	53.78	30.36	**86.40**	86.06	86.12	86.26
	ASR%	53.96	43.74	46.22	69.64	**13.60**	13.94	13.88	13.74

change the labels of the benign samples during training. Regarding ASR, we notice that FGold, Tolp, FLAME, and our method achieved comparable results to the baseline in the MNIST-non-IID benchmark (with 20% attackers). On the other hand, the attackers achieved attack success rates of about 100% with the other methods. For CIFAR10-IID (with 20% attackers), our method achieved the lowest ASR compared to the other methods, which (except FLAME) failed to counter the BA. For CIFAR10-non-IID (with 10% of attackers), only FLAME and our method stayed robust against the attack. For CIFAR10-non-IID (with 20% of attackers), only FLAME and our method were able to mitigate the attack compared to the other methods, which failed against it.

To sum up, our defense performed effectively against label-flipping attacks, while it improved over the state-of-art methods against backdoor attacks.

Next, we will explore the effects of various hyper-parameters and components on our defense approach while keeping the attacker ratio fixed at 20%.

Impact of the explained variance threshold. We studied the impact



(A) MNIST-non-IID ($K'\% = 20$)

(B) CIFAR10-IID ($K'\% = 20$)

(C) CIFAR10-non-IID ($K'\% = 10$)

(D) CIFAR10-non-IID ($K'\% = 20$)

FIGURE 6.3: Robustness against backdoor attacks

of the explained variance threshold τ on the performance of our defense. We used values of $\tau \in \{50\%, 60\%, 70\%, 80\%, 90\%, 95\%\}$ in the CIFAR10-non-IID benchmark under the backdoor attack. Figure 6.4 shows the overall accuracy and the attack success rate for different threshold values. It can be seen that higher threshold values for the explained variance (up to 90%) mitigated the attack better. However, a threshold greater than 90% achieved almost the same performance as 90%. This justifies our choice of $\tau = 90\%$.

Impact of the local data distribution. We studied the impact of the local data distribution on the performance of our defense in the CIFAR10

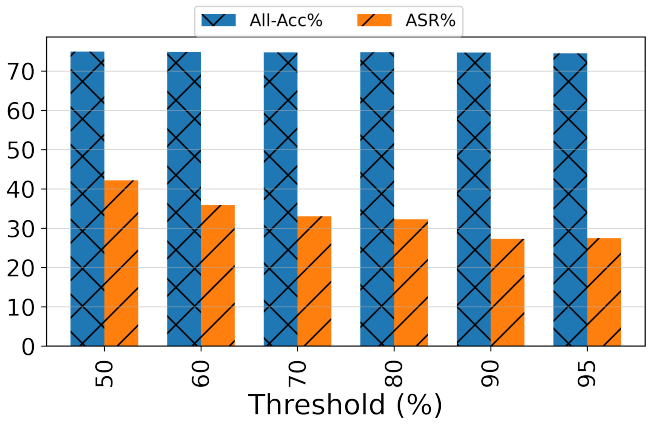


FIGURE 6.4: Impact of the explained variance threshold

benchmark under the label-flipping and backdoor attacks. We generated local data with different degrees of non-IIDness using different values of the Dirichlet distribution parameter $\alpha \in \{0.5, 0.7, 1, 3, 5, 10, 20\}$.

TABLE 6.5: Impact of the non-IIDness of the local data distribution on the performance of our defense using the α parameter of the Dirichlet distribution in CIFAR-10. Lower values of α correspond to higher non-IIDness.

$\alpha \rightarrow$ Attack↓	Metric↓	0.5	0.7	1	3	5	10	20
LF	TE	0.93	0.92	0.90	0.89	0.88	0.87	0.87
	All-Acc%	70.34	72.90	74.16	74.84	75.65	75.16	75.26
	Src-Acc%	53.52	64.3	65.40	63.60	64.54	63.51	63.75
	ASR%	21.03	15.52	13.24	13.71	14.40	13.43	12.72
BA	All-Acc%	71.48	73.36	74.67	75.42	75.43	75.85	75.95
	ASR%	9.32	21.44	27.31	7.85	6.36	5.21	3.77

Table 6.5 shows that the higher the non-IIDness of local data, the

lower the overall accuracy. In fact, the non-IIDness of the local data of peers is a challenge to the performance and convergence of FL, as it has been observed in several FL works such as Zhang and Li, 2021; Zhang and Li, 2022. It can also be seen that higher non-IIDness (lower α) of the local data distribution corresponds to higher LF ASRs and lower BA ASRs. A possible explanation for these opposite behaviors is that the high divergence in local updates caused by high non-IIDness i) makes it more challenging to differentiate between good and poisoned updates and thus increases the LF ASR, and ii) highly perturbs the few BA-related weights in poisoned updates and thus renders BAs less effective. Nevertheless, our defense, in general, was able to mitigate the attack impact without losing much accuracy on the main task.

Impact of feature similarity between the source and target class. We studied the impact of the similarity in features between the source and the target class samples on the attack's performance and our defense's performance. We conducted the LF attack in the CIFAR10-non-IID benchmark with FedAvg and our defense. Also, we conducted the attack with FedAvg when no attack was performed and used it as a baseline. We adopted the following three attack scenarios:

1. Source and target classes with high feature similarity: the attackers flipped the examples with the label *Dog* to *Cat*.

2. Source and target classes with moderate feature similarity: the attackers flipped the examples with the label *Truck* to *Car*.

3. Source and target classes with high feature dissimilarity: the attackers flipped the examples with the label *Frog* to *Plane*.

Table 6.6 shows the obtained results. We can see that the higher the feature similarity between the source and the target classes, the higher the attack success rate. That is because the more similar the two classes' features, the larger the overlapping between their decision boundaries, which makes the attacker's task easier. This also explains why ASR was

TABLE 6.6: Impact of feature similarity between the source and target class

Source-Target→		Dog-Cat	Truck-Car	Frog-Plane
Metric↓	Method↓			
All-Acc%	Baseline	75.88	75.88	75.88
	FedAvg	75.22	74.38	74.84
	Ours	74.16	74.18	74.21
Src-Acc%	Baseline	66.70	79.50	85.51
	FedAvg	44.20	65.10	83.93
	Ours	65.41	78.90	86.00
ASR%	Baseline	14.90	6.73	0.41
	FedAvg	29.60	18.30	0.88
	Ours	13.24	7.61	0.47

about 15% with the *Dog-Cat* scenario (when no attack was performed). On the other hand, the more distinct the features of the two classes, the clearer the separation between their decision boundaries, which makes the attack less effective, as with the *Frog-Plan* scenario.

We could also see that our defense effectively protected against the attack with all scenarios and achieved an ASR close to the baseline.

Effectiveness of scaling similarity scores by training rounds. The attackers may perform a *timing attack* to trick our defense and achieve higher ASR. We considered a scenario where the attackers behave honestly until they accumulate large similarity scores and then maliciously behave in the last training rounds to have a greater impact on the aggregated model. We performed the LF attack with the CIFAR-non-IID benchmark in the last (10^{th}) training round. We then used our defense (without and with the scaling factor) to counter the attack. Figure 6.5 shows the effectiveness of the scaling factor against this adaptive LF attack. With scaling, we achieved a higher source class accuracy and a lower attack success rate.

Impact of the attackers' malicious behavior rate. The attackers may

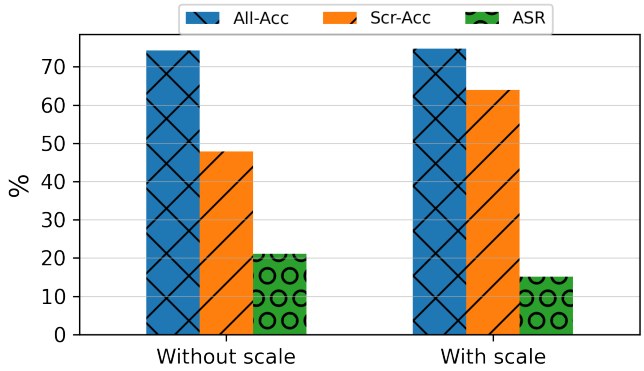


FIGURE 6.5: Effectiveness of scaling similarity scores by training rounds

also follow another strategy to increase the attack impact in the aggregated global model by not always acting maliciously. To study the impact of this strategy, we simulated the LF attack in CIFAR10-non-IID when the attackers launched the attack, at each local epoch, with a probability $p' \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$. Table 6.7 shows that our defense was effective against all the malicious behavior rates of the attackers.

TABLE 6.7: Impact of the attackers' malicious behavior rate (p')

Metric↓, $p' \rightarrow$	0%	25%	50%	75%	100%
TE	0.85	0.86	0.86	0.88	0.90
All-Acc%	75.25	75.12	74.93	74.31	74.16
Src-Acc%	65.01	65.25	65.54	64.83	65.41
ASR%	13.42	12.97	13.14	13.45	13.24

Runtime overhead. We measured the CPU runtime of our method and compared it with that of the other methods. Table 6.8 shows the per-iteration server runtime overhead in seconds for each method. The

results show that our method ranked second best after FoolsGold, which achieved the smallest runtime (excluding FedAvg, which just averages updates and is not meant to counter any attacks). Nevertheless, the runtime overhead of our method can be viewed as a good investment, given its effectiveness at combating the targeted attacks.

TABLE 6.8: CPU runtime per iteration on the server side (in seconds). Boldfaced values are the smallest runtimes among all defenses, excluding FedAvg. Underlined values are the second-smallest runtimes.

Benchmark↓ Method→	FedAvg	Median	TMean	MKrum	FGold	Tolp	FLAME	Ours
MNIST-CNN	0.0091	0.1110	0.0895	0.0297	**0.0137**	0.3783	0.0220	0.0170
CIFAR10-ResNet18	0.1993	1.6784	1.4432	1.9461	**0.2966**	1.1480	3.3530	0.6616
IMDB-BiLSTM	0.2000	2.0813	1.7949	2.2210	**0.2985**	0.5755	3.4681	0.5245

6.5 Conclusions

We have studied the behavior of targeted attacks against FL and we have found that robust features for attack detection can be extracted from the gradients of the last layers of deep learning models. Accordingly, we have engineered robust discriminative features for attack detection by computing the peer-wise similarities of gradient directions and then compressing them using PCA to reduce redundant information. Then, we have built on the engineered features to design FL-Defender, a novel and effective method to defend against attacks. FL-Defender re-weights the peers' local updates during the global model aggregation based on their historical deviation from the centroid of the engineered features. The empirical results show that our method performs very well at defending against label-flipping attacks regardless of the peers' data distribution or the model size. Also, it improves over the state of the art at mitigating backdoor attacks. Besides, it maintains the benign model performance on the main task and causes minimal computational overhead on the server.

Chapter 7

Enhanced security and privacy via fragmented federated learning

Achieving a balance between accuracy, privacy, and security is a tough challenge for FL (Li et al., 2020; Kairouz et al., 2019; Blanco-Justicia et al., 2021). On the one hand, good updates may reveal private information, whereas poisoned updates can compromise the model's availability and/or integrity. On the other hand, enhancing privacy through update distortion can harm accuracy, while doing so through update aggregation can harm security as it prevents the server from filtering out poisoned updates.

Our goal in this chapter is to address the following puzzle: *"Can we prevent a semi-honest server from performing privacy attacks on individual updates while learning an accurate global model and ensuring protection against poisoning attacks?"*

To do so, we propose *fragmented federated learning* (FFL), a framework in which peers randomly exchange fragments of updates among them before sending them to the server. In particular, we bring the following contributions:

- We propose a novel lightweight protocol that i) allows peers to privately exchange and mix random fragments of their updates, ii) enables the server to correctly aggregate the global model from the mixed updates, and iii) prevents the server from recovering the complete original updates or linking them to their originators.
- We propose a new reputation-based defense tailored to FFL against security attacks. Specifically, the server selects peers for training and adaptively aggregates their mixed updates according to their global reputations. Also, honest peers do not exchange fragments with peers having low local reputations. Reputations are computed based on the quality of the updates the peers send and the fragments they exchange.
- We provide extensive theoretical and empirical analyses to assess the accuracy, privacy and security offered by FFL, and we quantify the computation overhead and communication cost it incurs.

The contributions in this chapter have been published in (Jebreel et al., 2022a).

This chapter is organized as follows. Section 7.1 describes the attacks being considered. Section 7.2 presents the fragmented federated learning framework. Section 7.3 and Section 7.4 provide privacy and security analyses of FFL. Section 7.5 and Section 7.6 provide convergence and complexity analyses of FFL. Section 7.7 details the experimental setup and evaluates our approach w.r.t. accuracy, robustness against attacks, and runtime. Section 7.8 gathers the conclusions.

7.1 Attack models

Privacy attack model. We focus on a *semi-honest* server A^s, who follows the protocol honestly, but tries to infer information about the private local data of peers. Even though privacy attacks may also be orchestrated by peers based on the successive global models, the performance

of such attacks is quite limited and degrades significantly as the number of peers increases (Melis et al., 2019). Server-side attacks are much stronger, especially when the server sees local updates individually and can link them to their originators. Chapter 3 has discussed several privacy attacks against FL. To mount any of those attacks, the server needs to access individual local updates. Therefore, our goal is to disable privacy attacks by preventing the server from obtaining the peers' original updates.

Security attack model. In our work, we consider a number of attackers $K' \leq K/5$, that is, no more than 20% of the K peers in the system. Although some works in the literature assume larger percentages of attackers, finding more than 20% of attackers in real-world FL scenarios is highly unlikely. For example, with the millions of users (Davenport and Davenport, 2018) of Gboard (Hard et al., 2018), controlling even a small percentage of user devices requires the attacker(s) to compromise a large number of devices, which demands huge effort and resources and is therefore impractical. We assume the K' attackers carry out two types of attacks: (1) untargeted attacks based on Gaussian noise (Li et al., 2019b; Wu et al., 2020b) and (2) targeted attacks based on label-flipping (Biggio, Nelson, and Laskov, 2012; Fung, Yoon, and Beschastnikh, 2018). Furthermore, we assume that the attacker(s) have no control over the server or the honest peers.

7.2 Fragmented federated learning

In this section, we present the *fragmented federated learning* (FFL) framework. First, we give an overview of our framework and then present its design and protocols in detail. Table 7.1 summarizes the notation used in this chapter.

TABLE 7.1: Notation used in this chapter

Notation	Description
W	Federated learning model or update
W^t	Model or update at round t
$\|W\|$	Number of model parameters
λ	Bitlength of a parameter
w_l	Weight matrix of layer l
b_l	Bias vector of layer l
σ_l	Activation function of layer l
D	Dimensionality of the model
L	Number of layers of the model
D_L	Last-layer dimensionality
K	Number of peers
A	Aggregator server
T	Number of training rounds
C	Fraction of selected peers
n	Number of selected peers
S	Set of selected peers
Sc	Set of candidate peers
E	Number of local epochs
BS	Size of local batch size
η	Local learning rate
W_k	peer k's update
$W_{k,i}$	i-th parameter of W_k
K'	Number of malicious peers
γ^t	Global reputation vector held by the server at time t
γ_k^t	Global reputation given by the server to peer k at time t
$Q1_x$	First quartile of the values of magnitude x
ζ_k^t	Local reputation vector held by peer k at time t
$\zeta_{k,j}^t$	Local reputation given by peer k to peer j at time t
p	Large prime (2048-bit long)
$\mathbb{G}_p$	Multiplicative group of integers mod p
g	Generator of $\mathbb{G}_p$
$PRNG(.)$	Public pseudo-random number generator
$Enc_{pk_A}(\cdot)$	Encryption under A's public key
$Dec_{sk_A}(\cdot)$	Decryption under A's private key
$(W_k)_{mix}$	peer k's mixed update
$(W_k)'_{mix}$	peer k's encrypted mixed update
s_{r_k}	Seed to generate k's one-time pad (OTP)
r_k	OTP used to encrypt k's fragments
m	Binary mask: vector of 0's and 1's
$\neg m$	1's complement of m

7.2.1 Overview

Fig. 7.1 shows an overview of the FFL framework. The key idea is to have the peers randomly fragment and mix their updates before sending them to the server. Specifically, two peers agree on some symmetric random indices in their update vectors and exchange the parameters

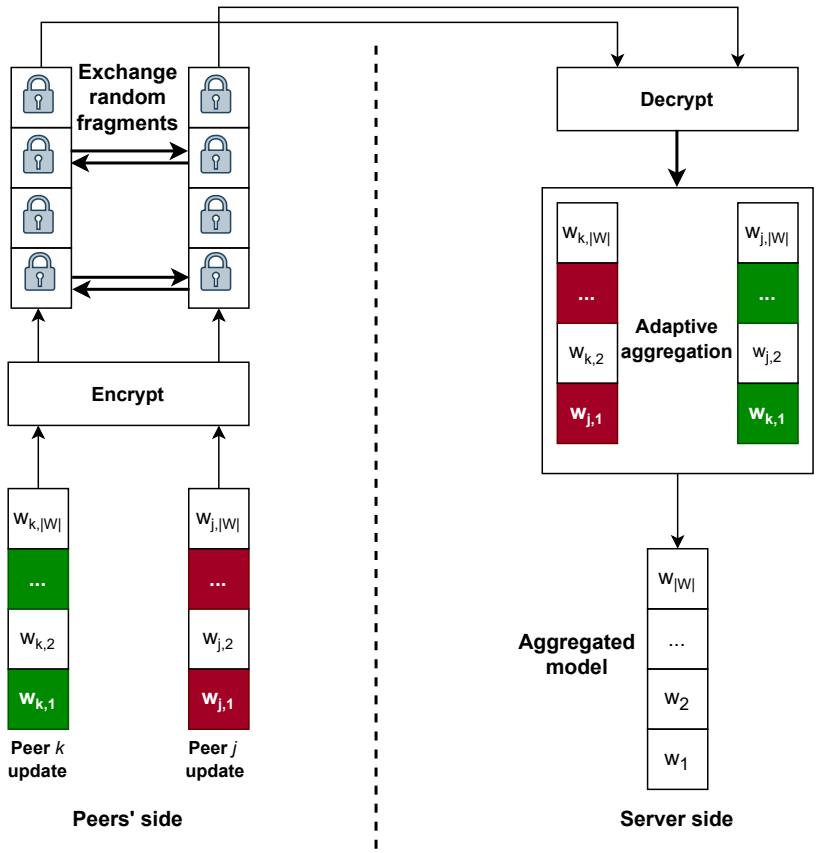


FIGURE 7.1: Overview of the FFL framework

of those indices after they are encrypted. The peers then send the encrypted mixed updates to the server instead of their original updates. Since exchanging parameters is done without changing their original coordinate positions, the server can calculate the average of the mixed updates after decrypting the encrypted mixed updates and obtain the same updated global model that would result from averaging the original updates.

Sending mixed updates instead of the original ones breaks the link between the updates and their originators, and does not give the server direct access to individual updates. Thus, FFL prevents a semi-honest server from mounting powerful privacy attacks. Also, the global model's accuracy is preserved because the parameters of the exchanged fragments are kept unaltered. Yet, exchanging fragments between peers should neither impose significant communication or computation overheads nor protect against server-side privacy attacks at the price of facilitating peer-side privacy attacks.

Thus, we design a fragment exchanging protocol based on lightweight cryptographic tools to: i) allow the peers to exchange and mix random fragments of their updates while incurring very minor communication and computation overheads, ii) prevent peers from seeing each other's original fragments, and iii) allow the server to correctly compute the updated global model from the mixed updates without being able to recover the individual original updates or link them to their originators.

However, averaging updates to compute the global model goes against countering security attacks. Also, the design of the exchange protocol gives n' attackers the chance of poisoning $2n'$ coordinates. We tackle both issues by designing a novel reputation-based defense tailored for FFL that builds trust in peers based on the quality of the mixed updates they send and the fragments they exchange. Specifically, the server holds a global reputation vector and uses it to select peers for training and adaptively aggregate their mixed updates. A peer who repeatedly sends poisoned mixed updates will have a lower global reputation, and thus will not be selected in future training rounds. Also, the mixed updates she sends will have little to no influence on the global model aggregation. On the other hand, each peer holds a local reputation vector and uses it to decide whether to exchange fragments with the other peers. The local reputation increases or decreases based on the quality of the fragments exchanged by peers. An attacker who exchanges poisoned fragments with honest peers will eventually find no honest peer

to exchange with. The similarity between the mixed updates and their centroid is used to measure their quality.

7.2.2 FFL design

There are $K+1$ parties in the FFL framework: K peers and an aggregator server A. We assume that K' peers (with $K' \leq K/5$) may be malicious and attack the global model's availability or integrity by sending poisoned updates to A. Also, we assume that the aggregator server is semi-honest and may use the peers' updates to infer information about their local private data. Before starting any training task, A generates a public/private key pair (pk_A, sk_A) and broadcasts pk_A to the K peers. Then, the server uses the metadata of the peers' devices (like IP or MAC addresses) to register them into the system. After that, A assigns each peer a unique pseudonym. All parties are assumed to possess pairwise secure communication channels using communication protocols such as TLS or HTTPS and communicate with each other using pseudonyms.

Protocol 2 formalizes the framework we propose. At the first global training round, A starts a federated learning task by initializing the global model W^0 and the global reputation vector $\gamma^0 \in \mathbb{R}^K$. The global reputation vector γ is used to select peers for training and re-weight their mixed updates in the global model aggregation. Also, each peer k initializes a local reputation vector $\zeta_k^0 \in \mathbb{R}^{K-1}$ that is used to store a local reputation value for each peer in the system different from k. In the first global training round, both γ^0 and ζ_k^0 are initialized with zero vectors. Later in this section, we explain how γ and ζ_k are calculated and used in our framework.

Adaptive selection of peers. At each round t, A uses Procedure 1 to select a set S of potential honest peers. First, the first quartile of the global reputation $Q1_{\gamma^t}$ is computed. Then, peers with a global reputation greater than or equal to $Q1_{\gamma^t}$ are assigned to the candidate set Sc. Finally, the server selects a random set S of $n = \max(C \cdot |Sc|, 2)$ peers

from Sc. As we explain later in this section, we expect the attackers (up to 20%) to have global reputations less than $Q1_{\gamma^t}$. Note that, in the standard FL setting, $n = \max(C \cdot K, 1)$, but we replace 1 by 2 because FFL needs at least 2 peers to be applicable. After that, A sends the global model W^t to the peers in S through a secure channel. The server also adds the pseudonyms of the selected peers to a public board seen by all peers.

Local training. Each peer $k \in S$ trains the received model on her private local data to obtain her local update W_k. Then, k scales her computed local update by the number of data points she holds, d_k. The next step is for k to randomly select another peer j from the public board to exchange a fragment of her encrypted update W_k with her.

Peer selection for exchange. To avoid making herself a bridge for poisoning the global model, k uses her local reputation vector ζ_k^t to decide whether to exchange fragments with any other peer j. To do so, k computes the first quartile of her local reputation vector $Q1_{\zeta_k^t}$ and assigns the peers in the public board that have local reputations greater than $Q1_{\zeta_k^t}$ to the set Sc_k. Then, k selects a random peer j from Sc_k and asks her to exchange fragments. When j receives the request from k for fragment exchange, she checks the local reputation of k, $\zeta_{j,k}^t$, and if $\zeta_{j,k}^t < Q1_{\zeta_j^t}$, j rejects k's request for exchange. Otherwise, j accepts to exchange fragments with k, and they call protocol EXCHANGE_FRAGMENTS. At the end of the protocol, both k and j obtain two encrypted mixed updates with their corresponding encrypted one-time pad (OTP) seeds: $(W_k)'_{mix}$ and $Enc_{pk_A}(s_{r_j})$ for peer k, and $(W_j)'_{mix}$ and $Enc_{pk_A}(s_{r_k})$ for peer j, and send them to A. Note that the seeds to generate the mixed updates' OTPs are encrypted under A's public key and swapped between k and j. Later in this section, we detail how local reputations are computed.

Fragments exchanging and mixing. The protocol for exchanging fragments is key in our approach. We design this protocol to privately and efficiently exchange fragments of updates between peers. Specifically, a peer k (the *Initiator*) seeks to exchange a random fragment of her

Procedure 2: Fragmented federated learning

Input: K, C, BS, E, η, T
Output: W^T, the global model after T training rounds

A initializes $W^0, \gamma^0 = \{\gamma_k^0 = 0\}_{k=1}^K$;
for *each peer* $k \in [1, K]$ **do**
  k initializes $\zeta_k^0 = \{\zeta_{k,j\neq k}^0 = 0\}$;
end
for *each round* $t \in [0, T-1]$ **do**
  $S \leftarrow$`SELECT_peerS`(C, γ^t);
  A sends W^t to all peers in S
  for *each peer* $k \in S$ ***in parallel*** **do**
    k calls $(W_k^{t+1})'_{mix}, Enc_{pk_A}(s_{r_j}) \leftarrow$ `peer_UPDATE`(k, W^t);
    k sends $(W_k^{t+1})'_{mix}, Enc_{pk_A}(s_{r_j})$ to A;
    A decrypts $Enc_{pk_A}(s_{r_j})$ with its private key to obtain s_{r_j};
    A generates $r_j \leftarrow PRNG(s_{r_j})$;
    A decrypts $(W_k^{t+1})'_{mix}$ with r_j as $(W_k^{t+1})_{mix} \leftarrow (W_k^{t+1})'_{mix} \oplus r_j$;
  end
  `COMPUTE_SIMILARITY()`;
  `UPDATE_REPUTATIONS()`;
  Let ν^t be the computed trust vector from Expression (7.8);
  A aggregates $W^t \leftarrow \frac{1}{\sum_{k\in S} d_k} \sum \nu_k^t (W_k^{t+1})_{mix}$;
end
Function `peer_UPDATE`(k, W^t)
  $W_k \leftarrow W^t$;
  for *each local epoch* $e \in [1, E]$ **do**
    for *each batch* β *of size* BS **do**
      $W_k \leftarrow W_k - \eta \nabla \mathcal{L}(W_k, \beta)$;
    end
  end
  $W_k \leftarrow d_k W_k$;
  exchanged=false;
  Let $Q1_{\zeta_k^t}$ be the first quartile in ζ_k^t;
  Let Sc_k be the set of peers with local reputations greater than $Q1_{\zeta_k^t}$;
  while *not exchanged* **do**
    $j \leftarrow$ select a random peer of Sc_k;
    if $\zeta_{j,k}^t \geq Q1_{\zeta_j^t}$ **then**
      $(W_{mix})', Enc_{pk_A}(s_{r_j}) \leftarrow$ `EXCHANGE_FRAGMENTS`(k, j);
      exchanged=true;
      return $(W_{mix})', Enc_{pk_A}(s_{r_j})$;
    end
  end
end

Procedure 1: Adaptive selection of peers

```
SELECT_peerS(C, γ^t):
    Sc ← [ ];
    Let Q1_{γ^t} be the first quartile of γ^t;
    for each global reputation γ_k^t ∈ γ^t do
        if γ_k^t ≥ Q1_{γ^t} then
            Add(k, Sc);
    n ← max(C · |Sc|, 2);
    S ← select a random set of n peers from Sc;
    return S;
```

update with another peer j (the *Acceptor*). Both peers leverage a key exchange protocol to jointly generate two complementing masks used to fragment their updates. The fragments are then encrypted using one-time pads, that is, random sequences added to the fragments, that allow *Initiator* and *Acceptor* to combine their updates but prevent each other from accessing their counterpart's update parameters. These one-time pads can only be removed by the central server after they have been mixed using the secret information provided by both peers.

Let $PRNG(\cdot)$ be a public pseudo-random number generator that takes an integer as a seed. Let $\mathbb{G}_p$ be the multiplicative group of integers modulo a large prime p (2048-bit long), and g a generator of the group. All subsequent operations with vectors are performed coordinate-wise. Protocol EXCHANGE_FRAGMENTS is as follows:

1. Peer k randomly generates integers a, s_{r_k} and s_{ρ_k}, and sends $g^a \mod p$ and $Enc_{pk_A}(s_{r_k})$ to j.

2. Peer j generates integers b, s_{r_j} and s_{ρ_j} and computes $g^b \mod p$, $r_j = PRNG(s_{r_j})$ and $\rho_j = PRNG(s_{\rho_j})$, with the last two numbers having bitlength $|W|\lambda$ (the bitlength of an update, that is, number of parameters times the bitlength of a parameter). Peer j generates a mask $m = PRNG(g^{ab} \mod p)$ of 0's and 1's of bitlength $|W|$ (equal to the number of update parameters) and its 1-complement

inverse mask $\neg m$ such that $m \oplus \neg m = \mathbf{1}_{|W|}$ (adding modulo 2 a mask to its 1-complement inverse yields the all ones mask). Then, peer j sends $g^b \mod p$, $Enc_{pk_A}(s_{r_j})$, $W_j \oplus r_j \oplus \rho_j$, and $(W_j \odot \neg m) \oplus \rho_j$ to k, where $\odot$ is an operator between an update and a mask that preserves the i-th update parameter if the i-th mask bit is 1, and clears to 0 the i-th update parameter if the i-th mask bit is 0.
The random uniform binary mask m will be used to fragment both peer j's and peer k's updates. Also, both r_j and ρ_j are OTPs known to j only, and used by peer j to hide her original fragments from peer k.

3. Peer k computes $r_k = PRNG(s_{r_k})$ and $\rho_k = PRNG(s_{\rho_k})$ both of length $|W|\lambda$. Peer k generates $m = PRNG(g^{ab} \mod p)$ and its inverse $\neg m$. Then, peer k computes her encrypted mixed update as follows:

$$\begin{aligned}(W_k)'_{mix} &= (W_k)_{mix} \oplus r_j \\ &= W_k \oplus (W_k \odot m) \\ &\oplus (W_j \oplus r_j \oplus \rho_j) \oplus ((W_j \odot \neg m) \oplus \rho_j). \end{aligned} \tag{7.1}$$

In Expression (7.1), the result of subexpression $W_k \oplus (W_k \odot m)$ is to clear to 0 all parameters of W_k at coordinates where the mask m has a 1. On the other hand, subexpression $W_j \oplus (W_j \odot \neg m)$ clears to 0 all parameters of W_j where the mask m has a 0. Bitwise adding both subexpressions yields the mixed update $(W_k)_{mix}$. Note that the two appearances of ρ_j cancel each other and r_j encrypts the mixed update into $(W_k)'_{mix}$. Since the mask m is random, the mixed update can be expected to contain the same number of 1s and 0s. Hence, $(W_k)_{mix}$ can be expected to contain half of the coordinates from W_j and the other half from W_k. Another important remark is that the mixed update is encrypted using r_j, whereas peer k has only $Enc_{pk_A}(s_{r_j})$, so k cannot obtain the full cleartext mixed update.

After that, peer k sends $(W_k \oplus r_k \oplus \rho_k)$, and $(W_k \odot \neg m) \oplus \rho_k$ to j. She also sends $(W_k)'_{mix}$, $Enc_{pk_A}(s_{r_j})$ to the server.

Receiving $Enc_{pk_A}(s_{r_j})$ allows the server to decrypt the encrypted mixed update but does not allow it to extract any original fragment separately or link it to the fragment's originator, as we show in Section 7.3. Moreover, the use of the Diffie-Hellman key exchange method (Diffie and Hellman, 1976) to generate seed g^{ab} ensures that no one but peer k and peer j knows the generated mask m.

4. Similarly, peer j computes her encrypted mixed update as:

$$\begin{aligned}(W_j)'_{mix} &= (W_j)_{mix} \oplus r_k \\ &= W_j \oplus (W_j \odot m) \\ &\oplus (W_k \oplus r_k \oplus \rho_k) \oplus ((W_k \odot \neg m) \oplus \rho_k).\end{aligned} \tag{7.2}$$

Finally, peer j sends $(W_j)'_{mix}$, $Enc_{pk_A}(s_{r_k})$ to the server.

A naive and simpler way to exchange fragments would be to encrypt the updates coordinate by coordinate by using the server public key before exchanging fragments. However, this would cause significant communication and computation overheads for both the server and the participants. Instead, our protocol uses OTPs as a means to encrypt the mixed updates. In other words, we rely on symmetric-key encryption, which is much more efficient when dealing with models that may contain millions of parameters.

Decryption of encrypted mixed updates. When at training round t the server receives a mixed encrypted update $(W_k^{t+1})'_{mix}$ and its corresponding encrypted seed $Enc_{pk_A}(s_{r_j})$, it uses its private key to obtain the clear seed $s_{r_j} = Dec_{sk_A}(Enc_{pk_A}(s_{r_j}))$. Then A regenerates r_j using s_{r_j} as a seed to $PRNG(\cdot)$ and bitwise adds r_j to $(W_k^{t+1})'_{mix}$ to obtain

$(W_k^{t+1})_{mix}$. A does the same for all $k \in S$ to get the plain mixed updates $\{(W_k^{t+1})_{mix} | k \in S\}$.

Computation of reputation and trust values. We explain how reputations are computed and used to neutralize potential attackers in the system. First, the similarities between the mixed updates and their centroid are computed to measure their quality. Second, the global and local reputations are updated based on the computed similarities. Third, the global reputations are used to compute the trust values for the senders of the mixed updates, and these trust values finally weight the mixed updates when aggregating them to obtain the new global model. The above three steps are detailed next:

1. Compute similarity. In this procedure, the server computes the corresponding mixed gradients of the set $\{(W_k^{t+1})_{mix} | k \in S\}$. For a mixed update $(W_k^{t+1})_{mix}$, A computes its corresponding gradient as

$$(\nabla_k^t)_{mix} = (W^t - (W_k^{t+1})_{mix})/\eta. \tag{7.3}$$

The impact of untargeted attacks is expected to be evident in the magnitudes of whole poisoned updates, whereas the impact of the targeted attacks is expected to be more evident in the last-layer gradients that carry the indicative features and map directly to the prediction probability (Fung, Yoon, and Beschastnikh, 2020). Therefore, we independently analyze the whole mixed gradients and the last layer of mixed gradients and then combine the results to simultaneously counter both untargeted and targeted attacks. To do so, A first computes the magnitude of each mixed gradient and obtains $\{||(\nabla_k^t)_{mix}|| | k \in S\}$. Then, it computes the median of the computed magnitudes, med_{mix}^t. Since most mixed updates are good, med_{mix}^t is expected to fall amid good mixed updates. After that, A computes the distance between med_{mix}^t and each magnitude $||(\nabla_k^t)_{mix}||$ as

$$ds_k^t = |med_{mix}^t - ||(\nabla_k^t)_{mix}||. \tag{7.4}$$

For each peer k, ds_k^t represents how far the magnitude of the peer's mixed update is from med_{mix}^t. In the case of untargeted attacks, poisoned mixed updates are expected to deviate more from med_{mix}^t and thus result in larger ds^t values. It would also be possible to compute distances based on the updates themselves and the median update rather than their magnitudes, but this would cause a higher computational overhead on the server. To capture the behavior of targeted attacks, A extracts the mixed gradients of the last layer L and obtains the set $\{(\nabla_k^t)_{mix,L} | k \in S\}$. Then, A computes $med_{mix,L}^t$ as the coordinate-wise median of the previous set. Since honest peers share the same objective and are a majority, the median of the mixed last-layer gradients is expected to lie in the same direction as the honest peers' last-layer gradients. Thus, A computes the cosine similarity between $med_{mix,L}^t$ and each $(\nabla_k^t)_{mix,L}$ as

$$cs_k^t = \cos\varphi = \frac{(\nabla_k^t)_{mix,L} \cdot med_{mix,L}^t}{||(\nabla_k^t)_{mix,L}|| \cdot ||med_{mix,L}^t||}. \tag{7.5}$$

This yields a cosine similarity value $cs_k^{t+1} \in [-1, 1]$ for each peer's mixed update. Note that poisoned mixed updates, being a minority, can be expected to have lower cosine similarity with $med_{mix,L}^t$ than good mixed updates.

To compute a combined similarity value that captures the behaviors of both untargeted and targeted attacks, A performs the following steps:

1. Normalize and invert the computed distances into the range $[0, 1]$ as $ds_k^t = 1 - ds_k^t / \max_{j \in S}(ds_j^t)$.

2. Normalize the cosine similarity vector cs^t into the range $[0, 1]$ as $cs_k^t = (cs_k^t + 1)/2$.

3. Compute the aggregated similarity value of k as $sim_k^t = \alpha ds_k^t + (1 - \alpha) cs_k^t$.

In this way, smaller sim_k^t values are likely to correspond to potential poisoned mixed updates. The hyperparameter $\alpha \in [0, 1]$ is used to tune the simultaneous detection of untargeted and targeted attacks.

2. Update reputations. After computing the vector sim^t containing similarities for all peers $k \in S$, A updates the global reputations of those peers as

$$\gamma_k^{t+1} = \gamma_k^t + (sim_k^t - Q1_{sim^t}), \tag{7.6}$$

where $Q1_{sim^t}$ is the first quartile of similarity values. Based on its computed similarity sim_k^t, k's global reputation increases or decreases: a similarity less than $Q1_{sim^t}$ causes a reputation decrease and may lead to negative reputation. If k is an attacker and always sends poisoned mixed updates, its global reputation will get smaller and smaller and thus he will be excluded by the server from future selection for training. However, an honest peer k could also experience a reputation decrease if she sent an update with a poisoned fragment during the exchange. Therefore, A also sends $sim_k^t - Q1_{sim^t}$ to k, who uses it to update the local reputation of the peer j with whom k has exchanged fragments in training round t:

$$\zeta_{k,j}^{t+1} = \zeta_{k,j}^t + (sim_k^t - Q1_{sim^t}). \tag{7.7}$$

This ensures that, when k obtains a small similarity value because of j's poisoned fragment, she reduces j's local reputation. As a result, if j exchanges poisoned fragments with the other peers, she will get a bad local reputation among all honest peers and thus become an outcast, so that no honest peer will accept to exchange fragments with him in the future.

3. Adaptive model aggregation. The server adaptively aggregates the received mixed updates using the global reputations of their senders. First, A computes the trust vector ν^t as

$$\nu^t = \max(\tanh(\gamma^{t+1} - Q1_{\gamma^{t+1}}), 0). \tag{7.8}$$

The hyperbolic tangent function (tanh) squashes its negative inputs into the range $[-1, 0[$, and its positive inputs into the range $]0, 1]$. Since the attackers are expected to have values lower than $Q1_{\gamma^{t+1}}$, tanh will make their trust scores in ν^t less than 0. The maximum in Expression (7.8) sets the attackers' trust values to 0. Note that, since the reputations of honest peers are likely to increase in every training round, their trust scores will converge to 1 as training evolves, due to the use of tanh. Finally, the server uses the values in the computed trust vector ν^t to re-weight and aggregate the mixed updates (line 18 of Protocol 2). Since the attackers' trust values are 0, their updates are neutralized in the aggregation.

7.3 Privacy analysis

In this section, we theoretically demonstrate the effectiveness of FFL against privacy attacks.

7.3.1 Privacy between peers

Proposition 1. *Two peers j and k exchanging fragments in Protocol EXCHANGE_FRAGMENTS do not learn each other's fragments.*

Proof. Since the protocol is symmetric, we only need to prove that peer k does not learn peer j's fragments. peer k receives the following from peer j: $g^b \mod p$, $Enc_{pk_A}(s_{r_j})$, $W_j \oplus r_j \oplus \rho_j$, and $(W_j \odot \neg m) \oplus \rho_j$. Then peer k can compute the common mask m, but she cannot decrypt s_{r_j}, which would allow her to re-create r_j. However, peer k can add $W_j \oplus r_j \oplus \rho_j$ and $(W_j \odot \neg m) \oplus \rho_j$, which, combined to k's knowledge of m, allows k to learn the bits of r_j that encrypt parameters for which bits in the mask are 0. However, in the mixed update $(W_k)_{mix}$ computed by peer k in Expression (7.1), all parameters from peer j corresponding to mask positions equal to 0 are cleared to 0. Hence, the bits of r_j discovered by k do not allow her to retrieve any parameter of j. □

Note that a third-party intruder observing the exchange of fragments cannot do better than any of the two peers at learning the other peer's parameters. In fact, the intruder is likely to be in a worse position, because he does not know m unless it is leaked by one of the peers.

7.3.2 Unlinking peers from their updates

Proposition 2. *Given a mixed update $(W_k)_{mix}$ obtained by a peer k after exchanging fragments, the probability that a certain subset of u parameters in W_k is entirely present in peer k's mixed update $(W_k)_{mix}$ is $(1/2)^u$.*

Proof. By construction of Protocol EXCHANGE_FRAGMENTS, in the mixed update $(W_k)_{mix}$ the original parameters of peer k are found only where the mask m has bits with value 0. Now, the probability of a specific set of u positions in m being 0 is $(1/2)^u$ if the generator $PRNG$ used to obtain m is good. □

A consequence of Proposition 2 is that mixing updates effectively unlinks them from their originators: indeed, the probability that a specific set of parameters in the original update survives in the mixed update decreases exponentially with the set size. The effectiveness of unlinking updates against privacy attacks is examined in the next sections.

7.3.3 Robustness against membership inference attacks

Membership inference attacks (MIAs) (Nasr, Shokri, and Houmansadr, 2019; Melis et al., 2019) leverage a peer's local update W_k to infer if a specific data point (x, y) was part of her training data. In Nasr, Shokri, and Houmansadr, 2019, a semi-honest server exploits the distinguishable pattern that (x, y) leaves on W_k. To carry out the attack, the server trains a binary classifier using some available data containing member and non-member data points and the components of a model trained on the member data points. The binary classifier predicts a membership score for any target data point (x, y). The membership score is the

probability that (x, y) belongs to a target peer's training data. The attack components include: the calculated gradient vector on the data point, the activations of the intermediate layers of the peer's model, the activation of the output layer, and a scalar representing the loss of the model on the data point. The authors demonstrate that the gradient vector on the target data point is the most important component for the success of the attack. According to the way a semi-honest server performs MIAs, we can state the following proposition.

Proposition 3. *Given a mixed update $(W_k)_{mix}$ of a peer k, it is not possible to correctly predict the membership score of a target data point (x, y) belonging to k by using $(W_k)_{mix}$.*

Proof. The binary classifier needs the learned pattern (that is, the correct attack components) to correctly predict the membership score of (x, y). However, in FFL, the semi-honest server will compute an intermediate layer activation $f_{k,l}(x) = \sigma_l((w_{k,l})_{mix} \cdot x + (b_{k,l})_{mix})$ instead of $f_{k,l}(x) = \sigma_l(w_{k,l} \cdot x + b_{k,l})$. This will result in random activations and a random loss scalar as well. Based on that, calculating the gradient vector by backpropagating from a wrong loss scalar through the parameters of a mixed model will result in a completely random attack component, and hence in a random membership score prediction for the data point (x, y). Therefore, FFL prevents the semi-honest server from correctly predicting the membership score of any target data point. □

7.3.4 Robustness against property inference attacks

Property inference attacks (Ganju et al., 2018; Melis et al., 2019) try to infer specific properties about the training data by recognizing patterns within a peer's local model. Melis et al., 2019 show how to infer properties of a peer's training data that are uncorrelated to the main task features. The idea of the attack is to use a peer's update to infer properties that characterize a subset of her training data. A semi-honest server can

use some auxiliary data, with and without the property, to generate updates with and without that property. Then, it uses the gradients of the generated updates as input features to train a binary classifier. The binary classifier is used to distinguish if an input gradient was computed on data with the same target property. Finally, when the server receives a target peer's update W_k, it extracts its gradient as

$$\nabla W_k = \frac{W - W_k}{\eta}. \tag{7.9}$$

Then the server passes ∇W_k to the binary classifier to determine if the peer's data have the target property.

We can notice that, like the classifier in the MIA attack, the classifier in this attack mainly depends on the original gradient vectors. FFL prevents the server from obtaining the whole original gradient of the target peer. In general, her mixed gradient is inconsistent with the pattern the binary classifier was trained on. Thus, the classifier decision will most likely be inaccurate.

7.3.5 Robustness against reconstruction attacks

Reconstruction attacks (Zhu and Han, 2020; Zhao, Mopuri, and Bilen, 2020; Geiping et al., 2020) are much stronger than the previous ones, since they can extract both the original training inputs and the labels from a peer's local gradient. The idea behind these attacks is that a peer's update W_k is computed based on both the global model W and the peer's training data (x_k, y_k). Since a semi-honest server has both W and W_k, it can obtain the training data by inverting the update gradient. First, the server computes peer k's gradient ∇W_k by using Expression (7.9). After that, it tries to invert the gradient and to find the unknown training data (x_k, y_k) that result in the same extracted gradient: it starts by randomly initializing a dummy input x^* and a label input y^*, and

feeds these "dummy data" to the global model W to get "dummy gradients" ∇W^* as

$$\nabla W^* = \nabla \mathcal{L}(W, (x^*, y^*)). \tag{7.10}$$

Then, the server repeatedly modifies the dummy data in an adversarial perturbation way, based on the difference between the dummy gradient ∇W^* and peer k's original gradient ∇W_k. A small distance between ∇W^* and ∇W_k means that the dummy data are similar to the original data. The authors of Zhu and Han, 2020; Zhao, Mopuri, and Bilen, 2020 use the Euclidean distance between ∇W_k and ∇W^* as the objective function to modify the dummy data, whereas Geiping et al., 2020 employ the cosine similarity. The objective function used in Geiping et al., 2020 is given by

$$argmin_{x^*,y^*} \left(1 - \frac{< \nabla W_k, \nabla W^* >}{||\nabla W_k|| ||\nabla W^*||}\right). \tag{7.11}$$

Based on the above we can state the following proposition.

Proposition 4. *A mixed gradient $(\nabla W_k)_{mixed}$, sent by a peer k, cannot be leveraged by the server to estimate a target data point (x, y) in k's local data.*

Proof. When using a mixed gradient $(\nabla W_k)_{mixed}$ instead of the original ∇W_k, the objective functions used in the reconstruction attacks will in general result in a random reconstructed data point because in general there is no original data point corresponding to that mixed gradient. □

Proposition 4 guarantees that by providing the server with mixed updates instead of the original ones, FFL effectively prevents the semi-honest server from performing reconstruction attacks.

7.4 Security analysis

When exchanging fragments with another honest peer k, an attacker j' can follow one of three strategies:

- **Strategy 1**. Exchange her poisoned fragment with k and send a poisoned mixed update to the server containing her other poisoned fragment and the good fragment of k. With an expected number of $n/5$ attackers in a training round, there is a chance of poisoning at most $2n' = 2n/5$ mixed updates and at most $n' = n/5$ coordinates.

- **Strategy 2**. Exchange her poisoned fragment with k and send a fully poisoned mixed update to the server. Thus, there is a chance of poisoning at most $2n' = 2n/5$ mixed updates and at most $2n' = 2n/5$ coordinates.

- **Strategy 3**. Exchange her poisoned fragments with the other peers and submit fully good updates to the server. The attacker does this to increase his global reputation and discredit the honest peers in front of the server. This strategy poisons fewer updates than Strategies 1 or 2: at most $n' = n/5$ mixed updates and at most $n' = n/5$ coordinates.

Now let us see how FFL can counter the above strategies and neutralize the impact of poisoned mixed updates on the global model aggregation. In *Strategy 1*, since the number of untouched good coordinates is a majority ($4n/5$), the poisoned mixed updates will have less similarity to the centroid of the mixed update. This lower similarity will decrease the global reputations of both attackers and some honest peers, and the local reputations of the attackers. But since an honest peer is more likely to select another honest peer for the exchange, honest peers will find an opportunity to increase their global reputations and offset the harm caused by the attackers' poisoned fragments. That is because the probability of an honest peer selecting another honest peer is $(n - n' - 1)/(n - 1) = (4n - 5)/(5n - 5)$, whereas the probability of selecting an attacker is $(n')/(n - 1) = n/(5n - 5)$. Moreover, as the training evolves, the attackers will obtain smaller and smaller local reputations (less than the first quartile in the local reputation vectors). As

a result, honest peers will not accept exchanging fragments with them. This will force attackers to send their poisoned updates directly to the server or keep them. If they send them, their global reputations will decrease more and more to be below the first quartile $Q1_\gamma$, which will completely neutralize their influence on the global model, because they will not be selected for future training. On the other hand, if they refrain from sending their poisoned updates, they will neutralize themselves. Note that, even if some attackers managed to have global reputations slightly greater than the first quartile in the early training rounds, they would have less influence on the global model aggregation than honest peers, because they will have small trust scores. Fig. 7.2 shows an example of how the average trusts of honest peers and attackers evolve as the training evolves when the attackers follow Strategy 1. In *Strategy 2*, the

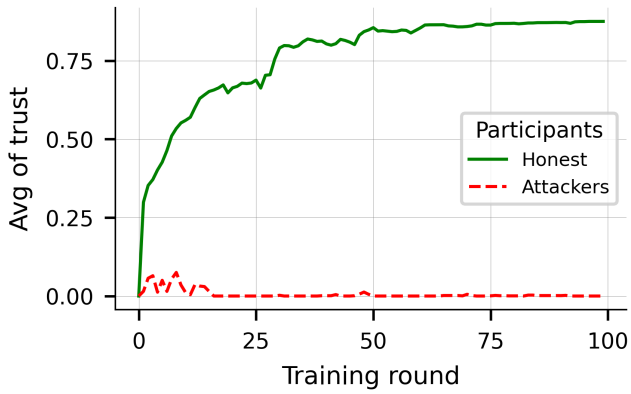


FIGURE 7.2: Peer average trust evolution during training in the CIFAR10-VGG16 benchmark

situation does not differ much. The attackers will get still lower similarity values than under Strategy 1, because they send fully poisoned updates, which will be farther from the centroid than the poisoned mixed updates sent by the honest peers. This will be reflected in their global

and local reputations, which will lead to attackers getting neutralized. As for *Strategy 3*, the attackers will get good global reputations because they always send good mixed updates; however, they will get low local reputations with honest peers. This will deter honest peers from making future exchanges with the attackers. Hence, the attackers will end up being unable to poison the mixed updates sent by honest peers.

7.5 Convergence analysis

To prove the convergence of FFL, first we need to prove that applying the adaptive aggregation in Protocol 1 on the mixed updates gives the same result as applying FedAvg on the original updates when all peers are honest.

Proposition 5. *Given that parameters move among peers but remain in their original coordinates when mixing updates, applying the adaptive aggregation in Protocol 1 on the mixed updates when all peers are honest produces the same result as applying FedAvg on the original updates.*

Proof. Let $g_c : \mathbb{R}^n \mapsto \mathbb{R}$ be the coordinate-wise average operator which aggregates each coordinate independently from the others, and $(W_{1,i}, \ldots, W_{n,i})$ be the i-th coordinate parameters for n local updates. Let $\pi : \mathbb{R}^n \mapsto \mathbb{R}^n$ be a random permutation function that is applied to the i-th coordinate parameters to yield one of $n!$ possible permutations. Given the permutation-insensitivity property of g_c, it holds that

$$g_c(W_{1,i}, \ldots, W_{n,i}) = g_c(\pi(W_{1,i}, \ldots, W_{n,i})) \tag{7.12}$$

for any permutation function π. Hence, applying g_c on mixed updates will produce the same result as applying it on the original updates as long as the exchanged parameters remain in their original coordinates.

FedAvg is a weighted average operator that aggregates the parameters coordinate-wise proportionally to the number of data points of their

senders. Let us assume the server gives a trust score of 1 to every sender as they are honest. Given that each peer k weights her original local update by her number of data points d_k, the adaptive aggregation process in Protocol 1 will produce the same global model as applying FedAvg on the original local updates. □

Proposition 6. *Given the process in Protocol 1, the convergence of FFL is guaranteed as long as FedAvg converges.*

Proof. Given Proposition S.5, we just need to prove that FFL fulfills two conditions: 1) it removes poisoned mixed updates from model aggregation and 2) it does not modify the parameters of good mixed updates in the adaptive aggregation phase.

Condition 1: As the training evolves, the attackers who exchange poisoned fragments or send poisoned updates are expected to obtain low global and local reputations and thus not be selected by the server or honest peers for future training or fragments exchange. Therefore, poisoned mixed updates are expected to be removed from the global model aggregation.

Condition 2: As the training evolves, peers with global reputations greater than $Q1_\gamma$ are expected to be honest and thus be repeatedly selected by the server for future training rounds. This will make their global reputations greater and greater, and thus their trust values will converge to 1 due to the use of the tanh functions. That is,

$$\lim_{t\to\infty} \tanh\left((\gamma_k^t | \gamma_k^t > Q1_{\gamma^t}) - Q1_{\gamma^t}\right) = 1. \tag{7.13}$$

As a result, the parameters of the mixed updates of those peers will stay unaltered. Since honest peers above $Q1_\gamma$ are about 75% of the K peers in the system, the average of their mixed updates is expected to be an unbiased estimator of the average of the K peers' original updates when all peers are honest. Let W_*^∞ be the global model obtained when all peers are honest, and let $(W_k^\infty)_{mix}$ and W_k^∞ be, respectively, peer k's mixed and

original updates, after ∞ iterations. Let H be the set of expected honest peers with global reputations greater than $Q1_{\gamma^t}$. Then we have

$$\begin{aligned}\lim_{t\to\infty} \frac{1}{\sum_{k\in H} d_k} \sum_{k\in H} \nu_k^t (W_k^t)_{mix} &\simeq \frac{1}{\sum_{k\in H} d_k} \sum_{k\in H} (W_k^\infty)_{mix} \\ &\simeq \frac{1}{\sum_{k\in H} d_k} \sum_{k\in H} d_k W_k^\infty \simeq W_*^\infty.\end{aligned} \tag{7.14}$$

This shows that Condition 2 is satisfied. □

7.6 Complexity analysis

In this section, we analyze the computation and communication cost overheads of FFL.

7.6.1 Computation overhead of FFL

We examine the computation overhead added by FFL to standard FL for the server and peers, and we compare it with related works in the state of the art. We also consider the last-layer dimensionality D_L when looking at FFL and FoolsGold (Fung, Yoon, and Beschastnikh, 2020). Note that the last layer usually contains a very small number of parameters compared to the whole model parameters and thus has a much lower dimensionality. For example, in the VGG16 model we use in our experiments, the model contains about 15 million parameters, whereas its last layer contains about $250K$ parameters.

Server side. The computation overhead on the server side can be broken down into the following parts: (1) selecting peers for training based on their global reputations, which costs $\mathcal{O}(n)$; (2) decrypting n encrypted seeds and their corresponding n OTP-encrypted mixed updates, which costs $\mathcal{O}(nc + nD)$, with c being a constant; (3) extracting the gradients of the mixed updates, computing their magnitudes, computing the median of the computed magnitudes and the distances between

the computed magnitudes and their median, which costs $\mathcal{O}(nD + nD + n\log n + n)$, (4) computing the median of the last-layer gradients of n mixed updates and the cosine similarity between the last-layer gradients to their median, which costs $\mathcal{O}(n\log nD_L + nD_L)$; and (5) computing the similarity vector sim^t, updating the global reputation vector and finally computing the trust vector v^t, which costs $\mathcal{O}(n)$. Therefore, the overall computation overhead of the server is $\mathcal{O}(n(D + \log n + D_L \log n + D_L + 1))$.

Peer side. The peer's computation overhead can be broken down into the following parts: (1) computing the first quartile of her local reputation vector using the Quickselect algorithm, which costs $\mathcal{O}(n)$; (2) computing the 2048-bit Diffie-Hellman power, which costs $\mathcal{O}(1)$; (3) fragmenting the local update, encrypting two updates using the two generated OTPs, and subtracting one OTP-encrypted update from the other, which costs $\mathcal{O}(D + 2D + D)$; and (4) encrypting a 3072-bit seed and finally updating the local reputation of a single peer in a single index, which costs $\mathcal{O}(1 + 1)$. Therefore, the overall computation overhead of a peer is $\mathcal{O}(n + D)$.

Now, let us compare the computation overhead of FFL with that of the following methods: BREA (So, Güler, and Avestimehr, 2020), the median (Yin et al., 2018), the trimmed mean (Yin et al., 2018), multi-Krum (Blanchard et al., 2017), and FoolsGold (Fung, Yoon, and Beschastnikh, 2020). Note that, while BREA tries to counter both privacy and security attacks, the other methods are only meant to counter security attacks.

Table 7.2 shows that FFL imposes less computation overhead on the server than the other methods. If we compare the computation overhead of FFL with that of BREA on both the server and the peer, FFL is much more efficient and thus more suitable for large-scale FL applications.

TABLE 7.2: Comparison of computation overhead

Framework	Server(s)	peer
FFL	$\mathcal{O}(n(D + \log n + D_L \log n + D_L + 1))$	$\mathcal{O}(n + D)$
BREA	$\mathcal{O}((n^2 + nD) \log^2 n \log \log n)$	$\mathcal{O}(nD + n^2)$
Median	$\mathcal{O}(n \log nD)$	0
Trimmed mean	$\mathcal{O}(n \log nD)$	0
Multi-Krum	$\mathcal{O}(n^2 D)$	0
FoolsGold	$\mathcal{O}(n^2 D_L)$	0

7.6.2 Communication cost of FFL

Here we turn to the total communication cost of FFL with respect to the model size D and the number of peers n. These two parameters are relevant because D has a high impact on the size of the transmitted messages whereas n has a high impact on their number. We also use c to denote a constant communication cost for any sent or received parameter with a constant size, such as the encrypted seeds.

Server side. The communication cost at the server side can be broken down into the following parts: (1) sending one global model to each peer and receiving one OTP-encrypted mixed update from each peer, which costs $\mathcal{O}(2nD)$; and (2) sending a 32-bit global reputation γ_k to each peer and receiving a 3072-bit encrypted seed with each OTP-encrypted mixed update, which costs $\mathcal{O}(\sim 3nKbit) = \mathcal{O}(nc)$. Therefore, the overall communication cost of the server is $\mathcal{O}(2nD + nc)$.

Peer side. The peer's communication cost can be broken down into the following parts: (1) receiving one global model from the server and sending one OTP-encrypted mixed update to the server, which costs $\mathcal{O}(2D)$; (2) sending two OTP-encrypted updates and receiving two OTP-encrypted updates, which costs $\mathcal{O}(4D)$; and (3) exchanging two 2048-bit Diffie-Hellman powers (see Lepinski and Kent, 2008 for justification of this 2048 length) and two 3072-bit encrypted seeds (see Barker et al., 2006 for justification of this 3072 length) with another peer, plus sending 3072-bit encrypted seed to the server and receiving a 32-bit global

TABLE 7.3: Comparison of communication cost, where δ is the message expansion factor resulting from HE, and c is a constant communication cost for any sent or received parameter with a constant size, such as the encrypted seeds.

Framework	Server(s)	peer
FL	$\mathcal{O}(2nD)$	$\mathcal{O}(2D)$
FFL	$\mathcal{O}(2nD + nc)$	$\mathcal{O}(6D + 6c)$
BREA	$\mathcal{O}(2nD + n^2c)$	$\mathcal{O}(2nD + D + n)$
PEFL	$\mathcal{O}(4\delta nD + \delta D)$	$\mathcal{O}(\delta D + D)$
ShieldFL	$\mathcal{O}(13\delta nD)$	$\mathcal{O}(2\delta D)$

reputation γ_k from it, which costs $\mathcal{O}(6c)$. Therefore, the overall communication cost for each peer is $\mathcal{O}(6D + 6c)$.

Now, let us compare the communication cost of FFL with that of each of the standard FL, BREA (So, Güler, and Avestimehr, 2020), PEFL (Liu et al., 2021) and ShieldFL (Ma et al., 2022b). Table 7.3 shows the communication complexity of each framework on the server(s) and the peer.

In standard FL, the server sends n copies of the global model to n peers and receives n local updates back, which costs $\mathcal{O}(2nD)$. FFL imposes the same cost on the server plus $\mathcal{O}(nc)$, which corresponds to the received n encrypted seeds, and the sent n global reputations. Since the server receives the encrypted seeds along with her encrypted mixed updates in the same messages and sends the 32-bit global reputations along with the updated global models in the same messages, FFL sends and receives the same number of messages as in standard FL. Moreover, the server only incurs 3072 bits $\approx$ 0.0004 MB communication overhead for every received mixed update and 32 bits for every sent updated global model, which is negligible compared to the sizes of the current state-of-art deep learning models. For example, the sizes of the three deep learning models we use in our experimental analysis below are 0.0874 MB (for a CNN model), 58.9131 MB (for a VGG model), and 50.6453 MB

(for a BiLSTM model). Therefore, FFL imposes almost the same communication cost on the server as standard FL.

On the peer's side, in FFL each peer sends and receives 6 updates in total and also incurs $6c$ additional cost, as note above. Therefore, the number of messages a peer needs to send or receive is 6 in total, which is three times as many as in standard FL. Nevertheless, the communication cost each peer incurs is reasonable in exchange for the benefit she receives in protecting her privacy and learning a more accurate global model, and also in comparison with the other works.

To put these values in context, let us compare them with the communication cost of BREA, PEFL, and ShieldFL, three state-of-the-art frameworks that provide a similar level of privacy and accuracy as FFL.

In BREA, the server sends and receives the same number of updates. However, it incurs an additional quadratic communication cost from receiving the locally computed Euclidean distances from the n peers. This quadratic term can make the server a bottleneck of communication as the number of peers in the system increases. FFL, however, imposes a much lower communication cost on the server, as we have shown. On the peer side, the communication cost of BREA depends on both the number of peers n and the model dimensionality D. In real-world FL scenarios, we expect both n and D to be large, and hence a significant communication cost will be imposed on the peer. In contrast, the communication cost on the peer side in FFL is independent of the number of peers in the system.

Both PEFL and ShieldFL are HE-based frameworks that expand updates by some factor δ (which is usually large) and thus add a necessarily significant communication and computation overhead on the server and peers. PEFL tries to reduce the size of an encrypted update by using a packing technique, which results in roughly a threefold message expansion, as the authors claim. This makes the server's communication cost about $\mathcal{O}(12nD + 3D)$ and the peer's cost about $\mathcal{O}(4D)$. Note that the

packing and HE cause a high computation overhead on the peers' devices. Moreover, the two non-colluding servers in these two frameworks exchange encrypted updates and some other parameters between them several times so that they can filter the poisoned updates while protecting privacy, which involves exchanging several large-size messages.

To summarize, FFL imposes almost the same communication cost on the server as standard FL (without privacy preservation), and only a marginally higher communication cost on peers. This makes it more suitable for real-world privacy-preserving FL than the state-of-the-art methods.

To confirm the theoretical communication analysis above, we next provide some actual communication times that may illustrate the feasibility of our approach's (and its practical benefits w.r.t. related works). To this end, we have considered the average speeds of mobile internet communication among countries worldwide [1]. Specifically, the current average download speed, upload speed and latency are, respectively, 31.01 megabits per second (*Mbps*), 8.66 *Mbps* and 29 milliseconds (*ms*).

According to these figures, we next report the communication and latency times of FFL for the MNIST-CNN and the CIFAR1010-VGG16 benchmarks, and we compare them with those of standard FL and BREA. Comparing with BREA makes sense because it is, like our framework, a single-server framework and it also involves exchanging messages between peers. We were not able to compare with PEFL or ShieldFL because they do not give the exact value of the expansion factor δ used and also because they are two-server frameworks. In this setting, the sizes of the CNN and VGG16 models are $0.0874MB$ ($0.6992Mbit$) and $58.9131MB$ ($471.3048Mbit$), which represent small and large DL models. Also, we consider $100K$ peers in the system, which would correspond to a large-scale FL system.

Communication times are based on the following constraints:

[1] https://www.speedtest.net/global-index. Checked on Aug. 1, 2022.

- When a peer receives (downloads) a message from the server, we consider her/his download speed.
- When a peer sends a message to the server, we consider her/his upload speed.
- When two peers exchange messages, we consider their upload speed.

We next report communication times for each framework's server-peer and peer-peer interaction for one training round.

FL communication time. In standard FL, each peer downloads a copy of the global model from the server and uploads her/his local update to the server, so there is no peer-peer communication. Thus, the server-peer communication time for the MNIST- CNN is $0.6992Mbit/31.01Mbps + 0.6992Mbit/8.66Mbps \approx 0.1033$ seconds. By adding one latency for downloading and one for uploading, we obtain that the total communication time is $0.1033 + 2 \times 29/1000 \approx 0.1613$ seconds. The communication time of CIFAR10-VGG16 can be computed following the same steps, which result in a total communication time of about 69.6797 seconds.

FFL communication time. In FFL, there are server-peer and peer-peer communications. In the server-peer communication, each peer downloads from the server a copy of the global model along with a 32-bit global reputation value in the same message. Also, each peer uploads one OTP-encrypted mixed update to the server along with a $3,072$ bit encrypted seed in the same message. Thus, the server-peer communication time for MNIST-CNN is $(0.6992Mbit + 0.000032Mbit)/31.01Mbps + (0.6992Mbit + 0.003072Mbit)/8.66Mbps \approx 0.1036$ seconds. In the peer-peer communication, each peer sends one $2,048$-bit Diffie-Hellman power and receives one $2,048$-bit Diffie-Hellman power. Also, s/he sends two OTP-encrypted updates plus a $3,072$ bit encrypted seed and receives the same. Thus, the peer-peer communication time is $(4 \times 0.6992Mbit + 2 \times 0.002048Mbit + 2 \times 0.003072Mbit)/8.66Mbps \approx 0.3241$ seconds. By

adding the latencies, we get a total communication time for MNIST-CNN of about $0.1036 + 2 \times 29/1000 + 0.3241 + 4 \times 29/1000 = 0.6018$ seconds. The communication time of CIFAR10-VGG16 can be computed following the same steps, which result in a total communication time of about 287.4900 seconds. Therefore, the FFL communication time overheads over the standard FL for MNIST-CNN and CIFAR10-VGG16 are 273.09% and 312.59%, respectively.

BREA communication time. In BREA (So, Güler, and Avestimehr, 2020), there are server-peer and peer-peer communications. In the server-peer communication, each peer downloads a copy of the global model from the server. Also, each peer uploads to the server a $100K$ floating-point vector (the locally computed Euclidean distances) and then uploads the aggregated shares. We consider the vector to be a 32-bit floating-point. Thus, the server-peer communication time for MNIST-CNN is $(0.6992Mbit)/31.01Mbps + (0.6992Mbit + 0.000032Mbit \times 100,000)/8.66Mbps \approx 0.4728$ seconds. In the peer-peer communication, each peer sends $100K$ shares and receives $100K$ shares. Thus, the peer-peer communication time is $(2 \times 0.6992Mbit \times 100,00)/8.66Mbps \approx 16,147.8060$ seconds. By adding latencies, we get a total communication for MNIST-CNN of about $0.4728 + 3 \times 29/1000 + 16,147.8060 + 200,000 \times 29/1000 = 21,948.3658$ seconds. The communication time for CIFAR10-VGG16 can be computed following the same steps, which result in a total communication time of about $10,890,507.4916$ seconds. Therefore, the BREA communication time overheads over standard FL for MNIST-CNN and CIFAR10-VGG16 are $13,607,070.37\%$ and $15,629,283.44\%$, respectively.

These figures, summarized in Table 7.4, show that the communication time imposed by FFL is significantly lower than that of the BREA, which provides similar security and privacy to FFL.

Note that we do not include the time required to establish communications, which is expected to be small compared to the message-dependent times we consider.

TABLE 7.4: Comparison of communication time in seconds for one training round for the MNIST-CNN and CIFAR10-VGG16 benchmarks. S-P indicates the server-peer communication time. P-P indicates the peer-peer communication time.

Framework/ Benchmark	MNIST-CNN				CIFAR10-VGG16			
	S-P	P-P	Latency	Total time	S-P	P-P	Latency	Total time
FL	0.1033	0	0.0580	0.1613	69.6217	0	0.0580	69.6797
FFL	0.1036	0.3241	0.1740	0.6018	69.6220	217.6939	0.1740	287.4900
BREA	0.4728	16,147.8060	5,800.0870	21,948.3658	69.9912	10,884,637.4134	5,800.0870	10,890,507.4916

7.7 Experiments

In this section, we report empirical results on three real data sets for the most relevant security and privacy attacks discussed above. Our code and data are available for reproducibility purposes[2].

7.7.1 Experimental setup

In all the experiments, the *Acceptor*, respectively the *Initiator*, looped through all the layers of her update and generated a random binary $mask_l$ for each layer $l \in [1, L]$, where L is the total number of layers in the global model W. At the end, the *Acceptor*, resp. the *Initiator*, set $mask = mask_l||\ldots||mask_L$, where $||$ is the concatenation operator, and used *mask* to exchange a random fragment.

We used the Diffie-Hellman key exchange protocol with the secure 2048-bit MODP group (Lepinski and Kent, 2008) to generate the secret shared seeds of the fragments' masks, and we used PyCryptodome, 2021[3] to encrypt and decrypt the OTP seeds with the recommended 3072-bit key size. Note that, with PyTorch, the parameter bitlength is $\lambda = 32$. We used a value of $\alpha = 0.2$ because we found it gives better simultaneous detection of untargeted and targeted attacks. That is because targeted attacks are stealthier than untargeted ones.

[2] https://github.com/anonymized30/FFL

[3] PyCryptodome is a Python package that includes a self-contained set of cryptographic primitives at a low-level.

Data distribution and training. We tested the robustness of FFL against poisoning attacks on three ML tasks: tabular data classification, image classification and sentiment analysis. We defined the following benchmarks by distributing the data from the data sets, described in Chapter 2 among the participating peers in the following way:

- Adult-MLP. We randomly and uniformly split the training examples of the Adult data set among 20 FL peers. We used a multilayer perceptron (MLP) with one input layer, one hidden layer and one output layer that contains about 5K learnable parameters. The output layer is followed by a Sigmoid function to produce the final predicted class for an input record. The MLP was trained during 100 rounds. In each round, the FL server selected 10 peers and asked them to train the model for 1 local epoch and a local batch size 64. The peers used the binary cross-entropy with logit loss function and the Adam optimizer with a learning rate = 0.001 to train their models.

- MNIST-CNN. We randomly and uniformly split the MNIST 60K training examples among 100 simulated peers of an FL setting. We used a two-layer convolutional neural network (CNN) with two fully connected layers. The CNN model was trained during 200 rounds. In each round, the FL server randomly chose 50 peers and asked them to train the model for 3 local epochs and a local batch size 64. The peers used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with a learning rate = 0.001 and momentum = 0.9 to train their models.

- CIFAR10-VGG16. We randomly and uniformly split the CIFAR10 50K training examples among 20 FL peers. We used the VGG16 CNN model with one fully connected layer (Simonyan and Zisserman, 2014). The VGG16 model was trained during 100 rounds. In each round, the FL server randomly chose 10 peers and asked them to train the model for 3 local epochs and a local batch size

32. The peers used the cross-entropy loss function and the SGD optimizer with a learning rate = 0.01 and momentum = 0.9 to train their models.

- IMDB-BiLSTM. We randomly and uniformly split the IMDB 40K training examples among 20 FL peers. We used a Bidirectional Long/Short-Term Memory (BiLSTM) model, which has an embedding layer of 100 dimensions for each token. The model ends with a linear layer followed by a Sigmoid function to produce the final predicted sentiment for an input review. The BiLSTM model was trained during 100 rounds. In each round, the FL server randomly chose 10 peers and asked them to train the model for 1 local epoch and a local batch size 32. The peers used the binary cross-entropy with logit loss function and the Adam optimizer with a learning rate = 0.001 to train their models.

To evaluate the effectiveness of FFL at defeating reconstruction attacks, we used the code provided by the authors of Geiping et al., 2020, who perform reconstruction attacks with the *ConvNet64* model described in their paper (with about 3 million parameters) and some images from the *CIFAR10* validation set.

Evaluation metrics. We used the following evaluation metrics on the test set examples to assess the impact of the attacks on the learned model and the performance of the proposed framework w.r.t. the state of the art:

- *Test error (TE)*. This is the error resulting from the loss functions used in training. The lower the test error, the more robust the method is against the attack.
- *Overall accuracy (All-Acc)*. This is the number of correct predictions divided by the total number of predictions.
- *Source class accuracy (Src-Acc)*. We evaluated the accuracy for the subset of test examples belonging to the source class. One may

achieve a good overall accuracy while degrading the accuracy of the source class.

- *Attack success rate (ASR)*. This is the proportion of targeted examples (with the source label) that are incorrectly classified into the label desired by the attacker.

An effective defense needs to retain the benign performance of the global model on the main task while reducing ASR.

7.7.2 Experimental evaluation

Robustness against poisoning attacks

We next report results on the robustness of FFL against two security attack strategies: in *Strategy 1*, the attacker generates two poisoned fragments, one of which he exchanges with another peer k, and he sends to the server a mixed update consisting of his other poisoned fragment and the good fragment of k; in *Strategy 2*, the attacker exchanges a poisoned fragment with k and sends a poisoned update to the server.

Also, we compare the performance of FFL with the performance of FedAvg (McMahan et al., 2017a), the median (Yin et al., 2018), the trimmed mean (Yin et al., 2018), multi-Krum (Blanchard et al., 2017), and FoolsGold (Fung, Yoon, and Beschastnikh, 2020) on standard FL. Notice that FedAvg does nothing to counter security attacks (it systematically aggregates all received updates).

We evaluated TE and All-Acc under the Gaussian noise untargeted attack, and TE, Src-Acc, and ASR under the label-flipping targeted attack. We used standard FL with FedAvg when no attacks were performed as a baseline to show the impact of attacks on the model performance. In our experiments, the percentage of attackers was 20% for all four benchmarks.

Gaussian noise attack. The attackers added Gaussian noise to their updates to prevent the model from converging (Li et al., 2019b; Wu et

TABLE 7.5: Robustness against Gaussian noise attacks.
Best scores are in bold.

	Benchmark/ Method	Adult-MLP		MNIST-CNN		CIFAR10-VGG16		IMDB-BiLSTM	
		TE	All-Acc%	TE	All-Acc%	TE	All-Acc%	TE	All-Acc%
FL	FedAvg (no attacks)	0.349	82.56	0.112	96.69	0.881	80.77	0.475	88.63
	FedAvg	1.198	75.08	2.322	9.65	10.324	10.0	0.618	67.7
	Median	0.349	**82.87**	0.115	96.62	1.020	78.91	0.524	88.33
	Trimmed mean	0.350	82.61	0.114	96.64	1.046	**80.23**	**0.457**	**88.79**
	Multi-Krum	0.350	82.69	0.126	96.18	0.998	78.86	0.565	87.72
	FoolsGold	44.051	69.47	2.741	8.92	12.813	9.02	2.694	51.61
FFL	Strategy 1	**0.349**	82.84	**0.112**	**96.67**	**0.931**	79.74	0.536	88.15
	Strategy 2	0.350	82.86	0.113	96.61	0.938	79.26	0.542	87.48

al., 2020b). Specifically, they added noise with 0 mean and 0.5 standard deviation for the MLP and CNN model parameters, and 0.2 standard deviation for VGG16 and BiLSTM parameters.

Table 7.5 shows the results under this attack. First, we can see the significant negative impact of the attack on the performance of FedAvg regarding both the test error and the overall accuracy. The case was even worse with FoolsGold because the added noise made the attackers' last layers more diverse than the honest peers'. FooldGold assumes peers with similar last layers to be attackers and those with diverse last layers to be honest. Thus, it considered poisoned updates and excluded good updates in the model aggregation. The rest of the methods, including FFL, achieved comparable results to the baseline. Since the added noise made the magnitudes of the poisoned updates different from those of good updates, we observe that i) the median and the trimmed mean were able to neutralize the poisoned parameters in model aggregation, ii) multi-Krum was able to exclude the poisoned updates due to the larger Euclidean distances they had, and iii) FFL was able to exclude poisoned mixed updates because their deviations from their medians were larger than those of good mixed updates. We can also see that, in most cases (Adult-MLP, MNIST-CNN and CIFAR10-VGG16), FFL achieved the lowest test error among all methods. As the training evolved, FFL set the trust values of honest peers to 1 and thus fully considered their contributions. Note that FFL achieved similar performance under attack strategies 1 and 2.

TABLE 7.6: Robustness against label-flipping attacks.
Best scores are in bold.

Benchmark/ Method		Adult-MLP			MNIST-CNN			CIFAR10-VGG16			IMDB-BiLSTM		
		TE	Src-Acc%	ASR%	TE	Src-Acc%	ASR%	TE	Src-Acc%	ASR%	TE	Src-Acc%	ASR%
FL	FedAvg (no attacks)	0.350	39.06	60.94	0.112	95.43	0.49	0.881	70.73	14.51	0.519	87.16	12.84
	FedAvg	0.374	10.98	89.02	0.130	93.39	1.26	0.913	53.82	29.34	0.562	63.61	36.39
	Median	0.354	31.28	68.72	0.117	93.48	1.07	0.943	58.01	24.81	0.643	59.92	40.08
	Trimmed mean	0.353	32.12	67.88	0.118	93.58	1.07	0.943	58.43	25.54	0.649	58.20	41.80
	Multi-Krum	**0.350**	39.34	60.66	0.113	**95.33**	**0.39**	0.896	42.60	39.33	0.886	39.05	60.95
	FoolsGold	0.351	**39.72**	**60.28**	**0.111**	95.14	**0.39**	0.989	67.41	16.62	0.549	**86.86**	**13.14**
FFL	Strategy 1	**0.350**	39.66	60.34	**0.111**	**95.33**	**0.39**	**0.849**	**68.90**	**13.20**	0.544	86.64	13.36
	Strategy 2	**0.350**	39.50	60.50	0.112	95.04	0.49	0.892	68.80	15.00	**0.524**	86.16	13.84

Label-flipping attack. In the label-flipping attack, attackers flip the labels of correct training examples from one class (a.k.a. the source class) to another class and train their models according to the latter (Biggio, Nelson, and Laskov, 2012; Fung, Yoon, and Beschastnikh, 2018). For Adult, the attackers flipped each example with the label "$>$ 50K" to "$\leq$ 50K", while they flipped each example with the label "7" to "1" for MNIST. For CIFAR10, the attackers flipped each example with the label "Cat" to "Dog". For IMBD, they flipped the "positive" reviews to "negative".

Table 7.6 shows the results under the label-flipping attack. For the Adult-MLP benchmark, the FedAvg performance under the attack significantly degraded for Src-Acc and ASR. However, TE slightly increased and kept close to that of the baseline (FedAVg - no attacks). The reason for that is that the Adult data set is highly imbalanced, with a skew toward the "$\leq$ 50K" class label. The median and the trimmed mean achieved low TE, but saw degraded performance for Src-Acc and ASR because they discarded a large number of coordinates in the global model aggregation. Multi-Krum, FoolsGold and FFL achieved similar results to the baseline with slightly greater Src-ACC and slightly lower ASR. Since the Adult data set's training data were randomly and uniformly distributed among the peers, some of them had larger percentages of the target class examples compared to the original class distribution of the data set. The original percentage of the examples belonging to the target class "$\leq$ 50K" in the data set is about 75.22%, which caused bias in the global model against the minority class "$>$ 50K". This made the

updates of those honest peers having higher percentages of the target class close to the attackers' updates. Since multi-Krum, FoolsGold and FFL excluded or penalized some honest peers with updates close to the attackers' updates, the global model became less biased against the minority class "$>$ 50K" and, hence, the source class accuracy slightly increased.

For MNIST-CNN, the performance of FedAvg, median and trimmed mean slightly degraded compared to the baseline.

An interesting note is that FedAvg was not highly affected by the attack. The reasons for that are the small size of the model and the simple and balanced distribution of the data set. That was also observed in Shejwalkar et al., 2021, where the authors argued that, in some cases, FedAvg could be robust against poisoning attacks.

For CIFAR-VGG16, Src-Acc degraded from 70.73% to 53.82%, and ASR increased from 14.51% to 29.34% with FedAvg. On the other hand, FedAvg achieved TE lower than that of the median, the trimmed median and FoolsGold. That is because the attackers flipped the labels in the examples for only one class and kept the labels of the other classes unchanged. The median and the trimmed mean decreased Src-Acc and increased ASR because of the large size of the VGG16 model. Chang et al., 2019 have shown that the estimation errors of the median and the trimmed mean scale up with the size of the model in a square-root manner. Multi-Krum achieved the worst performance regarding Src-Acc and ASR because the small impact of the attack was not detectable in the large model. Therefore, multi-Krum identified some attackers as honest while identifying some honest peers as attackers, which led to its poor performance. FFL achieved the best performance among all the methods for all three metrics. FoolsGold scored the second-best after FFL regarding Src-Acc and ASR. FFL and FoolsGold achieved such good performance because they analyzed the last-layer gradients, which contain more useful information for detecting the behavior of targeted poisoning attacks. However, FoolsGold achieved the greatest TE among

the other methods because it did not consider the full contributions of the honest peers. FFL, however, considered almost all the full contributions of the honest peers and thus achieved the lowest TE.

For the IMDB-BiLSTM benchmark, FFL and FoolsGold achieved the best performance among all methods, most of which were negatively impacted by the large size of the BiLSTM model. FFL and FoolsGold outperformed the other methods by a large margin in achieving good values for all metrics. FoolsGold performed well in this benchmark because it was its ideal setting: updates from honest peers were somewhat different due to the different reviews they gave, whereas updates for attackers became very close to each other because they shared the same objective.

To summarize, the results show that FFL can effectively defend against untargeted and targeted poisoning attacks while preserving the benign model performance. Moreover, FFL outperforms the state-of-the-art defenses in achieving good model performance while preventing the attackers from mounting successful security attacks and hindering the semi-honest server from mounting privacy attacks.

Protection against the reconstruction privacy attack

We next report results on the protection offered by FFL against the most powerful privacy attack, namely the reconstruction attack proposed in Geiping et al., 2020. Notice that this attack does not require auxiliary data and can estimate the private training data by inverting their corresponding gradients.

Fig. 7.3 shows the results when two peers, k and j, sent their updates computed on just their private images (left column of the figure) to the server. In the FL setting (middle column of the figure), the server was able to reconstruct their private images with high accuracy. However, when they mixed fragments of their updates before sending them (FFL

FIGURE 7.3: Reconstruction of two input images from the gradients of two peers. Left: Two input images. Middle: Reconstruction from original gradients (FL). Right: Reconstruction from mixed gradients (FFL).

setting, right column of the figure), the server was only able to obtain noise instead of the original images.

Similarly, Fig. 7.4 shows the results when two peers k and j sent their updates computed on a batch of 8 images to the server. The figure exemplifies on two different batches of input images; the first example is given in the three upper rows and the second one in the three lower rows. In the FL setting, the server was able to recover a lot of information about the peers' training data. However, when they mixed their updates before sending them to the server (FFL setting), the latter just got a totally random batch of images.

Runtime of FFL

Table 7.7 reports the CPU runtimes in seconds per training round for mixed update decryption, global reputation calculation and model aggregation on the server side, and local model computation, fragment exchanging and mixing on the peers' side. Note that the reported runtimes for exchanging and mixing fragments are the average for a single peer. We computed the runtime for the three benchmarks with 10, 50 and 100 updates per training round to illustrate how the runtime scales with the number of updates. On the server side, we report the total runtime for all methods; in contrast, on the peer side, we report the overhead (extra runtime) with respect to standard FL.

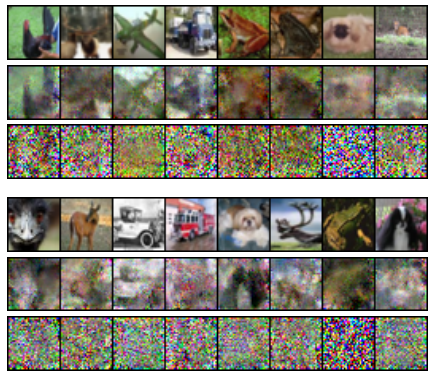


FIGURE 7.4: The three upper rows show the reconstruction by the server of a batch of 8 input images based on the gradients of a peer. The first row shows the input private images, the second row shows the reconstructions from the FL original gradients of the peer, and the third row the reconstructions from the FFL mixed gradients of the peer. The three lower rows report another analogous example with a different batch of input images.

It can be seen that, as the number of updates and the model size increase, the FFL server's runtime grows less than for the other methods, which confirms our computation cost analysis in Section 7.6.1. In particular, with CIFAR10-VGG16, FFL achieved the lowest runtime among all non-baseline methods. Furthermore, unlike FFL, the other methods are unable to thwart privacy attacks or provide adequate protection against security attacks.

Regarding the runtime overhead incurred by each peer, the maximum runtime overhead resulting from exchanging and mixing fragments was about 1.319 seconds for the largest model we used, which was VGG16. Also, it is worth noting that the increase in the number of updates had little impact on the overhead of the peers because their computations essentially depend on the model dimensionality.

Nevertheless, the FFL runtime is very small if we compare it with

TABLE 7.7: CPU runtime in seconds of FFL in comparison with standard FL for one training round. Columns Dec. and Aggr. contain the decryption runtime of encrypted mixed updates, and the global reputation calculation and aggregation runtime on the server side (S). Column Mix. reports the overhead (extra runtime) on the peer side (P) with respect to standard FL; this overhead results from local reputation calculation and the EXCHANGE_FRAGMENTS protocol of FFL.

Benchmark		Adult-MLP			MNIST-CNN			CIFAR10-VGG16			IMDB-BiLSTM		
# of updates	Method	S		P	S		P	S		P	S		P
		Dec.	Aggr.	Mix.	Dec.	Aggr.	Mix.	Dec.	Aggr.	Mix.	Dec.	Aggr.	Mix.
10	FedAvg (FL)	0	0.001	0	0	0.001	0	0	0.064	0	0	0.071	0
	Median (FL)		0.002			0.002			0.858			0.816	
	TMean (FL)		0.003			0.007			1.201			0.843	
	MKrum (FL)		0.004			0.013			0.822			0.690	
	FGold (FL)		0.01			0.017			0.497			0.256	
	FFL	0.064	0.021	0.231	0.113	0.016	0.578	0.372	2.308	1.106	0.321	1.773	1.081
50	FedAvg (FL)	0	0.002	0	0	0.002	0	0	0.194	0	0	0.275	0
	Median (FL)		0.007			0.008			6.843			4.135	
	TMean (FL)		0.015			0.045			7.097			4.267	
	MKrum (FL)		0.014			0.015			7.044			5.657	
	FGold (FL)		0.042			0.035			1.609			1.129	
	FFL	0.263	0.04	0.239	0.509	0.033	0.583	1.875	4.726	1.188	1.612	3.536	1.174
100	FedAvg (FL)	0	0.004	0	0	0.004	0	0	0.341	0	0	0.517	0
	Median (FL)		0.015			0.013			16.582			8.926	
	TMean (FL)		0.035			0.111			16.645			8.929	
	MKrum (FL)		0.028			0.031			27.875			16.893	
	FGold (FL)		0.086			0.063			6.180			2.273	
	FFL	0.359	0.043	0.249	1.01	0.035	0.594	3.675	4.883	1.319	3.236	3.583	1.282

that of Zhang et al., 2020, which provides a level of privacy similar to ours but without being able to neutralize poisoned updates. Specifically, for an LSTM model containing only 4.02 million parameters, Zhang et al., 2020 report a runtime overhead for each peer of about 176 seconds and a total runtime for the server of 174 seconds to aggregate 50 local updates. Note also that the hardware specifications employed by Zhang et al., 2020 are superior to ours. Therefore, the peers in FFL can turn a blind eye to the computational overhead they incur in exchange for protecting their privacy, countering security attacks and learning a more accurate global model.

7.8 Conclusions

In this chapter, we have presented *fragmented federated learning* (FFL), a novel approach based on cooperation among peers in FL systems to

preserve their privacy without renouncing robust and accurate aggregation of their updates to the global model. FFL offers a practical solution where peers privately and efficiently exchange random fragments of their updates before sending them to the server. We have also proposed a novel reputation-based defense tailored for FFL that builds trust in the peers based on the quality of the mixed updates they send and the fragments they exchange. We have demonstrated that the proposed framework can effectively counter privacy and security attacks. All the above is achieved while obtaining a global model's accuracy similar to that of standard FL (when no attack is performed) and imposing affordable communication cost and computation overhead on the participating parties. The efficiency of FFL makes it applicable to large-scale FL systems.

Chapter 8

Defending against backdoor attacks by layer-wise feature analysis

In Chapter 1, we have discussed how backdoor attacks can compromise both federated and centralized learning. We have also shown in Chapter 6 that countermeasures applied during the training phase of FL may not work well with large DL models and highly non-IID data among peers. In this chapter, we explore how to defend against BAs at the test time when the trained model is used for inference.

Among all backdoor defenses in the CL literature, backdoor detection is one of the most important defense paradigms, where defenders attempt to detect whether a suspicious object (*e.g.*, model or sample) is malicious. Currently, most existing backdoor detectors assume poisoned samples have different feature representations from benign samples, and they tend to focus on the layer before the fully connected layers (Chen et al., 2019a; Tang et al., 2021; Hayase and Kong, 2021). Two intriguing questions arise: **(1)** *Is this layer always the most critical place for backdoor detection?* **(2)** *If not, how to find the critical layer for designing more effective backdoor detection?*

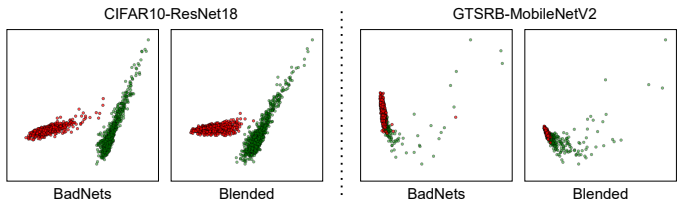


FIGURE 8.1: PCA-based visualization of features of benign (green) and poisoned samples (red) generated by the layer before the fully connected layers of models attacked by BadNets Gu et al., 2019 and Blended Chen et al., 2017. As shown in this figure, features of poisoned and benign samples are not well separable on the GTSRB benchmark.

In this chapter, we give a negative answer to the first question (see Figure 8.1). To answer the second one, we conduct a layer-wise feature analysis of poisoned and benign samples from the target class. We find out that the feature difference between benign and poisoned samples tends to reach the maximum at a critical layer, which can be easily located based on the behaviors of benign samples. Specifically, *the critical layer is the one or near the one that contributes most to assigning benign samples to their true class.* Based on this finding, we propose a simple yet effective method to filter poisoned samples by analyzing the feature differences (measured by cosine similarity) between incoming suspicious samples and a few benign samples at the critical layer. Our method can serve as a 'firewall' for deployed DNNs to identify, block, and trace malicious inputs. In short, the main contributions of the chapter are four-fold:

- We demonstrate that the features of poisoned and benign samples are not always clearly separable at the layer before fully connected layers, which is the one typically used in existing defenses.

- We conduct a layer-wise feature analysis aimed at locating the critical layer where the separation between poisoned and benign samples is neatest.
- We propose a backdoor detection method to filter poisoned samples by analyzing the feature differences between suspicious and benign samples at the critical layer.
- We conduct extensive experiments on two benchmark data sets to assess the effectiveness of our proposed defense.

The contributions in this chapter have been accepted for publication to The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2023).

This chapter is organized as follows. Section 8.1 recalls related work, including state-of-the-art BAs and defenses. Section 8.2 performs the layer-wise feature analysis. Section 8.3 describes the threat model being considered, and presents our defense method against BAs at the inference time. Section 8.4 describes the experimental setup and presents comprehensive results on two benchmark data sets. Section 8.5 outlines how the proposed defense can be adopted in federated learning. Section 8.6 is a conclusion.

8.1 Related work

In this chapter, we focus on backdoor attacks and defenses in image classification. Other deep learning tasks are out of our current scope.

Backdoor attacks. BadNets (Gu et al., 2019) was the first backdoor attack, which randomly selected a few benign samples and generated their poisoned versions by stamping a trigger patch onto their images and reassigning their label as the target label. Later Chen et al., 2017 noted that the poisoned image should be similar to its benign version

for stealthiness; these authors proposed a blended attack by introducing trigger transparency. However, these attacks are with poisoned labels and therefore users can still detect them by examining the image-label relation. To circumvent this, Turner, Tsipras, and Madry, 2019 proposed the clean-label attack paradigm, where the target label is consistent with the ground-truth label of poisoned samples. Specifically, in this paradigm, adversarial attacks were exploited to perturb the selected benign samples before conducting the standard trigger injection process. Nguyen and Tran, 2020b adopted image warping as the backdoor trigger, which modifies the whole image while preserving its main content. Besides, Nguyen and Tran, 2020a proposed the first sample-specific attack, where the trigger varies across samples. However, such triggers are visible and the adversaries need to control the whole training process. More recently, Li et al., 2021d introduced the first poison-only invisible sample-specific attack to address these problems.

Backdoor defenses. Existing backdoor defenses fall into three main categories: input filtering, input pre-processing, and model repairing.

- Input filtering: which intends to differentiate benign and poisoned samples based on their distinctive behaviors, like the separability of the feature representations of benign and poisoned samples. For example, Hayase and Kong, 2021 introduced a robust covariance estimation of feature representations to amplify the spectral signature of poisoned samples. Zeng et al., 2021 proposed to filter inputs inspired by the understanding that poisoned images tend to have some high-frequency artifacts. Gao et al., 2022 proposed to blend various images on the suspicious one, since the trigger pattern can still mislead the prediction no matter what the background contents are.

- Input pre-processing: which modifies each input sample before feeding it into the deployed DNN. Its rationale is to perturb potential trigger patterns and thereby prevent backdoor activation. Liu,

Xie, and Srivastava, 2017 proposed the first defense in this category where they used an encoder-decoder to modify input samples. Rosenfeld et al., 2020 employed randomized smoothing to generate a set of input neighbors and averaged their predictions. Further, Li et al., 2021c demonstrated that if the location or appearance of the trigger is slightly different from that used for training, the attack effectiveness may degrade sharply. Based on this, they proposed to pre-process images with spatial transformations.

- Model repairing: which aims at erasing backdoors contained in the attacked DNNs. For example, Liu, Xie, and Srivastava, 2017; Zhao et al., 2020; Li et al., 2021b showed that users can effectively remove backdoors by fine-tuning the attacked DNNs with a few benign samples. These methods were inspired by catastrophic forgetting (Kirkpatrick et al., 2017). Liu, Dolan-Gavitt, and Garg, 2018 revealed that model pruning can also remove backdoors effectively, because backdoors are mainly encoded in specific neurons. Very recently, Zeng et al., 2022 proposed to repair compromised models with adversarial model unlearning.

In this chapter, *we focus on input filtering*, which is very convenient to protect deployed DNNs.

8.2 Layer-wise feature analysis

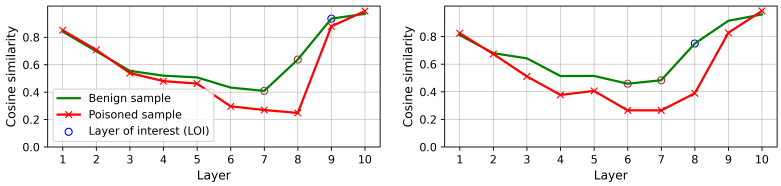
We notice that the predictions of attacked DNNs for both benign samples from the target class and poisoned samples are all the target label. The attacked DNNs mainly exploit class-relevant features to predict these benign samples while they use trigger-related features for poisoned samples. We suggest that defenders could exploit this difference to design effective backdoor detection. To explore their main differences, we conduct a layer-wise analysis, as follows.

Definition 1 (Layer-wise centroids of target class features)**.** *Let f' be an attacked DNN with a target class t. Let $X_t = \{x_i\}_{i=1}^{|X_t|}$ be benign samples with true class t, and let $\{a_i^1, \ldots, a_i^L\}_{i=1}^{|X_t|}$ be their intermediate features generated by f'. The centroid of t's benign features at layer l is defined as $\hat{a}_t^l = \frac{1}{|X_t|}\sum_{i=1}^{|X_t|} a_i^l$, and $\{\hat{a}_t^1, \ldots, \hat{a}_t^L\}$ is the set of layer-wise centroids of t's benign features.*

Definition 2 (Layer-wise cosine similarity)**.** *Let a_j^l be the features generated by layer l for an input x_j, and let cs_j^l be the cosine similarity between a_j^l and the corresponding t's centroid $\hat{a}_t^l$. The set $\{cs_j^1, \ldots, cs_j^L\}$ is said to be the layer-wise cosine similarities between x_j and t's centroids.*

Settings. We conducted six representative attacks on four classical benchmarks: CIFAR10-ResNet18, CIFAR10-MobileNetV2, GTSRB-ResNet18, and GTSRB-MobileNetV2. The six attacks were BadNets (Gu et al., 2019), the backdoor attack with blended strategy (Blended) (Chen et al., 2017), the label-consistent attack (LC) of Turner, Tsipras, and Madry, 2019, WaNet (Nguyen and Tran, 2020b), ISSBA (Li et al., 2021d), and IAD (Nguyen and Tran, 2020a). More details on the data sets, DNNs, and attack settings are presented in Section 8.4. Specifically, for each attacked DNN f' with a target class t, we estimated $\{\hat{a}_t^1, \ldots, \hat{a}_t^L\}$ using 10% of the benign test samples labeled as t. Then, for the benign and poisoned test samples classified by f' into t, we calculated the layer-wise cosine similarities between their generated features and the corresponding estimated centroids. Finally, we visualized the layer-wise means of the computed cosine similarities of the benign and poisoned samples to analyze their behaviors.

Results. Figure 8.2 shows the layer-wise means of cosine similarity for benign and poisoned samples with the CIFAR10-ResNet18 benchmark under the BadNets and ISSBA attacks. As we go deeper into the attacked DNN layers, the gap between the direction of benign and poisoned features gets larger until we reach a specific layer where the backdoor trigger is activated, causing poisoned samples to get closer to the



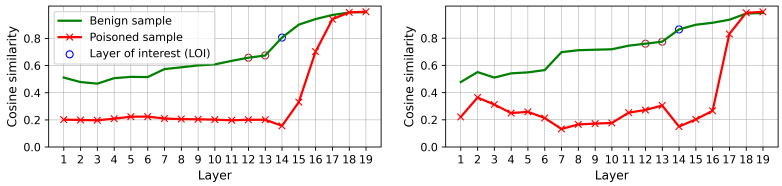
(A) BadNets (B) ISSBA

FIGURE 8.2: Layer-wise behaviors of benign samples from the target class and poisoned samples (generated by BadNets and ISSBA) on CIFAR-10 with ResNet-18

target class. Figure 8.3 shows the same phenomenon for the GTSRB-MobileNetV2 benchmark. Further, we can see that for BadNets the latent features of benign and poisoned samples are similar in the last layer of the features extractor (*i.e.*, layer 17).

Regardless of the attack or benchmark, when we enter the second half of DNN layers (which usually are class-specific), *benign samples start to get closer to the target class before the poisoned ones, that are still farther from the target class* because the backdoor trigger is not yet activated. This makes the difference in similarity maximum in one of those latter layers, which we call the *critical layer*. In particular, *this layer is not always the one typically used in existing defenses* (*i.e.*, the layer before fully-connected layers). Besides, we show that it is very likely to be either the layer that contributes most to assigning the benign samples to their true target class (which we name the *layer of interest or LOI*, circled in blue) or one of the two layers before the LOI (circled in brown).

Results under other attacks for these benchmarks are presented in Section 8.4.2. The results there also provide confirmation that the above distinctive behaviors hold regardless of the data sets or models being used. From the analysis above, we can conclude that focusing on those circled layers can help develop a simple and robust defense against backdoor attacks.



(A) BadNets (B) ISSBA

FIGURE 8.3: Layer-wise behaviors of benign samples from the target class and poisoned samples (generated by BadNets and ISSBA) on GTSRB with MobileNetV2

8.3 The proposed defense

Threat model. Consider a user that obtains a suspicious trained f_s that might contain hidden backdoors. We assume that the user has limited computational resources or benign samples, and therefore cannot repair f_s. The user wants to defend by detecting at inference time whether a suspicious incoming input x_s is poisoned, given f_s. Similar to existing defenses, we assume that a small set of benign samples X_{val} is available to the user/defender. We denote the available samples that belong to a potential class t as $X_{t_{val}}$. Let $m = |X_{t_{val}}|$ denote the number of available samples labeled as t.

Method design. Based on the lessons learned in Section 8.2, our method to detect poisoned samples at inference time consists of four steps. **1)** Estimate the layer-wise features' centroids of class t for each of layers $\lfloor L/2 \rfloor$ to L using the class's available benign samples. **2)** Compute the cosine similarities between the extracted features and the estimated centroids, and then compute the layer-wise means of the computed cosine similarities. **3)** Identify the layer of interest (LOI), sum up the cosine similarities in LOI and the two layers before LOI (sample-wise), and compute the mean and standard deviation of the summed cosine similarities. **4)** For any suspicious incoming input x_s classified as t by f_s, **4.1)**

compute its cosine similarities to the estimated centroids in the above-mentioned three layers, and **4.2)** consider it as a potentially poisoned input if its summed similarities fall below the obtained mean by a specific number τ of standard deviations (called threshold in what follows).

Algorithm 2 summarizes our defense. For each potential target class $t \in \{1, \dots \mathcal{C}\}$, we first feed the available m benign samples to f_s and extract their intermediate features in the second half of layers to obtain the set $\{(a_i^{L/2}, \dots, a_i^L)\}_{i=1}^m$ (if L is odd, take the integer part of $L/2$ instead of $L/2$ here and in what follows). Note that we can reduce computation by focusing on the second half of layers because the LOI and the two layers before the LOI are among the latter layers of the DNN. After that, we compute the layer-wise centroids of the extracted features for each layer $l \in \{L/2, \dots, L\}$ (Line 5). Then, we compute the cosine similarity between the benign features of each layer and their corresponding centroid (Line 6).

Then, we aggregate the computed similarities to approximate the similarity centroid in each layer $l \in \{L/2, (L/2)+1, \dots, L-1, L\}$ (Line 7).

Next, we use $\{\hat{cs}_t^{L/2}, \hat{cs}_t^{(L/2)+1}, \dots, \hat{cs}_t^L\}$, to locate the layer of interest LOI_t that contributes most to assigning t's benign samples to their true class t. We use Procedure 1 to locate LOI_t. We compute the difference between the approximated similarity of each layer and its preceding one, and we identify the layer with the maximum difference as LOI_t. For example, if the maximum difference is $|\hat{cs}_t^l - \hat{cs}_t^{l-1}|$, then layer l is the layer of interest.

Once we locate LOI_t, we estimate the behavior of benign samples in that layer and in the two layers previous to it. For each sample $x_i \in X_{t_{val}}$, we sum up its computed cosine similarities in the three layers (Line 9). After computing the summed similarities of the m samples and obtaining the set $\{cs_i\}_{i=1}^m$, we compute the mean μ_t and the standard deviation σ_t of the set.

To detect potentially poisoned samples, for any suspicious incoming

Algorithm 2: Detecting backdoor attacks via layer-wise feature analysis

Input: Suspicious trained DNN f_s; Validation samples X_{val}; Threshold τ; Suspicious input x_s
Output: Boolean value (True/False) tells if x_s is poisoned

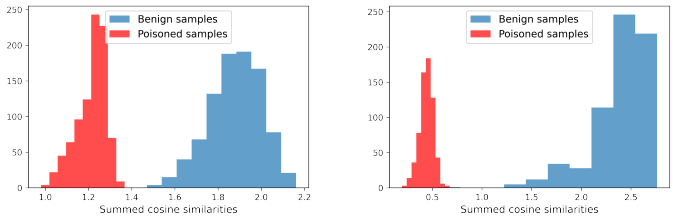
for *each potential target class* $t \in \{1, \ldots, \mathcal{C}\}$ **do**
  `// An offline loop`
  $X_{t_{val}} \leftarrow$Split t's benign samples from X_{val}
  $m \leftarrow |X_{t_{val}}|$
  $\{a_i^{\lfloor L/2 \rfloor}, \ldots, a_i^L\}_{i=1}^m \leftarrow$Layers' features generated by f_s for $\{x_i \in X_{t_{val}}\}$
  $\hat{a}_t^l \leftarrow \frac{1}{m}\sum_{i=1}^m a_i^l$ `// Estimate` t`'s centroid at layer` $l \in \{\lfloor L/2 \rfloor, \ldots, L\}$
  $cs_i^l \leftarrow$ `CosineSimilarity`$(a_i^l, \hat{a}_t^l)$ `// Similarity of` a_i^l `to its centroid`
  $\hat{cs}_t^l \leftarrow \frac{1}{m}\sum_{i=1}^m cs_i^l$ `// Aggregate computed benign similarities at layer` l
  $LOI_t \leftarrow$ `IdentifyLayerOfInterest`$(\{\hat{cs}_t^{\lfloor L/2 \rfloor}, \ldots, \hat{cs}_t^L\})$
  $cs_i \leftarrow cs_i^{LOI_t-2} + cs_i^{LOI_t-1} + cs_i^{LOI_t}$
  $\mu_t, \sigma_t \leftarrow$ `MEAN`$(\{cs_i\}_{i=1}^m)$, `STD`$(\{cs_i\}_{i=1}^m)$
end
IsPoisoned $\leftarrow$ *False*
$\hat{y}_s \leftarrow f_s(x_s)$ `//` $\hat{y}_s$ `is the predicted class by` f_s `for` x_s
for *each potential target class* $t \in \{1, \ldots, \mathcal{C}\}$ **do**
  if $\hat{y}_s = t$ **then**
    $\{cs_s^{LOI_t-2}, cs_j^{LOI_t-1}, cs_j^{LOI_t}\} \leftarrow$ `CosineSimilarity`$(\{a_s^l, \hat{a}_t^l\}_{l=LOI_t-2}^{LOI_t})$
    $cs_s \leftarrow cs_s^{LOI_t-2} + cs_s^{LOI_t-1} + cs_s^{LOI_t}$
    if $cs_s < (\mu_t - \tau \times \sigma_t)$ **then**
      IsPoisoned $\leftarrow$ *True*
end
return *IsPoisoned*

input x_s classified as t by f_s at inference time, we extract its features in LOI_t and the two preceding layers, compute their cosine similarities to

Procedure 1: Identifying layer of interest (LOI)

```
IdentifyLayerOfInterest({ĉs^⌊L/2⌋, ..., ĉs^L}):
    max_diff ← ĉs^(⌊L/2⌋+1) − ĉs^⌊L/2⌋
    LOI ← ⌊L/2⌋ + 1
    for l ∈ {⌊L/2⌋ + 2, ..., L} do
        l_diff ← ĉs^l − ĉs^(l−1)
        if l_diff > max_diff then
            max_diff ← l_diff
            LOI ← l
    return LOI
```

the corresponding estimated centroids $\{cs_s^{LOI_t-2}, cs_s^{LOI_t-1}, cs_s^{LOI_t}\}$, and sum them up to get cs_s. Then, we identify x_s as a potentially poisoned sample if $cs_s < \mu_t - \tau \times \sigma_t$, where τ is an input threshold chosen by the defender that provides a reasonable trade-off between the true positive rate TPR and the false positive rate FPR. Figure 8.6 shows an example of the distributions of the summed cosine similarities of benign and poisoned features to the estimated benign centroids (in the three identified layers) under the label-consistent attack of Turner, Tsipras, and Madry, 2019.



(A) CIFAR10-ResNet18 (B) GTSRB-MobileNetV2

FIGURE 8.6: Distributions of the summed cosine similarities of benign and poisoned samples under the label-consistent attack on CIFAR10 with ResNet18 and GTSRB with MobileNetV2 benchmarks

TABLE 8.1: Statistics of the used data sets and DNNs

Data set	# Available samples	DNN model	# Layers
CIFAR-10	1,000	ResNet18	10
GTSRB	1,263	MobileNetV2	19

8.4 Experiments

8.4.1 Experimental setup

We used the BackdoorBox (Li et al., 2022) open toolbox for conducting all attacks and re-implemented the other defenses used in our work. The source code, pre-trained models, and poisoned test sets of our defense are available at https://github.com/NajeebJebreel/DBALFA.

Data sets and models. We used two classic benchmark data sets, namely CIFAR10 (Krizhevsky, 2009) and GTSRB (Stallkamp et al., 2011). We used the ResNet18 (He et al., 2016) on CIFAR10 and the MobileNetV2 (Sandler et al., 2018) on GTSRB. The data sets and DNN models employed, along with the number of benign samples at the defender's disposal, are outlined in Table 8.1. Note that, for ease of computation, we consider as a layer each convolutional block other than the first convolutional layer and the last fully connected layer.

Training setting. We used the cross-entropy loss and the SGD optimizer with a momentum 0.9 and weight decay 5×10^{-4} on all benchmarks. We used initial learning rates 0.1 for ResNet18 and 0.01 for MobileNetV2, and trained models for 200 epochs. The learning rates were decreased by a factor of 10 at epochs 100 and 150, respectively. We set the batch size to 128 and trained all models until they converged.

Attack baselines. We evaluated each defense under the six attacks mentioned in Section 8.2: BadNets, Blended, LC, WaNet, ISSBA, and IAD. They are representative of visible attacks, patch-based invisible attacks, clean-label attacks, non-patch-based invisible attacks, invisible sample-specific attacks, and visible sample-specific attacks, respectively.

Attack setting. For both CIFAR10 and GTSRB, we took the following settings. The target class on all data sets was 1 for BadNets (Gu et al., 2019), the backdoor attack with blended strategy (Chen et al., 2017) (Blended), the invisible sample-specific attack (Li et al., 2021d) (ISSBA), and the input-aware dynamic attack (Nguyen and Tran, 2020a) (IAD). The target classes for the label-consistent attack (Turner, Tsipras, and Madry, 2019) and WaNet (Nguyen and Tran, 2020b) were 2 and 0, respectively, on all data sets. We used a 2×2 square as the trigger pattern for BadNets (as suggested in Gu et al., 2019). We adopted the random noise pattern, with a 10% blend ratio, for Blended (as suggested in Chen et al., 2017). The trigger pattern adopted for the LC attack was the same used in BadNets with maximum perturbation size 16. For WaNet, we set the noise rate to $\rho_n = 0.2$, the control grid size to $k = 4$, and the warping strength to $s = 0.5$ on all data sets, as suggested in the WaNet paper (Nguyen and Tran, 2020b).

For WaNet, ISSBA, and IAD, we took their default settings. Besides, we set the poisoning rate to 5% for BadNets, Blended, LC, and ISSBA. For WaNet and IAD, we set the poisoning rate to 10%. We implement baseline attacks based on the codes in `BackdoorBox` (Li et al., 2023). For IAD (Nguyen and Tran, 2020a), we trained the classifier and the trigger generator concurrently. We attached the dynamic trigger to the samples from other classes and relabeled them as the target label. Figure 8.7 shows an example of poisoned samples generated by different attacks.

Defense baselines. We compared our defense with six representative defenses, namely randomized smoothing (RS) (Rosenfeld et al., 2020), ShrinkPad (ShPd) (Li et al., 2021c), activation clustering (AC) (Chen et al., 2019a), STRIP (Gao et al., 2022), SCAn (Tang et al., 2021), and fine-pruning (FP) (Liu, Dolan-Gavitt, and Garg, 2018). RS and ShPd are two defenses with input pre-processing; AC, STRIP, and SCAn are three advanced input-filtering-based defenses; FP is based on model repairing.

Defense setting. For RS, we generated 100 neighbors of each input

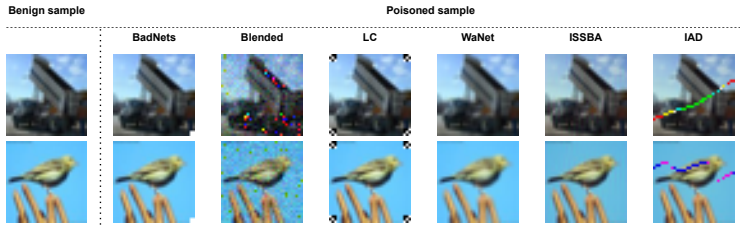


FIGURE 8.7: Example benign samples and their poisoned versions generated by six representative backdoor attacks

with a mean = 0 and a standard deviation = 0.1, as suggested in Cohen, Rosenfeld, and Kolter, 2019. We set the shrinking rate to 10% for ShPd and padded shrinked images with 0-pixels to expand them to their original size, as suggested in Li et al., 2021c. For FP, we pruned 95% of the dormant neurons in the last convolution layer and fine-tuned the pruned model using 5% of the training set. We adjusted RS, ShPd, and FP to be used as detectors for poisoned samples by comparing the change in prediction before and after applying them to an incoming input. For AC, STRIP, SCAn, and our defense, we randomly selected 10% from each benign test set as the available benign samples. Then, for AC, we used the available benign samples, from each class, for normalizing benign and poisoned test samples and identifying potential poisoned clusters. For STRIP, we blended each input with 100 random inputs from the available benign samples using a blending value $\alpha = 0.5$, as suggested in Gao et al., 2022. Then, we identified inputs with entropy below the 10-th percentile of the entropies of benign samples as potentially poisoned samples. For SCAn, we identified classes with scores larger than e as potential target classes, as suggested in Tang et al., 2021, and identified the cluster that did not contain the available benign samples as a poisoned cluster. For our defense, we used a threshold $\tau = 2.5$, which gave us a reasonable trade-off between TPR and FPR on both benchmarks.

TABLE 8.2: MA% and ASR% under the selected backdoor attacks on the CIFAR10-ResNet18 and the GTSRB-MobileNetV2 benchmarks. Best scores are in bold.

Benchmark↓	Metric↓,Attack→	BadNets	Blended	LC	WaNet	ISSBA	IAD
CIFAR10-ResNet18	MA%	91.45	92.19	91.98	91.13	**94.74**	94.42
	ASR%	97.20	**100.0**	99.96	99.04	**100.0**	99.66
GTSRB-MobileNetV2	MA%	97.00	97.27	97.45	96.09	98.43	**98.81**
	ASR%	95.49	**100.0**	**100.0**	91.82	**100.0**	99.63

Evaluation metrics. We used the main accuracy (MA) and the attack success rate (ASR) to measure attack performance. Specifically, MA is the number of correctly classified benign samples divided by the total number of benign samples, and ASR is the number of poisoned samples classified as the target class divided by the total number of poisoned samples. We adopted TPR and FPR to evaluate the performance of all defenses, where TPR is computed as the number of detected poisoned inputs divided by the total number of poisoned inputs, whereas FPR is the number of benign inputs falsely detected as poisoned divided by the total number of benign inputs.

8.4.2 Results

Performance of attacks. Table 8.2 shows the performance of the selected attacks on the CIFAR10-ResNet18 and the GTSRB-MobileNetV2 benchmarks. It can be seen that sample-specific attacks (*e.g.*, ISSBA and IAD) performed better than other attacks in terms of MA and ASR.

Main results. We ran each defense five times and computed the average results for a fair comparison. As shown in Tables 8.3 and 8.4, existing defenses failed to detect attacks with low TPR or high FPR in many cases, especially on the GTSRB data set. For example, AC failed in most cases on GTSRB, although it had promising performance on CIFAR-10. In contrast, our method had good performance in detecting all attacks on both data sets. There were only a few cases (4 over 28) where our approach

TABLE 8.3: Main results (%) on the CIFAR-10 data set. Boldfaced values are the best results among all defenses. Underlined values are the second-best results.

Attack→	BadNets		Blended		LC		WaNet		ISSBA		IAD		Avg	
Defense↓	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
RS	9.84	8.00	7.35	5.76	9.21	7.52	98.48	10.00	8.83	8.72	13.28	6.36	24.50	7.73
ShPd	94.28	13.31	49.72	12.89	69.87	13.18	36.25	17.69	95.22	5.50	42.74	7.56	64.68	11.69
FP	96.10	17.13	96.23	16.16	94.76	17.31	96.01	18.64	98.98	19.53	97.08	22.52	96.53	18.55
AC	**99.52**	31.14	**100.00**	30.69	**100.00**	31.16	**99.18**	32.44	**99.94**	34.22	82.99	31.32	96.94	31.83
STRIP	68.70	11.70	65.20	11.70	66.00	12.80	7.90	12.30	56.20	11.40	2.10	14.00	44.35	12.32
SCAn	96.60	**0.77**	**100.00**	**0.00**	0.02	5.05	98.55	**1.06**	99.89	2.61	84.19	**0.13**	79.88	1.60
Ours	99.38	1.35	**100.00**	1.59	**100.00**	**1.20**	91.04	1.48	98.97	**1.17**	**99.12**	1.26	**98.09**	**1.34**

TABLE 8.4: Main results (%) on the GTSRB data set. Boldfaced values are the best results among all defenses. Underlined values are the second-best results.

Attack→	BadNets		Blended		LC		WaNet		ISSBA		IAD		Avg	
Defense↓	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
RS	13.20	22.10	10.12	20.40	9.23	19.15	10.10	17.20	8.61	16.98	17.70	17.60	11.49	18.91
ShPd	94.97	12.16	11.58	10.68	96.16	10.60	66.11	14.81	95.92	8.26	31.07	16.10	65.97	12.10
FP	89.05	18.80	30.56	**3.70**	94.71	50.02	67.12	3.24	94.22	7.05	94.37	5.75	78.34	14.76
AC	0.30	8.84	0.00	5.67	4.83	**5.42**	0.42	25.87	99.06	17.48	43.85	10.73	24.74	12.34
STRIP	32.00	9.00	80.40	10.80	7.40	11.00	34.20	11.40	13.00	13.60	6.60	10.60	28.93	11.07
SCAn	46.05	**2.57**	46.02	4.03	30.45	11.39	54.07	**1.88**	96.85	**0.17**	0.09	19.41	45.59	6.58
Ours	**99.99**	6.23	**100.00**	6.72	**100.00**	5.95	**100.00**	6.49	**100.00**	5.43	**100.00**	**4.67**	**100.00**	**5.92**

was neither optimal nor close to optimal. In these cases, our detection was still on par with state-of-the-art methods, and another indicator (*i.e.*, TPR or FPR) was significantly better than them. For example, when defending against the blended attack on the GTSRB data set, the TPR of our method was 69.44% larger than that of FP, which had the smallest FPR in this case. These results confirm the effectiveness of our detection.

Resistance to adaptive attacks. The adversary may adapt his attack to bypass our defense by optimizing the model's original loss $\mathcal{L}_{org}$ and minimizing the layer-wise angular deviation between the features of the poisoned samples and the features' centroids of the target class's benign samples. We studied the impact of this strategy by introducing the *cosine distance* between the features of poisoned samples and the target class centroids as a secondary loss function $\mathcal{L}_{cd}$ in the training objective function. Also, we introduced a penalty parameter β, which yielded a

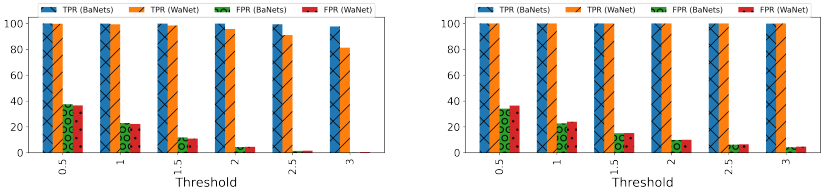
TABLE 8.5: Adaptive attack. Top, impact of penalty factor β on MA and ASR. Bottom, impact of penalty factor β on TPR and FPR.

β	0	0.5	0.6	0.7	0.8	0.9	0.91	0.92	0.95
MA (%)	91.45	92.96	92.06	92.65	92.63	90.33	79.97	69.13	10
ASR (%)	97.20	96.72	96.93	96.63	96.29	96.88	96.41	97.36	100

Defense↓	$\beta \rightarrow$ Metric (%)↓	0	0.5	0.6	0.7	0.8	0.9	0.91	0.92	0.95
AC	TPR	99.52	99.20	99.16	45.69	26.26	26.22	23.81	13.38	0.00
	FPR	31.14	29.46	28.85	8.21	7.72	6.21	0.25	7.80	0.00
SCAn	TPR	96.60	96.55	96.60	72.80	56.19	0.00	0.00	0.00	0.00
	FPR	0.77	1.38	4.60	1.14	0.10	0.00	0.00	0.00	0.00
Ours	TPR	99.38	99.41	98.18	97.43	97.52	94.20	24.20	0.00	0.00
	FPR	1.35	1.96	1.44	1.15	0.53	1.40	4.17	0.00	0.00

modified objective function $(1-\beta)\mathcal{L}_{org} + \beta\mathcal{L}_{cd}$. The role of β is to control the trade-off between the angular deviation and the main accuracy loss. We then launched BadNets on CIFAR10-ResNet18 under the modified objective function. Table 8.5 (top subtable) shows MA and ASR with different penalty factors. We can see that values of $\beta < 0.9$ slightly increased the main accuracy because the second loss acted as a regularizer to the model's parameters, which reduced over-fitting. Also, ASR stayed similar to the non-adaptive ASR (when $\beta = 0$). However, the main accuracy degraded with greater β values, because the original loss function was dominated by the angular deviation loss.

Table 8.5 (bottom subtable) shows the TPRs and FPRs of AC, SCAn, and our defense with different penalty factors. As β increased (up to $\beta = 0.9$), the TPR of our defense decreased from 99.38% to 94.20% while FPR was almost unaffected. This shows that the adversary gained a small advantage with $\beta = 0.9$. On the other hand, the other defenses achieved limited or poor robustness compared to ours with the same β values. With $\beta \geq 0.91$, AC, SCAn, and our method defense failed to counter the attack. However, looking at Table 8.5 (top subtable) we can see the main accuracy degraded with these high β values, which made it easy to reject the model due its low performance.



(A) CIFAR10-ResNet18 (B) GTSRB-MobileNetV2

FIGURE 8.8: Impact of detection thresholds on TPR (%) and FPR (%)

TABLE 8.6: Impact of poisoning rates

Poisoning Rate↓, Metric→	MA (%)	ASR (%)	TPR (%)	FPR (%)
1%	91.52	94.15	99.64	1.25
3%	92.28	96.31	99.32	1.32
5%	91.45	97.20	99.36	1.35
10%	91.45	97.56	99.83	1.62

Effects of the detection threshold. Figure 8.8 shows the TPRs and FPRs of our defense with threshold $\tau \in \{0.5, 1, 1.5, 2, 2.5, 3\}$ for BadNets and WaNet. It can be seen that a threshold 2.5 is reasonable, as it offers a high TPR while keeping a low FPR. Note that the larger the threshold, the smaller the TPR and FPR. Users should choose the threshold based on their specific needs.

Effects of the poisoning rate. We launched BadNets on CIFAR10-ResNet18 using different poisoning rates $\in \{1\%, 3\%, 5\%, 10\%\}$ to study the impact of poisoning rates on our defense. Table 8.6 shows the attack success rate (ASR) increases with the poisoning rate. However, the poisoning rate has minor effects on our TPR and FPR. These results confirm again the effectiveness of our method.

Effectiveness of our layer selection. We compared the performance of AC, SCAn, and our method at detecting BadNets on the GTSRB-MobileNetV2 benchmark using latent features and critical features. We generated latent features based on the feature extractor (*i.e.*, the layer

TABLE 8.7: Effectiveness of defenses with different features. Latent features denote those generated by the feature extractor that is typically used in existing defenses. Critical features are extracted by our method from the identified layers.

Metric→	TPR (%)		FPR (%)	
Defense↓, Features→	Latent Features	Critical Features	Latent Features	Critical Features
AC	0.3	**96.32**	8.84	**7.67**
SCAn	46.05	**86.19**	2.57	**1.96**
Ours	1.31	**99.99**	**4.93**	6.23

TABLE 8.8: Performance of features from individual layers compared to identified layers by our defense. The LOI of WaNet and IAD are 9 and 8, respectively.

Layer		1	2	3	4	5	6	7	8	9	10	Ours
WaNet	TPR (%)	0.00	0.10	0.05	0.00	0.01	0.00	68.82	98.08	59.82	0.00	91.04
	FPR (%)	0.09	0.82	0.24	0.20	0.21	0.04	2.06	1.52	2.06	0.65	1.48
IAD	TPR (%)	19.32	34.03	6.44	30.49	61.09	78.65	88.81	99.65	99.10	2.36	99.12
	FPR (%)	1.65	1.38	1.44	1.60	2.27	1.70	1.29	1.13	1.09	1.24	1.26

before fully-connected layers) that is typically adopted in existing defenses. The critical features were extracted by the layer of interest (LOI) used in our method. Table 8.7 shows that using our features led to significantly better performance in almost all cases. In other words, existing detection methods can also benefit from our LOI selection. Also, we compared the performance of our method on CIFAR10-ResNet18 under WaNet and IAD when using the features of every individual layer, and when using LOI and the two layers before LOI. Table 8.8 shows that as we approach the critical layer, which was just before LOI with WaNet and at LOI with IAD, the detection performance gets better. Since our method included the critical layer, it also was effective. These results confirm the effectiveness of our layer selection and partly explain our method's good performance.

Effectiveness of cosine similarity. We also tried the Euclidean distance as a metric to differentiate between benign and poisoned samples, as we

TABLE 8.9: Comparison between Euclidean distance and cosine similarity as metrics to differentiate between benign and poisoned samples (±: standard deviation). Best scores are in bold.

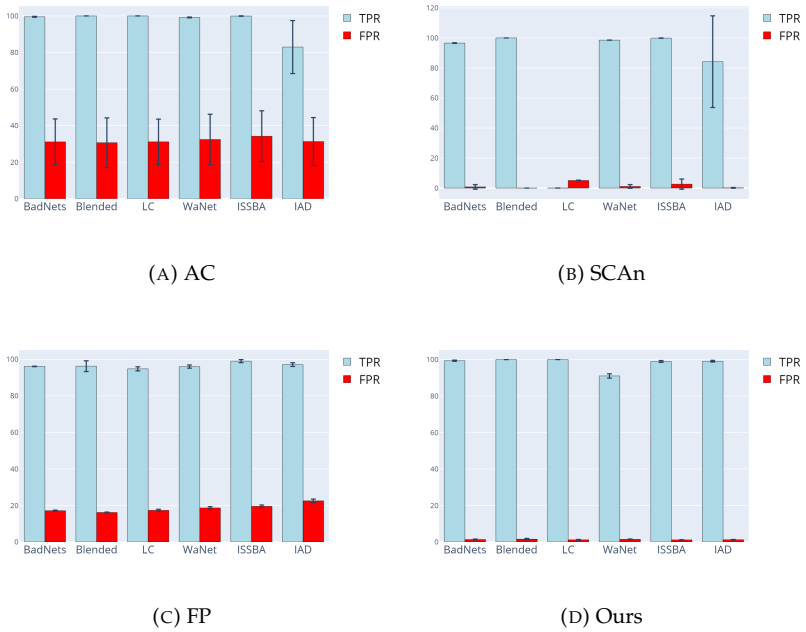
Threshold→	0.5		1		1.5		2		2.5		3	
Evaluation metric→ Similarity metric↓	TPR%	FPR%	TPR%	FPR%	TPR%	FPR%	TPR%	FPR%	TPR%	FPR%	TPR%	FPR%
Euclidean distance	99.98 (±0.01)	**24.77** (±**0.64**)	97.12 (±3.13)	**13.64** (±0.90)	95.73 (±2.41)	**8.67** (±**0.36**)	70.84 (±10.00)	4.69 (±0.36)	53.13 (±12.94)	3.21 (±1.78)	15.71 (±10.61)	1.56 (±**0.06**)
Cosine similarity	**100.00** (±**0.00**)	33.65 (±1.11)	**99.99** (±**0.00**)	18.74 (±**0.73**)	**99.91** (±**0.05**)	8.71 (±0.88)	**99.76** (±**0.04**)	**3.89** (±**0.32**)	**99.12** (±**0.45**)	**0.17** (±**0.13**)	**95.99** (±**3.04**)	**0.40** (±0.18)

did with cosine similarity. The only difference was considering any suspicious input with a summed distance greater than the mean of benign samples with τ standard deviations as potentially poisoned. Table 8.9 shows the detection performance of our defense with each of the two metrics in the CIFAR10-ResNet18 benchmark under the IAD backdoor attack with different thresholds. It can be seen that cosine similarity provides a better differentiation between benign and poisoned samples. A possible explanation is that the direction of features is more important for detection than their magnitude.

Stability comparison. We compared the stability of our defense with that of AC, SCAn, and FP on the CIFAR10-ResNet18 and GTSRB-MobileNetV2 benchmarks. We ran each defense five times and we report the average TPR and FPR with their standard deviations. Error bars in Figure 8.9 and Figure 8.10 show that our defense, in general, is more stable than the others.

Runtime comparison We compared the average CPU runtime (in seconds) of our defense with that of AC and SCAn on the whole benign and poisoned test sets. Figure 8.11 shows that our defense had the shortest runtime on CIFAR10-ResNet18 and the second shortest on GTSRB-MobileNetV2. It had a runtime slightly longer than that of AC on GTSRB-MobileNetV2 because MobileNetV2 contains a larger number of intermediate layers, which increases the time required to analyze them.

Additional results on layer-wise feature analysis Figure 8.12 shows the layer-wise behavior of benign and poisoned features w.r.t. the target



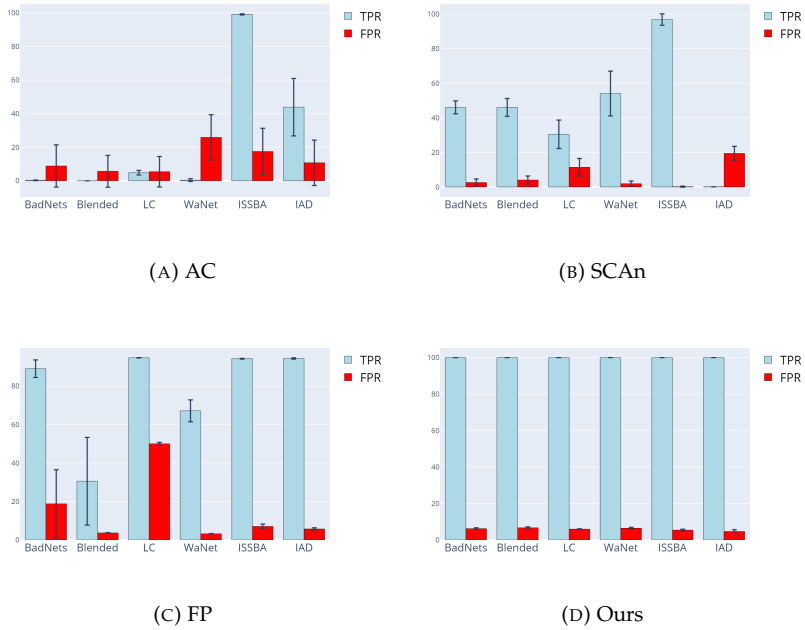
(A) AC (B) SCAn

(C) FP (D) Ours

FIGURE 8.9: Stability on the CIFAR10-ResNet18 benchmark

class on the CIFAR10-ResNet18 benchmark under all the used attacks. Figure 8.13 shows the same on the GTSRB-MobileNetV2 benchmark.

It can be seen that the layer with the maximum difference in cosine similarity is likely to be one of the three circled layers (the LOI and the two preceding layers). This happens in all cases, except for WaNet on GTSRB-MobileNetV2. We can also notice that the layer-wise gaps are smaller for WaNet, which is stealthier than the other attacks. Nevertheless, no matter how stealthy the attack is, the difference is always evident in one of the circled layers.



(A) AC (B) SCAn

(C) FP (D) Ours

FIGURE 8.10: Stability on the GTSRB-MobileNetV2 benchmark

8.5 Extending the proposed defense to FL

So far, we have seen how our proposed defense can effectively fend off backdoor attacks, outperforming its counterparts in centralized learning. In this section, we sketch how our defense design allows for its adoption in federated learning, specifically on the final global model used for inference. To utilize our defense in FL, the server or any of the peers need a small set of samples from the test data distribution. In situations where data is IID among peers, especially in cross-silo FL, this set may be available to peers. However, in non-IID settings, and especially in cross-device FL, the set may not be available. Nevertheless, peers can

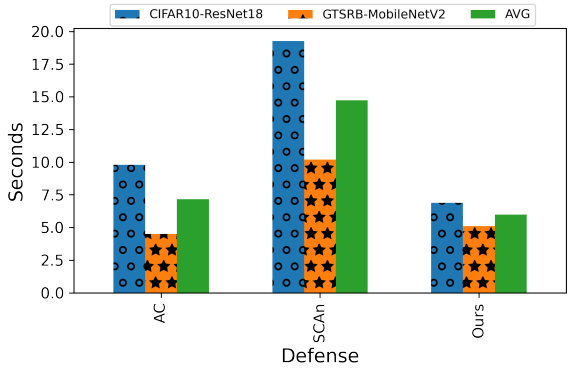
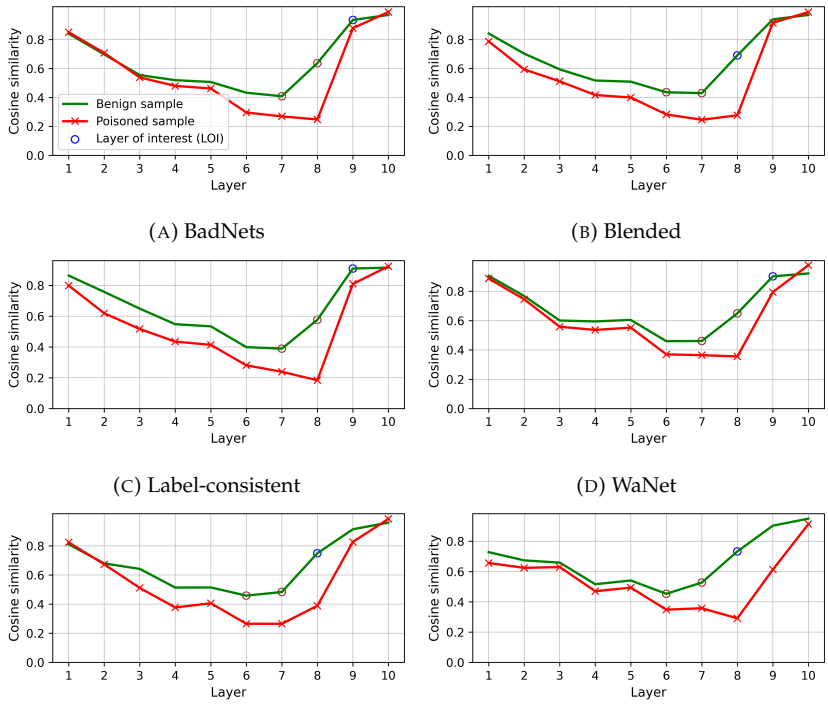


FIGURE 8.11: Average CPU running time in seconds.

collaborate under the server coordination to compute averages of intermediate activations and other parameters required for our defense, such as LOI, μ_t, and σ_t. Peers can then use these parameters to detect poisoned samples targeting the trained global model during inference. It is worth noting that BAs typically do not affect the model's accuracy on the main task, and defending against them during training can impose a high computational cost on the server. Therefore, adopting a defense strategy during inference may be more cost-effective and efficient.

8.6 Conclusions

In this chapter, we conducted a layer-wise feature analysis of the behavior of benign and poisoned samples generated by attacked DNNs at inference time. We found that the feature difference between benign and poisoned samples tends to reach the maximum at a critical layer, which can be easily located based on the behaviors of benign samples. From this finding, we proposed a simple yet effective backdoor detection method to determine whether a given suspicious testing sample is poisoned by analyzing the differences between its features and those of a



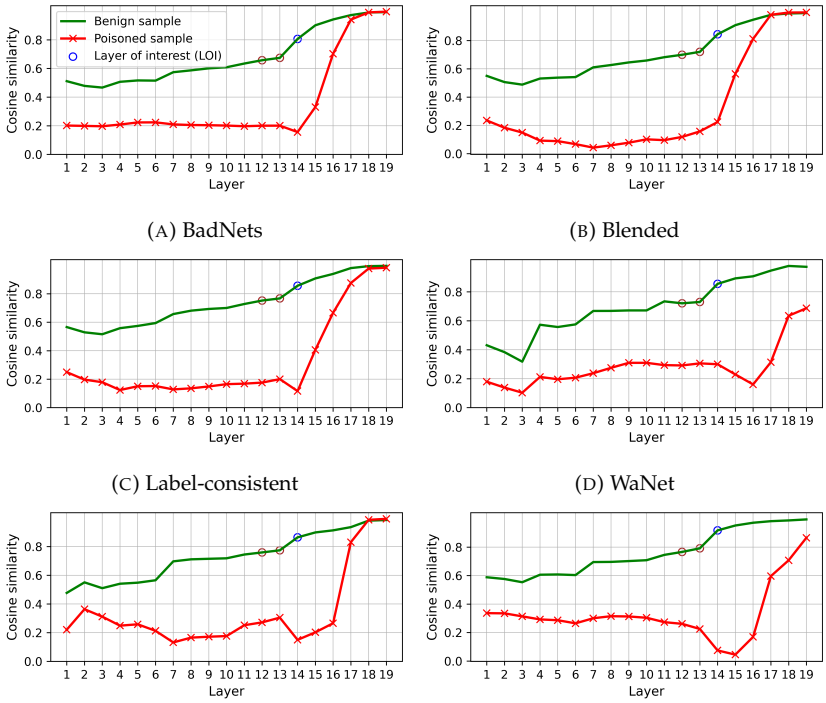
(A) BadNets (B) Blended

(C) Label-consistent (D) WaNet

(E) ISSBA (F) IAD

FIGURE 8.12: Layer-wise behavior of benign and poisoned samples w.r.t. the target class in the CIFAR10-ResNet18 benchmark, under all implemented attacks

few local benign samples. Moreover, we outlined how our proposed defense can be extended to the federated learning paradigm, allowing for easy adoption by the server or participating peers during the inference stage. Our extensive experiments on benchmark data sets confirmed the effectiveness of our detection. We hope our work can provide a deeper understanding of attack mechanisms, to facilitate the design of more effective and efficient backdoor defenses and more secure DNNs.



(A) BadNets

(B) Blended

(C) Label-consistent

(D) WaNet

(E) ISSBA

(F) IAD

FIGURE 8.13: Layer-wise behavior of benign and poisoned samples w.r.t. the target class in the GTSRB-MobileNetV2 benchmark, under all implemented attacks

Chapter 9

KeyNet: An asymmetric key-style framework for watermarking deep learning models

Safeguarding deep learning models from theft or unauthorized use is crucial for their owners, whether they are single entities (e.g., technology companies) in centralized learning or multiple entities (e.g., banks or hospitals) in cross-silo federated learning.

In this chapter, we propose *KeyNet*, a novel watermarking framework that meets a wide range of desirable requirements for effective watermarking. In particular it offers fidelity, robustness, reliability, integrity, capacity, security, authentication, uniqueness, and scalability.

KeyNet depends on three components: the WM carrier set distribution, the signature, and the marked model and its corresponding private model. The private model is trained together with the original model to decode the WM information from the marked model's predictions. The WM information is only triggered by passing a sample from the WM carrier set signed by the legitimate owner to the corresponding marked

model. The predictions of the marked model represent the encoded WM information that can be decoded only by the corresponding private model. The private model takes the predictions as input and decodes the WM information.

Unlike in previous works (discussed in Section 9.2), a watermarked input can take more than one label, which corresponds to the position of the owner's signature. Besides, the number of WM classes can be greater than the original task classes.

To successfully embed the WM and preserve the original task accuracy, the owner leverages multi-task learning (MTL) to learn both the original and the watermarking tasks together. After that, the owner distributes the marked original model, and keeps the private model secret. The owner uses the private model as a private key to decode the original model's outputs on the WM carrier set.

The main contributions of this chapter can be summarized as follows:

- *KeyNet* provides a strong link between the owner and her marked model by integrating two reliable authentication methods, namely a cryptographic hashing algorithm and a verification protocol. Also, the use of a cryptographic hash improves the capacity of embedding WM information. Besides being robust against DL model modifications such as compression and fine-tuning, *KeyNet* does not allow the WM to be overwritten by attackers without losing the accuracy of the original task.

- We demonstrate the ability of our framework to scale and fingerprint different unique copies of a DL model for different users in the system. The information of a user is combined with the owner's information, the carrier is signed with the combined information, and then a unique pair of a pre-trained model along with its corresponding private model is fine-tuned before being

delivered to the user. After that, the owner can identify the point of leakage with a small number of queries.

- We conduct extensive experiments and evaluate the proposed framework on different DL model architectures. The results show that *KeyNet* can successfully embed WMs with reliable detection accuracy while preserving the accuracy of the original task. Also, it yields a small number of false positives when tested with unmarked models.

The contributions in this chapter have been published in NajeebJebreel, 2020.

The remainder of the chapter is organized as follows. Section 9.1 describes the requirements of an effective DL models watermarking framework. Section 9.2 discusses related work. Section 9.3 describes the attack model to watermarking systems. Section 9.4 presents our framework in detail. Section 9.5 describes the experimental setup and reports the results on a variety of data sets. Section 9.6 outlines how KeyNet can be adopted in federated learning. Finally, Section 9.7 gathers conclusions.

9.1 Requirements for watermarking of DL models

Digital watermarking techniques have been widely used in the past two decades as a means to protect the ownership of multimedia contents like photos, videos and audios (Hartung and Kutter, 1999; Sebé, Domingo-Ferrer, and Herrera, 2000; Furht and Kirovski, 2004; Lu, 2004). The general idea of watermarking is to embed secret information into a data item (without degrading its quality) and then use the embedded secret to claim ownership of the item.

This concept of watermarking can also be extended to DL models. Several authors have proposed to use digital WMs to prove the ownership of models and address IP infringement issues (Uchida et al., 2017; Zhang et al., 2018; Adi et al., 2018; Chen et al., 2019b; Le Merrer, Perez, and Trédan, 2020; Li et al., 2019a; Rouhani, Chen, and Koushanfar, 2018; Chen, Rouhani, and Koushanfar, 2019; Xu, Li, and Yuan, 2020). The proposed methods fall into two main classes: i) *white-box methods*, which directly embed the WM information into the model parameters and then extract it by accessing those parameters; and ii) *black-box methods*, which embed WMs in the output predictions of DL models. The latter type of methods employ so-called trigger (or carrier) data samples that trigger an unusual prediction behavior: these unusual trigger-label pairs constitute the model watermark and they can be used by the model owner to claim her ownership.

For a watermarking framework to be effective, several requirements should be fulfilled (Uchida et al., 2017; Rouhani, Chen, and Koushanfar, 2018; Adi et al., 2018; Li et al., 2019a; Boenisch, 2020). These requirements include:

- Fidelity: The accuracy of the marked model on the original task shall not be degraded as a result of watermark embedding.
- Robustness: The watermark shall be robust against model modifications such as model fine-tuning, model compression or WM overwriting.
- Reliability: Watermark extraction shall exhibit a minimal false negative rate to allow legitimate owners to detect the WM with high probability.
- Integrity: Watermark extraction shall result in a minimal amount of false positives; unmarked models must not be falsely claimed.
- Capacity: It must be possible to include a large amount of watermark information in the target model.

- Security: The watermark must not leave a detectable footprint on the marked model; an unauthorized party must not be able to detect the existence of a WM.
- Unforgeability: An attacker must not be able to claim ownership of another party's watermark, or to embed additional watermarks into a marked model.
- Authentication: A strong link between the owner and her watermark must be provided; reliable verification of the legitimate owner's identity shall be guaranteed.
- Generality: The watermarking methodology must be applicable to different DL architectures and data sets.
- Efficiency: Embedding and extracting WMs shall not entail a large overhead.
- Uniqueness: The watermarking methodology must be able to embed a unique watermark for each user in order to distinguish each distributed marked model individually.
- Scalability: Unique watermarking must scale to many users.

Nonetheless, simultaneously satisfying all of these requirements is difficult to achieve (Boenisch, 2020).

9.2 Related work

The use of digital watermarking techniques has recently been extended from traditional domains such as multimedia contents to deep learning models. Related works can be categorized based on their application scenario as follows.

White-box watermarking. In this scenario, the model internal weights are publicly accessible. Uchida et al., 2017 embed an N-bit string WM

into specific parameters of a target model via regularization. To this end, they add a regularizer term to the original task loss function that causes a statistical bias on those parameters and use this bias to represent the WM. To project the parameters carrying the WM information, they use an embedding parameter X for WM embedding and verification. Based on the same idea, Wang et al., 2020b use an additional neural network instead of the embedding parameter to project the WM. The additional network is kept secret and serves for WM verification. Other works (Fan, Ng, and Chan, 2019; Wang and Kerschbaum, 2019) also adopt the same approach for embedding the WM information in the internal weights of DL models.

Black-box watermarking. Assuming direct access to the weights of a DL model to extract the WM is often unrealistic, particularly when someone wishes to extract the WM to claim legitimate ownership of a seemingly stolen model in someone else's power. To overcome this problem, several black-box watermarking methods have been proposed. These methods assume access to the predictions of the model and, thus, embed the WM information into the model's outputs. The idea of these methods is to use some samples and assign each sample a specific label within the original task classes (Le Merrer, Perez, and Trédan, 2020; Adi et al., 2018; Rouhani, Chen, and Koushanfar, 2018; Zhang et al., 2018; Guo and Potkonjak, 2018). The trigger-label pairs form what is called a trigger set or a carrier set. The carrier set is then used to embed the WM into the target model by training the target model to memorize its trigger-label pairs along with learning the original task. Since DL models are over-parameterized, it is possible to make them memorize the trigger-label samples through over-fitting (Hitaj and Mancini, 2018). Such embedding methods are known as back-dooring methods (Adi et al., 2018). The triggers are used later to query a remote model and compare its predictions with the corresponding labels. A high proportion of matches between the predictions and the labels is used to prove the ownership of the model.

Trigger set methods can be classified into several types. A first type of methods is based on assigning a random label to each trigger. The trigger samples themselves may be random samples from different distributions (Adi et al., 2018) or adversarial samples (Le Merrer, Perez, and Trédan, 2020). This approach has many drawbacks. Beyond its limited capacity regarding the number of triggers that can be used for verification, it does not establish a strong link between the owner and her WM. Thus, it is easy for an attacker to insert his WM by using a set of trigger-label pairs, giving them random labels, and then claim ownership of the owner's model. This type of attack is called the ambiguity attack (Fan, Ng, and Chan, 2019).

A second type of methods relies on inserting the WM information into the original data. The inserted information may be a graphical logo (Li et al., 2019c), the owner's signature (Guo and Potkonjak, 2018), a specific text string (which could be the company name) (Zhang et al., 2018), or some pattern of noise (Zhang et al., 2018). These methods may affect the accuracy of the model in the original task. Besides, the WM may be vulnerable to model fine-tuning aimed at destroying the WM. That is possible because the WM samples will be close to their counterparts in the same class in the feature space. Hence, fine-tuning may cause the WM pattern to be ignored and those samples to be classified into their original classes again (Cao, Jia, and Gong, 2019).

Another type of black-box methods proposes to exploit the discarded capacity in the intermediate distribution of DL models' output to embed the WM information (Xu, Li, and Yuan, 2020). They use non-classification images as triggers and assign each trigger a serial number (SN) as label. SN is a vector that contains n decimal units where n is the number of neurons in the output layer. The value of SN serves as an identity bracelet that proves ownership of a marked model. To embed the WM information in the softmax layer predictions, they train the target model

to perform two tasks simultaneously: the original task, which is a multi-class classification task, and the watermarking task, which is a regression task. They use the mean square error (MSE) as a loss function for the watermarking task to minimize the difference between the predicted value of a trigger and its corresponding SN. To link the owner with her marked model, they create an endorsement by a certification authority on the generated SNs. Ownership verification is performed by sending some trigger inputs, extracting their corresponding SNs, and having them verified by the authority. This method preserves the original task accuracy and also creates a link between the owner and her WM model. However, it has several drawbacks. The length of the SN depends on the size of the output layer in the model. This may prevent the owner from embedding a large WM. Moreover, by relying on values after the decimal point to express a specific symbol in the SN, if some decimal values are slightly changed, the entire SN will be corrupted. A modification like model fine-tuning would lead to destroying the WM information. In this respect, the authors do not evaluate two important types of modifications that could affect the WM, namely, model fine-tuning and WM overwriting.

Wu et al., 2020a proposes to watermark DL models that output images. They force the marked model to embed a certain WM (*e.g.*, logo) in any image output by that model. They train two models together: the marked model and the extractor model. The latter extracts the WM from the output of the former. The marked model is distributed while the extractor is kept secret by the owner. The paper does not evaluate the robustness of the method against the basic attacks that may target the marked model, such as model fine-tuning, model compression, and WM overwriting. Besides, there is a high probability that the WM extractor has memorized the WM in its weights; as a result, when it receives images from models different from the marked one, it might generate the same WM each time.

A shortcoming of most of the aforementioned WM methods is that

the WM is the same in all copies of the model Boenisch, 2020. Hence, if the owner distributes more than one copy of a model, it is impossible for the legitimate owner to determine which of the authorized users has leaked it.

9.3 Attack model

To ensure the robustness of a watermarking methodology, it should effectively overcome (at least) three potential attacks:

- **Model fine-tuning**. In this attack, an attacker who has a small amount of the original data retrains the WM model with the aim of removing the WM while preserving the accuracy in the original task.
- **Model compression**. The compression of a DL model's weights minimizes its size and speeds up its performance. Model compression may compromise the WM within the marked model, thereby affecting its detection and extraction.
- **Watermark overwriting**. This type of attack is a major threat to the WM because it might result in the attacker being able to overwrite the owner's WM, or also to embed another WM of his own and thus seize ownership of the WM model. We make the following assumptions about the attacker. First, the attacker is assumed to have a small amount of training data when compared to the owner. Otherwise, he can use his data to train a new model from scratch, or use the predictions of the WM model to create an unmarked copy of it by predictive model-theft techniques (Tramèr et al., 2016). Second, the attacker is assumed to be aware of the methodology used to embed the WM but to be unaware of the carrier set distribution, the owner's signature, or the topology of the owner's private model. An attack is considered to be successful if the attacker manages to overwrite the WM without losing

much accuracy in the original task. The goal is to prevent the attacker from overwriting the WM without significantly impairing the original task accuracy, proportional to the amount of data he knows. To make a realistic trade-off, the relationship between the size of the data known by the attacker and the loss in original task accuracy should be inversely proportional: the smaller the attacker data, the greater the accuracy loss is.

9.4 The *KeyNet* framework

Instead of having the model memorize the WM through over-fitting, in our approach we design the watermarking task as a standalone ML task with its logical context and rules. Firstly, this task performs a one-vs-all classification so that it can distinguish WMs from original samples with different distributions. Second, the watermarking task learns the features that enable it to identify the spatial information of the legitimate owner's signature. Third, it learns to distinguish the pattern of the owner's signature from the patterns of fake signatures. The purpose of designing the watermarking task in this way is: i) to increase the difficulty of the task so that an attacker with little training data cannot add his WM without losing the accuracy of the original task; ii) to provide a reliable verification method that strengthens the owner's association with her marked model; iii) to achieve greater security by keeping the private model in the hands of the owner; iv) to embed a robust WM without affecting the accuracy of the original task; v) to produce different unique copies of a DL model for different users of the system based on the same carrier by signing the carrier with the joint signature of the user/owner; and vi) to scale for a large number of users and identify the leakage point with high confidence and little effort.

KeyNet consists of two main phases: watermark embedding and watermark extraction and verification. Figure 9.1 shows the global workflow of *KeyNet*. The marked DL model is used as a remote service, so

that the user can only obtain its final predictions. *KeyNet* passes the final predictions of the remote DL model to its corresponding private model, which uses them to decode the WM information. In the ownership verification protocol we exploit the fact that each sample in the owner's WM carrier set can take different labels based on the position of the owner's signature in it. We next briefly explain the workflow of each phase.

Watermark embedding. *KeyNet* takes four main inputs in the WM embedding phase: the target model (pre-trained or from scratch), the original data set, the owner's WM carrier set, and the owner's information string. The output is the marked model, its corresponding private model, and the owner's signature. The WM carrier set samples are signed using the owner's signature. After that, the signed WM carrier set is combined with the original data set and they are used to fine-tune (or train) the targeted model. The private model takes the final predictions of the original model as inputs and outputs the position of the owner's signature on the WM sample. To embed the WM information and preserve the main task accuracy at the same time, WM embedding leverages multi-task learning (MTL) to train the two models jointly.

MTL is an ML approach that allows learning multiple tasks in parallel by sharing the feature representation among them (Caruana, 1997; Ruder, 2017). Many MTL methods (Mrkšić et al., 2015; Li, Liu, and Chan, 2014; Zhang et al., 2016) show that different tasks can share several early layers, and then have task-specific parameters in the later layers. MTL also helps the involved tasks to generalize better by adding a small amount of noise that helps them reduce over-fitting (Neelakantan et al., 2015; Ndirango and Lee, 2019).

In our framework, the original model parameters are shared among the original task and the watermarking task. When the marked model receives unmarked data samples, its predictions represent the classification decision on those samples. However, when it receives a watermarked sample, its output represents the features that the private model needs to distinguish the signature position on that sample. For this to be

possible, the private model forces the shared layer (the original model parameters) to produce a different representation of the WM samples. We can see the private model as a private key held only by the owner that decodes the WM information from the original model predictions. More details about this phase are given in Sections 9.4.2 and 9.4.3.

Watermark extraction and verification. The owner can extract the WM information from a suspicious remote DL by taking a random sample from her WM carrier set, putting her signature on one of the predefined positions, and querying the remote model. After that, she passes the remote model's predictions to the private model. If the private model decodes the WM information and provides the position of the signature with high accuracy, then the owner can claim her ownership.

To verify the ownership of a remote black-box DL model, the owner first delivers the WM carrier set and her signature to the authority. She also tells the authority about the methodology used to sign the WM samples along with the predefined positions where the WM may be placed. The authority (*i.e.*, the *verifier*) randomly chooses a sample from the carrier set, puts the signature in a random position, queries the remote DL model, and sends the model's predictions to the owner. The owner (*i.e.*, the *prover*) takes the predictions, passes them to her private model, and tells the authority the position of her signature on the image. The authority repeats the proof as many times as she desires. After that, the owner's answer accuracy is evaluated according to a minimum threshold. If the owner surpasses the threshold, her ownership is regarded as proven by the authority. More details about this phase are given in Section 9.4.4.

The following subsections describe each phase in detail. First, we formalize the problem. Then, we describe the methodology for signing and labeling the WM carrier set using the hashed value of the owner's information. After that, we describe the WM embedding phase by training the original model and the private model on the original and the watermarking tasks jointly. Finally, we explain the WM extraction and

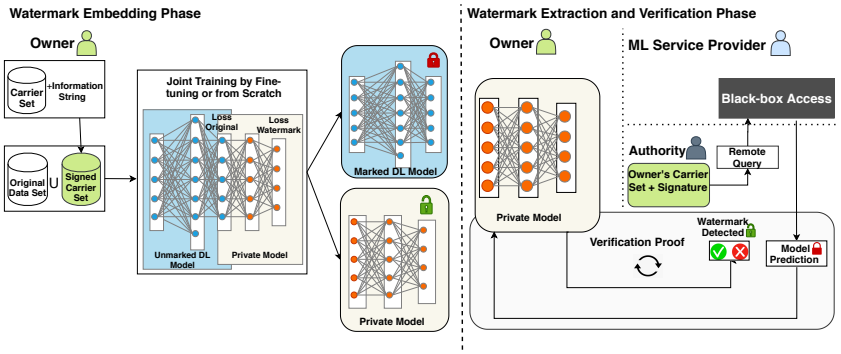


FIGURE 9.1: *KeyNet* global workflow.

verification phase.

9.4.1 Problem formulation

The key idea of our framework is to perform two tasks at the same time: the original classification task T_{org} and the watermarking task T_{wm}. To do so, *KeyNet* leverages the multi-task learning (MTL) approach to achieve high accuracy in both tasks by sharing the parameters of the original model between the two tasks. *KeyNet* adds a private model to the original model. The original model's objective is to correctly classify the original data samples into their corresponding labels, while the private model's objective is to correctly predict the position of the owner's signature in a sample of the WM carrier set using the original model predictions. We can formally represent as follows the problem being tackled:

- **Representation of the original, private and combined models.** Let $D_{org} = \{(x_i, y_i)\}_{i=1}^{n}$ be the original task data and $D_{wm} = \{c_j\}_{j=1}^{m}$ be the WM carrier set data. Let h be the function of the original

model and f be the function of the private model. Let θ_1 the parameters of h and θ_2 be the parameters of f. Let *signature* be the owner's signature and $PL = \{(p_k, l_k)\}_{k=1}^{z}$ be the set of predefined position-label pairs (*e.g.* (position:top left, label: 1), (position: bottom right, label: 4)) where z is the total number of positions at which the signature can be located on a WM carrier set sample. Let *sign* be the function that puts a *signature* on a carrier set sample c and returns the signed sample c^{p_k} and its corresponding label l_k as

$$(c^{p_k}, l_k) = sign(signature, c, p_k).$$

Let D_{wm}^{signed} be the signed carrier set samples that contain all the (c^p, l) pairs. We use D_{org} and D_{wm}^{signed} to train both h and f to perform T_{org} and T_{wm}.

Typically, the function h tries to map each $x_i \in D_{org}$ to its corresponding y_i, that is, $h(x_i) = y_i$.

Let $f(h)$ be the composite function that aims at mapping each $c_j^{p_k}$ to its corresponding l_k, that is, $f(h(c^{p_k})) = l_k$.

- **Embedding phase.** We formulate the embedding phase as an MTL problem where we jointly learn two tasks that share some parameters in the early layers and then have task-specific parameters in the later layers. The shared parameters in our case are θ_1, while θ_2 are the WM task specific parameters. We compute the weighted combined loss L as

$$L = \alpha Loss(h(x), y) + (1 - \alpha) Loss(f(h(c^p)), l),$$

where $h(x)$ represents the predictions on the original task samples, $f(h(c^p)$ represents the predictions on D_{wm}^{signed} samples, $Loss(h(x), y))$

is the loss function that penalizes the difference between the original model outputs $h(x)$ and the original data targets y, $Loss(f(h(c^p)), l)$ is the loss function that penalizes the difference between the composite model outputs $f(h(c^p))$ and the signed WM carrier set's target l, and α is the combined weighted loss parameter. Then we seek θ_1 and θ_2 that make L small enough to get acceptable accuracy on both T_{org} and T_{wm}. Once this is done, the WM has been successfully embedded while preserving the accuracy of the original task T_{org}.

- **Verification phase.** The verification function V checks whether a claimer (a.k.a. the *prover*), who has delivered her *signature* and WM carrier set D_{wm} to the authority (a.k.a. the *verifier*), is the legitimate owner of a remote model $h^{'}$. If the prover is the legitimate owner of $h^{'}$, she will be able to pass the verification process and thus prove her ownership of $h^{'}$. That is because she possesses the private model f, which was trained to decode $h^{'}$ predictions on her signed D_{wm}.

  Here, r represents the number of the required rounds in the verification process and T denotes the threshold needed to prove the ownership of $h^{'}$. Note that the authority also knows the signing function $sign$ used to sign D_{wm} samples in order to obtain (c^{p_k}, l_k) pairs.

  The function V can be expressed as $V(\{(f(h^{'}(c^{p_k})), l_k, p_k)\}_{k=1}^{r}, T) = \{True, False\}$.

9.4.2 Watermark carrier set signing and labeling

The methodology we use for labeling the WM carrier set is key in our approach. In contrast to related works, which assign a unique label to each of the WM carrier set samples, our labeling method allows for a

single sample to carry more than one label. More precisely, any sample $c \in D_{wm}$ can take one of z labels $\{l_k\}_{k=1}^{z}$, where z is the number of predefined positions $\{p_k\}_{k=1}^{z}$ at which the *signature* of the owner can be placed. Besides the z positions, if a sample is not signed by the legitimate owner, it uses label 0 by default. Also, if the sample is not in D_{wm}, it uses label 0 by default even if it is signed by the owner. Function 1 formalizes the method used to sign and label the D_{wm} samples.

First and foremost, the owner's information and the metadata of her model are endorsed by the authority. This information is a string of arbitrary length. After that, Function 1 returns the signed D_{wm}^{signed} WM carrier set consisting of pairs (signed D_{wm} sample, label), a signed D_{dif}^{signed} consisting of pairs (signed sample from a set D_{dif} of different distribution, 0), and the owner's signature *signature* used to sign the samples. The inputs to the function are:

1. The owner's information string $infStr$ that has been endorsed by the authority.
2. The size of the signature s to be placed on the WM carrier set samples.
3. The owner's WM carrier set D_{wm}.
4. The set of positions-labels pairs $PL = \{p_k, l_k\}_{k=1}^{z}$ that defines the signature positions and their corresponding labels.
5. A small set of samples D_{dif} from other distributions than D_{wm}.

The function starts its work by taking the hash value for $infStr$ and then converting it to a squared array of size s, as follows. In our implementation we use *SHA256* which yields 256 bits, that are converted to 64 characters by digesting them to hexadecimal. The last s (with $s \leq 64$) among the digested characters are converted to a decimal vector of length s. The decimal vector values are normalized between 0 and

1 by dividing them by the maximum value in the vector. The normalized vector is then reshaped into a squared array. The resulting array represents the owner's *signature*. Note that it is also possible to use any hashing function different from SHA256.

Once we obtain *signature*, we start the labeling step of the WM carrier set D_{wm}. For each $c \in D_{wm}$, we replicate c for z times where z is the total number of possible positions $\{p_k\}_{k=1}^{z}$ of the signature. Then we use the *sign* function to place *signature* in the position p_k to obtain the (c^{p_k}, l_k) pair.

We also leave one copy of each sample without signing and assign it the label 0. That is, if a carrier set sample c is not signed with *signature*, it will be represented by the $(c, 0)$ pair.

We then do two steps:

1. We generate a fake information string $fakeStr$ by making a slight modification to $infStr$. Then, we generate $signature_{fake}$ following steps similar to those above followed to generate the real owner's signature *signature*. After that, we sign the D_{wm} samples each with a different fake signature and a randomly chosen position p_k and, instead of assigning to the signed sample the corresponding label l_k, we assign it the label 0. To obtain a new fake signature, we again make a slight random modification in $infStr$ and generate the fake signature in the same way as above.

2. We take samples from other distributions D_{dif} and sign them with the real owner's signature as we did with the D_{wm} samples. We assign them the label 0. We use samples from different distributions to avoid triggering the WM just with the owner's real signature. In other words, we make the triggering of the WM from a marked model dependent on the carrier set distribution in addition to the pattern of the owner's signature.

The goal of the above two steps is to make the marked model h^* output the information that tells the position of a signature only if we pass

to it a sample that *belongs to D_{wm} and is signed with the real signature*. Otherwise, h^* ignores the presence of any different signature from *signature* on D_{wm} samples. This also avoids h^* responding to samples from different distributions than the D_{wm} distribution.

Finally, Function 1 returns the signed WM samples D_{wm}^{signed}, the signed samples from different distributions D_{dif}^{signed}, and the signature *signature* that will be used to trigger the marked model h^*. Note that this process is performed only once, before the WM is embedded.

9.4.3 Watermark embedding

To successfully embed the WM in the original model h without compromising the accuracy of the original task, we jointly train both h and the private models f simultaneously. Since a large amount of the carrier set samples have been signed, we first randomly select one-fifth of D_{wm}^{signed} for training. The random selection allows for the representation of all the possible states while reducing the carrier set size. The signed samples from other distributions D_{dif}^{signed} are combined with those randomly selected and assigned to D_2. In the end, we add D_2 to the original task data D_{org}. The resulting combined data set $D = D_{org} \cup D_2$ are used in the training step as specified next.

During joint training, a batch b is taken from D. Then b is separated into two sub-batches: $\{x, y\} \in D_{org}, \{c, l\} \in D_2$. $\{x\}$ is passed to the original model h and the loss $L_f(h(x), y)$ is calculated. On the other hand, $\{c\}$ is first passed to h, and then the predictions of the original model $h(c)$ are passed to the private model f; the loss $L_f(f(h(c)), l)$ is afterwards calculated. Since we deal with two classification tasks, the cross-entropy loss function is used to calculate the loss for both tasks. We use the parameter α to balance the weight of $L_f(h(x), y)$ and $L_f(f(h(c)), l)$ before we add them up in the joint loss L. Parameter α allows us to choose the best combination of the weighted loss that preserves the accuracy of

T_{org} while embedding WM successfully. Then, the parameters of h, f are optimized to minimize L.

Reducing $L_f(h(x), y)$ forces h to predict the correct class for x, while reducing the watermarking task loss $L_f(f(h(c)), l)$ forces the private model to distinguish the distribution of the WM carrier set D_{wm}, and predict the location of the owner's signature in its samples. The original model, in addition to performing the original task, also executes the first part of the watermarking task by outputting the features needed to find the position of the signature. By using these features as input, the private model performs the second part of the watermarking task, which consists in identifying the signature position.

Regarding the architecture of the private model f, the number of inputs corresponds to the size of h predictions, whereas the number of outputs corresponds to the number of classes of the WM task $z + 1$. We also add at least one hidden layer in-between. The hidden layer enriches the information coming from the original model before passing it to the output layer of the private model.

Algorithm 1 summarizes the process of embedding the WM. It takes an unmarked model h that might be pre-trained or be trained from scratch, the private model f, the original data set D_{org}, the signed WM carrier set D_{wm}^{signed}, signed samples from other distributions D_{dif}^{signed} and the joint loss balancing parameter α. The output of the embedding phase is a marked model h^* along with its corresponding private model f.

9.4.4 Watermark extraction and verification

The verification process of ownership involves a would-be owner in the role of prover and the authority in the role of verifier. The would-be owner claims that a remote model $h^{'}$ is part of her IP. The authority is given the WM carrier set D_{wm}, the would-be owner's *signature*, the signing function *sign*, and remote access to $h^{'}$. The authority sets an accuracy threshold T and a number of required verification rounds r to

decide whether $h^{'}$ is the IP of the would-be owner. In each round, the authority randomly selects a sample c from D_{wm}, signs it using *signature* in a random position p_k, and sends the signed sample c^{p_k} to the remote model $h^{'}$. The predictions $h^{'}(c^{p_k})$ (which contain the encoded WM information) are forwarded to the would-be owner. The latter passes them to her private model f to obtain $l_k = f(h^{'}(c^{p_k}))$. Since the relationship between positions and labels is one-to-one, the would-be owner can use l_k to tell the authority the position p_k of her signature in c. After r rounds, the accuracy acc of the would-be owner at detecting the positions is the number of correct answers divided by r. If $acc \geq T$, then authority certifies that $h^{'}$ is owned by the would-be owner.

Note that the authority can also send the samples without signing them or sign them using fake signatures different from *signature*. In this case, the would-be owner should tell the authority that this sample does not contain her signature. That is possible because in these cases the private model gives them the label 0. Protocol 1 formalizes the verification process.

9.5 Experiments

In this section, we evaluate the performance of *KeyNet* on two image classification data sets and with two different DL model architectures. First, we present the experimental setup. After that, we evaluate the proposed framework performance against the requirements stated in Section 9.1. We focus on robustness, authentication, scalability, capacity, integrity and fidelity but, since our framework partly fulfills the rest of requirements, we also assess its performance on each of them.

9.5.1 Experimental setup

Original task data sets and DL models. We used two image classification data sets: CIFAR10 (Krizhevsky, 2009) and FMNIST5. CIFAR10

has 10 classes, while FMNIST5 is a subset of the public data set Fashion-MNIST (Xiao, Rasul, and Vollgraf, 2017); FMNIST5 contains the samples that belong to the first five classes in Fashion-MNIST (classes from 0 to 4). Table 9.1 summarizes the original task data sets, the carrier set, and the DL models and their corresponding private models.

Watermark carrier sets. We employed three different data sets as WM carrier sets: STL10 (Coates, Ng, and Lee, 2011), MNIST (LeCun and Cortes, 2010), and Fashion-MNIST (the latter was used only in attacks). We applied Function 1 to label the carrier set's images. Then, we passed the carrier set, the owner's information, the signature size, a fake signature, some samples from different distributions, and a list containing the labeling order of the positions of the owner's signature in the carrier set. We used the following labeling order: (1: Top left, 2: Top right, 3: Bottom left, 4: Bottom right, and 5: Image center). Function 1 assigns label 0 to an image if i) the image belongs to the carrier set but does not carry any signature; ii) the image belongs to the carrier set but carries a signature different from the owner's signature; iii) the image does not belong to the carrier set distribution (even if it is signed with the owner's real signature). For WM accuracy evaluation, we randomly sampled 15% of the WM carrier set. After that, we signed them in different random positions and assigned them the corresponding labels. Figure 9.2 shows some examples of signed carrier set images and their corresponding labels.

Attacker configurations. We assumed the attacker has varying percentages of the training data, ranging from 1% to 30% of the original training data. The attacker's training data were randomly sampled from the original training data. We also assumed that the WM carrier set distribution is a secret between the owner and the authority, so we assigned the attacker different WM carrier sets from those of the owner. The attacker's private model was slightly different as well, because the owner's private model and its architecture are secret. The rest of the attacker's configurations and hyper-parameters were the same as the

FIGURE 9.2: Examples of signed STL10 carrier set images employed with the CIFAR10 data set. Each image shows the signature position and its corresponding label.

owner's. Table 9.2 summarizes the attacker's WM carrier sets and private model architectures.

Performance metric. We used *accuracy* as performance metric to evaluate all the original and WM tasks. *Accuracy* is the number of correct predictions divided by the total number of predictions.

Training hyper-parameters. We used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with learning rate = 0.001, momentum = 0.9, weight decay = 0.0005, and batch size = 128. We trained all the original unmarked models for 250 epochs. To embed the WM from scratch, we trained the combined model for 250 epochs. To embed the WM in a pre-trained model, we combined the private model and fine-tuned the combination for 30 epochs.

To jointly train the original and private models, we used parameter α to weight the original task loss and the WM task loss before optimization. For CIFAR10 we used $\alpha = 0.9$ when embedding the WM in a pre-trained model, while we used $\alpha = 0.95$ when embedding the WM into a DL model from scratch. For FMNIST5 we used $\alpha = 0.85$ to embed the WM in a pre-trained model, while we used $\alpha = 0.9$ to embed the WM

from scratch. The experiments were implemented using Pytorch 1.6 and Python 3.6.

9.5.2 Experiments and results

First, we made sure that an accurate private model could not be obtained (and therefore the ownership of a DL model could not be claimed) by using only the predictions of a black-box DL model. To do so, we queried different unmarked DL models with all the signed samples in the owner's WM carrier set, and we used the predictions as input features to train the private model. Table 9.3 shows the performance of the private models obtained in this way after 250 epochs.

It can be seen that the average accuracy of the private model at detecting the signature position inside the WM carrier set is as low as 32.27%. This accuracy was obtained by granting unconditional query access to the black-box model and by using its predictions as input to train the private model. Based on that, we decided to set threshold $T = 0.9$, which is nearly three times greater than the above average accuracy. Therefore, to prove her ownership of a black-box DL model, the owner's private model must detect the signature positions in the WM carrier set with an accuracy greater than or equal to 90%.

In the following, we report the results of *KeyNet* on several experiments that test its fulfillment of the requirements described in Section 9.1.

Fidelity. Embedding the WM should not decrease the accuracy of the marked model on the original task. As shown in Table 9.4, the marked model's accuracy is very similar to that of the unmarked model. This is thanks to the joint training, which simultaneously minimizes the loss for the original task and the WM task. Also, *KeyNet* did not only preserve the accuracy in the original task, but sometimes it even led to improved accuracy. That is not surprising, because the watermarking task added a small amount of noise to the marked model, and this helped reduce over-fitting and thus generalize better.

KeyNet therefore fulfills the fidelity requirement by reconciling accuracy preservation for the original task and successfully embedding of the WM in the target models.

Reliability and robustness. *KeyNet* guarantees a robust DL watermarking and allows legitimate owners to prove their ownership with accuracy greater that the required threshold $T = 90\%$. Table 9.4 shows that WM detection accuracy was almost 100%, and thus our framework was able to reliably detect the WM.

We assess the **robustness** of our framework against three types of attacks: *fine-tuning* (Tajbakhsh et al., 2016), *model compression* (Han, Mao, and Dally, 2015; Han et al., 2015) and *WM overwriting* (Uchida et al., 2017; Shafieinejad et al., 2019):

- *Model fine-tuning.* Fine-tuning involves retraining a DL model with some amount of training data. It may remove or corrupt the WM information from a marked model because it causes the model to converge to another local minimum. In our experiments, we sampled 30% of the original data and used them to fine-tune the marked model by optimizing its parameters based only the loss of the original task. Table 9.5 outlines the impact of fine-tuning on the WM detection accuracy with all benchmarks. We can notice that *KeyNet* is robust against fine-tuning and was able to preserve a WM detection accuracy of about 97% after 200 epochs. The explanation for this strong persistence against fine-tuning is that *KeyNet* does not embed the WM information within the decision boundaries of the original task classes. Hence, the effect of fine-tuning on the WM is very small.

- *Model compression.* We used the compression approach proposed in Han et al., 2015 to prune the weight parameters in the marked DL models. To prune a particular layer, we first sorted the weights in the specified layer by their magnitudes. Then, we masked to zero the smallest magnitude weights until the desired pruning

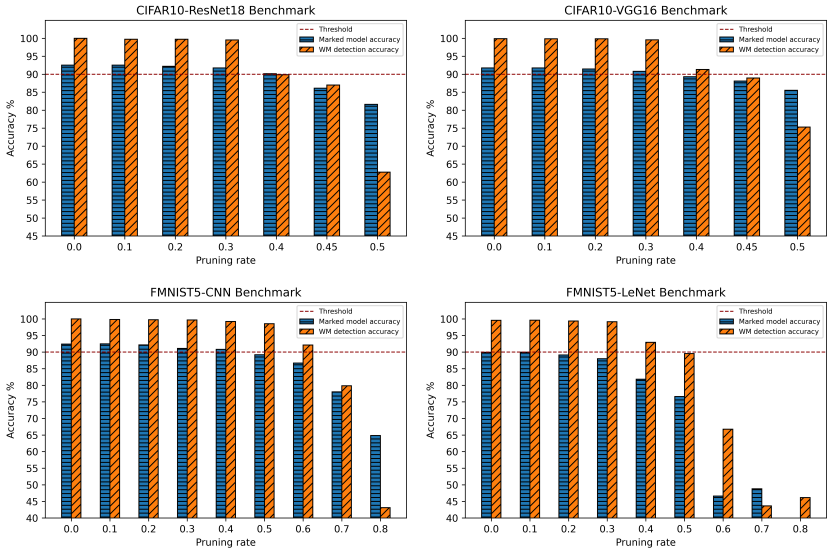


FIGURE 9.3: Robustness against model compression. The X-axes indicate the pruning levels we used for each marked model. The blue bars indicate the marked model accuracy in the original task, while the orange bars indicate the accuracy of WM detection. The horizontal dotted line indicates the threshold $T = 90\%$ used to verify the ownership of the model.

level was reached. Figure 9.3 shows the impact of model compression on both WM detection accuracy and original task accuracy with different pruning rates. We see that *KeyNet* is robust against model compression, and the accuracy of the WM remains above the threshold $T = 90\%$ as long as the marked model is still useful for the original task. This is consistent, because when the marked model becomes useless due to excessive compression, the owner will not be interested in claiming its ownership.

- *Watermark overwriting.* Assuming that the attacker is aware of the methodology used to embed the WM, he may embed a new WM

that may damage the original one. In our experiments, we assumed that the attacker knows the methodology but knows neither the owner's carrier set nor the owner's private model architecture. We studied the effect on the WM of the attacker's knowing various fractions of the original training data, ranging from 1% to 30%. We chose the lower bound 1% based on Aiken, Kim, and Woo, 2020; the authors of that paper demonstrate that an attacker with less than 1% of the original data is able to remove the watermark with a slight loss in the accuracy of the original task.

To overwrite the WM, the attacker selected her own carrier set and signed it using Function 1 with her signature. Then she trained her private model along with the marked model in the same way as in Algorithm 1. Tables 9.6 and 9.7 summarize the results of WM overwriting experiments. The attacker was able to successfully overwrite the original WM and successfully embed her new WM, but this was done at the cost of a substantial accuracy loss in the original task when using a fraction of the training data up to 10%. Thus, our watermark easily survives the attacks in the conditions described in Aiken, Kim, and Woo, 2020. For fractions above 10%, the accuracy of the marked model became competitive, but an attacker holding such a large amount of training data can easily train her own model and has no need to pirate the owner's model Li et al., 2019a.

Integrity. *KeyNet* meets the integrity requirement by yielding low WM accuracy detection with unmarked models, and thus it does not falsely claim ownership of models owned by a third party. In our experiments, there were 6 classes for the watermarking task. Looking at Table 9.8, the accuracy of falsely claimed ownership of unmarked models is not far from guessing 1 out of 6 numbers randomly, which equals approximately 16.6%.

Authentication. *KeyNet* fulfills the authentication requirement by

design. Using a cryptographic hash function such as *SHA256* to generate the owners' signatures establishes a strong link between owners and their WMs. Furthermore, the verification protocol of *KeyNet* provides strong evidence of ownership. When the authority uses a fake signature, the marked model does not respond. This dual authentication method provides unquestionable confidence in the identity of the legitimate owner.

Security. Since *KeyNet* embeds the WM in the dynamic content of DL models through joint training, and since modern deep learning models contain a huge number of parameters, detecting the presence of the WM in such models is infeasible. In case the attacker knows that a model contains WM information and wants to destroy it, he will only be able to do so by also impairing the accuracy of the model in the original task. Regarding the security of the owner's signature, the use of a strong cryptographic hash function, such as SHA256, provides high security, as we next justify. On the one hand, if the signature size s is taken long enough, it is virtually impossible for two different parties to have the same signature: the probability of collision for s hexadecimal digits is $1/16^s$, so $s = 25$ should be more than enough. On the other hand, even if the owner's signature is known by an attacker, the cryptographic hash function makes it impossible to deduce the owner's information from her signature.

Unforgeability. To prove ownership of a DL model that is not his, an attacker needs to pass the verification protocol (Protocol 1). However, the private model allowing watermark extraction is kept secret by the legitimate owner. Without the private model, even if the attacker knows both the WM carrier set and the owner's signature, the attacker can only try a random strategy. Yet, the probability of randomly guessing the right position at least a proportion T of r rounds is at most $1/z^{\lfloor Tr \rfloor}$. This probability can be made negligibly small by increasing the number r of verification rounds.

Thus, *KeyNet* partially meets the unforgeability requirement: an attacker

can embed additional WMs into a marked model, but cannot claim ownership of another party's WM.

Capacity. Capacity can be viewed from two perspectives: i) the framework allows the inclusion of a large amount of WM information, and ii) triggers available in the verification process are large enough. Given that hashes are one-way functions with fixed length, the information that can be embedded in them is virtually unlimited. In our experiments, we used a medium-sized signature of $s = 25$ characters. Nevertheless, *KeyNet* allows flexibility in specifying various signature sizes and in using hash functions other than SHA256. On the other hand, *KeyNet* can use a large number of samples in WM verification. In addition to using all samples belonging to a certain distribution (the WM carrier set), it allows using samples from other distributions due to the method of labeling and training used. The marked model gives the signature information if the signature is placed on top of a WM carrier set's sample, while samples from different distributions are given the label 0 (even if they are signed with the signature of the legitimate owner).

Uniqueness and scalability. *KeyNet* can be easily extended to produce unique copies of a DL model for each user, as well as scale to cover a large number of users in the system. Also, it can link a remote copy of a DL model with its user with minimal effort and high reliability.

In our experiments, we distributed two unique copies of the FMNIST5-CNN model: one for *User1* and another for *User2*, each copy having its corresponding private model. We took a pre-trained FMNIST5-CNN model and fine-tuned it for 30 epochs to embed the WM linked to a specific user. To do so, we signed two copies of the WM carrier set, where each copy was signed using different joint signatures. Once we got two unique carrier sets, we trained two unique marked models, each one with its corresponding private model using Algorithm 1. In the end, we got two unique marked copies of the model with their corresponding private models and users. We distributed each copy to its corresponding user.

We then assumed that *User 1*, respectively *User 2*, leaked their model, and we tried to find the leaker as follows:

1. We took a small set (say 6 samples) of the WM carrier set.
2. We signed a copy of these samples in random places with *User1*'s joint signature; another copy was also signed in the same way for *User2*.
3. We queried $h^{'}$ (the allegedly leaked model) with the samples signed by *User1* to obtain their predictions. We did the same with *User2* samples.
4. We passed the predictions from samples signed with *User1*'s signature to her private model and calculated the accuracy at detecting the WM. We did the same with *User2*'s predictions and his private model.

Figure 9.4a shows the results of model owner detection if *User1* leaked her model. We see that we were able to determine that the model copy was most likely leaked by *User1*. Figure 9.4a1 shows the normalized confusion matrix of *User1*'s private model in detecting the WM information using the predictions of *User1*'s remote model. It shows that the accuracy at detecting signature positions was almost 100% when we sent the samples signed by *User1*. Figure 9.4a2 shows the normalized confusion matrix of *User2*'s private model in detecting the WM information by using the predictions of *User1*'s remote model when the samples were signed with *User2*' signature. Since *User1*'s model was trained to distinguish only *User1*'s signature position, it output features that led *User2*'s private model to provide label 0 for samples signed by *User2*.

Figure 9.4b provides similar results when *User2* leaked his model. The same conclusions hold. Note that the private models were unable to distinguish the signature positions and output the label 0 when they were fed with predictions of non-corresponding marked models and non-corresponding signatures. This is an interesting feature of *KeyNet*,

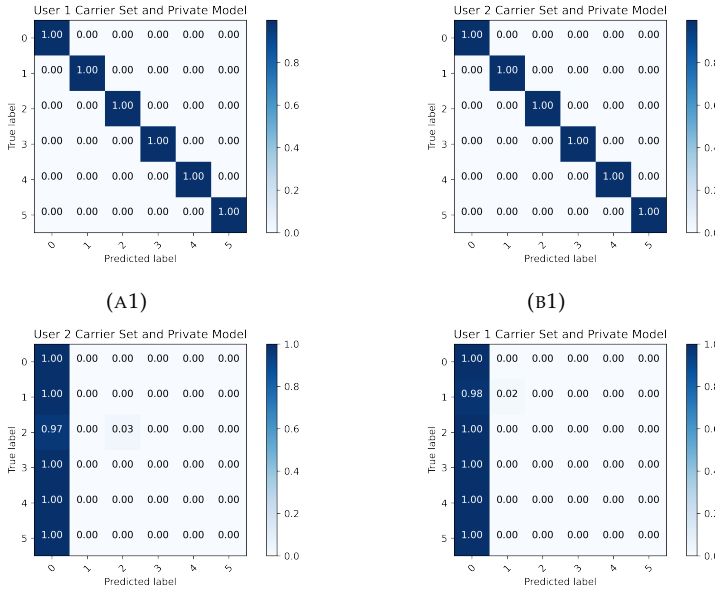
since all the remote models and their private models learned a common representation of label 0.

Regarding **scalability**, if we want to query a remote model in case we have u users and we decide to use m signed samples for verification, then the number of remote model queries will be $u \times m$, and hence linear in u.

Efficiency and generality. The efficiency of *KeyNet* is related to the size of the output of the model to be marked. The smaller the number of output neurons, the fewer the parameters of the private model. On the other hand, our framework allows embedding the WM from scratch or by fine-tuning; the latter contributes to efficiency. Regarding **generality**, even though in our work we use image classification tasks that output softmax layer probabilities (confidence) for the input image with each class, *KeyNet* can be extended to cover a variety of ML tasks that take images as input and output multiple values such as multi-labeling tasks, semantic segmentation tasks, image transformation tasks, etc.

9.6 Extending the proposed defense to FL

We have seen how KeyNet can effectively embed successful watermark in DL models in the centralized learning paradigm. In this section, we discuss how can we extend it to the federated learning, where several parties share the ownership of the trained global model. This can be accomplished by having the server select and sign the watermark carrier set using Function 1 of the KeyNet framework. Fortunately, the signature method used in Function 1 allows the server to include the information of all participating peers and obtain a joint signature that includes the information of all parties and the model they intend to train. After the WM carrier set is signed, the server randomly and uniformly distributes it to the peers. In addition to the original model, the server provides them with the private model combined with the original, as is



(A1) (B1)

(A2) (B2)

(A) With predictions of *User1*'s copy (B) With predictions of *User2*'s copy

FIGURE 9.4: Normalized confusion matrices of the accuracy in detecting the individual copies of the FMNIST5-CNN model distributed among two users. Figure 9.4a shows the detection accuracy of the predictions of *User1*' copy. Figure 9.4a1 shows the confusion matrix of *User1*'s private model when *User1*'s copy was queried by samples signed by *User1*. Figure 9.4a2 shows the confusion matrix of *User2*'s private model when *User1*'s copy was queried by samples signed by *User2*. Figure 9.4b shows the same results for *User2*'s model.

the case with KeyNet. Since all parties share the signature used, everyone can establish joint ownership of the trained global model without any of them being able to use that signature to claim sole ownership of

the public model.

9.7 Conclusions

We have presented *KeyNet*, a novel watermarking framework to protect the IP of DL models. We use the final output distribution of deep learning models to include a robust WM that does not fall in the same decision boundaries of original task classes. To make the most of this advantage, we design the watermarking task in an innovative way that makes it possible i) to embed a large amount of WM information, ii) to establish a strong link between the owner and her marked model, iii) to thwart the attacker from overwriting the WM information without losing accuracy in the original task, and iv) to uniquely fingerprint several copies of a pre-trained model for a large number of users in the system.

The results we obtained empirically prove that *KeyNet* is effective and can be generalized to various data sets and DL model architectures. Besides, it is robust against a variety of attacks, it offers a very strong authentication linking the owners and their WMs, and it can be easily used to fingerprint different copies of a DL model for different users.

Function 1: Signing a WM carrier Set

Input: Owner's information $infStr$; owner's signature size s; owner's WM carrier set D_{wm}; signature positions/labels set PL; other distributions' samples D_{dif}

Output: signed labeled WM samples D_{wm}^{signed}; signed samples from different distributions D_{dif}^{signed}; owner's signature $signature$

$signature \leftarrow hashAndReshape(infStr, s)$ //`The owner signature`
$fakeStr \leftarrow modify(infStr)$ //`Fake information string.`
$signature_{fake} \leftarrow hashAndReshape(fakeStr, s), signature_{fake} \neq signature$//`A fake signature.`
$D_{wm}^{signed}, D_{dif}^{signed} = [\,], [\,]$
for *each sample* c *in* D_{wm} **do**
  for *each position, label* (p_k, l_k) *in* PL **do**
    $c^{p_k} \leftarrow sign(signature, c, p_k)$
    $Add((c^{p_k}, l_k), D_{wm}^{signed})$
  end
  $p_k \leftarrow selectRandomPosition()$
  $c^{p_k} \leftarrow sign(signature_{fake}, c, p_k)$
  $Add((c^{p_k}, 0), D_{wm}^{signed})$
  $Add((c, 0), D_{wm}^{signed})$
  $fakeStr \leftarrow modify(infStr)$ //`New fake information string.`
  $signature_{fake} \leftarrow hashAndReshape(fakeStr, s), signature_{fake} \neq signature$
end
for *each sample* d *in* D_{dif} **do**
  for *each position, label* (p_k, l_k) *in* PL **do**
    $d^{p_k} \leftarrow sign(signature, d, p_k)$
    $Add((d^{p_k}, 0), D_{dif}^{signed})$
  end
end
return $D_{wm}^{signed}, D_{dif}^{signed}, signature$

Algorithm 1: Watermark Embedding

Input: Unmarked DL model h; private model f; original data set D_{org}; signed WM carrier set D_{wm}^{signed}; signed samples from other distributions D_{dif}^{signed}; batch size BS; weighted loss parameter α

Output: Marked model h^*, corresponding private model f

$s = size(D_{wm}^{signed})/5$
$D_2 \leftarrow randomSample(D_2, s)$
$D_2 \leftarrow D_2 \cup D_{dif}^{signed}$
$D \leftarrow D_{org} \cup D_2$
$L_f = crossEntropy()$ `//Loss function`
for *each batch b of size BS in D* **do**
    $\{x, y\}, \{c, l\} \leftarrow split(b)$, with $\{x, y\} \in D_{org}$ and $\{c, l\} \in D_2$
    $L \leftarrow \alpha L_f(h(x), y) + (1 - \alpha) L_f(f(h(c)), l)$
    $optimize(L)$
end
return h^*, f

Protocol 1: Watermark Verification

Input: Remote access to $h^{'}$, threshold T, number of rounds r
Output: Boolean decision d (True or False) on $h^{'}$'s ownership

$correct = 0$
$d = False$ //`Decision on the ownership of` $h^{'}$.
for *each round* $i = 1, 2, \ldots r$ **do**
  $c \leftarrow randomSample(D_{wm})$
  $p_k \leftarrow$ randomPosition(), k $\in 1, 2, \ldots z$
  $c^{p_k} \leftarrow sign(signature, c, p^k)$
  $predictions \leftarrow h^{'}(c^{p_k})$
  $l_k \leftarrow f(predictions)$
  $answer \leftarrow$ Position corresponding to l_k
  if $answer = p_k$ **then**
    $correct \leftarrow correct + 1$
end
$acc = correct/r$
if $acc \geq T$ **then**
  $d \leftarrow True$
return d

TABLE 9.1: Data sets and deep learning model architectures. $C(3, 32, 5, 1, 2)$ denotes a convolutional layer with 3 input channels, 32 output channels, a kernel of size 5×5, a stride of 1, and a padding of 2, $MP(2, 1)$ denotes a max-pooling layer with a kernel of size 2×2 and a stride of 1, and $FC(10, 20)$ indicates a fully connected layer with 10 inputs and 20 output neurons. We used *ReLU* as an activation function in the hidden layers. We used *LogSoftmax* as an activation function in the output layers for all DL models. The rightmost column contains the architecture of the corresponding private models.

Data set	WM carrier set	DL model	DL model architecture	Private model architecture
CIFAR10	STL10	ResNet18	See He et al., 2016.	FC(10,20),FC(20,10), FC(10, 6) (496 learnable parameters)
		VGG16	See Simonyan and Zisserman, 2014.	
FMNIST5	MNIST	CNN	C(3,32,5,1,2), MP(2,1),C(32,64,3,1,2), MP(2,1), FC(4096,4096),FC(4096,5)	FC(5, 10), FC(10,20), FC(20,6) (411 learnable parameters)
		LeNet	See LeCun et al., 2015.	

TABLE 9.2: Attacker's WM carrier sets and private models. The attacker's WM carrier set and private model differ from the owner's.

Data set	Owner's WM carrier set	Attacker's WM carrier set	Attacker's private model architecture
CIFAR10	STL10	Fashion-MNIST	FC(10,20),FC(20,30), FC(30, 6) (980 learnable parameters)
FMNIST5	MNIST	STL10	FC(5, 20), FC(20,10), FC(10,6) (360 learnable parameters)

TABLE 9.3: Accuracy of the private models at detecting the position of the owner's signature in the WM carrier set when trained for 250 epochs with the predictions of black-box models.

Data set	Black-box DL model	Watermark detection accuracy %
CIFAR10	ResNet18	31.25
	VGG16	30.14
FMNIST5	CNN	34.42
	LeNet	33.26

TABLE 9.4: Fidelity results. Column 3 shows the accuracy of the unmarked models in the original tasks (baseline accuracy) before embedding the WM. Columns (4, 5) show the accuracy of the marked model in the original task after embedding WM by fine-tuning a pre-trained model or by training the combined model from scratch. Columns (6, 7) show the accuracy of the private model in detecting the WM using the predictions of the corresponding marked model. To embed the WM in a pre-trained model, we fine-tuned it for 30 epochs while we trained models from scratch for 250 epochs.

Data set	DL model	Unmarked model Accuracy %	Marked model Accuracy %		Watermark detection Accuracy %	
			By Fine-tuning (30 epochs)	From Scratch (250 epochs)	By Fine-tuning	From Scratch
CIFAR10	ResNet18	91.96	92.07	92.53	99.96	99.97
	VGG16	90.59	90.52	91.74	99.68	99.89
FMNIST5	CNN	92.08	92.42	92.32	99.98	99.90
	LeNet	90.68	89.94	89.94	99.55	99.79

TABLE 9.5: Fine-tuning results. In the fine-tuning attack, the marked models were retrained based on the original task loss only.

Data set	**CIFAR10**						**FMNIST5**					
DL model	ResNet18			VGG16			CNN			LeNet		
Number of epochs	50	100	200	50	100	200	50	100	200	50	100	200
Marked model Accuracy %	92.40	92.33	92.47	91.31	91.64	91.69	92.30	92.52	92.40	89.84	90.06	90.12
Watermark detection Accuracy %	98.19	98.05	99.12	97.20	94.72	96.67	97.35	97.02	96.92	98.23	97.42	96.4

TABLE 9.6: Overwriting attack results with CIFAR10 marked models. The table shows the accuracy before and after overwriting each marked model and its corresponding private model depending on the fraction of training data known by the attacker (from 1% to 30%).

Data set	**CIFAR10**											
DL model	**ResNet18**						**VGG16**					
Data fraction %	1	3	6	10	20	30	1	3	6	10	20	30
Marked model Accuracy before %	92.53	92.53	92.53	92.53	92.53	92.53	91.74	91.74	91.74	91.74	91.74	91.74
Marked model Accuracy after %	39.05	63.75	83.31	86.35	89.91	90.9	34.61	71.86	81.2	83.9	88.07	89.67
Owner's WM detection Accuracy after %	24.53	20.35	22.27	28.3	37.33	41.15	32.32	30.88	28.61	32.48	30.32	44.4
Attacker's WM detection Accuracy %	99.97	99.89	99.95	99.9	99.94	99.97	99.69	99.68	99.89	99.87	99.9	99.96

TABLE 9.7: Overwriting attack results with FMNIST5 marked models. The table shows the accuracy before and after overwriting each marked model and its corresponding private model depending on the fraction of training data known by the attacker (from 1% to 30%).

Data set	**FMNIST5**											
DL model	**CNN**						**LeNet**					
Data fraction %	1	3	6	10	20	30	1	3	6	10	20	30
Marked model Accuracy before %	92.32	92.32	92.32	92.32	92.32	92.32	89.94	89.94	89.94	89.94	89.94	89.94
Marked model Accuracy after %	76.18	81.88	87.86	89.1	91.38	91.84	74.64	79.44	86.36	88.86	89.58	89.52
Owner's WM detection Accuracy after %	26.97	28.53	38.52	46.83	70	68.03	26.32	18.71	43.99	57.4	75.83	65 .66
Attacker's WM detection Accuracy %	100	100	100	100	100	100	99.25	99.65	99.84	99.92	99.89	99.92

TABLE 9.8: Integrity results with unmarked models. Each private model was tested with two different unmarked models: one model has the same topology as its corresponding marked model, the other one has a different topology. The last four columns show the accuracy detection obtained with the unmarked models.

Data set	**DL model**	**Watermark detection accuracy with marked models%**	**Watermark detection accuracy with unmarked models %**			
			Same topology	Accuracy	Different topology	accuracy
CIFAR10	**ResNet18**	99.97%	ResNet18	18.92%	VGG16	19.80%
CIFAR10	**VGG16**	99.89%	VGG16	7.92%	ResNet18	12.32%
FMNIST5	**CNN**	99.98%	CNN	12.96%	LeNet	10.97%
FMNIST5	**LeNet**	99.55%	LeNet	17.93%	CNN	17.75%

Chapter 10

Conclusions and future work

In this thesis, we have found that defenses against poisoning attacks impose high computational overhead on the server, adopt unrealistic assumptions on the peers' data distribution, or are ill-suited for high-dimensional DL models. Based on that, we have proposed three defenses against poisoning attacks to overcome the limitations of existing defenses and make FL more secure. Our proposed defenses against poisoning attacks include a method that analyzes the biases of the last layer to neutralize Byzantine poisoning attacks efficiently; LFighter, which dynamically extracts and clusters the relevant gradients of the label-flipping (LF) attack from the last layer to counter the attack; and FL-Defender, which extracts squeezed discriminative features from the peers' last-layer gradients and uses them to mitigate the impact of the targeted poisoning attacks. On the other hand, we have found that existing defenses against privacy attacks have different trade-offs between accuracy, privacy, security, and efficiency. Motivated by that, we have proposed fragmented federated learning (FFL), which addresses the security-privacy-accuracy conflict by extending cooperation between peers in FL systems to preserve their privacy without renouncing robust, efficient, and accurate aggregation of their updates to the global model.

We also identified two security vulnerabilities shared by federated and centralized learning, which are backdoor and model stealing attacks. With that motivation, we have developed two robust defense mechanisms to thwart these attacks within the centralized learning paradigm, and have demonstrated how they can be readily adapted to the FL paradigm.

We have conducted extensive experiments on real data sets, which demonstrate the effectiveness of the proposed defenses at making ML more secure and private.

Specifically, our contributions are the following:

- In Chapter 4, we have proposed a new method to detect Byzantine attacks in FL that is both efficient and low-cost on the server side. The method focuses on analyzing only the biases of the last layer of deep learning models, which enables the detection and elimination of poisoned updates at each training round. The effectiveness of this method has been demonstrated through tests on two different data sets and different DL architectures, and the results have been compared with three state-of-the-art methods. This comparison shows that our approach achieves similar attack detection performance as the best baseline method, multi-Krum, while significantly reducing the runtime required for update verification and aggregation at each training round.

- In Chapter 5, we have presented a novel defense against the label-flipping (LF) attack, called *LFighter*, which is effective regardless of the peers' data distribution or model dimensionality. First, we have conducted in-depth analyses of the attack behavior and found that the gradients connected to the source and target class neurons in the output layer are good discriminative features for attack detection. These features stay robust under different data distributions and model sizes. Based on that, the LFighter we propose dynamically extracts the potential source and target classes' gradients from the peers' local updates, clusters those gradients, and

analyzes the resulting clusters to filter out potential poisoned updates before model aggregation. We have demonstrated the effectiveness of LFighter against the LF attack through extensive experiments on three data sets with different deep learning model sizes, peers' local data distributions, and ratios of attackers. Additionally, we have compared our approach with several state-of-the-art defenses. We have shown its superiority at simultaneously delivering low test error, high overall accuracy, high source class accuracy, low attack success rate, and stability of the source class accuracy.

- In Chapter 6, we have analyzed the behavior of label-flipping (LF) and backdoor attacks, and observed that the last-layer gradients can be used to extract useful features for detecting targeted attacks. Then, we have designed *FL-Defender*, which leverages those gradients to mitigate targeted attacks against FL regardless of model dimensionality or the distribution of the peers' data. On the one hand, FL-Defender engineers more robust discriminative features by computing the peer-wise angle similarity for the peers' last-layer gradients and compressing the computed similarity vectors using PCA to reduce redundant information. On the other hand, FL-Defender penalizes peers' updates at the model aggregation stage based on their angular deviation from the centroid of the compressed similarity vectors. Experimental results on three data sets with different DL model sizes and peer data distributions have demonstrated the effectiveness of FL-Defender at defending against targeted attacks. FL-Defender outperforms several state-of-the-art defenses in retaining the accuracy of the global model on the main task, reducing the attack success rate, and causing minimal computational overhead on the server.

- In Chapter 7, we have proposed the fragmented federated learning (*FFL*) framework to solve the security-privacy-accuracy conflict in

FL. FFL introduces a lightweight protocol that enables peers to privately exchange and mix random fragments of their updates. This makes it difficult for the server to link the updates to their originators or recover the complete original updates, thereby preventing the semi-honest server from conducting successful privacy attacks. The protocol also allows the server to aggregate the mixed updates into a global model correctly. To defend against poisoning attacks, a novel reputation-based defense mechanism is being integrated into the FFL design that uses the quality of the updates and fragments exchanged to assign global and local reputations to peers. The server then selects peers for training and adaptively aggregates their mixed updates based on their global reputations, whereas honest peers are incentivized not to exchange fragments with peers having low local reputations. Experiments on four real data sets have demonstrated that FFL can effectively counter privacy and security attacks while maintaining the global model's accuracy. All of the above is achieved while imposing affordable communication cost and computation overhead on the participating parties, making FFL applicable to large-scale FL systems.

- In Chapter 8, we have presented a novel approach for detecting backdoor attacks in DNNs at inference time. Specifically, we have proposed a mechanism to identify the critical layer within the DNN where the distinction between malicious and benign samples is most apparent. We can then apply a filtering mechanism by using feature differences at that identified layer to screen any questionable samples that enter the system. This innovative method allows for real-time detection of poisoned samples in DNNs, which can be applied during inference to the models trained in centralized learning and to the final global models trained through federated learning.

- In Chapter 9, we have introduced a novel digital watermarking

framework known as *KeyNet*, which satisfies the majority of the criteria necessary for an effective DNN watermarking framework. The framework consists of three main components: a watermark (WM) carrier set, owner information, and a marked model with its corresponding private model, which are utilized to embed and verify WM information. Additionally, KeyNet allows for the creation of unique fingerprints for various copies of a DL model, intended for different users within the system. Furthermore, it is possible to extend the framework to the FL paradigm, and offer a useful means to protect the intellectual property of the owners of DL models in centralized and federated learning.

Overall, the FL-related proposed methods present promising solutions for enhancing the security and privacy of FL, which could help its successful adoption in real-world applications. Also, the two defenses against backdoor and model stealing attacks can improve the security of both FL and CL.

10.1 Publications

Next, we enumerate the publications that back the contents of this thesis, ordered as outlined above:

1. Najeeb Jebreel, Alberto Blanco-Justicia, David Sánchez and Josep Domingo-Ferrer. "Efficient detection of Byzantine attacks in federated learning using last layer biases." In *Modeling Decisions for Artificial Intelligence: 17th International Conference, MDAI 2020*, Sant Cugat, Catalonia, September 2–4, 2020, pp. 154-165. Springer International Publishing, 2020. **CORE ranking: B.**

2. Najeeb Moharram Jebreel, Josep Domingo-Ferrer, David Sánchez and Alberto Blanco-Justicia. "Defending against the label-flipping attack in federated learning." **Submitted to a 1st-decile journal.**

3. Najeeb Moharram Jebreel and Josep Domingo-Ferrer. "FL-Defender: Combating targeted attacks in federated learning." *Knowledge-Based Systems* 260 (2023): 110178. **Impact Factor: 8.139 (1st decile).**

4. Najeeb Moharram Jebreel, Josep Domingo-Ferrer, Alberto Blanco-Justicia and David Sánchez. "Enhanced security and privacy via fragmented federated learning." *IEEE Transactions on Neural Networks and Learning Systems* (2022). **Impact Factor: 14.255 (1st decile).**

5. Najeeb Moharram Jebreel, Josep Domingo-Ferrer and Yiming Li. "Defending Against Backdoor Attacks by Layer-wise Feature Analysis." Accepted to *The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2023)*. **CORE ranking: A. Best Paper Award** (**143** papers were accepted out of **822** submissions).

6. Najeeb Moharram Jebreel, Josep Domingo-Ferrer, David Sánchez and Alberto Blanco-Justicia. "Keynet: An asymmetric key-style framework for watermarking deep learning models." *Applied Sciences* 11(3) (2021): 999. **Impact Factor: 2.838 (2nd quartile).**

10.2 Future work

There are several potential future work directions that can be explored:

1. We have evaluated our method to detect Byzantine attacks by using the biases of the last layer against the random attacks based on Gaussian noise under the IID setting. To fully evaluate its effectiveness, we plan to extend our analysis to other types of Byzantine attacks, such as the multi-label flipping attack (Fang et al., 2020). Also, we plan to evaluate its effectiveness under various non-IID settings.

2. We plan to further investigate the performance of LFighter with other DNN architectures, such as transformers, and data sets with

higher-resolution images and a larger number of classes, such as Imagenet. Additionally, we intend to explore the defense's ability to withstand different attack strategies. For instance, we will investigate its efficacy against a scenario where some attackers flip the label from c_i to c_j, and others flip the label from c_k to c_l. We aim to provide insights into the defense mechanism's versatility and applicability to a broad range of scenarios.

3. Combining multiple attack features could provide a more robust defense against poisoning attacks. For example, one could combine the features obtained by LFighter with those obtained by FL-Defender to improve targeted attack detection. Those features could also be combined with extracted features from internal layers to enable a comprehensive defense against untargeted and targeted poisoning attacks. However, this integration will need to be evaluated through experiments comparing the performance of different defense combinations. Besides, it will increase the computation overhead on the server. The latter consequence motivates developing an efficient combination of such features.

4. While FFL provides a more efficient solution for private and secure FL compared to several existing works, there is still room for making it more efficient by reducing the number of exchanged parameters. To accomplish this, we propose to study the sensitivity of each layer in the model to privacy leakage and to perform the exchange protocol on only the most sensitive layers. For example, in CNNs, earlier layers extract common features among all classes, which are less sensitive to privacy than later layers that extract more class-specific features. By selectively exchanging only the sensitive layers, we can reduce communication and computation costs and improve the overall efficiency of FFL. We also plan to investigate the performance of FFL when working with non-IID

data and different DNN architectures to gain insights into the robustness and generalizability of FFL.

5. When facing backdoor attacks with highly non-IID data distributions and large model sizes, FL-Defender and the other state-of-the-art defenses perform poorly against the attack. This shows that defending against BAs during training under such a scenario is still challenging. To address this challenge, we can think on integrating our proposed defense against BAs at inference time with both LFighter and FL-Defender. This approach is expected to enhance the overall robustness of FL against targeted attacks, including both LF and BA. However, it is important to note that the effectiveness of the proposed defense strategy will need to be thoroughly tested and evaluated in different scenarios before it can be widely adopted.

6. We also plan to conduct further comprehensive experiments to validate the effectiveness of our last two defense mechanisms against backdoor and model stealing attacks in FL.

7. We plan to explore another threat model for DL model stealing, which involves stealing a remote black-box model accessed via an API through its predictions. To counter this threat, we plan to investigate an effective way to watermark the created surrogate models on the adversary side. This would involve embedding a unique and identifiable watermark into the surrogate model, which could then be used to detect and verify the presence of the watermark in the model. By doing so, owners of trained DL models will have an additional means to protect their IP besides KeyNet.

8. Although there has been extensive research on protecting against poisoning and privacy attacks in the typical horizontal FL framework, relatively few attacks and defense mechanisms have been

proposed for the vertical FL or the federated transfer learning. As interest in these frameworks continues to grow, it becomes increasingly crucial to investigate their vulnerabilities to privacy and poisoning attacks. This line of research has the potential to yield novel attack models and defense strategies that can be tailored to the unique characteristics of vertical FL and federated transfer learning. By conducting further studies on the security of these frameworks, we can ensure that they are well-protected against potential threats and continue to be valuable tools in the field of machine learning.

9. This thesis focused on fully-supervised federated learning where all local data are labeled. In the future, we plan to investigate the vulnerabilities of semi-supervised federated learning (SSFL) to poisoning and privacy attacks and then propose effective defenses against those attacks. In SSFL, peers train the shared global model using their local data, which consists of labeled and unlabeled examples. The global model benefits from the additional information the unlabeled data provides, resulting in improved performance and reduced reliance on large amounts of labeled data. SSFL thus offers a practical approach for situations where data privacy is crucial and labeled data is limited.

10. In this thesis, the focus was on classification tasks, which are widely used in machine learning applications. In the future, we plan to explore other ML tasks, including regression, object detection, semantic segmentation, and machine translation.

Bibliography

Adi, Yossi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet (2018). "Turning your weakness into a strength: Watermarking deep neural networks by backdooring". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1615–1631.

Adiwardana, Daniel, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al. (2020). "Towards a human-like open-domain chatbot". In: *arXiv preprint arXiv:2001.09977*.

Aiken, William, Hyoungshick Kim, and Simon Woo (2020). "Neural Network Laundering: Removing Black-Box Backdoor Watermarks from Deep Neural Networks". In: *arXiv preprint arXiv:2004.11368*.

Aono, Yoshinori, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. (2017). "Privacy-preserving deep learning via additively homomorphic encryption". In: *IEEE Transactions on Information Forensics and Security* 13.5, pp. 1333–1345.

Awan, Sana, Bo Luo, and Fengjun Li (2021). "CONTRA: Defending against Poisoning Attacks in Federated Learning". In: *In European Symposium on Research in Computer Security (ESORICS)*. Springer, pp. 455–475.

Bagdasaryan, Eugene, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov (2020). "How to backdoor federated learning". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2938–2948.

Barker, Elaine, Elaine Barker, William Burr, William Polk, Miles Smid, et al. (2006). *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration.

Bhagoji, Arjun Nitin, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo (2019). "Analyzing federated learning through an adversarial lens". In: *International Conference on Machine Learning*. PMLR, pp. 634–643.

Bhowmick, Abhishek, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers (2018). "Protection against reconstruction and its applications in private federated learning". In: *arXiv preprint arXiv:1812.00984*.

Biggio, Battista, Blaine Nelson, and Pavel Laskov (2012). "Poisoning attacks against support vector machines". In: *arXiv preprint arXiv:1206.6389*.

Blanchard, Peva, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer (2017). "Machine learning with adversaries: Byzantine tolerant gradient descent". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 118–128.

Blanco-Justicia, Alberto, Josep Domingo-Ferrer, Sergio Martínez, David Sánchez, Adrian Flanagan, and Kuan Eeik Tan (2021). "Achieving security and privacy in federated learning systems: Survey, research challenges and future directions". In: *Engineering Applications of Artificial Intelligence* 106, p. 104468.

Blanco-Justicia, Alberto, David Sánchez, Josep Domingo-Ferrer, and Krishnamurty Muralidhar (2023). "A critical review on the use (and misuse) of differential privacy in machine learning". In: *ACM Computing Surveys* 58.8, 160:1–160:16.

Boenisch, Franziska (2020). "A Survey on Model Watermarking Neural Networks". In: *arXiv preprint arXiv:2009.12153*.

Bonawitz, Keith, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. (2019). "Towards federated learning at scale: System design". In: *arXiv preprint arXiv:1902.01046*.

Bonawitz, Keith, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth (2017). "Practical secure aggregation for privacy-preserving machine learning". In: *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191.

Brisimi, Theodora S, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi (2018). "Federated learning of predictive models from federated electronic health records". In: *International journal of medical informatics* 112, pp. 59–67.

Campello, Ricardo JGB, Davoud Moulavi, and Jörg Sander (2013). "Density-based clustering based on hierarchical density estimates". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer, pp. 160–172.

Cao, Xiaoyu, Jinyuan Jia, and Neil Zhenqiang Gong (2019). "IPGuard: Protecting the Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary". In: *arXiv preprint arXiv:1910.12903*.

Caruana, Rich (1997). "Multitask learning". In: *Machine learning* 28.1, pp. 41–75.

Chang, Hongyan, Virat Shejwalkar, Reza Shokri, and Amir Houmansadr (2019). "Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer". In: *arXiv preprint arXiv:1912.11279*.

Chen, Bryant, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava (2019a). "Detecting backdoor attacks on deep neural networks by activation clustering". In: *AAAI Workshop*.

Chen, Huili, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar (2019b). "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models". In: *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pp. 105–113.

Chen, Huili, Bita Darvish Rouhani, and Farinaz Koushanfar (2019). "BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks". In: *arXiv preprint arXiv:1904.00344*.

Chen, Xinyun, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song (2017). "Targeted backdoor attacks on deep learning systems using data poisoning". In: *arXiv preprint arXiv:1712.05526*.

Chen, Yu, Fang Luo, Tong Li, Tao Xiang, Zheli Liu, and Jin Li (2020). "A training-integrity privacy-preserving federated learning scheme with trusted execution environment". In: *Information Sciences* 522, pp. 69–79.

Chen, Yudong, Lili Su, and Jiaming Xu (2017). "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.2, pp. 1–25.

Coates, Adam, Andrew Ng, and Honglak Lee (2011). "An Analysis of Single Layer Networks in Unsupervised Feature Learning". In: *AISTATS*. https://cs.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf.

Cohen, Jeremy, Elan Rosenfeld, and Zico Kolter (2019). "Certified adversarial robustness via randomized smoothing". In: *ICML*.

Damaskinos, Georgios, El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, and Sébastien Rouault (2019). "Aggregathor: Byzantine machine learning via robust gradient aggregation". In: *Proceedings of Machine Learning and Systems* 1, pp. 81–106.

Dargan, Shaveta, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar (2019). "A survey of deep learning and its applications: A new paradigm to machine learning". In: *Archives of Computational Methods in Engineering*, pp. 1–22.

Davenport, Corbin and Corbin Davenport (2018). *Gboard passes one billion installs on the play store*. Accessed: 2022-04-2. URL: https://www.androidpolice.com/2018/08/22/gboard-passes-one-billion-installs-play-store/.

Dayan, Ittai, Holger R Roth, Aoxiao Zhong, Ahmed Harouni, Amilcare Gentili, Anas Z Abidin, Andrew Liu, Anthony Beardsworth Costa, Bradford J Wood, Chien-Sung Tsai, et al. (2021). "Federated learning for predicting clinical outcomes in patients with COVID-19". In: *Nature medicine* 27.10, pp. 1735–1743.

Deng, Li and Dong Yu (2014). "Deep learning: methods and applications". In: *Foundations and Trends in Signal Processing* 7.3–4, pp. 197–387.

Denil, Misha, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas (2013). "Predicting parameters in deep learning". In: *arXiv preprint arXiv:1306.0543*.

Diffie, Whitfield and Martin Hellman (1976). "New directions in cryptography". In: *IEEE transactions on Information Theory* 22.6, pp. 644–654.

Domingo-Ferrer, Josep, Alberto Blanco-Justicia, Jesús Manjón, and David Sánchez (2022). "Secure and Privacy-Preserving Federated Learning via Co-Utility". In: *IEEE Internet of Things Journal* 9.5, pp. 3988–4000.

Domingo-Ferrer, Josep, Alberto Blanco-Justicia, David Sánchez, and Najeeb Jebreel (2020). "Co-utile peer-to-peer decentralized computing". In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, pp. 31–40.

Domingo-Ferrer, Josep, David Sánchez, and Alberto Blanco-Justicia (2021). "The limits of differential privacy (and its misuse in data release and machine learning)". In: *Communications of the ACM* 64.7, pp. 34–36.

Dwork, Cynthia, Aaron Roth, et al. (2014). "The algorithmic foundations of differential privacy." In: *Foundations and Trends in Theoretical Computer Science* 9.3-4, pp. 211–407.

European Commission (2016). *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data*

Protection Regulation) (Text with EEA relevance). URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

Fan, Lixin, Kam Woh Ng, and Chee Seng Chan (2019). "Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks". In: *Advances in Neural Information Processing Systems*, pp. 4714–4723.

Fang, Minghong, Xiaoyu Cao, Jinyuan Jia, and Neil Gong (2020). "Local model poisoning attacks to Byzantine-robust federated learning". In: *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 1605–1622.

Fung, Clement, Chris JM Yoon, and Ivan Beschastnikh (2018). "Mitigating sybils in federated learning poisoning". In: *arXiv preprint arXiv:1808.04866*.

Fung, Clement, Chris JM Yoon, and Ivan Beschastnikh (2020). "The Limitations of Federated Learning in Sybil Settings". In: *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, pp. 301–316.

Furht, Borko and Darko Kirovski (2004). *Multimedia security handbook*. CRC press.

Ganju, Karan, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov (2018). "Property inference attacks on fully connected neural networks using permutation invariant representations". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–633.

Gao, Yansong, Yeonjae Kim, Bao Gia Doan, Zhi Zhang, Gongxuan Zhang, Surya Nepal, Damith C. Ranasinghe, and Hyoungshick Kim (2022). "Design and Evaluation of a Multi-Domain Trojan Detection Method on Deep Neural Networks". In: *IEEE Transactions on Dependable and Secure Computing* 19.4, pp. 2349–2364.

Geiping, Jonas, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller (2020). "Inverting Gradients–How easy is it to break privacy in federated learning?" In: *arXiv preprint arXiv:2003.14053*.

Gentry, Craig (2009). "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178.

Geyer, Robin C, Tassilo Klein, and Moin Nabi (2017). "Differentially private federated learning: A client level perspective". In: *arXiv preprint arXiv:1712.07557*.

Gu, Tianyu, Brendan Dolan-Gavitt, and Siddharth Garg (2017). "BadNets: Identifying vulnerabilities in the machine learning model supply chain". In: *arXiv preprint arXiv:1708.06733*.

Gu, Tianyu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg (2019). "BadNets: Evaluating backdooring attacks on deep neural networks". In: *IEEE Access* 7, pp. 47230–47244.

Guo, Jia and Miodrag Potkonjak (2018). "Watermarking deep neural networks for embedded systems". In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, pp. 1–8.

Han, Song, Huizi Mao, and William J Dally (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149*.

Han, Song, Jeff Pool, John Tran, and William Dally (2015). "Learning both weights and connections for efficient neural network". In: *Advances in Neural Information Processing Systems*, pp. 1135–1143.

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

Hardy, Stephen, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne (2017). "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption". In: *arXiv preprint arXiv:1711.10677*.

Hartigan, John A and Manchek A Wong (1979). "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the royal statistical society. series c (applied statistics)* 28.1, pp. 100–108.

Hartung, Frank and Martin Kutter (1999). "Multimedia watermarking techniques". In: *Proceedings of the IEEE* 87.7, pp. 1079–1107.

Hayase, Jonathan and Weihao Kong (2021). "Spectre: Defending against backdoor attacks using robust covariance estimation". In: *ICML*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hitaj, Briland, Giuseppe Ateniese, and Fernando Perez-Cruz (2017). "Deep models under the GAN: information leakage from collaborative deep learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 603–618.

Hitaj, Dorjan and Luigi V Mancini (2018). "Have you stolen my model? evasion attacks against deep neural network watermarking techniques". In: *arXiv preprint arXiv:1809.00615*.

Huang, Jiwei, Zeyu Tong, and Zihan Feng (2022). "Geographical POI recommendation for Internet of Things: A federated learning approach using matrix factorization". In: *International Journal of Communication Systems*, e5161.

Jagielski, Matthew, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li (2018). "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 19–35.

Jebreel, Najeeb, Alberto Blanco-Justicia, David Sánchez, and Josep Domingo-Ferrer (2020). "Efficient Detection of Byzantine Attacks in Federated Learning Using Last Layer Biases". In: *International Conference on Modeling Decisions for Artificial Intelligence*. Springer, pp. 154–165.

Jebreel, Najeeb Moharram and Josep Domingo-Ferrer (2023). "FL-Defender: Combating targeted attacks in federated learning". In: *Knowledge-Based Systems* 260, p. 110178.

Jebreel, Najeeb Moharram, Josep Domingo-Ferrer, Alberto Blanco-Justicia, and David Sánchez (2022a). "Enhanced Security and Privacy via Fragmented Federated Learning". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15. DOI: 10.1109/TNNLS.2022.3212627.

Jebreel, Najeeb Moharram, Josep Domingo-Ferrer, David Sánchez, and Alberto Blanco-Justicia (2022b). "Defending against the Label-flipping Attack in Federated Learning". In: *arXiv preprint arXiv:2207.01982*.

Kairouz, Peter, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. (2019). "Advances and open problems in federated learning". In: *arXiv preprint arXiv:1912.04977*.

Karimireddy, Sai Praneeth, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh (2020). "Scaffold: Stochastic controlled averaging for federated learning". In: *International Conference on Machine Learning*. PMLR, pp. 5132–5143.

Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang (2016). "On large-batch training for deep learning: Generalization gap and sharp minima". In: *arXiv preprint arXiv:1609.04836*.

Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. (2017). "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13, pp. 3521–3526.

Kohavi, Ron et al. (1996). "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid." In: *Kdd*. Vol. 96, pp. 202–207.

Krizhevsky, Alex (2009). *Learning multiple layers of features from tiny images*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2017). "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6, pp. 84–90.

Lamport, Leslie, Robert Shostak, and Marshall Pease (2019). "The Byzantine generals problem". In: *Concurrency: the works of leslie lamport*, pp. 203–226.

Le Merrer, Erwan, Patrick Perez, and Gilles Trédan (2020). "Adversarial frontier stitching for remote neural network watermarking". In: *Neural Computing and Applications* 32.13, pp. 9233–9244.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/.

LeCun, Yann, Patrick Haffner, Léon Bottou, and Yoshua Bengio (1999). "Object recognition with gradient-based learning". In: *Shape, contour and grouping in computer vision*. Springer, pp. 319–345.

LeCun, Yann et al. (2015). "LeNet-5, convolutional neural networks". In: *URL: http://yann. lecun. com/exdb/lenet* 20.5, p. 14.

Lepinski, Matt and Stephen Kent (Jan. 2008). *Additional Diffie-Hellman Groups for Use with IETF Standards*. DOI: 10.17487/RFC5114. URL: https://rfc-editor.org/rfc/rfc5114.txt.

Leshno, Moshe, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken (1993). "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural networks* 6.6, pp. 861–867.

Li, Huiying, Emily Wenger, Ben Y Zhao, and Haitao Zheng (2019a). "Piracy Resistant Watermarks for Deep Neural Networks". In: *arXiv preprint arXiv:1910.01226*.

Li, Liping, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling (2019b). "RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 1544–1551.

Li, Shenghui, Edith Ngai, Fanghua Ye, and Thiemo Voigt (2021a). "Auto-weighted Robust Federated Learning with Corrupted Data Sources". In: *arXiv preprint arXiv:2101.05880*.

Li, Sijin, Zhi-Qiang Liu, and Antoni B Chan (2014). "Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 482–489.

Li, Tian, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith (2020). "Federated learning: Challenges, methods, and future directions". In: *IEEE Signal Processing Magazine* 37.3, pp. 50–60.

Li, Tian, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith (2018). "Federated optimization in heterogeneous networks". In: *arXiv preprint arXiv:1812.06127*.

Li, Yige, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma (2021b). "Neural attention distillation: Erasing backdoor triggers from deep neural networks". In: *ICLR*.

Li, Yiming, Mengxi Ya, Yang Bai, Yong Jiang, and Shu-Tao Xia (2022). "BackdoorBox: A Python Toolbox for Backdoor Learning". In: URL: https://github.com/THUYimingLi/BackdoorBox.

Li, Yiming, Mengxi Ya, Yang Bai, Yong Jiang, and Shu-Tao Xia (2023). "BackdoorBox: A python toolbox for backdoor learning". In: *arXiv preprint arXiv:2302.01762*.

Li, Yiming, Tongqing Zhai, Yong Jiang, Zhifeng Li, and Shu-Tao Xia (2021c). "Backdoor attack in the physical world". In: *ICLR Workshop*.

Li, Yuezun, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu (2021d). "Invisible backdoor attack with sample-specific triggers". In: *ICCV*.

Li, Zheng, Chengyu Hu, Yang Zhang, and Shanqing Guo (2019c). "How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN". In: *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 126–137.

Liu, Kang, Brendan Dolan-Gavitt, and Siddharth Garg (2018). "Fine-pruning: Defending against backdooring attacks on deep neural networks". In: *RAID*.

Liu, Xiaoyuan, Hongwei Li, Guowen Xu, Zongqi Chen, Xiaoming Huang, and Rongxing Lu (2021). "Privacy-Enhanced Federated Learning against Poisoning Adversaries". In: *IEEE Transactions on Information Forensics and Security* 16, pp. 4574–4588. DOI: 10.1109/TIFS.2021.3108434.

Liu, Yuntao, Yang Xie, and Ankur Srivastava (2017). "Neural Trojans". In: *ICCD*.

Lopuhaa, Hendrik P (1989). "On the relation between S-estimators and M-estimators of multivariate location and covariance". In: *The Annals of Statistics*, pp. 1662–1683.

Lu, Chun-Shien (2004). *Multimedia security: steganography and digital watermarking techniques for protection of intellectual property: steganography and digital watermarking techniques for protection of intellectual property*. IGI Global.

Lyu, Lingjuan, Han Yu, Xingjun Ma, Chen Chen, Lichao Sun, Jun Zhao, Qiang Yang, and S Yu Philip (2022). "Privacy and robustness in federated learning: Attacks and defenses". In: *IEEE transactions on neural networks and learning systems*.

Ma, Xu, Xiaoqian Sun, Yuduo Wu, Zheli Liu, Xiaofeng Chen, and Changyu Dong (2022a). "Differentially Private Byzantine-robust Federated Learning". In: *IEEE Transactions on Parallel and Distributed Systems* 33.12, pp. 3690–3701.

Ma, Zhuoran, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H Deng (2022b). "ShieldFL: Mitigating Model Poisoning Attacks in Privacy-Preserving Federated Learning". In: *IEEE Transactions on Information Forensics and Security* 17, pp. 1639–1654.

Maas, Andrew, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts (2011). "Learning word vectors for sentiment analysis". In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150.

Mammen, Priyanka Mary (2021). "Federated learning: Opportunities and challenges". In: *arXiv preprint arXiv:2101.05428*.

McMahan, Brendan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas (2017a). "Communication-efficient learning of deep networks from decentralized data". In: *Artificial Intelligence and Statistics*. PMLR, pp. 1273–1282.

McMahan, H Brendan, Daniel Ramage, Kunal Talwar, and Li Zhang (2017b). "Learning differentially private recurrent language models". In: *arXiv preprint arXiv:1710.06963*.

Melis, Luca, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov (2019). "Exploiting unintended feature leakage in collaborative learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 691–706.

Minaee, Shervin, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao (2021). "Deep Learning–based Text Classification: A Comprehensive Review". In: *ACM Computing Surveys (CSUR)* 54.3, pp. 1–40.

Minka, Thomas (2000). *Estimating a Dirichlet distribution*.

Mothukuri, Viraaji, Prachi Khare, Reza M Parizi, Seyedamin Pouriyeh, Ali Dehghantanha, and Gautam Srivastava (2022). "Federated learning-based anomaly detection for IoT security attacks". In: *IEEE Internet of Things Journal* 9.4, pp. 2545–2554.

Mothukuri, Viraaji, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava (2021). "A survey on security and privacy of federated learning". In: *Future Generation Computer Systems* 115, pp. 619–640.

Mrkšić, N, DO Séaghdha, B Thomson, M Gašić, PH Su, D Vandyke, TH Wen, and S Young (2015). "Multi-domain dialog state tracking using recurrent neural networks". In: *ACL-IJCNLP 2015-53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*. Vol. 2, pp. 794–799.

NajeebJebreel (2020). *NajeebJebreel/KeyNet v1.0.0*. Version v1.0.0. DOI: 10.5281/zenodo.4282992. URL: https://doi.org/10.5281/zenodo.4282992.

Naseri, Mohammad, Jamie Hayes, and Emiliano De Cristofaro (2020). "Toward robustness and privacy in federated learning: Experimenting with local and central differential privacy". In: *arXiv preprint arXiv:2009.03561*.

Nasr, Milad, Reza Shokri, and Amir Houmansadr (2019). "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning". In: *2019 IEEE symposium on security and privacy (SP)*. IEEE, pp. 739–753.

Ndirango, Anthony and Tyler Lee (2019). "Generalization in multitask deep neural classifiers: a statistical physics approach". In: *Advances in Neural Information Processing Systems*, pp. 15862–15871.

Neelakantan, Arvind, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens (2015). "Adding gradient noise improves learning for very deep networks". In: *arXiv preprint arXiv:1511.06807*.

Nelson, Blaine, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles Sutton, J Doug Tygar, and Kai Xia (2008). "Exploiting machine learning to subvert your spam filter." In: *LEET* 8, pp. 1–9.

Nguyen, Thien Duc, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, et al. (2022). "{FLAME}: Taming

Backdoors in Federated Learning". In: *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1415–1432.

Nguyen, Thien Duc, Phillip Rieger, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Ahmad-Reza Sadeghi, Thomas Schneider, et al. (2021). "FLGUARD: Secure and Private Federated Learning". In: *arXiv preprint arXiv:2101.02281*.

Nguyen, Tuan Anh and Anh Tran (2020a). "Input-aware dynamic backdoor attack". In: *NeurIPS*.

Nguyen, Tuan Anh and Anh Tuan Tran (2020b). "WaNet-Imperceptible Warping-based Backdoor Attack". In: *International Conference on Learning Representations*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

PyCryptodome (2021). *PyCryptodome a self-contained Python package of low-level cryptographic primitives*. URL: https://pycryptodome.readthedocs.io/ (visited on 09/27/2021).

Ribeiro, Mauro, Katarina Grolinger, and Miriam AM Capretz (2015). "Mlaas: Machine learning as a service". In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, pp. 896–902.

Rosenfeld, Elan, Ezra Winston, Pradeep Ravikumar, and Zico Kolter (2020). "Certified robustness to label-flipping attacks via randomized smoothing". In: *ICML*.

Rouhani, Bita Darvish, Huili Chen, and Farinaz Koushanfar (2018). "Deepsigns: A generic watermarking framework for ip protection of deep learning models". In: *arXiv preprint arXiv:1804.00750*.

Roy, Serge (1994). "Factors influencing the choice of a learning rate for a backpropagation neural network". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 1. IEEE, pp. 503–507.

Ruder, Sebastian (2017). "An overview of multi-task learning in deep neural networks". In: *arXiv preprint arXiv:1706.05098*.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen (2018). "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *CVPR*.

Sebé, Francesc, Josep Domingo-Ferrer, and Jordi Herrera (2000). "Spatial-domain image watermarking robust against compression, filtering, cropping, and scaling". In: *International Workshop on Information Security*. Springer, pp. 44–53.

Shafieinejad, Masoumeh, Jiaqi Wang, Nils Lukas, Xinda Li, and Florian Kerschbaum (2019). "On the robustness of the backdoor-based watermarking in deep neural networks". In: *arXiv preprint arXiv:1906.07745*.

Shejwalkar, Virat, Amir Houmansadr, Peter Kairouz, and Daniel Ramage (2021). "Back to the drawing board: A critical evaluation of poisoning attacks on federated learning". In: *arXiv preprint arXiv:2108.10241*.

Shen, Shiqi, Shruti Tople, and Prateek Saxena (2016). "Auror: Defending against poisoning attacks in collaborative deep learning systems". In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 508–519.

Siegel, Andrew F (1982). "Robust regression using repeated medians". In: *Biometrika* 69.1, pp. 242–244.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

So, Jinhyun, Başak Güler, and A Salman Avestimehr (2020). "Byzantine-resilient secure federated learning". In: *IEEE Journal on Selected Areas in Communications* 39.7, pp. 2168–2181.

So, Jinhyun, Başak Güler, and A Salman Avestimehr (2021). "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning". In: *IEEE Journal on Selected Areas in Information Theory* 2.1, pp. 479–489.

Stallkamp, Johannes, Marc Schlipsing, Jan Salmen, and Christian Igel (2011). "The German traffic sign recognition benchmark: a multi-class classification competition". In: *IJCNN*.

Steinhardt, Jacob, Pang Wei Koh, and Percy Liang (2017). "Certified defenses for data poisoning attacks". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 3520–3532.

Sun, Ziteng, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan (2019). "Can you really backdoor federated learning?" In: *arXiv preprint arXiv:1911.07963*.

Tajbakhsh, Nima, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang (2016). "Convolutional neural networks for medical image analysis: Full training or fine tuning?" In: *IEEE Transactions on Medical Imaging* 35.5, pp. 1299–1312.

Tang, Di, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang (2021). "Demon in the Variant: Statistical Analysis of *DNNs* for Robust Backdoor Contamination Detection". In: *USENIX Security*.

Tolpegin, Vale, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu (2020). "Data poisoning attacks against federated learning systems". In: *European Symposium on Research in Computer Security*. Springer, pp. 480–501.

Tramèr, Florian, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart (2016). "Stealing machine learning models via prediction apis". In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 601–618.

Turner, Alexander, Dimitris Tsipras, and Aleksander Madry (2019). "Label-consistent backdoor attacks". In: *arXiv preprint arXiv:1912.02771*.

Uchida, Yusuke, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh (2017). "Embedding watermarks into deep neural networks". In: *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pp. 269–277.

Wang, Binghui and Neil Zhenqiang Gong (2018). "Stealing hyperparameters in machine learning". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 36–52.

Wang, Hongyi, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos (2020a). "Attack of the tails: Yes, you really can backdoor federated learning". In: *Advances in Neural Information Processing Systems* 33, pp. 16070–16084.

Wang, Jiangfeng, Hanzhou Wu, Xinpeng Zhang, and Yuwei Yao (2020b). "Watermarking in deep neural networks via error back-propagation". In: *Electronic Imaging* 2020.4, pp. 22–1.

Wang, Tianhao and Florian Kerschbaum (2019). "Robust and Undetectable White-Box Watermarks for Deep Neural Networks". In: *arXiv preprint arXiv:1910.14268*.

Wang, Xiaofei, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen (2019). "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning". In: *IEEE Network* 33.5, pp. 156–165.

Wold, Svante, Kim Esbensen, and Paul Geladi (1987). "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3, pp. 37–52.

Wu, Hanzhou, Gen Liu, Yuwei Yao, and Xinpeng Zhang (2020a). "Watermarking Neural Networks with Watermarked Images". In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Wu, Zhaoxian, Qing Ling, Tianyi Chen, and Georgios B Giannakis (2020b). "Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks". In: *IEEE Transactions on Signal Processing* 68, pp. 4583–4596.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747*.

Xie, Cong, Oluwasanmi Koyejo, and Indranil Gupta (2020). "Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation". In: *Uncertainty in Artificial Intelligence*. PMLR, pp. 261–270.

Xu, Huazhe, Yang Gao, Fisher Yu, and Trevor Darrell (2017). "End-to-end learning of driving models from large-scale video datasets". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2174–2182.

Xu, Xiangrui, Yaqin Li, and Cao Yuan (2020). ""Identity Bracelets" for Deep Neural Networks". In: *IEEE Access* 8, pp. 102065–102074.

Yao, Andrew C (1982). "Protocols for secure computations". In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, pp. 160–164.

Yao, Yuanshun, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao (2017). "Complexity vs. performance: empirical analysis of machine learning as a service". In: *Proceedings of the 2017 Internet Measurement Conference*, pp. 384–397.

Yin, Dong, Yudong Chen, Ramchandran Kannan, and Peter Bartlett (2018). "Byzantine-robust distributed learning: Towards optimal statistical

rates". In: *International Conference on Machine Learning*. PMLR, pp. 5650–5659.

Zeng, Yi, Si Chen, Won Park, Z Morley Mao, Ming Jin, and Ruoxi Jia (2022). "Adversarial unlearning of backdoors via implicit hypergradient". In: *ICLR*.

Zeng, Yi, Won Park, Z Morley Mao, and Ruoxi Jia (2021). "Rethinking the backdoor attacks' triggers: A frequency perspective". In: *ICCV*.

Zhang, Chengliang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu (2020). "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning". In: *2020 USENIX Annual Technical Conference*, pp. 493–506.

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals (2021a). "Understanding deep learning (still) requires rethinking generalization". In: *Communications of the ACM* 64.3, pp. 107–115.

Zhang, Jialong, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy (2018). "Protecting intellectual property of deep neural networks with watermarking". In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 159–172.

Zhang, Wei and Xiang Li (2021). "Federated transfer learning for intelligent fault diagnostics using deep adversarial networks with data privacy". In: *IEEE/ASME Transactions on Mechatronics* 27.1, pp. 430–439.

Zhang, Wei and Xiang Li (2022). "Data privacy preserving federated transfer learning in machinery fault diagnostics using prior distributions". In: *Structural Health Monitoring* 21.4, pp. 1329–1344.

Zhang, Wenlu, Rongjian Li, Tao Zeng, Qian Sun, Sudhir Kumar, Jieping Ye, and Shuiwang Ji (2016). "Deep model based transfer and multi-task learning for biological image analysis". In: *IEEE Transactions on Big Data*.

Zhang, Yuhui, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng (2021b). "ShuffleFL: gradient-preserving federated learning using trusted

execution environment". In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*, pp. 161–168.

Zhang, Zhuosheng, Jiarui Li, Shucheng Yu, and Christian Makaya (2021c). "SAFELearning: Enable Backdoor Detectability In Federated Learning With Secure Aggregation". In: *arXiv preprint arXiv:2102.02402*.

Zhao, Bo, Konda Reddy Mopuri, and Hakan Bilen (2020). "idlg: Improved deep leakage from gradients". In: *arXiv preprint arXiv:2001.02610*.

Zhao, Pu, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin (2020). "Bridging mode connectivity in loss landscapes and adversarial robustness". In: *ICLR*.

Zhao, Yue, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra (2018). "Federated learning with non-iid data". In: *arXiv preprint arXiv:1806.00582*.

Zhu, Ligeng and Song Han (2020). "Deep leakage from gradients". In: *Federated Learning*. Springer, pp. 17–31.

UNIVERSITAT ROVIRA i VIRGILI