

Imperial College of Science, Technology and Medicine  
Dyson School of Design Engineering

# Privacy-preserving Machine Learning System at the Edge

Fan Mo

Submitted in part fulfilment of the requirements for the degree of  
**Doctor of Philosophy and Diploma**  
of  
**Imperial College London**  
May 2022



## Abstract

Data privacy in machine learning has become an urgent problem to be solved, along with machine learning’s rapid development and the large attack surface being explored. Pre-trained deep neural networks are increasingly deployed in smartphones and other edge devices for a variety of applications, leading to potential disclosures of private information. In collaborative learning, participants keep private data locally and communicate deep neural networks updated on their local data, but still, the private information encoded in the networks’ gradients can be explored by adversaries. This dissertation aims to perform dedicated investigations on privacy leakage from neural networks and to propose privacy-preserving machine learning systems for edge devices.

Firstly, the systematization of knowledge is conducted to identify the key challenges and existing/adaptable solutions. Then a framework is proposed to measure the amount of sensitive information memorized in each layer’s *weights* of a neural network based on the generalization error. Results show that, when considered individually, the last layers encode a larger amount of information from the training data compared to the first layers. To protect such sensitive information in weights, *DarkneTZ* is proposed as a framework that uses an edge device’s Trusted Execution Environment (TEE) in conjunction with model partitioning to limit the attack surface against neural networks. The performance of *DarkneTZ* is evaluated, including CPU execution time, memory usage, and accurate power consumption, using two small and six large image classification models. Due to the limited memory of the edge device’s TEE, model layers are partitioned into more sensitive layers (to be executed inside the device TEE), and a set of layers to be executed in the untrusted part of the operating system. Results show that even if a single layer is hidden, one can provide reliable model privacy and defend against state of art membership inference attacks, with only a 3% performance overhead.

This thesis further strengthens investigations from neural network weights (in on-device machine learning deployment) to gradients (in collaborative learning). An information-theoretical framework is proposed, by adapting usable information theory and considering the attack outcome as a probability measure, to quantify private information leakage from network gradients. The private original information and latent information are localized in a layer-wise manner. After that, this work performs sensitivity analysis over the gradients w.r.t. private information to further explore the underlying cause of information leakage. Numerical evaluations are

conducted on six benchmark datasets and four well-known networks and further measure the impact of training hyper-parameters and defense mechanisms. Last but not least, to limit the privacy leakages in gradients, I propose and implement a Privacy-preserving Federated Learning (*PPFL*) framework for mobile systems. TEEs are utilized on clients for local training, and on servers for secure aggregation, so that model/gradient updates are hidden from adversaries. This work leverages greedy layer-wise training to train each model's layer inside the trusted area until its convergence. The performance evaluation of the implementation shows that *PPFL* significantly improves privacy by defending against data reconstruction, property inference, and membership inference attacks while incurring small communication overhead and client-side system overheads. This thesis offers a better understanding of the sources of private information in machine learning and provides frameworks to fully guarantee privacy and achieve comparable ML model utility and system overhead with regular machine learning framework.

## Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Hamed Haddadi, for his support, guidance, and encouragement throughout my Ph.D. journey. He not only mentors my research but also gives me many valuable suggestions for my lifelong career. Thank you for the trust and freedom to let me choose my research. Thank you for providing me with many opportunities to explore my journey in both academic and industry ways. Without his support, this thesis would not exist.

I would like to sincerely thank Kleomenis Katevas and Soteris Demetriou, who have also guided me a lot when I just started my Ph.D. They advised me with patience and time which helped me get with my research.

I want to thank Nicolas Kourtellis who is my supervisor during my internship at Telefonica Research, Shale Xiong and Dominic P. Mulligan who are my supervisors at Arm Research. Thank you all for guiding me, and it was a great experience to work with you and the team.

I also want to deeply thank friends, colleagues, and collaborators, Mohammad Malekzadeh, Anastasia Borovykh, Yuchen Zhao, Ranya Aloufi, Anna Maria Mandalari, Ali S. Shamsabadi, Andrea Cavallaro, Ilias Leontiadis, and many more that I cannot mention here because the list will be very long. Thank you all very much for the great collaborations. Thank you all for your great support and advice.

I would also like to thank the supervisor of my Master's program in China, Jia Zhou, who has put great effort to encourage and help me to pursue a Ph.D. degree. I will always be grateful for your trust and encouragement.

Most of all, I would like to thank my beloved family, my wife Linfei Feng, and my parents Zhifang Liu and Bin Mo. They gave me so many to support my needs in my Ph.D. study. Without their faith and support, I cannot reach this stage.



I dedicate this thesis to my wife and my parents  
for their patience, faith, and supports.





## **Statement of Originality**

This is to certify that this thesis embodies the results of my own course of study and research.  
All else from other published or unpublished work is appropriately referenced.



## Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives (CC BY-NC) licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Abbreviations</b>	<b>xxiii</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Motivation and Objectives . . . . .	2
1.2 Contributions . . . . .	10
1.3 Thesis Outline . . . . .	12
1.4 Publications . . . . .	14
<b>2 Background and Literature Review</b>	<b>16</b>
2.1 Machine Learning . . . . .	17
2.1.1 Pipeline . . . . .	17
2.1.2 Paradigm . . . . .	19
2.1.3 Attack surface . . . . .	20
2.2 Confidential Computing and Trusted Execution Environments . . . . .	22

2.2.1	Threat models . . . . .	22
2.2.2	Key components . . . . .	23
2.2.3	Partitioning frameworks for developers . . . . .	26
2.3	Confidential Computing Solution for Machine Learning . . . . .	27
2.3.1	Overview . . . . .	27
2.3.2	Key challenges . . . . .	29
2.3.3	Existing solutions . . . . .	30
2.4	Confidential Computing Helping Machine Learning Integrity . . . . .	34
2.4.1	Key challenges . . . . .	34
2.4.2	Existing/adoptable solutions . . . . .	36
<b>3</b>	<b>Privacy Measure of Neural Network Weights</b>	<b>38</b>
3.1	Introduction and Related Work . . . . .	38
3.2	Proposed Approach . . . . .	39
3.2.1	Problem definition . . . . .	39
3.2.2	Sensitive information exposure approach . . . . .	40
3.3	Measuring Information Exposure . . . . .	42
3.3.1	Model and datasets . . . . .	42
3.3.2	Results and discussion . . . . .	42
3.4	Exploration of Protection using TEEs . . . . .	45
3.4.1	Evaluation setup . . . . .	45
3.4.2	Results and discussion . . . . .	47

---

3.5	Summary . . . . .	48
<b>4</b>	<b>On-device Privacy-preserving ML System</b>	<b>50</b>
4.1	Introduction and Related Work . . . . .	51
4.1.1	Membership inference attack (MIA) . . . . .	51
4.1.2	Deep learning in the TEE . . . . .	52
4.1.3	Privacy-preserving methods . . . . .	53
4.2	DarkneTZ Framework . . . . .	54
4.2.1	Threat model . . . . .	54
4.2.2	Design overview . . . . .	55
4.2.3	Model preparation . . . . .	56
4.2.4	DNN partitioned execution . . . . .	57
4.3	Experiment Settings . . . . .	58
4.3.1	Models and datasets . . . . .	58
4.3.2	Implementation and evaluation setup . . . . .	59
4.3.3	Measuring privacy in MIAs . . . . .	61
4.4	Evaluation Results . . . . .	62
4.4.1	CPU execution time . . . . .	64
4.4.2	Memory usage . . . . .	66
4.4.3	Power consumption . . . . .	68
4.4.4	Privacy measurement . . . . .	70
4.5	Discussion and Summary . . . . .	72

4.5.1	System performance . . . . .	72
4.5.2	Models' privacy . . . . .	74
4.5.3	Limitations & Next steps . . . . .	75
<b>5</b>	<b>Privacy Measure of Neural Network Gradients</b>	<b>76</b>
5.1	Introduction and Related Work . . . . .	77
5.1.1	Gradient sharing . . . . .	77
5.1.2	Attack and defense measurements . . . . .	78
5.1.3	Information flow via neural networks . . . . .	79
5.2	Problem Formulation . . . . .	81
5.2.1	Usable information definition . . . . .	81
5.2.2	Privacy terminology . . . . .	83
5.2.3	Attacker: Original information . . . . .	84
5.2.4	Attacker: Latent information . . . . .	86
5.2.5	The role of computational power . . . . .	88
5.3	Sensitivity Measure on Private Information . . . . .	88
5.3.1	Sensitivity justification . . . . .	89
5.3.2	Sensitivity of gradients w.r.t. original information . . . . .	90
5.3.3	Sensitivity of gradients w.r.t. latent information . . . . .	91
5.4	Numerical Evaluation . . . . .	92
5.4.1	Evaluation setup . . . . .	93
5.4.2	Layer-wise localization of information leakages . . . . .	94



---

5.4.3	Sensitivity analysis . . . . .	96
5.4.4	Impacts of training hyperparameters . . . . .	98
5.4.5	Impact of defense mechanisms . . . . .	100
5.5	Discussion and Summary . . . . .	102
5.5.1	Localization of the most sensitive layers . . . . .	102
5.5.2	Insights gained from gradient sensitivity . . . . .	102
5.5.3	Localization for the design of protection mechanisms . . . . .	103
5.5.4	Insights for defense mechanisms . . . . .	103
5.5.5	Limitations & Next steps . . . . .	104
<b>6</b>	<b>Privacy-preserving Federated Learning System</b>	<b>105</b>
6.1	Introduction and Related Work . . . . .	106
6.1.1	Revisiting TEEs . . . . .	106
6.1.2	Privacy risks in FL . . . . .	107
6.1.3	Layer-wise DNN training for FL . . . . .	108
6.2	Threat Model and Assumptions . . . . .	109
6.3	PPFL Framework . . . . .	110
6.3.1	System overview . . . . .	110
6.3.2	Layer-wise training and aggregation . . . . .	113
6.4	Implementation & Evaluation Setup . . . . .	117
6.4.1	PPFL prototype . . . . .	117
6.4.2	Models and datasets . . . . .	118

6.4.3	Performance metrics . . . . .	119
6.5	Evaluation Results . . . . .	121
6.5.1	Effectiveness of PPFL thwarting known privacy-related attacks . . . . .	121
6.5.2	Communication cost . . . . .	122
6.5.3	PPFL performance comparable to state-of-art FL . . . . .	126
6.5.4	Client-side system cost . . . . .	128
6.5.5	PPFL ML and system costs when blocks of layers were trained in clients	129
6.5.6	Bootstrapping the PPFL with public models help . . . . .	131
6.5.7	Bootstrapping the PPFL with public datasets help . . . . .	133
6.6	Discussion and Summary . . . . .	135
<b>7</b>	<b>Conclusion and Future Outlook</b>	<b>138</b>
7.1	Thesis Achievement . . . . .	138
7.2	Open Challenges and Future Outlook . . . . .	140
	<b>Bibliography</b>	<b>144</b>

# List of Tables

2.1	Summary of hardware-assisted Trusted Execution Environments . . . . .	25
2.2	Open source framework for Trusted Execution Environments . . . . .	25
2.3	Previous research that uses Trusted Execution Environments to guarantee the confidentiality of machine learning . . . . .	31
4.1	Training and testing accuracy (Acc.) and corresponding MIA precision (Pre.) with or without DarkneTZ (DTZ) of all models and datasets. . . . .	69
5.1	Frequently used notation in this chapter for privacy measures in neural network gradients. . . . .	77
5.2	Related works of measurements on attacks and defenses on neural network gradients. . . . .	79
5.3	Models and datasets used in experiments. . . . .	93
6.1	DNNs used in the evaluation of PPFL. . . . .	119
6.2	Results of three privacy-related attacks (DRA, AIA, and MIA) on PPFL vs. end-to-end (E2E) FL. Average score reported with 95% confidence interval in parenthesis. . . . .	122
6.3	Communication overhead (rounds and amount) of PPFL to reach the same accuracy as end-to-end FL system. . . . .	124

6.4	Time duration of FL phases in <i>one communication round</i> , when training LeNet, AlexNet and VGG9 models with PPFL and end-to-end (E2E) FL. . . . .	124
6.5	Reduction of communication rounds and amount when training 2-layer instead of 1-layer blocks. . . . .	130

# List of Figures

- 1.1 Popularity of deep learning (DL) applications vs non-DL applications in terms of Million Downloads and Thousand Reviews, data source from research [XLL<sup>+</sup>19]. 2
- 1.2 An overview of disclosing private information from a pre-trained DL model (Left) and the technique of the Trusted Execution Environment (Right). . . . . 4
- 2.1 Overview flow of Confidential Computing which utilizes the untrusted host's Trusted Execution Environment to protect the model and data in machine learning. 27
- 2.2 Server-side ML (Left) and client-side ML (Right) protection using Trusted Execution Environments. . . . . 28
- 2.3 Partition on the ML process (shown with model architectures) in order to protect it or a part of it inside TEEs. . . . . 33
- 3.1 The proposed framework for measuring the risk of exposing sensitive information in a deep neural network  $M$  trained on a private dataset  $S$ .  $M_b$  and  $M_s$  are obtained by fine-tuning the parameters of a target layer  $l$  on the whole training set  $X$  (i.e., both  $S$  and non-private training set  $T$ ) and  $S$ , respectively. . . . . 40
- 3.2 Generalization errors of  $M_s$  and  $M_b$  trained on half of the training set,  $S$ , of (a) MNIST, (b) Fashion-MNIST, and (c) CIFAR-10 for fine-tuning each target layer. Error bars represent 95% confidence intervals. . . . . 43

3.3	The risk of sensitive information exposure of VGG-7 per layer on MNIST, Fashion-MNIST and CIFAR-10. Error bars represent 95% confidence intervals. . . . .	44
3.4	Risk per neuron for each layer on MNIST, Fashion-MNIST, and CIFAR-10. Error bars represent 95% confidence intervals. . . . .	44
3.5	Using a TEE to protect the most sensitive layers (last layers) of an on-device deep neural network. . . . .	45
3.6	Execution time, memory usage, and power usage for protecting layers of VGG-7 trained on MNIST (left column) and CIFAR-10 dataset (right column) using the TrustZone of device. The x-axis corresponds to several last layers included in the TrustZone. <i>O</i> refers to the calculation of cost function; <i>SM</i> , <i>FC</i> , <i>D</i> , <i>MP</i> , and <i>C</i> refer to the softmax, fully connected, dropout, maxpooling, convolutional layers of VGG-7. The number of layers with trainable parameters in the TrustZone are 1, 2, 3, and 4. The dashed line represents the baseline, which runs all the layers outside the TrustZone. Error bars represent 95% confidence intervals. . . . .	46
4.1	DarkneTZ uses on-device TEE to protect a set of layers of a deep neural network for both inference and fine-tuning. . . . .	55
4.2	The CPU time of each step of training models or conducting inference on CIFAR-100 and ImageNet Tiny, protecting consecutive last layers using TrustZone. . . .	63
4.3	The CPU time of each step of training models or conducting inference on CIFAR-100, protecting consecutive last layers using TrustZone. . . . .	64
4.4	The CPU execution time in user mode and kernel mode of each step of training the model or conducting inference on CIFAR-100, protecting consecutive last layers using TrustZone. . . . .	65
4.5	The memory usage and power consumption of training models, while conducting training or inference on CIFAR-100 and ImageNet Tiny, protecting consecutive last layers using TrustZone. . . . .	67

4.6	Performance on protecting the last layer of models trained on ImageNet in TrustZone for inference. . . . .	68
4.7	Precision and recall of white-box membership inference attacks when first or last layers of the model, trained on CIFAR-100, are protected using TrustZone. . . .	70
4.8	Precision of white-box membership inference attacks on models trained on CIFAR-100 when only outputs are protected using TrustZone. . . . .	71
5.1	Markov chain in forward propagation (from layer 1 to $L$ ) and backward propagation (from layer $L$ to 1). $W$ refers to weights; $G$ refers to gradients; $T$ refers to intermediate representations. . . . .	80
5.2	Original information leakage measured by usable information with thresholds from 0.05 to 0.8 (with a step size of 0.05) in each 2-layer set. The left figure shows the results of Infimum over attack family [ZLH19a, ZMB20, GBDM20]. The right figure shows the results of iDLG model only [ZMB20]. . . . .	95
5.3	Latent information leakage, measured by usable information, in each layer's gradients. Scattered points refer to private attributes being measured for different datasets and classification tasks. Blue lines give the LOESS regression curve. Dashed lines (--) separate the feature extractor and the classifier of the model. .	96
5.4	Original information leakage risks, measured by sensitivity ([Left] $F$ -norm; [Right Bottom] 1-norm; [Right Top] $\infty$ -norm), in each layer's gradients, based on CIFAR100 (—), LFW (—), CelebA (—), PubFig (—) dataset. Error bars are 95% confidence intervals. . . . .	96
5.5	Latent information leakage risks, measured by subspace distances, in each layer's gradients. Scattered points correspond to private attributes. Blue lines give the LOESS regression. Dashed lines (--) refer to the feature extractor-classifier connection. . . . .	97

5.6	Influence of gradient aggregation on risks of original information [Left two] and latent information [Right two], measured on LeNet trained on CIFAR100 and LFW, respectively (Note: Points corresponding to failed trials of attacks are not plotted. Logarithmic scale is used in some plots for better visualization). . . . .	99
5.7	Influence of training epoch on risks of original information [Left] and latent information [Right], measured on LeNet trained on CIFAR100 and LFW, respectively.	100
5.8	Usable information of the ‘race’ attribute in a fixed number of gradients (Top) and a fraction of gradients (Bottom) from each layer of the neural network. . . . .	101
5.9	Influence of training epoch on risks of original information [Left] and latent information [Right], measured on LeNet trained on CIFAR100 and LFW, respectively.	101
6.1	Schematic diagram of PPFL. Main phases follow the system design in [BEG <sup>+</sup> 19].	110
6.2	Test accuracy of training LeNet, AlexNet, and VGG9 models on IID and Non-IID datasets when using PPFL. . . . .	126
6.3	System performance of the client devices when training LeNet, AlexNet, and VGG9 using PPFL, measured on <i>one step of training</i> (i.e., one batch of data). . . . .	127
6.4	Test accuracy of training AlexNet and VGG9 models on CIFAR10 (IID and Non-IID) when using PPFL with blocks of two layers in TEE. . . . .	129
6.5	System performance of the client devices when training AlexNet and VGG9 models on CIFAR10 when using PPFL with blocks of two layers in TEE. . . . .	130
6.6	Test accuracy of training on CIFAR10 (IID and Non-IID) with public models MobileNetv2 and VGG16, pre-trained on ImageNet. . . . .	131
6.7	System performance of client devices when training with transferred public models on CIFAR10, measured on 1 step of training. . . . .	131
6.8	Test accuracy when learning with public datasets. The short red line (–) starting from y-axis refers to end-to-end FL. . . . .	134



# Abbreviations

**AIA** Attribute Inference Attack

**API** Application Programming Interface

**AUC** Area Under the Curve

**C/Conv** Convolutional

**CA** Client Application

**CC** Confidential Computing

**CI** Confidence Interval

**CNN** Convolutional Neural Network

**CPU** Central Processing Unit

**D** Dropout

**DL** Deep Learning

**DLG** Deep Leakage from Gradients

**DNN** Deep Neural Network

**DP** Differential Privacy

**DP-SGD** Differentially Private Stochastic Gradient Descent

**DPI** Data Processing Inequality

**DRA** Data Reconstruction Attack

**DTZ** DarkneTZ

**E2E** End-to-End

**FC** Fully Connected

**FL** Federated Learning

**GPU** Graph Processing Unit

**HE** Homomorphic Encryption

**IID** Independent and Identically Distributed

**MB** Megabits

**MiB** Megabits

**MI** Mutual Information

**MIA** Membership Inference Attack

**ML** Machine Learning

**MP** Maxpooling

**MSE** Mean Squared Error

**OP-TEE** Open Portable Trusted Execution Environment

**OS** Operating System

**PPFL** Privacy-Preserving Federated Learning

**PSNR** Peak Signal-to-Noise Ratio

**ReLU** Rectifier Linear Unit

**REE** normal Rich Execution Environment

**RoT** Root of Trust

**SDK** Software Development Kits

**SGD** Stochastic Gradient Descent

**SGX** Software Guard eXtensions

**SM** Softmax

**SSIM** Structural Similarity Index Measure

**TA** Trusted Application

**TCB** Trusted Computing Base

**TEE** Trusted Execution Environment

**TLS** Transport Layer Security



# Chapter 1

## Introduction

### 1.1 Motivation and Objectives

Advances in memory and processing resources and the urge to reduce data transmission latency have led to a rapid rise in the deployment of various Deep Neural Networks (DNNs) on constrained edge devices (e.g., wearable, smartphones, and consumer Internet of Things (IoT) devices). According to a large-scale investigation of deep learning usage on smartphones, utilizing deep learning functions in applications becomes more popular [XLL<sup>+</sup>19] and gains more downloads and reviews (see Figure 1.1). Among these applications, 81% regard deep learning as their core features, and 71% of them are for image-related usages.

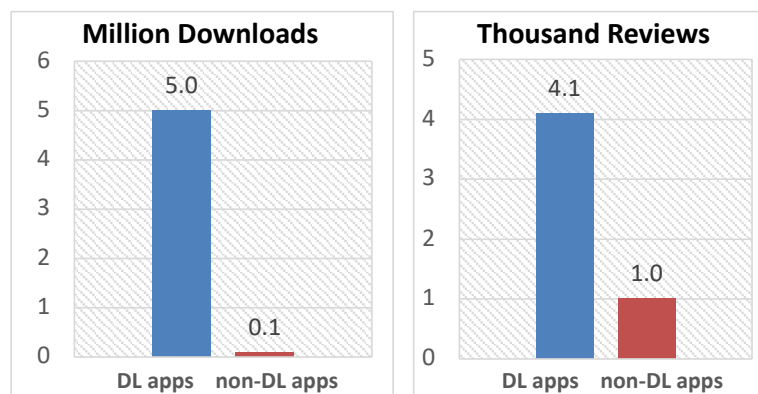


Figure 1.1: Popularity of deep learning (DL) applications vs non-DL applications in terms of Million Downloads and Thousand Reviews, data source from research [XLL<sup>+</sup>19].

Compared with centralized infrastructures (i.e., Cloud-based systems), hybrid and edge-based learning techniques enable methods for preserving users' privacy, as raw data can stay local [OST<sup>+</sup>18]. Nonetheless, recent work demonstrated that there is still private information leakage from local models [YLP<sup>+</sup>19, HAPC17, YGFJ18, MSDCS19, SZH<sup>+</sup>18].

**Challenge 1: Understanding private information leakage from local models' *weights*.**

The local model can be used by adversaries aiming to compromise the confidentiality of the model itself or that of the participants in training the model [SSSS17, YGFJ18]. The latter is part of a more general class of attacks, known as *Membership Inference Attacks* (refer to as MIAs henceforth). MIAs can have severe privacy consequences [LQS<sup>+</sup>13, SZH<sup>+</sup>18] motivating several studies to focus on tackling them [ACG<sup>+</sup>16, Mir17, JE19a]. Predominantly, such mitigation approaches rely on differential privacy [DR<sup>+</sup>14, EPK14], whose improvement in privacy preservation comes with an adverse effect on the model's prediction accuracy. Figure 1.2 shows an overview that information leaks from the trained model, while several techniques can be used to mitigate such leakage. One mitigation that deserves more investigation is *Trusted Execution Environment* (TEE). Nevertheless, the defenders are still unclear about privacy issues they will encounter; how, where, and in which degrees the privacy leakage happens. For example, we do not know if there are types of private information that can leak other than membership information. The first research question this thesis aims to answer is **Q1: How, where, and in which degrees does the private information leak from a DNN model in general?** This seems to be a tough question to answer and may relate to the unsolved problem, the generalization mystery of DNN models, but we aim to first have an overall understanding of such privacy leakage. At least when it comes to TEE protection, we can have a general view of which parts of DNN models are more sensitive and are the priority to protect.

**Challenge 2: Developing privacy-preserving edge system for on-device ML.** It is observed that edge devices are now increasingly equipped with a set of software and hardware security mechanisms powered by processor (CPU) designs offering strong isolation guarantees. System designs such as Arm TrustZone [Arm09] can enforce memory isolation between an untrusted part of the system operating in a *Rich Execution Environment* (REE), and a smaller

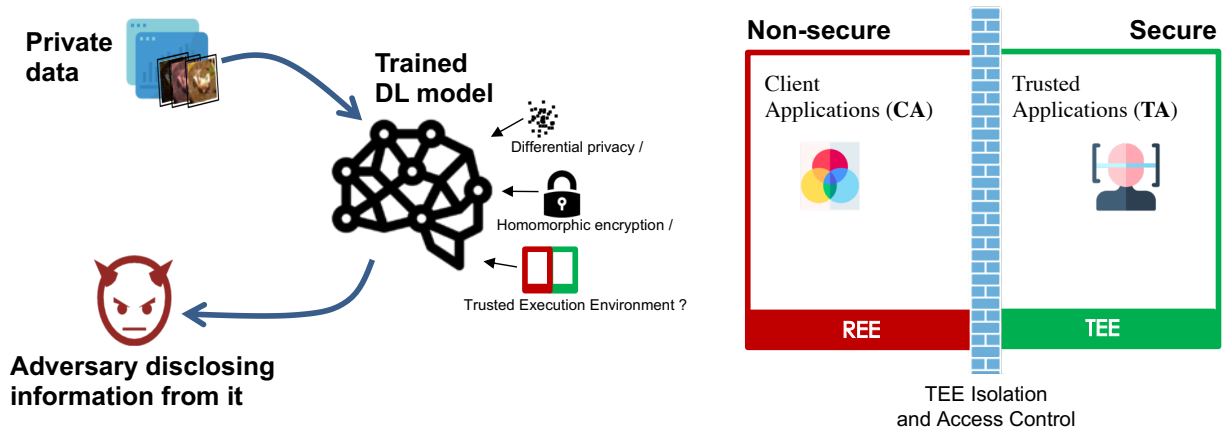


Figure 1.2: An overview of disclosing private information from a pre-trained DL model (Left) and the technique of the Trusted Execution Environment (Right).

trusted component operating in hardware-isolated TEE, responsible for security-critical operations. If one could efficiently execute sensitive DNNs inside the trusted execution environments of mobile devices, this would allow us to limit the attack surface of models without impairing their classification performance. Previous work has demonstrated promising results in this space; recent advancements allow for high-performance execution of sensitive operations within a TEE [HSS<sup>+</sup>18, HZG<sup>+</sup>18, TGS<sup>+</sup>18, GHZ<sup>+</sup>18b, TB18]. These works have almost exclusively experimented with integrating DNNs in cloud-like devices equipped with Intel Software Guard eXtensions (SGX). However, this paradigm does not translate well to edge computing due to significant differences in the following three factors: attack surface, protection goals, and computational performance. The *attack surface* on servers is exploited to steal a user’s private data, while the adversary on a user’s edge device focuses on compromising a model’s privacy. Consequently, the *protection goal* in most works combining deep learning with TEEs on the server (i.e., [GHZ<sup>+</sup>18b] and [HSS<sup>+</sup>18]) is to preserve the privacy of a user’s data during inference, while the protection on edge devices preserves both the model privacy and the privacy of the data used in training this model. Lastly, edge devices (such as IoT sensors and actuators) have *limited computational resources* compared to cloud computing devices; hence one cannot merely use performance results derived on an SGX-enabled system on the server to extrapolate measurements for TEE-enabled embedded systems. In particular, blindly integrating a DNN, or specifically, the most widely-used architecture Convolutional Neural Network (CNN) or one feature extractor followed by a classifier, in an edge device’s TEE might not be computationally

practical or even possible. A systematic measurement of the effects of such designs on edge-like environments would be useful for later development, which is one important work this thesis aims to perform. Thus, the research question I aim to answer here is: **Q2:** *Is it practical to store and execute a sequence of sensitive DNN's layers inside the TEE of an edge device?*

Since DNNs follow a layered architecture, this can be exploited to partition a DNN, having a sequence of layers executed in the untrusted part of the system while hiding the execution of sensitive layers in the trusted, secure environment. This part of the work utilizes the TEE (i.e., Arm TrustZone) and performs a unique layer-wise analysis on Convolutional Neural Networks to illustrate the privacy repercussions of an adversary on relevant neural network models on edge devices with the corresponding performance effects. Specifically, one research question this thesis aims to answer here is **Q3:** *Are any partitions useful to both effectively and efficiently tackle realistic attacks against DNNs on mobile devices?* Answering the above two questions helps us to understand running machine learning at the edge. However, there are still two factors requiring further investigations: i) other privacy-related attacks besides MIA, and ii) the privacy issues when sharing the on-device trained/fine-tuned DNNs. These two issues become more complicated, especially in sharing intermediate model updates in collaborative or federated ML.

**Challenge 3: Understanding private information leakage from *gradients*.** Using collaborative learning [MMR<sup>+</sup>17, YLCT19], *private data owners* (having personal [HRM<sup>+</sup>18], medical [RHL<sup>+</sup>20], financial [LTJZ20], or governmental [ZZS<sup>+</sup>22] dataset) are increasingly incentivized to participate in training a deep neural networks (DNN) model for a *target* task. Instead of sharing their private datasets, *participants* only share the *gradients* (or updated parameters) of the locally trained model at each iteration. The motivation of these methods is to utilize local computational power and comply with data protection regulations [EUd]. However, as noted by previous works that propose privacy-related attacks [ZLH19a, ZMB20, MSDCS19, GBDM20, NSH19], the shared gradients might be enough for an *attacker* (either the central server or one of the participants) to infer the private attributes of the data owners, known as *Attribute/Property Inference Attack* (PIA), or to reconstruct their private data, known as



*Data Reconstruction Attacks* (DRA). These ever-increasing attacks on the shared gradients call into doubt the privacy promised by collaborative/federated machine learning algorithms. More importantly, it is not fundamentally understood which *layers* of a DNN can potentially leak more private information, and to what *extent* different types of private information are stored in the shared gradients of each layer.

Despite many empirical privacy attacks on specific applications of collaborative learning, only a few works have recently started preliminary investigations on the performance of these attacks and the possibility of defending against them [HGS<sup>+</sup>21, LWH<sup>+</sup>22, URPPK22]. [LWH<sup>+</sup>22] further developed a Python-based software that can be reused partially for conducting and measuring attacks.

Following previous works, there still exist several open questions. i) Available measurements are mainly experimental; although there are some works [ZB20, FNJ<sup>+</sup>20] that explain why data reconstruction attacks can be successful in certain applications, we still lack a *unifying theory* for measuring different types of information contained in shared gradients. ii) Previous investigations are coarse-grained and look at the DNN model as a whole, meaning they do not *localize* which layers in a target DNN contain more private information for specific attacks. iii) Previous approaches only present the results of applying existing attacks and defenses, and do not allow us to understand the *underlying justifications* of how in general information leakage can be linked to other properties of DNN gradients; e.g., the sensitivity to the changes in the input data. Specifically, the research questions summarized to understand private information leakage from gradients are **Q4**: *Can we propose a theoretical-based measurement (e.g., information theory) to quantify and localize private information leakages in a fine-grained way?* and **Q5**: *Can we use any measurement to understand the underlying justifications of who these information leakages link to properties of gradients?*

In this thesis, I first propose a measurement framework based on information theory. Information theory has been used for measuring the amount of original or latent information that is captured in the outputs of different layers of a DNN and how this information evolves during the training procedure [TPB00, GVDBG<sup>+</sup>19, SZT17, SBD<sup>+</sup>19, APS19]. To understand the

information flow through DNN layers in the forward pass, the Shannon mutual information (MI) [SZT17, SBD<sup>+</sup>19, GVDBG<sup>+</sup>19] between each layer’s intermediate representations and the input data (i.e., the original info) or the output label (i.e., the public latent info) is estimated during each iteration. Nevertheless, it is well-known that the estimation of Shannon entropy and MI for high-dimensional data can have a high bias or variance [Pan03, BBR<sup>+</sup>18]. Furthermore, these estimations are even more difficult if one wants to adapt them to gradient produced in backpropagation, where analyses based on the data processing inequality (DPI) are not applicable anymore (see section 5.1.3 for an explanation of this difficulty).

This part of the work aims to bridge the gap between the aforementioned line of works and propose an information-theoretical framework to measure private information leakage from DNN gradients in a fine-grained and computable manner. Instead of the classical Shannon MI, this thesis adapts *usable information* [XZS<sup>+</sup>20] (i.e., predictive  $\mathcal{V}$ -information) to measure the ‘likelihood’ of a successful attack using a particular (family of) attack models, given that the attacker is able to obtain a subset of the gradients. Disregard the other types of attacks, I focus on two main types of private information: i) *original information*, the observed data in training datasets which are aimed by DRA, and ii) *latent information*, i.e., attributes of the observed data which are aimed by AIA; in this way, the framework covers most existing, critical information leakages on gradients. The proposed measure is based on the probability of recovering certain information (for attribute inference attacks) or of obtaining certain similarities (for data reconstruction attacks). To enable the measure to be generalized to private information of any granularity/dimensions, I consider the outcomes of every attack as a probability distribution over the *attacker’s aim*, and I approximate the density of this probability distribution. Such a framework is able to quantify and localize the private information in a layer-wise manner to better understand the “private information flow” through gradients of a DNN’s layers.

In support of this proposed measure and to further explore the underlying cause of information leakage, a sensitivity analysis can be performed over the gradients w.r.t. the two types of private information. This approach is inspired by the concept that sensitivity of gradients w.r.t. input/output can reflect model robustness [NBA<sup>+</sup>18], and robustness, in turn, is related to privacy risks [HCY21]. As sensitivity computation relies only on the trained model itself, it

can help to understand leakages independently of the attack model. Therefore, these sensitivity analyses are used to validate the results of the proposed usable information measure.

**Challenge 4: Developing privacy-preserving system for collaborative learning.** With such dedicated privacy leakage analysis on shared gradients, we can design the framework to perform collaborative learning, or more specifically federated learning, in a privacy-preserving way. Federated learning (FL) is a particular way of training deep neural networks (DNNs) on multiple devices locally and building an aggregated global model on a server. It has drawn significant attention from academia (e.g., [GKN17, KMA<sup>+</sup>19, LKX<sup>+</sup>20]) and industry, and is even being deployed in real systems (e.g., Google Keyboard [BEG<sup>+</sup>19]). Unlike traditional machine learning (ML), where a server collects all user data at a central point and trains a global model, in FL, users only send the locally updated model parameters to the server. This allows training a model without the need for users to reveal their data, thus preserving their privacy. As above mentioned, recent works have shown that adversaries can execute attacks to retrieve sensitive information from the model parameters themselves [ZLH19a, MSDCS19, GBDM20, HAPC17]. Prominent examples of such attacks are reconstruction [HAPC17, GBDM20] and various types of inference attacks [HAPC17, MSDCS19]; specifically, they are data reconstruction attacks, attribute inference attacks, and membership inference attacks as mentioned. Note that in FL scenarios, such attacks can be launched both at the server and client sides.

Motivated by these attacks, researchers have recently introduced several countermeasures to prevent them. Existing solutions can be grouped into three main categories depending on whether they rely on: (i) homomorphic encryption (e.g., [AHW<sup>+</sup>17, LKX<sup>+</sup>20]), (ii) multi-party computation (e.g., [BIK<sup>+</sup>17]), or (iii) differential privacy (e.g., [GKN17, MRTZ18, DR<sup>+</sup>14]). While homomorphic encryption is practical in both high-end and mobile devices, it only supports a limited number of arithmetic operations in the encrypted domain. Alternatively, the use of fully homomorphic encryption has been employed to allow arbitrary operations in the encrypted domain, thus supporting ML. Yet, this comes with too much computational overhead, making it impractical for mobile devices [NLV11, SEA20]. Similarly, current multi-party computation-based solutions incur significant computational overhead making it impossible to

be used on edge devices. The mechanism of differential privacy (DP) provides a theoretically grounded way to add noise for privacy guarantee and has been already used in practice (e.g., Apple’s iPhone DP system), but it can fail to provide sufficient privacy. This is highly likely to happen when the granularity of privacy is small, e.g., pixel-level private information on images [ZLH19a]. To guarantee such privacy, the level of differentially private noise addition goes very high which can even fully destroy the model utility (i.e., impossible to train the model). Besides, differentially private training impacts the fairness of the model. For example, when the classes of the training dataset are unbalanced, differentially private training tends to ignore classes of minority due to the stronger noises added to them for reducing the probability of re-identifying them [JE19a, BPS19]. This leads to more difficulties in federated learning settings. Current differentially private training also increases the system overhead (execution time and memory usage) because batch-based training is broken into micro-batches as the need for pre-sample gradient clipping [TM21, SVK20]. Overall, none of the existing solutions meets all requirements, hampering their adoption. Thus, this part of the work still considers hardware-based Trusted Execution Environments (TEEs) as a promising way to preclude attacks against DNN model parameters and gradients. All these advantages – together with the recent commoditization of TEEs both in high-end and mobile devices – make TEEs a suitable candidate to allow fully privacy-preserving ML modeling. However, in order to keep the Trusted Computing Base (TCB) as small as possible, current TEEs have limited memory. This makes it impossible to simultaneously place all DNN layers inside the TEE, as above mentioned, it may be more realistic to use TEEs to conceal only the most sensitive DNN layers from adversaries, leaving other layers unprotected [GHZ<sup>+</sup>18b, MSK<sup>+</sup>20]. While this approach was sufficient to mitigate some attacks against traditional ML where clients obtain only the final model, in FL scenarios the attack surface is significantly larger. FL client devices are able to observe distinct snapshots of the model throughout the training, allowing them to realize attacks at different stages [MSDCS19, HAPC17]. Therefore, it is of utmost importance to protect all DNN layers using the TEE. The last research question is **Q6**: *How to protect all layers, and what are the performance and cost if all layers are protected with TEEs during federated learning?*

## 1.2 Contributions

**Information theory-based privacy leakage measures.** By answering research question Q1 and in combination with research questions Q4 and Q5, this thesis proposes to utilize layer-wise generalization errors to measure memorized privacy in models. I further introduce an information-theoretical explanation for private information leakage under the *usable information* theory [XZS<sup>+</sup>20]. The proposed measure allows us to quantify the privacy risks of any selected subset of the shared gradients through a probabilistic view of the attacker’s aim. Due to the nature of usable information theory, this framework enables us to differentiate attackers with different levels of computational power, and thus better understand how much latent and original information leakage can occur. New attacks proposed in the future can be easily integrated into this framework by including them into the defined attack family. This may require re-performing some evaluations based on new attacks, but the final measurement results are grounded under usable information theory.

Utilizing the proposed framework, I perform a *layer-wise* localization of private information flow in the backpropagation process of DNNs. The characterizations reveal that fully-connected layers in benchmark DNN classifiers usually contain the highest latent and original information. It is believed that such understanding can help to achieve better data protection in environments where one can choose to hide or protect a subset of the DNN layers, e.g., split learning [TCCS20], differential privacy [MAE<sup>+</sup>18], or trusted execution environments [MSK<sup>+</sup>20].

Using a heuristic, a general-purpose measure based on gradient sensitivity, the “predictability” of information leakage is examined. This gives us an additional tool to validate the proposed measure of usable information in an attack-independent manner.

I extensively evaluate the effect of training hyperparameters and defense mechanisms on the probability of information leakage. It is shown that more *epochs* do not significantly reduce the amount of information leakage, but gradient *aggregation* can significantly reduce it. The result also shows that applying dropout and differential privacy on more sensitive layers can achieve better performance in alleviating information leakage.

**On-device privacy-preserving ML framework.** To first answer the two questions, Q2 and Q3, this thesis designs a framework, namely *DarkneTZ*, which enables an exhaustive layer-by-layer resource consumption analysis during the execution of a DNN model. *DarkneTZ* partitions a model into a set of non-sensitive layers ran within the system’s REE and a set of sensitive layers executed within the trusted TEE. I use *DarkneTZ* to measure, for a given DNN on two small and six large image classification models, the underlying system’s CPU execution time, memory usage, and accurate power consumption for different layer partition choices. I demonstrate the prototype of *DarkneTZ* using the Open Portable TEE (OP-TEE)<sup>1</sup> software stack running on a Hikey 960 board.<sup>2</sup> OP-TEE is compatible with the mobile-popular Arm TrustZone-enabled hardware, while the choice of hardware closely resembles common edge devices’ capabilities [YAA<sup>+</sup>18, PZLL19]. The results show that *DarkneTZ* only has 10% overhead when fully utilizing all available secure memory of the TEE for protecting a model’s layers. These results illustrate that REE-TEE partitions of certain DNNs can be efficiently executed on resource-constrained devices.

Then, I develop a threat model considering state-of-the-art MIAs against DNNs. I implement the respective attacks and use *DarkneTZ* to measure their effectiveness (adversary’s success rate) for different model partition choices. Results of this part of the work show that by hiding a single layer (the output layer) in the TEE of a resource-constrained edge device, the adversary’s success rate degrades to random guess while (a) the resource consumption overhead on the device is negligible (3%) and (b) the accuracy of inference remains intact. I also demonstrate the overhead of fully utilizing TrustZone for protecting models, and show that *DarkneTZ* can be an effective first step towards achieving hardware-based model privacy on edge devices.

**Privacy-preserving federated learning framework.** To answer the last research question Q6, this thesis proposes Privacy-preserving Federated Learning (PPFL), the first practical framework to *fully* prevent private information leakage at both *server* and *client-side* under FL scenarios. PPFL is based on *greedy layer-wise* training and aggregation, overcoming the

---

<sup>1</sup> <https://www.op-tee.org/> <sup>2</sup> <https://www.96boards.org/product/hikey960/>

constraints posed by the limited TEE memory, and providing comparable accuracy of complete model training at the price of a tolerable delay. This layer-wise approach supports sophisticated settings such as training one or more layers (block) each time, which can potentially better deal with heterogeneous data at the client-side and speed up the training process.

To show its feasibility, I implement and evaluate a full prototype of PPFL system including server-side (with Intel SGX), client-side (with Arm TrustZone) elements of the design, and the secure communication between them. The experimental evaluation shows that PPFL provides full protection against data reconstruction, property inference, and membership inference attacks, whose outcomes are degraded to random guessing (e.g., white noise images or 50% precision scores). PPFL is practical as it does not add significant overhead to the training process. Compared to regular end-to-end FL, PPFL introduces a  $3\times$  or higher delay for completing the training of all DNN layers. However, PPFL achieves comparable ML performance when training only the first few layers, meaning that it is not needed to train all DNN layers. Due to this flexibility of layer-wise training, PPFL can provide a similar ML model utility as end-to-end FL, with fewer communication rounds ( $0.54\times$ ), and a similar amount of network traffic ( $1.002\times$ ), with only  $\sim 15\%$  CPU time,  $\sim 18\%$  memory usage, and  $\sim 21\%$  energy consumption overhead at client-side.

This work is the first to build a DNN model in a FL setting with privacy-preserving guarantees using TEEs, by leveraging the greedy layer-wise training and training each DNN layer inside each FL client’s TEE. Thus, PPFL satisfies the constraint of TEE’s limited memory while protecting the model from the aforementioned privacy attacks. Interestingly, the classifiers built atop each layer may also provide personalization opportunities for the participating FL clients.

### 1.3 Thesis Outline

**Chapter 2** provides a literature review on privacy-preserving ML and the potential use of TEEs. I systematize the knowledge based on existing research and products that applies con-

confidential computing techniques (e.g., TEEs) for machine learning. Specifically, § 2.1 describes the pipeline and paradigms of ML and existing attack surfaces; § 2.2 outlines the threat model, key components, and tools around confidential computing; § 2.3 summarizes key challenges and existing solutions of using confidential computing for ML; § 2.4 presents the challenges and solutions of guaranteeing ML *integrity* with confidential computing.

**Chapter 3** presents a layer-wise privacy analysis method for the DNN model, more specifically, weights based on layers' maximum and minimum generalization errors. § 3.1 gives an introduction, and then § 3.2 presents the proposed approach including the problem definition and the method of sensitive information exposure; § 3.2 shows the evaluation results using the information exposure method; § 3.4 shows some initial measurements when using TEEs to protect machine learning models; § 3.5 discusses the results and then concludes this chapter.

**Chapter 4** provides a dedicated on-device privacy-preserving ML solution using TEEs based on the previous development. After introduction and related work (§ 4.1), the design and main components of DarkneTZ are presented (§ 4.2). § 4.3 provides implementation details and describes the evaluation setup (the implementation is available online<sup>3</sup>), while § 4.4 presents the system performance and privacy evaluation results. Lastly, § 4.5 discusses further performance and privacy implications that can be drawn from the systematic evaluation.

**Chapter 5** further presents an information-theoretical framework, by adapting usable information theory and considering the attack outcome as a probability measure, to quantify private information leakage from network gradients. Specifically, the used notation, gradient sharing, and information flow concept are clarified in § 5.1; § 5.2 formulates the problem based on usable information with computational constraints for both initial private information and latent private information; § 5.3 further provides gradient sensitivity measure to reason about the leakage; § 5.4 shows the numerical evaluation using information theory and sensitivity analysis; § 5.5 discusses and concludes the gradient leakage measurements.

**Chapter 6** provides the proposed framework to achieve privacy-preserving federated learning

---

<sup>3</sup> <https://github.com/mofanv/darknetz>



(PPFL). § 6.1 gives the introduction and related work needed to understand the design atop TEEs and greedy layer-wise training; § 6.2 describes the threat model and assumptions in this federated learning setting; § 6.3 presents an overview of the proposed system and its functionalities and detail how the framework employs layer-wise training and aggregation in conjunction to TEEs; § 6.4 describes the implementation of the system and evaluation setup; § 6.5 shows the evaluation results, and § 6.6 gives the conclusion.

**Chapter 7** concludes by summarizing the contribution and achievements, and then discusses open challenges and future outlook.

## 1.4 Publications

Publications and unpublished work included in this thesis:

- **Fan Mo**, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. “PPFL: privacy-preserving federated learning with trusted execution environments.” In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 94-108. 2021. (Best Paper Awards).  
Chapter 6
- **Fan Mo**, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. “PPFL: Enhancing Privacy in Federated Learning with Confidential Computing.” *GetMobile: Mobile Computing and Communications* 25, no. 4 (2022): 35-38.  
Chapter 6
- **Fan Mo**, Anastasia Borovykh, Mohammad Malekzadeh, Hamed Haddadi, and Soteris Demetriou. “Quantifying and Localizing Usable Information Leakage from Neural Network Gradients.” Submitted to *IEEE Transactions on Information Forensics and Security (TIFS)*. arXiv preprint arXiv: 2105.13929. Chapter 5
- **Fan Mo**, Anastasia Borovykh, Mohammad Malekzadeh, Hamed Haddadi, and Soteris Demetriou. “Layer-wise characterization of latent information leakage in federated learn-

ing.” In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), PPML Workshop*, 2020. **Chapter 5**

- **Fan Mo**, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. “DarkneTZ: towards model privacy at the edge using trusted execution environments.” In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 161-174. 2020. **Chapter 4**
- **Fan Mo**, and Hamed Haddadi. “Efficient and private federated learning using TEE.” In *Proceedings of the Fourteenth EuroSys Conference (EuroSys), PhD Workshop*, 2019. **Chapter 6**
- **Fan Mo**, Ali Shahin Shamsabadi, Kleomenis Katevas, Andrea Cavallaro, and Hamed Haddadi. “Towards characterizing and limiting information exposure in DNN layers.” In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), Poster, Short paper*, 2019. **Chapter 3**

Publications and unpublished work not included in this thesis, but related to my PhD work:

- Yuchen Zhao, Sayed Saad Afzal, Waleed Akbar, Osvy Rodriguez, **Fan Mo**, David Boyle, Fadel Adib, and Hamed Haddadi. “Towards battery-free machine learning and inference in underwater environments.” In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications (HotMobile)*, pp. 29-34. 2022.
- Mathias Brossard, Guilhem Bryant, Basma El Gaabouri, Xinxin Fan, Alexandre Ferreira, Edmund Grimley-Evans, Christopher Haster, Evan Johnson, Derek Miller, **Fan Mo**, Dominic P. Mulligan, Nick Spinale, Eric Van Hensbergen, Hugo J. M. Vincent, Shale Xiong. “Private delegated computations using strong isolation.” arXiv preprint arXiv:2205.03322.

## Chapter 2

# Background and Literature Review

Machine learning (ML) has been widely recognized as the most ubiquitous approach to learning patterns from data. ML applications exist across data processing of online surfing, financial data, health care, autonomous cars, and almost every field around us. Due to the wide applications, ML is run on heterogeneous devices – from distributed IoT/mobile devices to the high-performance computing center. This at the same time also opens a large attack surface in ML’s security and privacy, which has been explored by recent research, e.g., ML model stealing [OSF19, TZJ<sup>+</sup>16], model inversion [FJR15, HZL19], model/data poisoning [FCJG20, CLL<sup>+</sup>17], data reconstruction [ZLH19a, YMV<sup>+</sup>21], membership/attribute inference [SSSS17, MSDCS19].

ML faces security and privacy issues mainly due to i) the complexity of pipelines that involve many system/software stacks and ii) the weak robustness and low interpretability of ML algorithms. A full ML pipeline covers the collecting of raw data, the complete training and inference phases, and the later use of the trained ML model for prediction. Because data owners, the host of ML computation, the model owners, and result receivers are most likely to be different entities, the pipeline can be segmented across mutually mistrusting individuals and is left with a broad attack surface. In addition, considering that the ML algorithm at the current stage has relatively weak robustness (e.g., shown with adversarial examples [GSS14] or poisoning attacks [CLL<sup>+</sup>17]), a small change to the training process could cause large negative impacts that are hard to detect.

Meanwhile, Confidential Computing (CC) has emerged as one promising approach to achieving confidential ML. CC<sup>1</sup> is the protection of data in use, in addition to data protection in storage and transmission, by performing the computation in a hardware-based Trusted Execution Environment (TEE). The TEE is an isolation technique that establishes an isolated area (called enclaves or trust world, conventionally) where one can run codes inside in parallel with the normal operating system. The TEE has already been incorporated into most modern processors (e.g., Arm and Intel). The rapid growth of CC also pushes a line of research work that adopts it for ML [OSF<sup>+</sup>16, TB18, HSS<sup>+</sup>18, LLP<sup>+</sup>19, MHK<sup>+</sup>21] and it is also increasingly built as commercial products (e.g., CC service from Microsoft Azure [cca], Amazon Web Services [ccn], Google Cloud [cgg], etc.) Although many attempts have been made and several limitations are found, there is still no systematization of the challenges, and it is still unclear what are the key scientific and engineering obstacles and how to overcome them in order to run ML with confidential computing. This chapter presents a systematic analysis of CC-assisted solutions to alleviate the security and privacy problems in ML.

## 2.1 Machine Learning

Machine learning (ML) automates the pattern learning from large datasets in the sense that analysts do not need to manually explore data's latent features and their correlations [JM15, LBH15]. This section looks into the ML pipeline, common paradigms, and attack surfaces that exist among them.

### 2.1.1 Pipeline

The ML pipeline codifies and automates the workflow to produce and apply ML algorithms or models. It generally covers most cases when using ML, which consist of multiple sequential steps: data preparation, model training, model deployment/inference, and possible retraining/updates on the existing model.

---

<sup>1</sup> Definition in the whitepapers [Comb, Cona] by Confidential Computing Consortium, one project at the Linux Foundation.

**Data preparation.** Data are the basis for ML as we can imagine because ML requires tons of inputs to attempt “trial and error” and learn the patterns. In supervised learning [LBH15], all data are labeled, and this usually needs extensive efforts of human annotation; in semi-/un-supervised learning [ZG09, BCG<sup>+</sup>19, Le13], parts/all of the data are unlabeled due to its difficulty or saving the manpower; in reinforcement learning [SB18], data are in the form of sequences of action, observations, and rewards. No matter in any form, ML is always data-hungry. Before the learning starts, many efforts have to be made to obtain a large amount of data in order to cover all possible features they have in their distribution. Still, data augmentation [SK19b, XDH<sup>+</sup>20] is necessary to increase the amount of data by adding slightly modified copies or synthetic copies of existing data. Collected data are usually partitioned as the training set and the test set (sometimes additionally, the validation set) for training and evaluating the ML model.

**Model training.** Most model training aims to acquire a function  $f_\theta(x)$ , where model parameters  $\theta \in \Theta$  the parameter space, capable of mapping an input  $x$  to a predicted decision  $\hat{y} = f_\theta(x)$ , a value near to the true decision  $y$ . The form of  $f_\theta(\cdot)$  excluding  $\theta$  is called model hyperparameters, which are determined by selected model architectures e.g., the type of layers, the number of neurons, etc. Searching of the best  $\theta$  set usually is achieved by minimizing the loss  $\ell(y, f_\theta(x))$  using Stochastic Gradient Descent (SGD) [LBB<sup>+</sup>98, Bot10] on training sets, i.e., stepped descent by  $\theta \leftarrow \theta - \eta \nabla_\theta f_\theta(x)$ , where  $\eta$  is the step size multiplier known as the learning rate. The loss is backward propagated through the model, which is called backward propagation. After reaching a good performance on the validation set (or the other subset apart from the training set in cross-validation [K<sup>+</sup>95, AC10]), the model training stops. Then,  $f_\theta(\cdot)$ 's final model utility/accuracy is reported on the test set.

**Model deployment/Inference.** By feeding one unseen input data  $x_{\text{new}}$  into the ML function, one can get the prediction  $\hat{y} = f_\theta(x_{\text{new}})$ . This is called the inference stage. This inference could happen at both the *model provider* side or the *data owner* side depending on the trust and collaboration form between the model provider and the users. In typical cases of ML inference, the model needs to be deployed i) on a server-side device to support centralized service e.g., ML

inference as a service, or ii) on the users' devices for local inference distributively.

**Retraining/Updates.** ML models require retraining/updating their parameters, in order to adapt to data with unseen features over time. This usually is achieved by fine-tuning the model with newly collected data, especially on the last layers of the model when having limited data samples. Two examples of such a technique are i) back-propagation on the last layer only [Mar18] and ii) low-shot learning with weights imprinting on the last layer [QBL18]. The reason for updating the last layers preferentially is that the first layers tend to learn more general features that all input data should have, while the last layers learn more specific features that do not cover some new incoming data [PY09, YCBL14, ZF14].

### 2.1.2 Paradigm

Due to the increasing privacy/security concern raised by sharing data and network/computation-resource constraints, the stages of the ML pipeline are located to different participants for a better trade-off among utility, privacy, and cost. Based on the location of the main ML computation (e.g., training or inference), there are two main categories: centralized machine learning and federated machine learning.

**Centralized machine learning.** In centralized ML, the training or inference computations are conducted at a central place such as a server [ABC<sup>+</sup>16, YXW<sup>+</sup>17]. For both training and inference, the data have to be collected and stored at/near the center, and then the ML function performs training or inference on them. However, this raises privacy concerns about others' data. Therefore, centralized ML is usually chosen for the cases where the collected data are public (or non-private) or the data contributors are collaborators or have any form of trust/contracts with the center.

**Distributed/federated machine learning.** When data owners do not have the willingness to share data, distributed/federated ML is one solution, where, instead of the private data, the *ML model* is sharing among participants [LAP<sup>+</sup>14, MMR<sup>+</sup>17]. In ML inference, the model owner *distributes* its ML model to end-users, so that users can perform inference locally on their

devices. In terms of training, the terminology ‘federated learning (FL)’ [MMR<sup>+</sup>17, LSTS20, KMA<sup>+</sup>19] has been used intensively referring to the paradigm that users send the locally updated model parameters to the server without the need of revealing their data. Unfortunately, recent works have shown that adversaries can still execute attacks to retrieve sensitive information from the model parameters themselves [GBDM20, MSDCS19, ZLH19a].

### 2.1.3 Attack surface

The attack surface can cover many places within the complete pipeline. Here, I categorize existing attacks based on the core underpinning of information security<sup>2</sup> into: i) *confidentiality*, i.e., privacy of data and intellectual property of models, ii) *integrity* of the ML process. The attacks on every vulnerability in the ML pipeline can be viewed in light of confidentiality and/or integrity.

**Confidentiality-related attacks.** These attacks are curious-but-honest, i.e., interested in exploring (unauthorized) sensitive information of data and the model, but honestly performing ML training/inference without changing the computation results. They usually exist in the pipeline stages of model training, model deployment/inference, or model updates, due to that data and models owned by different participants have to be used. In centralized ML, without advanced protections, the computation host can directly access incoming data [RGC15, HTGW18]. Similarly, in federated ML, the host who orchestrates all clients’ local training can access their updated models and further infer private information about their local data using these models [GBDM20, MSDCS19, NSH19]. Several common attacks are i) Data Reconstruction Attacks [ZLH19a, GBDM20, HAPC17], aiming at reconstructing original input data based on the observed model or its gradients, ii) Attribute Inference Attacks [MSDCS19, JG18], aiming at inferring the value of users’ private properties in the training data, and iii) Membership Inference Attacks [NSH19, SSSS17], aiming at learning whether specific data instances are present in the training dataset. As one example, if one patient has medical-related images that have

<sup>2</sup> The full CIA triad includes Confidentiality, Integrity, and Availability, but the attacks specifically aim at ML system’s Availability, e.g., Denial-of-Service (DoS) attacks, have not drawn much attention from both industry and academia currently.

been used to train a ML model, without observing the original images, one adversary aims to disclose private information from the model. By performing data reconstruction attacks, it reveals the original images directly; by performing attribute inference attacks, it reveals the properties of the images such as the patient’s gender, race, etc.; by conducting membership inference attacks, it discloses whether these images have been participating in training the ML model.

On the other side, the *confidentiality of models* can be targeted. A model that has been trained with great effort (e.g., computation power, data cleaning, and collection) is considered the intellectual property of its owner, e.g., machine learning as a service (MLaaS) hosted by Amazon Web Services (AWS). In such cases in inference, one possible attack, called model stealing, is to counterfeit the functionality of this model by exploiting black-box access [OSF19, YYZ<sup>+</sup>20, TZJ<sup>+</sup>16], e.g., querying a large number of prediction results.

**Integrity-related attacks.** There are dishonest attacks that aim at actively changing the training/inference results, which harms the ML integrity. One famous attack is the adversarial example [GSS14, MDFFF17]. By adding calibrated noises to one image/audio, it leads to wrong prediction results but the input with noisy perturbation is invisible to human beings. Although this attack manipulates input data, it breaks the integrity of ML inference. Another example is the model poisoning attack in FL that could lead to Byzantine fault [FCJG20]. Such fault can easily occur because the server does not control all clients’ local training and cannot verify that their behaviors follow the promised training processes. For example, if one error in values of local updated parameters has been aggregated into the global model, the complete global model will become unusable. Furthermore, one can manipulate the training set e.g., by adding data with calibrated noisy labels [TTGL20]. This fools the classifier to have wrong boundaries for some specifically chosen data points. Backdoors can be added in such a way, and the attackers/other users will be able to trigger such backdoors in later use after model deployment [LMBL20, BVH<sup>+</sup>20].

**While-box or black-box.** Another wide-used way to understand attack surfaces is based on whether one attack requires access to the internal architecture of a ML model [PMSW18,



SDS<sup>+</sup>19]. Regarding the black-box attack, for example, model stealing usually starts from the outside without any a-priori information and aims to learn the model itself, which is black-box. Membership inference attacks on data privacy are usually black-box because it is already efficient according to previous research [SDS<sup>+</sup>19, YGFJ18], and white-box access does not significantly increase the attack ‘advantage’. The white-box attack includes almost all data reconstruction attacks, some adversarial example attacks, attribute inference attacks, etc. These attacks usually require detailed model parameters/gradients in order to be performed or have reasonable performance. The readers are referred to [PMSW18] for more details about security and privacy vulnerability in ML.

## 2.2 Confidential Computing and Trusted Execution Environments

Confidential computing (CC) [Pro] is a quickly emerging technology that provides a level of assurance of confidentiality and integrity when executing codes on data using Trusted Execution Environments (TEE). Nowadays, most Cloud service vendors have started providing CC services (e.g., Google Cloud, Microsoft Azure, etc) using the TEE solution supplied by processor manufacturers such as Arm, Intel, and AMD. This section first presents the threat model considered in CC, its components, and then Software Development Kits (SDK) useful for ML application developers.

### 2.2.1 Threat models

The CC provides a higher level of trust than normal execution by considering a stronger adversary who has full privileged access to the normal Rich Operating System (OS) [Glo15]. This adversary could be the device owner itself, malicious third-party software installed on the devices, or a malicious or compromised OS. Indeed, in CC, a group of individuals can be mutually mistrusting. When one individual demands a remote host to perform computation

with its (sensitive) code and/or data, the remote host usually can directly access the code and data and consequently learn sensitive information from it.

The TEE [Glo15], an isolated environment running in parallel with the Rich OS, allows to securely store data and execute arbitrary code on an untrusted device almost at native speed through secure memory compartments. The recent commoditization of TEEs both in high-end and mobile devices makes it an ideal candidate to achieve confidentiality and integrity in ML. However, TEEs are vulnerable to side-channel attacks [CVM<sup>+</sup>21], physical attacks [LKO<sup>+</sup>21], those that exploit weaknesses in TEEs [LJJ<sup>+</sup>17] and their SDKs [Van19], etc [CSFP20]. Also, Denial-of-Service (DoS) attack on the availability is hard to defend against and is not considered within the standard TEE threat model.

### 2.2.2 Key components

**Trust establishment and attestation.** The CC establishes a high level of trust among mutually distrusting individuals based on the TEEs and (remote) attestation. Its Root of Trust (RoT) is based on hardware-assisted cryptographic keys (usually called Endorsement keys), which will be attested to ensure the integrity of the TEE system [Glo15, CD16]. Thus, one local individual can trust a remote TEE; after it transmits its data or codes to another participant (i.e., the host)'s TEE, the computation host with root privilege cannot access or counterfeit the promised execution.

Remote attestation [CD16, HCF04] is the method that lets the user determine the level of trust/integrity of the remote TEE. It enables a user to authenticate the hardware and software configuration of a remote host, and check that the intended software is securely running inside the TEE. This attestation is usually conducted by a third party besides the user and host [CGL<sup>+</sup>11]. A typical attestation server is the processor manufacturer/vendor such as the SGX attestation service provided by Intel [JSR<sup>+</sup>16]. Thus, by default, the assumption is that the host who uses the SGX equipment, e.g., a Cloud, is not Intel and does not collude with

Intel, which also usually is the case in practice.<sup>3</sup>

**Trusted Execution Environments.** The core component in CC is the isolation called Trusted Execution Environments [Glo15]. In general, the TEE is a secure region inside the main processor. Both hardware and software approaches are used to isolate this region. TEE implementations differ from each other for different processor manufacturers (e.g., Arm, Intel, AMD). Specifically, implementing all these TEEs requires hardware-level modifications<sup>4</sup>, and due to that processors are different implementations of the Instruction Set Architecture (ISA), modifications on the hardware for TEEs, consequently, the changes on the software stack, cannot follow the same design. For example, Intel Software Guard eXtensions (SGX) [CD16] only has secure aware components inside the CPU including Instruction Decoder, Page Miss Handler, and Memory Encryption Engine; Arm TrustZone [MBG19, NMB<sup>+</sup>16] includes Processor and AXI bus with secure bit extension as the main components inside its System on Chip (SOC), and more other components such as Interrupt Controller, TrustZone Memory Adapter (TZMA), etc.

Table 2.1 summarizes common hardware-assisted TEEs with their providers, supported processors, and other features. Among them, Arm TrustZone is developed from a very early period; therefore, it is the most widely-used TEE on mobile/ubiquitous devices but constricted with the smallest secure memory size, the only single secure world supported, and restricted third-party application-level changes. To overcome it, Arm proposes Confidential Compute Architecture (CCA) working in parallel with TrustZone. Most recent TEEs support multi-enclaves and larger secure memory sizes with different Trust Compute Base (TCB) sizes. TEEs that include an OS inside a software stack usually have a large TCB size. However, a larger TCB size can probably provide higher usability to developers and users. For example, if the TEEs consider *kernel*, instead of *application*, as the highest access level, there is no requirement to change when deploying normal applications into TEEs.

---

<sup>3</sup> Nitro Enclave provided by AWS uses its own remote attestation service (e.g., AWS Key Management Service). In this case, one AWS Nitro Enclave customer provides assurances to their downstream customer. <sup>4</sup> There are software-assisted isolation techniques without the needs of hardware supports (e.g., seL4), but we usually do not consider them as they have lower security assurance and are not the mainstream in confidential computing.

Table 2.1: Summary of hardware-assisted Trusted Execution Environments

Open-source Hardware TEEs	ISA & Year	Supported Processors	Highest Access	Number of Isolation	TCB Size (excl. CPU/Soc pack.)	Secure Memory Size	Attestation	Application	App. Un-changed	Protections
Software Guard Extensions (SGX)	Intel (2013)	6th Intel CPU +	Ring 3 (app)	multi-enclaves	small. Page Table Manager	up to 128MB	remote	desktop-level	○	confidentiality integrity
TrustZone (TZ)	Arm (2005)	ARMv6-A + / Armv8-M +	EL-0 (app)	single secure world	large. Trusted OS, firmware	typically up to 16MiB 64MiB	-	mobile-level desktop-level	○	confidentiality integrity
Secure Encrypted Virtualization (SEV)	AMD (2016)	AMD EPYC +	Ring 0 (kernel)	multi-VMs	large. VM's OS, firmware	up to available system RAM	secure processor	enterprise-level	●	confidentiality
Physical Memory Protection (PMP)	RISC-V (2017)	Xilinx Artix-7 SiFive E31, U54	U-mode (app)	multi-enclaves	changeable. Runtime, firmware	changeable	-	mobile-level desktop-level	○	confidentiality integrity
Nitro Enclaves	AWS (2020)	Nitro Cards & Chip (EC2)	Ring 0 (kernel)	multi-enclaves	changeable. (OS), Nitro Hyper.	up to available instance RAM	remote	desktop-level enterprise-level	●	confidentiality integrity
Confidential Compute Arch. (CCA)	Arm (2021)	Arm9-A (not release)	EL-0 (app)	multi-realms	large. OS kernel, firmware	up to available system RAM	remote	mobile-level desktop-level	●	confidentiality integrity

Application Unchanged: ○ Major changes needed; ● Light changes needed; ● No change needed

Table 2.2: Open source framework for Trusted Execution Environments

Open Source TEE Framework	Leading	Supported TEEs	On Top Of	Main Functions	Attestation	Languages	ML Examples
Intel SGX SDK [int]	Intel	SGX	firmware & driver	API and librars for SGX app.	●	C/C++	●
OP-TEE [opt]	Linaro	TrustZone	firmware & driver	API and librars for TZ app.	○	C/C++	●
MultiZone [mul]	HEX-five	RISC-V	firmware & driver	API and librars for RISC-V Enclave app.	●	C/C++	○
Keystone [key]	UC Berkeley	RISC-V	firmware & driver	API and librars for RISC-V Enclave app.	●	C/C++	●
Open Enclave SDK [ope]	Microsoft	SGX, TrustZone	SGX SDK & OP-TEE	generalized API for TEE app.	●	C/C++	●
Asylo [asy]	Google	SGX	SGX SDK	generalized API for TEE app.	●	C/C++	●
Teaclave [tea]	Apache	SGX	SGX SDK	API for Rust programming	●	Rust	●
Gramine [gra]	Invisible Things Lab / Intel	SGX	SGX SDK	lightweight library OS to host app.	●	Native exec*	●
Occlum [occl]	Tsinghua / Ant Group	SGX	SGX SDK	lightweight library OS to host app.	●	Native exec*	●
SCONE [sco]	Scontain	SGX	container & driver	TEE-based container image generation	●	Docker image*	●
Enarx [ena]	Red Hat	SGX, SEV, CCA	firmware & driver	WebAssembly sandbox	●	Wasn binary*	○
Veracruz [ver]	Arm	SGX, TrustZone, Nitro Enclaves, CCA	SGX SDK & OP-TEE Nitro Enclaves SDK	proxy attestation, policy WebAssembly sandbox	●	Wasn binary*	●

Attestation: ○ Not available; ● Self-configuration required; ● Work off-the-shelf \* Most modern languages supported by compiling as a target executable. ML examples: ○ Not exist; ● Exists However, One non-exist example can be developed using existing ML framework with corresponding languages.

### 2.2.3 Partitioning frameworks for developers

With the low-level hardware and system stacks, it is still hard for application developers to utilize TEEs. Hence, depending on the use-cases, various software vendors provide deployment and partitioning frameworks for running a diverse set of workloads within enclaves. For example, more recently Amazon Nitro enclaves and Red Hat’s Enarx support running the same binary within different types of TEEs. These systems are inspired by Haven[BPH15] that ports Drawbridge[PBWH<sup>+</sup>11], a Windows library OS, inside an SGX enclave. Similarly, Graphene-SGX [TPV17] ports the Linux-based Graphene library OS in an enclave. Scone [ATG<sup>+</sup>16] tries to reduce TCB by porting musl-libc and a portion of Linux Kernel Library (LKL) <sup>5</sup>. TrustShadow [GLX<sup>+</sup>17] also protects unmodified applications from the host OS by trapping applications exceptions and system calls inside TrustZone, passing the calls to the OS, and verifying the output after parameter marshaling.

As another approach, various TEE partitioning frameworks are proposed to split applications into trusted and untrusted components; such as Intels SGXSDK [Cor19a], Microsofts Open Enclave [Cor19b], Googles Asylo [Goo18], OP-TEE [opt], and Keystone [LKS<sup>+</sup>20]. There are also language-specific partitioning frameworks such as Civet [TSJ<sup>+</sup>20] for porting Java classes into an SGX enclave, TLR (Trusted Language Runtime) [SRSW14] for running portions of C# applications inside TrustZone, and Glamdring [LPM<sup>+</sup>17] a compiler for partitioning applications into SGX enclaves via code annotation. Table 2.2 summarises primary commercial TEE frameworks. However, not all of them are feature-rich enough to support ML use cases. Particularly, mobile vendors (e.g., Qualcomm TEE [Qua19], Trustonic TEE [Tru], or Huawei TEE [BWM20]) only allow for limited TEE operations such as secure storage to avoid security risks inside TrustZone secure world which enables attackers to take full control of the device. This is a key reason for TEE applications to utilize SGX due to less privileged execution of enclaves in userspace as well as ease of programming.

---

<sup>5</sup> sgx-lkl <https://github.com/llds/sgx-lkl>.

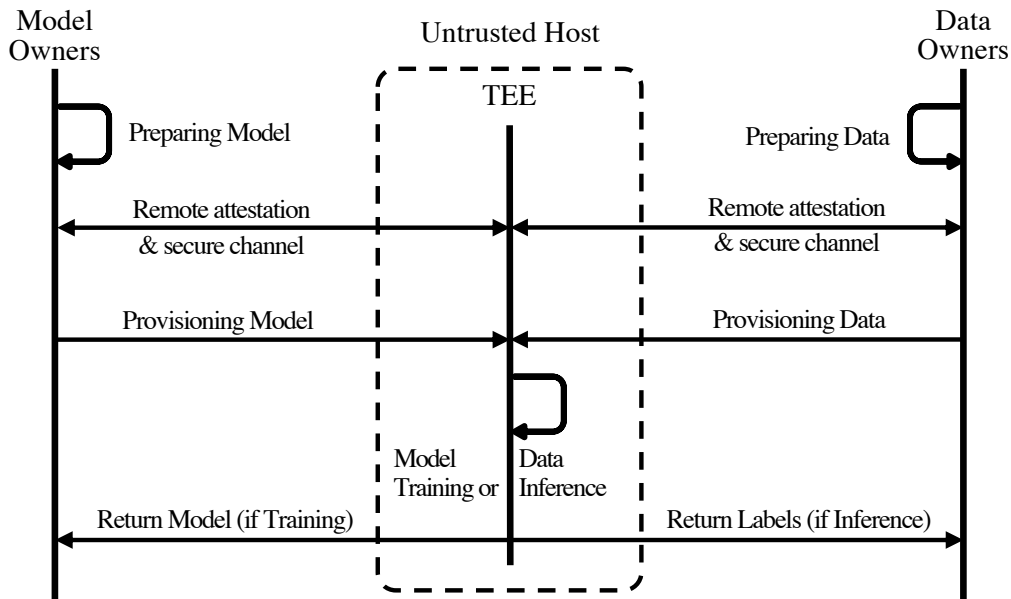


Figure 2.1: Overview flow of Confidential Computing which utilizes the untrusted host’s Trusted Execution Environment to protect the model and data in machine learning.

## 2.3 Confidential Computing Solution for Machine Learning

### 2.3.1 Overview

In CC, data/model owners provision their data/model to the untrusted host’s TEE for ML (see Figure 2.1). Specifically, after they finish the preparation of the model and/or data, they perform remote attestation to assure the integrity of the remote TEE, and then establish secure channels with the TEE. Afterward, data/models are provisioned to the TEE, where model training or inference will be performed. The final results will be returned, i.e., a trained model will be transmitted out in training, or the data label will be returned back to the user in inference. Note that in practice the host, model owners, data owners, and result receivers can be distinct entities, and thus, some may not exist in some cases. I further classify these cases according to the nature of the host: Server or Client (see Figure 2.2).

**Server-side ML protection.** A server-like host aims to provide ML service to its customers. Based on the specific ML function, it can provide i) Inference as a Service (e.g., [GTS<sup>+</sup>18,

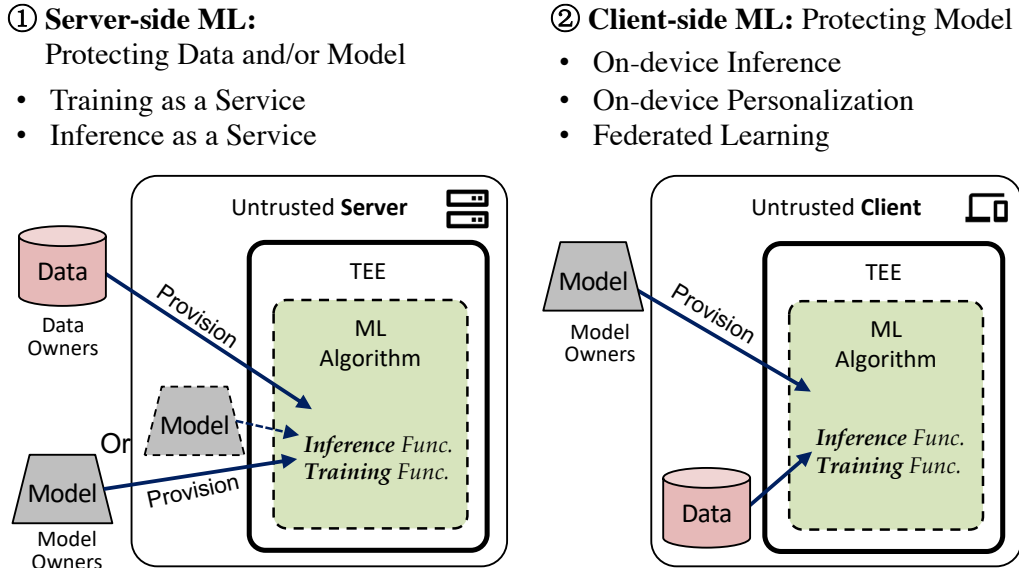


Figure 2.2: Server-side ML (Left) and client-side ML (Right) protection using Trusted Execution Environments.

LLP<sup>+</sup>19]) or ii) Training as a Service (e.g., [HSS<sup>+</sup>18, HCS18]). In both cases, schemes should protect data privacy. That is, data have to be secretly provisioned to the untrusted server’s TEE no matter for inference or training. However, the ownership of the ML model can belong to not only a third party (e.g., extra model owners/providers) but also to the untrusted server itself. The latter case only exists for ML inference, because inference does not change the model parameters. In contrast with it, training leads to updates on the model parameters, which can consequently leak private information of the data.

**Client-side ML protection.** A client-like host usually acts as the downstream user of a server, and thus it performs ML based on some ML functions or models provided by the server. Several common use cases are i) on-device inference (e.g., [BFJ<sup>+</sup>20]), the client conducts predictions on its own data with another individual’s model, ii) on-device personalization (e.g., [MSK<sup>+</sup>20]), the client personalizes the model based on its data for later inference, and iii) federated learning (e.g., [MHK<sup>+</sup>21]), the client trains the model continuously to contribute a global model owned by another party (e.g., server). In all these cases, the client owns the data. However, the model could be considered to be confidential, e.g., the model itself can leak private information, or as intellectual property, e.g., the model costs its owner or the server significant effort to train.

### 2.3.2 Key challenges

The key challenges mostly lie in the feasibility and effectiveness of deploying ML into TEEs due to TEEs' limitations.

**Constrained execution environments.** TEEs usually have constrained resources for trading off the size of TCB. As given in Table 2.1, commonly used hardware TEEs, e.g., SGX, provide 128MB secure memory, while TrustZone only provides up to 16~64MiB secure memory depending on software stack implementations. Such memory sizes are highly limited compared to the memory consumed by current ML algorithms, which can reach hundreds of MBs or GB. Although page swapping could increase available memory, it leads to significant overhead (e.g., 100~1000×) [LLP<sup>+</sup>19, KQG<sup>+</sup>19]. In addition, processor capabilities are also limited when running in the TEE mode. For example, TrustZone supports single-threading only. SGX supports multi-threading but this could lead to bugs that can be exploited by attackers (e.g., AsyncShock [WKPK16]). Indeed, increasing computational resources is always preferable for ML developers but it can reduce the reliability of the TEE.

**SDK supports.** Modern ML framework typically requires numerous libraries and cross-compilation to support efficient data loading and computations e.g., matrix multiplication. However, most TEEs that are built on hardware only provide basic low-level SDKs. Even though open source TEE frameworks have been developed to support sandbox/container (see Table 2.2), it is still hard to port all necessary libraries into it. For example, WebAssembly supports compiling binaries from many modern languages, e.g., Rust, which could have available ML libraries, such as autograd [Asa], but they have much-limited functionality compared to Tensorflow or PyTorch [PGM<sup>+</sup>19]. OS-based containers can provide richer libraries, but still, the lightweight version OS cannot have a rich environment same as the normal OS. Porting a normal OS to TEEs is also infeasible because it either exceeds the TEEs' size limits or leads to a huge TCB.

**Privacy protection effectiveness.** By deploying ML inside TEEs (for training/inference), the untrusted host itself cannot access the ML computations anymore. However, running



inside TEEs does not automatically disable all potential privacy leakages. This is because the other parts of the ML pipeline can leak private information somehow, and due to that the produced result, e.g., trained models or predicted data labels, will be transmitted out of the TEE and can be used to exploit private information [ZLH19a, MSDCS19, JSB<sup>+</sup>19]. Therefore, one particular strategy could have inconsistent protection effectiveness, for different ML scenarios and different types of defined privacy [MBM<sup>+</sup>21a, MBM<sup>+</sup>21b]. As one example, while hiding model parameters (along with gradient updates and activations) inside TEEs can defend against data reconstruction attacks, it has very low efficiency for membership inference attacks as most membership information can leak from the model’s outputs, i.e., the prediction results transmitted out of the TEE [JSB<sup>+</sup>19, SDS<sup>+</sup>19, GHZ<sup>+</sup>18b]. Therefore, for one specific TEE use case, the target privacy should be defined clearly, and the achieved protection efficiency also requires explicit analyses.

### 2.3.3 Existing solutions

Previous research has been dedicated to achieving ML with TEEs, and some aim to overcome the above challenges. Previous research is summarized in Table 2.3.

**Complete ML training/inference inside TEEs.** The most straightforward way is to deploy the complete ML training/inference process inside TEEs. In such a case, the maximum capability of the ML task is strictly limited to the space constraint. The strategy to take is to use memory space efficiently – trading off the total number of the model’s layers and the number of neurons – and to maximize the ML efficiency of each “bit” inside TEEs. Specifically, several practical approaches are listed as follows. i) Conducting inference instead of training. Training consumes much more computational resources because of backward propagation (e.g., memory used to save model gradients). ii) Choosing a small batch size. A large batch size leads to large memory usage because every sample in this batch produces its own activations for all model layers. iii) Balancing the feature extractor (e.g., convolutional layers) and the classifier (e.g., fully connected layers). A properly designed feature extractor can decrease the feature dimension and capture key features, enabling a small classifier to achieve good performance.

Table 2.3: Previous research that uses Trusted Execution Environments to guarantee the confidentiality of machine learning

Work & Year	Trusted Computing		Machine Learning				Advanced Features
	TEE Type	SDK	DL Framework	Training	Location	Protect Aim	
Obliv. MP'16 [OSF+16]	SGX	Intel SGX SDK	fast CNN	●	S	D	Data-obliviousness
Chiron'18 [HSS+18]	SGX	Intel SGX SDK	Theano	●	S	D	Multi-enclaves, Ryoan VM
PRIVADO'18 [GTS+18]	SGX	Intel SGX SDK	ONNX	○	S	D	Compiler, data-obliv.
Myelin'18 [HCS18]	SGX	Intel SGX SDK	TVM	●	S	D	DP for data obliviousness
Slalom'18 [TB18]	SGX	Intel SGX SDK	Eigen	○	S	D	Freivalds algorithm + GPU
Graviton'18 [VVB18]	GPU-TEE	CUDA RT driver	Caffe	●	S	D	secure CUDA for TEE on GPUs
Occlumency'19 [LLP+19]	SGX	Intel SGX SDK	Caffe	○	S	D	On-demand loading, Channel-based partition
TensorSCONE'19 [KQG+19]	SGX	Intel SGX SDK	Tensorflow	●	S	D	Docker supported, Compiler
YerbaBuena'19 [GHZ+19]	SGX	Intel SGX SDK	Darknet	○	S	D	Layer-wise partition, Privacy measurement
OMG'20 [BFJ+20]	TrustZone	SANCTUARY	TFLite Micro	○	C	M	-
DarkneTZ'20 [MSK+20]	TrustZone	OP-TEE	Darknet	●	C	M	Layer-wise partition, Privacy measurement
TrustFL[ZLZ+20]	SGX	Intel SGX SDK	TensorFlow	●	C	TI	GPU-outsourcing, Random sampling
Telekine'20 [HJM+20]	GPU-TEE	ROCm & CUDA	MXNet	●	S	D	Timing attacks, data-obliv.
MLCapsule'21 [HZG+21]	SGX	Intel SGX SDK	Eigen	○	C	M	-
Trusted-DNN'21 [LLX+21]	TrustZone	OP-TEE	-	○	C	M	Weights & Feature-map partition
Mem-Eff.'21 [TGGW21]	SGX	Azure CC (VM)	Darknet	○	S	D	Channel & Y-plane partition
PPFL'21 [MHK+21]	SGX + TrustZone	OE SDK + OP-TEE	Darknet	●	C + S	D	Layerwise training, Privacy measurement
Goten'21 [NCW+21]	SGX	Intel SGX SDK	Eigen	●	S	D	GPU-outsourcing, Non-colluding servers
Citadel'21 [ZXY+21]	SGX	SCONE	Tensorflow	●	S	M + D	Multi enclaves for training and aggregation

Training: ○ Not available; ● Available

Location: S = Server; C = Client

Protection Aim: D = Data privacy; M = Model intellectual property; TI = Training integrity

Here note that most modern neural networks being tested during the evaluation of these works are such convolutional neural networks because of their wide range of applications. That is, convolutional layers are well designed for feature extraction, and fully connected layers are suitable for the classifier atop the former extractor.

**Partitioned execution.** To avoid exceeding the maximum secure computational resource and memory swapping, one way is to utilize partitioned execution to actively optimize the memory usage in ML. Figure 2.3 shows two types of partitioned execution: i) Layer-based partition [GHZ<sup>+</sup>18b, MSK<sup>+</sup>20, MHK<sup>+</sup>21], which works for models with layer architecture in general, and ii) Feature map-based partition [LLX<sup>+</sup>21, TGGW21], which specifically aims to convolutional layers due to their high memory cost. Among them, whether to keep the first part or the last part inside TEEs (cases ① and ②) depends on the privacy protection goal. Rolling on layers inside TEEs (case ③) naturally works for ML inference, because one input propagates forward throughout layers and never goes back; that is, it works in the way that the next layers are loaded into a TEE until one layer’s computation executes finished in the TEE. Rolling on feature maps (cases ④ and ⑤) reduce the realtime memory usage by applying `GEMM` (i.e., General matrix multiply) and `img2col` (i.e., Image to column format transform) functions on divided sections and channels of feature maps. However, this disables parallelization and can increase computation time. Furthermore, due to that training involves backward propagation where more resources are required, partitioned execution is more necessary than inference. This could lead to higher overhead for cases ①, ②, and ③, because a higher volume of buffer is transmitted between the TEE and normal OS. Note that the partition should not change the model architecture (unlike the model partition in the design of AlexNet [KSH12]), in order to enjoy higher flexibility to support generic ML models.

**GPU outsourcing.** In addition to optimizing the limited secure space, one can also extend the space, e.g., to outsource the computation to faster GPUs. However, GPUs are untrusted; one way to address this issue is to use CPU-TEE (e.g., SGX) to coordinate the computation delegation and to establish a secret sharing protocol between the CPU-TEE and untrusted GPUs. In such a way, GPUs are able to perform executions on a “nonsensitive version” of the

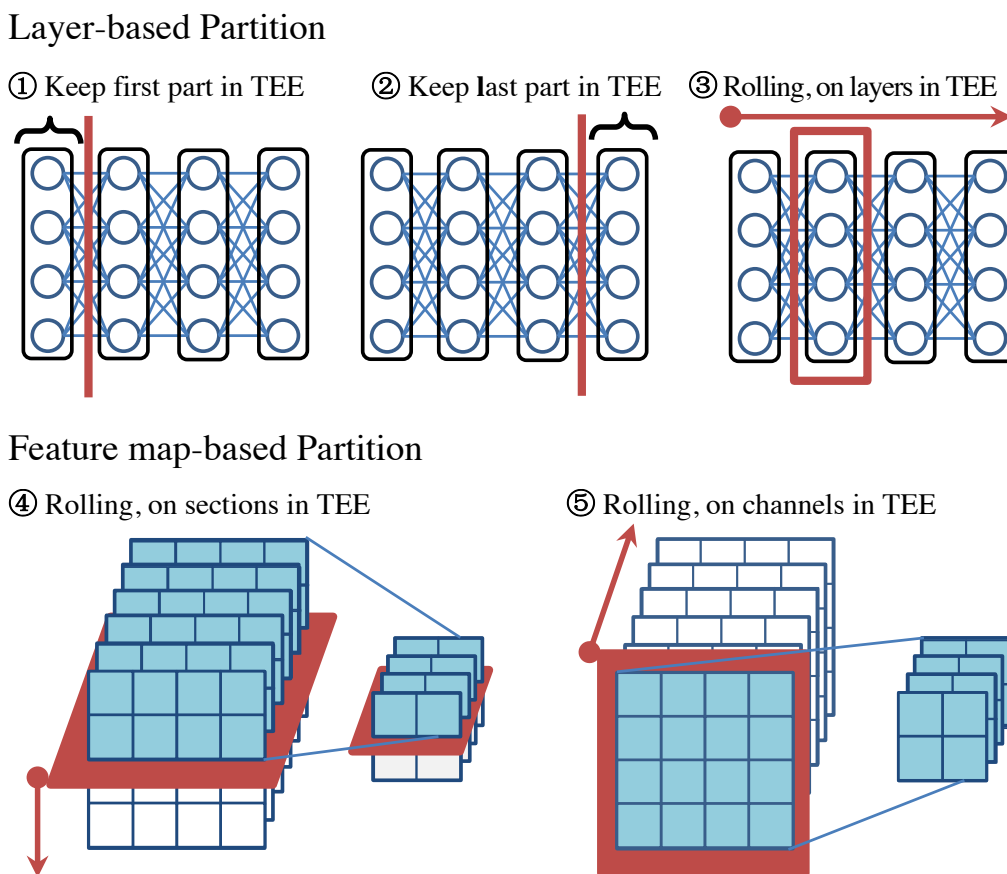


Figure 2.3: Partition on the ML process (shown with model architectures) in order to protect it or a part of it inside TEEs.

(TEE) computation, e.g., masked by one-time pads [TB18, NCW<sup>+</sup>21]. This can also accelerate the ML’s linear computation i.e., matrix multiplication. Another way is to support the TEE on GPUs, e.g., GPU-TEE. Previous research [VVB18, HJM<sup>+</sup>20] achieves GPU-TEE by providing the necessary software stack such as runtime, drivers, API, etc. Since existing GPU-TEE solutions are on software only, it lacks a hardware-based root and thus has a lower level of trust. Also, GPU outsourcing may weaken the trust model, which deserves careful investigation in a later stage of research.

**Attacks for privacy measurement.** Measuring the protection effectiveness is usually done by performing a form of attack directly. Whether to and which attack to perform depends on the threat model defined when using TEEs for protection. For example, for the original input protection with TEEs, attacks to reconstruct input data are performed [GHZ<sup>+</sup>18b], while there is no need to conduct membership inference attacks. By contrast, TEE protection on

membership information does not take care of other private information and only be measured with membership inference attacks [MSK<sup>+</sup>20]. As far as we know, there is still a lack of theoretical analysis on privacy leakage when utilizing TEEs for protection.

## 2.4 Confidential Computing Helping Machine Learning Integrity

Intuitively, deploying ML training/inference process inside TEEs can avoid malicious modifications to this process and therefore provides integrity. However, in practice, as the other parts of the ML pipeline still can be breached, protecting only the training/inference stage does not always guarantee integrity. Indeed, for partitioned execution to tackle space-constrained problems, the training happening outside can also be modified maliciously. In this section, we present the key challenges in guaranteeing ML integrity with TEEs and the solutions that exist or can be adapted.

### 2.4.1 Key challenges

**Large attack surface.** There is a large attack surface to breach integrity; however, one cannot expect to deploy the complete ML life-cycle inside TEEs to achieve an integrity guarantee due to the TEE’s small size. This resource limit issue is even more serious compared with the issue we met when guaranteeing confidentiality. Specifically, while a curious-but honest (i.e., confidentiality-related) adversary has the goal to explore specific private information, the integrity-related adversary can be diverse – it cannot only aim at actively changing the ML process but also just want to break the process by covertly changing one bit. Indeed, to disclose confidential information, adversaries usually explore the target data or the model which contains memorized information. Remained information may also gradually decrease along with more aggregated information in training. However, to breach integrity, one just needs to change one step in the training process (even in one bit) among participants (e.g., Byzantine

attacks [FCJG20]), and this malicious change can also be triggered at other stages e.g., data preparation, etc.

**Uncontrolled input/output space.** While deploying ML training/inference inside TEEs can avoid unauthorized direct changes in the produced result/models, the upstream pipeline, i.e., input space, is uncontrolled. Editing the inputs and their labels can affect the later ML process (e.g., called “dirty” inputs/labels), which is also the practice of performing some attacks (e.g., poisoning attacks). Adversarial examples [GSS14, MDFFF17] to add calibrated noises can seriously perturb the integrity of the input space because the noisy perturbation is hard to be detected. In addition to that, the downstream pipeline can also be edited maliciously, e.g., an server-side adversary fakes the result after the ML algorithm inside the TEE makes a prediction and transmits this result out.

**Low interpretability of ML.** The difficulty of detecting integrity perturbation is also caused by the low interpretability of current ML/DL [LXL<sup>+</sup>21]. Specifically, current practitioners train ML models using SGD, i.e., weights of neurons are updated iteratively to find the optimal solution by “itself”. Also, there is no comprehensive theoretical framework to interpret the internal steps of ML training. Therefore, one cannot determine whether the weight value of a neuron is uncompromised or not without rerunning the ML process again using the same setting. With such low interpretability, malicious changes, such as the adversarial perturbation in inputs and the changes in model training, are hard to trace and detect. Currently, algorithm-based approaches are being proposed to alleviate this situation, such as by removing outliers that are considered unreasonable bits [FCJG20], by further training the model on adversarial inputs [TKP<sup>+</sup>18], or by cryptography protocols [CGJvdM21]. However, using a system-based isolation technique (i.e., TEEs) to deal with the interpretability issue is not straightforward. One may still need to find a way to assure key elements of the ML pipeline is trustworthy by using TEEs.

## 2.4.2 Existing/adoptable solutions

**Assurance mechanisms.** To assure the integrity of ML training/inference, one way is to rerun the complete or a part of the process inside TEEs. As an example, the heaviest computation in ML is the matrix multiplication which is preferable to be outsourced to distrusted GPUs for acceleration. To verify whether this matrix multiplication is honestly performed, one can use Freivald’s algorithm which utilizes randomization to reduce time complexity from  $\mathcal{O}(n^{2.373})$  (best-known matrix multiplication algorithm) to  $\mathcal{O}(kn^2)$  with a probability of failure less than  $2^{-k}$  [MR96, TB18]. Besides, based on the nature of iterative training of ML, one can also sample and verify only a fraction of the training process in TEEs to reduce the overhead [ZLZ<sup>+</sup>20]. This will require backups for every several training iterations to reach a checkpoint faster, where Merkle hash tree-based method [Mer80] can be used to reduce the storage overhead. It is shown to be sufficient that the soundness error is less than 1% even for the adversaries who have honestly completed 90% of training rounds [ZLZ<sup>+</sup>20]. One can expect that there will be more efficient ways of sampling and checking, but such assurance only guarantees the integrity of the training/inference process and does not provide the guarantee for other stages in the ML pipeline.

**Input/output space control.** The integrity of input/output in the ML pipeline needs to be assured considering that the input/output space is far more uncontrollable than the training/inference process. One way that is generally used for data assurance and can also be adopted in ML input control is to require the digital signature for the data generation. This is done by hashing the generated sensor data (inside TEEs for example) to produce hash values, and after that, sensor data can be authenticated using this hash value [Mer89, Coh87, KFPC16]. At a later stage, the digital signature would also work in a similar way as remote attestation when outputting prediction results, considering that the result receiver trusts the TEE’s behavior. However, it is not easy especially for supervised learning because the data generation usually involves human annotation that is out of control of the digital signature. Even in semi-/un-supervised learning, it is possible to generate adversarial data without breaching sensors/TEEs, e.g., by changing physical surroundings or doing abnormal behaviors that

the ML does not expect to learn. Indeed, one can also adopt detection mechanisms, such as [BF99, CIKW16, BJG20], trying to identify fake/dirty inputs (e.g., inside TEEs). There is still a large area to explore in order to assure the integrity of the input space in the ML pipeline.



# Chapter 3

## Privacy Measure of Neural Network Weights

### 3.1 Introduction and Related Work

On-device DNNs have achieved impressive performance on a broad spectrum of services based on images, audio, and text. Examples include face recognition for authentication [VFGJ16], speech recognition for interaction [MPA<sup>+</sup>16] and natural language processing for auto-correction [BD17]. However, DNNs memorize the training data in their parameters information [ZBH<sup>+</sup>17, YGFJ18, ZF14]. Thus, keeping DNNs accessible in user devices leads to privacy concerns when training data contains sensitive information.

Previous works have shown that a reconstruction of the original input data is easier from the first layers of a DNN, when using for inference the layer's output (activation) [GHZ<sup>+</sup>18b, OST<sup>+</sup>18, OST<sup>+</sup>17]. In addition, the functionality of the parameters of each layer is different. For example, parameters of first layers trained (on images) output low-level features, whereas parameters of later layers learn higher-level features, such as faces [ZF14].

It is hypothesized that the memorization of sensitive information from training data differs across the layers of a DNN and, in this chapter to answer Research Question 1, I present an approach to measure this sensitive information. The result shows that each layer behaves

differently on the data they were trained on compared to data seen for the first time, by quantifying the generalization error (i.e., the expected distance between prediction accuracy of training data and test data [YGFJ18, SSSSS10]). This work further quantifies the risk of sensitive information exposure of each layer as a function of its maximum and minimum possible generalization error. The larger the generalization error, the easier the inference of sensitive information from training set data.

I perform experiments by training VGG-7 [SZ14] on three image datasets: MNIST [LCB10], Fashion-MNIST [XRV17], and CIFAR-10 [KH09]. The results show that the last layers memorize more sensitive information about training data, and the risk of information exposure of a layer is independent of the dataset.

To protect the most sensitive layers from potential white-box attacks [MSDCS19, HAPC17, NSH18], I leverage a resource-limited Trusted Execution Environment (TEE) [CBL<sup>+</sup>18, OSF<sup>+</sup>16, HSS<sup>+</sup>18] unit, Arm’s TrustZone, as a protection example. Experiments are conducted by training the last layers in the TEE and the first layers outside the TEE. Results show that the overhead in memory, execution time, and power consumption is minor, thus making it an affordable solution to protect a model from potential attacks.

## 3.2 Proposed Approach

### 3.2.1 Problem definition

Let  $M(\theta)$  be a DNN with  $L$  layers, parameterized by  $\theta = \{\theta_l\}_{l=1}^L$ , where  $\theta_l$  is the matrix with the parameters of layer  $l$ . Let  $X = \{\mathbf{X}_k\}_{k=1}^K$  be the training set of  $K$  images  $\mathbf{X}_k$ . Let  $S = \{\mathbf{S}_k\}_{k=1}^{K_s} \subset X$ , a randomly selected subset of  $X$  with  $K_s = \lfloor K/2 \rfloor$ , be the *private dataset* and  $T = \{\mathbf{T}_k\}_{k=1}^{K_t} \subset X$ , with  $T \cap S = \emptyset$ , be the *non-private dataset*.

As training  $M(\theta)$  on  $S$  might embed some information of  $\mathbf{S}_k$  in the parameters of each layer, this work aims to quantify the exposure of sensitive information in each  $\theta_l$ . The sensitive

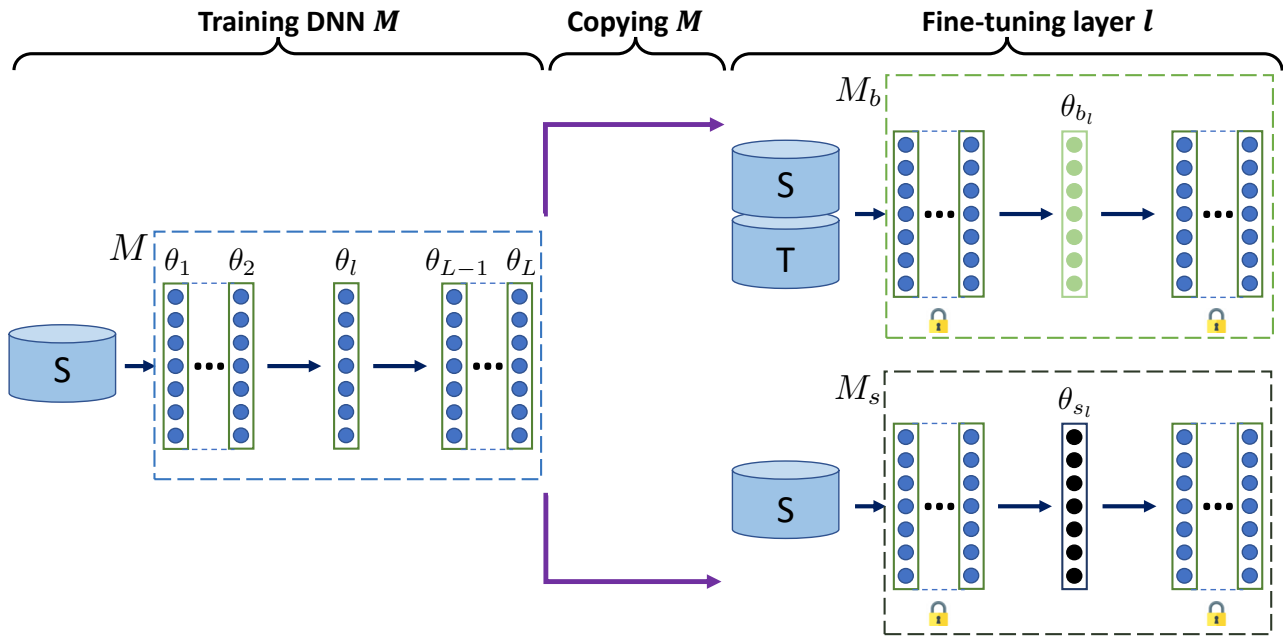


Figure 3.1: The proposed framework for measuring the risk of exposing sensitive information in a deep neural network  $M$  trained on a private dataset  $S$ .  $M_b$  and  $M_s$  are obtained by fine-tuning the parameters of a target layer  $l$  on the whole training set  $X$  (i.e., both  $S$  and non-private training set  $T$ ) and  $S$ , respectively.

information we are interested in analyzing is the absence or presence of any  $S_k$  in the training data.

### 3.2.2 Sensitive information exposure approach

This research leverages the fact that  $M(\theta)$ , trained on  $S$ , has a higher accuracy of predicting classes of data points from  $S$  than from another dataset,  $T$ . The difference in prediction accuracy indicates the generalization error [YGFJ18, SSSSS10] of  $M(\theta)$  and how easy it is to recognize whether a data point  $X_k$  was in  $S$  during training. I define the risk of sensitive information exposure of each  $\theta_l$  based on the maximum and minimum possible generalization errors (see Figure 3.1). A larger difference in the maximum and minimum of generalization error could show the more sensitive information exposure which results in inferring more accurately the absence or presence of data in the training data (i.e., membership inference attack [YGFJ18]).

To obtain the maximum generalization error, one can increase the chance of overfitting  $\theta_l$  to  $S$  by fine-tuning  $\theta_l$  and by freezing parameters of other layers of  $M$ . We call this model  $M_s(\theta_s)$ .

If  $C(M_s, \mathbf{S}_i)$  is the distance between  $M_s$  and  $\mathbf{S}_i$  measured by the cost function used in training, we quantify  $\epsilon_s$ , the generalization error of  $M_s(\theta_s)$ , based on its different behaviour on  $S$  and  $T$ :

$$\epsilon_s = \mathbb{E}_{\mathbf{T}_i \in T}[C(M_s, \mathbf{T}_i)] - \mathbb{E}_{\mathbf{S}_i \in S}[C(M_s, \mathbf{S}_i)], \quad (3.1)$$

where  $\mathbb{E}[\cdot]$  is the mathematical expectation.

To obtain the minimum generalization error without forgetting  $S$ , we can create a baseline  $M_b(\theta_b)$  by fine-tuning  $\theta_l$  on  $X$  and by freezing the parameters of the other layers of  $M$ . This fine-tuning makes  $\theta_{b_l}$  generalized on both  $T$  and  $S$ , which can be quantified as:

$$\epsilon_b = \mathbb{E}_{\mathbf{T}_i \in T}[C(M_b, \mathbf{T}_i)] - \mathbb{E}_{\mathbf{S}_i \in S}[C(M_b, \mathbf{S}_i)]. \quad (3.2)$$

$M(\theta)$ ,  $M_s(\theta_s)$  and  $M_b(\theta_b)$  share the same layers, except the target layer  $l$ . Therefore, the differences in each pair of

$$\{\mathbb{E}_{\mathbf{T}_i \in T}[C(M_s, \mathbf{T}_i)], \mathbb{E}_{\mathbf{T}_i \in T}[C(M_b, \mathbf{T}_i)]\},$$

and

$$\{\mathbb{E}_{\mathbf{S}_i \in S}[C(M_s, \mathbf{S}_i)], \mathbb{E}_{\mathbf{S}_i \in S}[C(M_b, \mathbf{S}_i)]\},$$

are due to different parameters of layer  $l$ .

Therefore, we quantify  $\mathcal{R}^{M_s}$ , the risk of sensitive information exposure of layer  $l$ , by comparing the generalization error of  $M_s$  and  $M_b$ :

$$\mathcal{R}^{M_s} = \frac{\epsilon_s - \epsilon_b}{\epsilon_s}. \quad (3.3)$$

The larger  $\mathcal{R}^{M_s}$ , the higher the risk of exposing sensitive information.

## 3.3 Measuring Information Exposure

### 3.3.1 Model and datasets

I use VGG-7 as the DNN  $M$ , which has seven convolutional layers (16C3-16C3-MP-32C3-32C3-MP-32C3-32C3-MP-64FC-10SM). Each layer is followed by Rectifier Linear Unit (ReLU) [NH10] activation function.

I use three datasets: MNIST, Fashion-MNIST, and CIFAR-10. MNIST includes 60k training images of  $28 \times 28 \times 1$  handwritten digits of 10 classes (i.e., 0 to 9). Fashion-MNIST contains 60k  $28 \times 28 \times 1$  images of 10 classes of clothing, namely T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. CIFAR-10 includes 50k training  $32 \times 32 \times 3$  images of 10 classes including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

I split each training set into set  $S$  and set  $T$ , as explained in Sec. 3.2.1. I use 20 epochs for MNIST, 40 epochs for Fashion-MNIST, and 60 epochs for CIFAR-10 to reach a relatively reasonable accuracy (similar to previous research [LCB10, KNHa]). The accuracy of VGG-7 on MNIST, Fashion-MNIST, and CIFAR-10 is 99.29%, 90.55%, and 71.63%, respectively. I then fine-tune  $M$  as  $M_s$  and  $M_b$  with 10 epochs for MNIST, 20 epochs for Fashion-MNIST, and 30 epochs for CIFAR-10. These epochs are half of the previous numbers in order to fine-tune models but do not overfit them.

### 3.3.2 Results and discussion

**Generalization error.** Figure 3.2 shows the generalization errors of  $M_s$  and  $M_b$ . For all three datasets, the baseline model  $M_b$ , as expected, has higher generalization errors than the model  $M_s$ , whose layer  $l$  is overfitted to dataset  $S$ , while the generalization error of CIFAR-10 is greater than that of Fashion-MNIST that in turn is greater than that of MNIST. A more complex dataset (e.g. CIFAR-10) is associated with a larger difference between  $S$  and  $T$

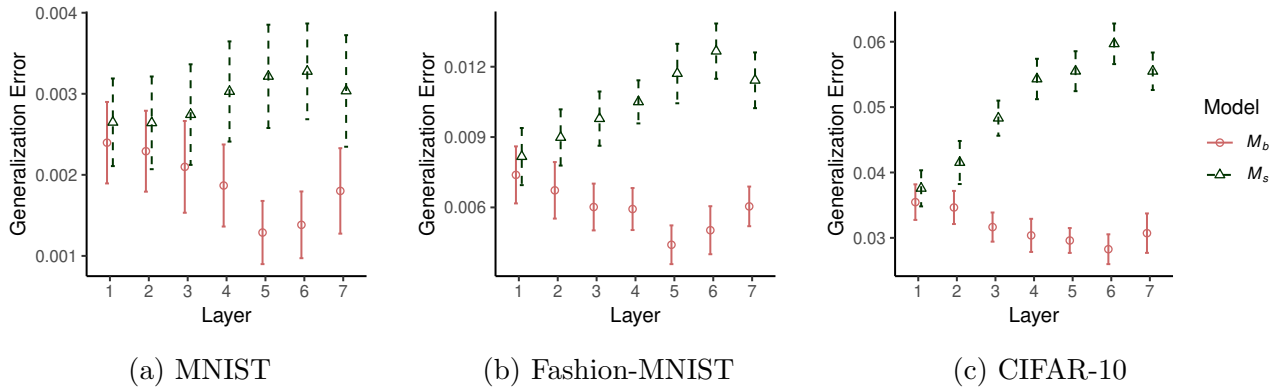


Figure 3.2: Generalization errors of  $M_s$  and  $M_b$  trained on half of the training set,  $S$ , of (a) MNIST, (b) Fashion-MNIST, and (c) CIFAR-10 for fine-tuning each target layer. Error bars represent 95% confidence intervals.

compared to a less complex dataset (e.g. MNIST), so it is harder to generalize the model to predict  $T$  by training with  $S$ .

As one goes through the convolutional layers, the generalization error of  $M_s$  increases, while the generalization error of  $M_b$  decreases until the 5<sup>th</sup> or 6<sup>th</sup> layer. A possible explanation is that the first layers memorize generic information (e.g. colors, and corners), whereas the last layers memorize more specific information that can be used to identify a specific image. For example, fine-tuning the last layers using  $S$  leads  $M_s$  to memorize specific information of  $S$ , which consequentially increases the generalization errors of  $M_s$  when predicting  $T$ .

**Sensitive information exposure.** Figure 3.3 shows the risk of sensitive information exposure for each layer of VGG-7 on all three datasets. The first layer has the lowest risk, and the risk increases as one go through the layers, with the last convolutional layer having the highest sensitive information exposure, which is 0.63 for both MNIST and Fashion-MNIST and 0.5 for CIFAR-10. This confirms the bigger derivation of the generalization error of  $M_s$  from  $M_b$  in the last layers than in the first layers. In addition, the order of layers in terms of sensitive information exposure is almost the same across all three datasets.

I also compute the risk per neuron for each layer by normalizing the risk of sensitive information exposure by the total number of neurons of the layer (Figure 3.4). This excludes the impact of layers' sizes and then gives a deeper understanding of which layers' neurons contain more sensitive information per unit. Such an understanding provides us with which layers' neurons to

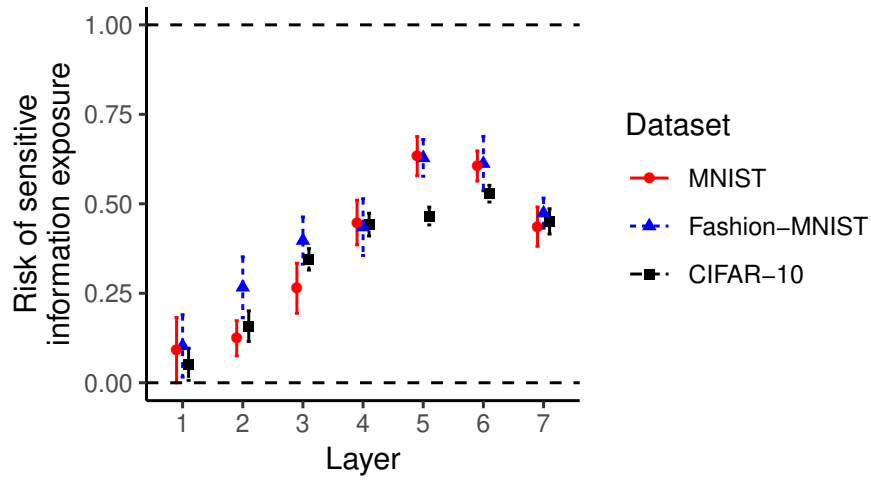


Figure 3.3: The risk of sensitive information exposure of VGG-7 per layer on MNIST, Fashion-MNIST and CIFAR-10. Error bars represent 95% confidence intervals.

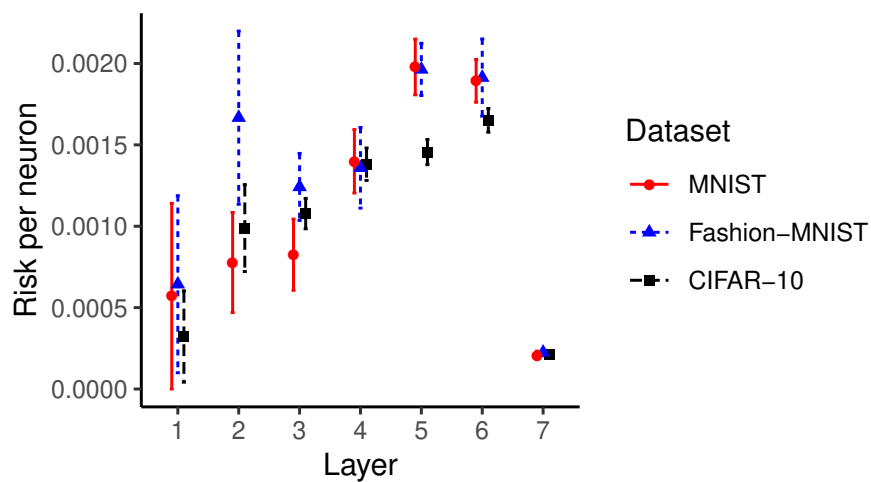


Figure 3.4: Risk per neuron for each layer on MNIST, Fashion-MNIST, and CIFAR-10. Error bars represent 95% confidence intervals.

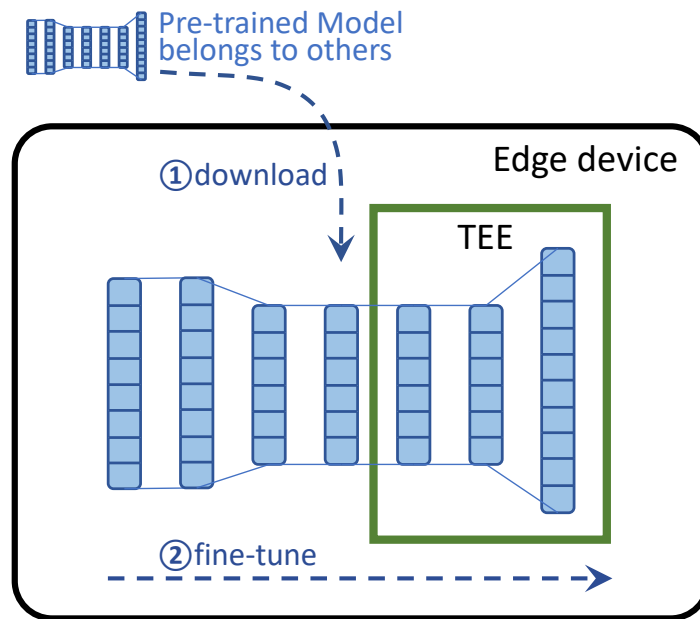


Figure 3.5: Using a TEE to protect the most sensitive layers (last layers) of an on-device deep neural network.

protect with a higher priority considering the fixed size of the protection area (e.g., TEE). The results show the risk per neuron increases as one moves through convolutional layers. Neurons in the late convolutional layers have high capabilities in memorizing sensitive information, whereas the fully connected layer (layer 7) has a much smaller risk per neuron.

## 3.4 Exploration of Protection using TEEs

### 3.4.1 Evaluation setup

This section presents an implementation and evaluates the cost of protecting the last layers of an on-device DNN during fine-tuning by deploying them in the TrustZone of a device (see Figure 3.5). TrustZone is ARM’s TEE implementation that establishes a private region on the main processor. Both hardware and software approaches isolate this region to allow trusted execution. As TEEs are usually small, one can only protect the most sensitive layers of the model and use the normal execution environment for the other layers.

I use Darknet [Red16] DNN library in Open Portable TEE (OP-TEE)<sup>1</sup>, a TEE framework based

<sup>1</sup> <https://www.op-tee.org>



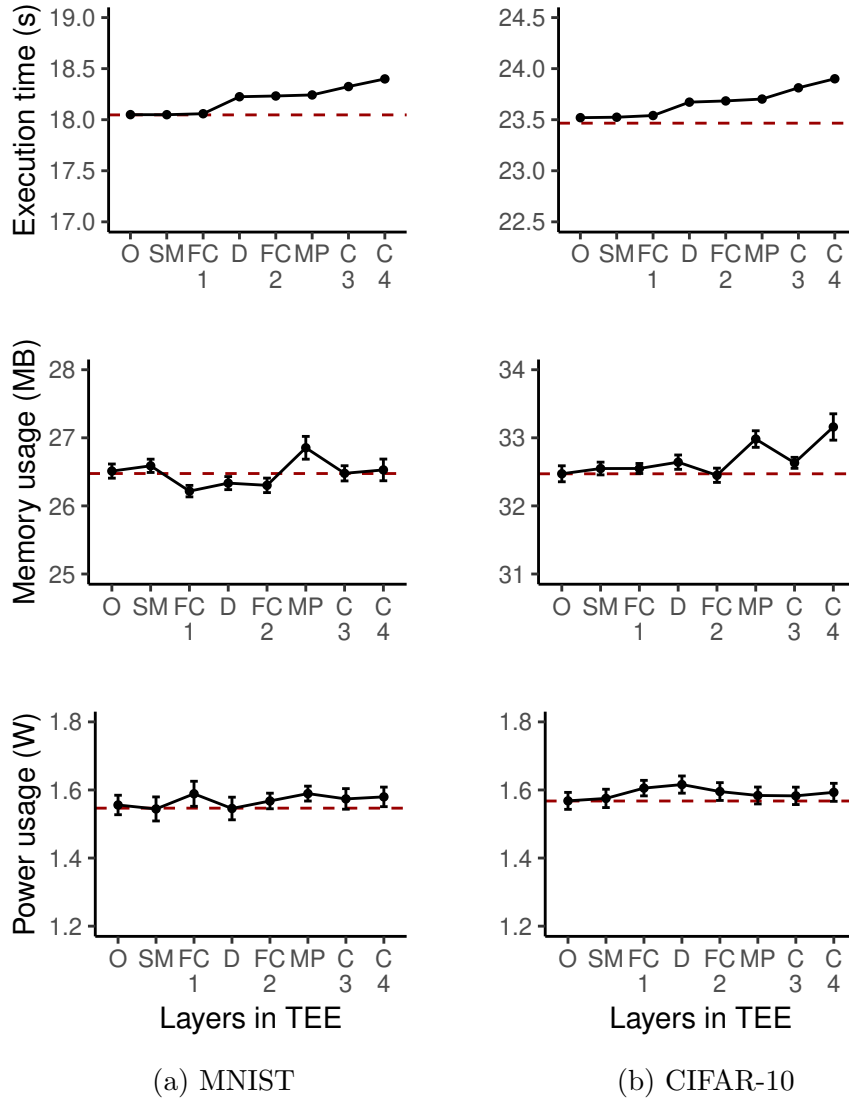


Figure 3.6: Execution time, memory usage, and power usage for protecting layers of VGG-7 trained on MNIST (left column) and CIFAR-10 dataset (right column) using the TrustZone of device. The x-axis corresponds to several last layers included in the TrustZone. *O* refers to the calculation of cost function; *SM*, *FC*, *D*, *MP*, and *C* refer to the softmax, fully connected, dropout, maxpooling, convolutional layers of VGG-7. The number of layers with trainable parameters in the TrustZone are 1, 2, 3, and 4. The dashed line represents the baseline, which runs all the layers outside the TrustZone. Error bars represent 95% confidence intervals.

on TrustZone, of a Raspberry Pi 3 Model B. This model of Raspberry Pi 3 runs instances of OP-TEE with 16 mebibytes (MiB) TEE’s memory. The choice of Darknet [Red16] is due to its high performance and small dependencies.

I fine-tune the pre-trained VGG-7 (from the previous section) with MNIST and CIFAR-10, respectively. Continuous layers are deployed in the TrustZone from the end for simplicity, including both layers with (i.e., the convolutional and fully connected layer) and without (i.e., the dropout and maxpooling layer) trainable parameters.

### 3.4.2 Results and discussion

Figure 3.6 shows the execution time (in seconds), memory usage (in MB), and power consumption (in Watt, using RuiDeng USB Tester (UM25C)<sup>2</sup>) of securing a part of the DNN in the TrustZone, starting from the last layer, and continuing adding layers until the maximum number of layers the zone can hold.

The resulting execution times are MNIST:  $F_{(7,232)} = 3658$ ,  $p < 0.001$ ; CIFAR-10:  $F_{(7,232)} = 2396$ ,  $p < 0.001$  and memory usage is MNIST:  $F_{(7,232)} = 11.62$ ,  $p < 0.001$ ; CIFAR-10:  $F_{(7,232)} = 20.01$ ,  $p < 0.001$ . The increase however is small compared to the baseline (Execution time: 1.94% for MNIST and 1.62% for CIFAR-10; Memory usage: 2.43% for MNIST and 2.19% for CIFAR-10). Moreover, running layers in the TrustZone did not significantly influence the power usage (MNIST:  $F_{(7,232)} = 1.49$ ,  $p = 0.170$ ; CIFAR-10:  $F_{(7,232)} = 1.61$ ,  $p = 0.132$ ).

Specifically, deploying the dropout layer and the maxpooling layer in the TEE increases both the execution time and memory usage. The reason is that these two types of layers have no trainable parameters, and for Darknet, the dropout and maxpooling are directly operated based on the trainable parameters of their front layer. Therefore, to run these two types of layers in the TEE, their front layer (i.e., fully connected/convolutional layers) needs to be copied into the TEE, which increases the cost. For layers with parameters that I aim to protect (1, 2, 3, and 4 in Figure 3.6), deploying fully connected layers (i.e., 1, 2) in the TEE does not increase the execution time accumulated on the first layers, and does not increase the memory

<sup>2</sup> <http://www.ruidengkeji.com>

usage. Deploying convolutional layers (i.e., 3 and 4) leads to an increase in execution time but does not increase memory usage when using MNIST. The second convolutional layer (i.e., 4) only increases memory usage when using CIFAR-10. However, exhausting the most available memory of the TEE can also cause an increase in overhead, so the reason for this increment of memory usage needs more analysis. Overall, for the implementation, protecting fully connected and convolutional layers has lower costs than other layers without trainable parameters with the TEE.

### 3.5 Summary

In this chapter, I proposed a method to measure the exposure of sensitive information in each layer of a pre-trained DNN model based on their generalization errors. It is shown that the closer the layer is to the output, the higher the likelihood that sensitive information of training data is exposed, which is opposite to the exposure risk of layers' activation from test data [GHZ<sup>+</sup>18b]. I did initial evaluations on the use of TEE to protect individual sensitive layers (i.e., the last layers) of a deployed DNN. The results show that TEE has a promising performance at a low cost.

It is worth noting that because there can exist interactions across layers, training one layer while freezing all other layers may lead to biases because training two layers separately and adding them performance gain will have different results than training these two layers together. On one side, the former one could be an estimation of the second one as shown in [LZC<sup>+</sup>22]. However, we still need to be aware of such differences and better test the privacy leakage based on real attacks or other metrics.

Furthermore, the evaluation can be improved by more fine-grained measurements, such as separating kernel space and user space's execution time and memory usage. Besides, the power measurement (UM25C) unit only has a precision of 3Hz which may be not sufficient enough to capture changes in power usage when switching between TrustZone's normal world and secure world. Thus, utilizing more accurate power measurement equipment in the next step can give a

---

deeper understanding of that. In addition, more datasets and models need to be tested to show the consistency of the results. The next step would also include the investigation of protecting the later layers of a DNN against, among other attacks, such as white-box membership inference attacks [NSH18] to demonstrate the real privacy guarantee performance of the such defense.

## Chapter 4

# On-device Privacy-preserving ML System

In this chapter, I aim to develop a dedicated edge ML framework preserving the private information leakage from DNN models aiming at Research Questions 2 and 3. This follows the results of the privacy measure of DNNs' weights in the previous chapter; I aim to develop this system to protect the last layers. Moreover, each layer of the model memorizes different information about the input, and they are more empirical studies showing similar patterns. Yosinki et al. [YCBL14] found that the first layers (closer to the input) are more transferable to new datasets than the last layers. That is, the first layers learn more *general* information (e.g., ambient colors in images), while the last layers learn information that is more *specific* to the classification task (e.g., face identity). The memorization difference per layer has been verified both in convolutional layers [ZF14, YCN<sup>+</sup>15] and in generative models [ZSE17]. Evidently, an untrusted party with access to the model can leverage the memorized information to infer potentially sensitive properties about the input data which leads to severe privacy risks.

Starting with the related work and knowledge, I focus on the defense against popular private information leakage related to membership inference, specifically one SOTA white-box membership inference attack, and polish the on-device privacy-preserving ML system to achieve high defense performance with low system cost.

## 4.1 Introduction and Related Work

### 4.1.1 Membership inference attack (MIA)

. With successful training (i.e., the model converging to an optimal solution), a DNN model “memorizes” features of the input training data [YGFJ18, RBU18], which it can then use to recognize unseen data exhibiting similar patterns. However, models have the tendency to include more specific information about the training dataset unrelated to the target patterns (i.e., the classes that the model aims to classify) [YGFJ18, CLG01].

MIAs form a possible attack on devices that leverage memorized information on a model’s layers to determine whether a given data record was part of the model’s training dataset [SSSS17]. A real-world example of MIA is that one adversary queries a remote ML model with one patient’s medical records to determine whether these medical records have participated in training the model, and consequently, the patient’s membership information is leaked. In a *black-box MIA*, the attacker leverages models’ outputs (e.g., confidence scores) and auxiliary information (e.g., public datasets or public prediction accuracy of the model) to train shadow models or classifiers without accessing internal information of the model [SSSS17, YGFJ18]. However, in a *white-box MIA*, the attacker utilizes the internal knowledge (i.e., gradients and activation of layers) of the model in addition to the model’s outputs to increase the effectiveness of the attack [NSH18]. It is shown that the last layer (model output) has the highest membership information about the training data [NSH18]. A white-box adversary is considered as the threat model, as DNNs are fully accessible after being transferred from the server to edge devices [XLL<sup>+</sup>19]. In addition to this, a white-box MIA is a *stronger* adversary than a black-box MIA, as the information the adversary has access to in a black-box attack is a subset of that used in a white-box attack.

### 4.1.2 Deep learning in the TEE

**Trusted execution environment (TEE).** A TEE is a trusted component that runs in parallel with the untrusted Rich operating system Execution Environment (REE) and is designed to provide safeguards for ensuring the confidentiality and integrity of its data and programs. This is achieved by establishing an isolated region on the main processor, and both hardware and software approaches are utilized to isolate this region. The chip includes additional elements such as unchangeable private keys or secure bits during manufacturing, which helps ensure that untrusted parts of the platform (even privileged OS or hypervisor processes) cannot access TEE content [CD16, Arm09].

In addition to strong security guarantees, TEEs also provide better computational performance than existing software protections, making them suitable for computationally-expensive deep learning tasks. For example, advanced techniques such as fully homomorphic encryption enable operators to process the encrypted data and models without decryption during deep learning but significantly increase the computation cost [NLV11, AAUC18]. Conversely, TEE protection only requires additional operations to build the trusted environment and the communication between trusted and untrusted parts, so its performance is comparable to normal executions in an untrusted environment (e.g., an OS).

**Deep learning with TEEs.** Previous work leveraged TEEs to protect deep learning models. Apart from the unique attack surface and thus protection goals considered in this research, these also differ from this approach in one more aspect: they depend on an underlying computer architecture that is more suitable for cloud environments. Recent work has suggested executing a complete deep learning model in a TEE [CD16], where during training, users' private data is transferred to the trusted environment using trusted paths. This prevents the host Cloud from eavesdropping on the data [OSF<sup>+</sup>16]. Several other studies improved the efficiency of TEE-resident models using Graphics Processing Units (GPU) [TB18], multiple memory blocks [HSS<sup>+</sup>18], and high-performance ML frameworks [HCS18]. More similar to my approach, Gu et al. [GHZ<sup>+</sup>18b] partitioned DNN models and only enclosed the first layers in

an SGX-powered TEE to mitigate input information disclosures of real-time fed device user images. In contrast, membership inference attacks become more effective by accessing information in the last layers. All these works use an underlying architecture based on Intel’s SGX, which is not suitable for edge devices. Edge devices usually have chips designed using Reduced Instruction Set Computing (RISC), peripheral interfaces, and much lower computational resources (around 16 mebibytes (MiB) memory for TEE) [EKA14]. Arm’s TrustZone is the most widely used TEE implementation in edge devices. It involves a more comprehensive trusted environment, including the security extensions for the AXI system bus, processors, interrupt controller, TrustZone address space controller, etc. Camera or voice input connected to the APB peripheral bus can be controlled as a part of the trusted environment by the AXI-to-APB bridge. Utilizing TrustZone for on-device deep learning requires more development and investigation because of its different features compared to SGX.

### 4.1.3 Privacy-preserving methods

An effective method for reducing the memorization of private information of training data in a DNN model is to avoid *overfitting* via imposing constraints on the parameters and utilizing dropouts [SSSS17]. *Differential Privacy* (DP) can also obfuscate the parameters (e.g., adding Gaussian noise to the gradients) during training to control each input’s impact on them [ACG<sup>+</sup>16, YLP<sup>+</sup>19]. However, DP may negatively affect the utility (i.e., the prediction accuracy) if the noise is not carefully designed [RRL<sup>+</sup>18]. In order to obfuscate private information only, one could apply methods such as generative neural networks [XRZ<sup>+</sup>19] or adversarial examples [JSB<sup>+</sup>19] to craft noises for one particular data record (e.g., one image), but this requires additional computational resources which are already limited on edge devices.

**Server-Client model partition.** General information processed in the first layers [YCBL14] during the forward propagation of deep learning often includes more important indicators for the inputs than those in the last layers (which is opposite to membership indicators), since reconstructing the updated gradients or activation of the first layers can directly reveal private information of the input [AHW<sup>+</sup>18, DB16]. Based on this, hybrid training models have been



proposed which run several first layers at the client-side for feature extraction and then upload these features to the server-side for classification [OST<sup>+</sup>17]. Such partition approaches delegate parts of the computation from the servers to the clients, and thus, in these scenarios, striking a balance between privacy and performance is of paramount importance.

Gu et al. [GHZ<sup>+</sup>18b] follow a similar layer-wise method and leverage TEEs on the cloud to isolate the more private layers. Clients' private data are encrypted and then fed into the cloud TEE so that the data and the first several layers are protected. This method expands the clients' trusted boundary to include the server's TEE and utilizes an REE-TEE model partition at the server which does not significantly increase clients' computation cost compared to running the first layers on client devices. To further increase training speed, it is also possible to transfer all linear layers outside a cloud's TEE into an untrusted GPU [TB18]. All these partitioning approaches aim to prevent leakage of private information of users (to the server or others), yet *do not* prevent leakage from trained models when models are executed on the users' edge devices.

## 4.2 DarkneTZ Framework

I now describe DarkneTZ, a framework for preserving DNN models' privacy on edge devices, and I start with the threat model.

### 4.2.1 Threat model

An adversary with full access to the REE of an edge device (e.g., the OS) on edge devices is considered: this could be the actual user, malicious third-party software installed on the devices, or a malicious or compromised OS. One only trusts the TEE of an edge device to guarantee the integrity and confidentiality of the data and software in it. In particular, it is assumed that a DNN model is pre-trained using private data from the server or other participating nodes. It is further assumed that the model providers can fully guarantee the model privacy during

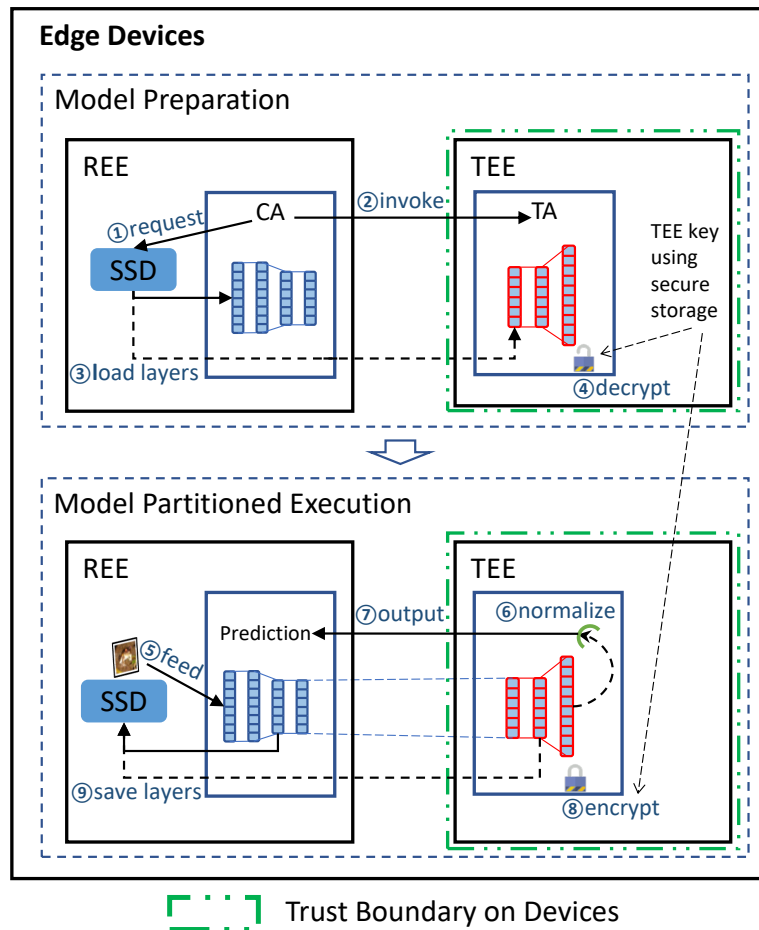


Figure 4.1: DarkneTZ uses on-device TEE to protect a set of layers of a deep neural network for both inference and fine-tuning.

Note: The trusted compute base—or trust boundary—for the model owner on edge devices is the TEE of the device.

training on their servers by utilizing existing protection methods [OSF<sup>+</sup>16] or even by training the model offline, so the model can be secret provisioned to the user devices without other privacy issues.

## 4.2.2 Design overview

DarkneTZ design aims at mitigating attacks on on-device models by protecting layers and the output of the model with low cost by utilizing an on-device TEE. It should be compatible with edge devices. That is, it should integrate with TEEs which can run on hardware technologies that can be found on commodity edge devices (e.g. Arm TrustZone), and use standard TEE

system architectures and corresponding APIs.

This research proposes DarkneTZ, illustrated in Figure 4.1, a framework that enables DNN layers to be partitioned as two parts to be deployed respectively into the REE and TEE of edge devices. DarkneTZ allows users to do inference with or fine-tuning of a model seamlessly—the partition is transparent to the user—while at the same time it considers the privacy concerns of the model’s owner. Corresponding Client Application (CA) and Trusted Application (TA) perform the operations in REE and TEE, respectively. Without loss of generality, DarkneTZ’s CA runs layers 1 to  $l$  in the REE, while its TA runs layers  $l+1$  to  $L$  located in the TEE during fine-tuning or inference of a DNN. This DNN partitioning can help the server to mitigate several attacks such as MIAs [NSH18, MSK<sup>+</sup>19], as the last layers have a higher probability of leaking private information about training data.

DarkneTZ expects sets of layers to be pre-provisioned in the TEE by the analyst (if the framework is used for offline measurements) or by the device OEM if a version of DarkneTZ is implemented on consumer devices. Note that in the latter case, secret provisioning of sensitive layers can also be performed over the air, which might be useful when the sensitive layer selection needs to be dynamically determined and provisioned to the edge device after supply. In this case, one could extend DarkneTZ to follow a variation of the SIGMA secure key exchange protocol [Kra03], modified to include remote attestation, similar to [ZZQ<sup>+</sup>19]. SIGMA is provably secure and efficient. It guarantees perfect forward secrecy for the session key (to defend against replay attacks) while its use of message authentication codes ensures server and client identity protection. Integrating remote attestation guarantees that the server provisions the model to a non-compromised edge device.

### 4.2.3 Model preparation

Once the model is provisioned, the CA requests the layers from devices (e.g., solid-state disk drive (SSD)) and invokes the TA. The CA will first build the DNN architecture and load the parameters of the model into normal memory (i.e., non-secure memory) to process all calculations and manipulations of the non-sensitive layers in the REE. When encountering

(secretly provisioned) encrypted layers that need to be executed in the TEE, which is determined by the model owner’s setting, the CA passes them to the TA. The TA decrypts these layers using a key that is securely stored in the TEE (using secure storage), and then it runs the more sensitive layers in the TEE’s secure memory. The secure memory is indicated by one additional address bit introduced to all memory system transactions (e.g., cache tags, memory, and peripherals) to block non-secure access [Arm09]. At this point, the model is ready for fine-tuning and inference.

#### 4.2.4 DNN partitioned execution

The *forward pass* of both inference and fine-tuning passes the input  $\mathbf{a}^0$  to the DNN to produce activation of layers until the last layer, i.e., layer  $l$ ’s activation is calculated by  $\mathbf{a}^l = f(\mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l)$ , where  $\mathbf{w}^l$  and  $\mathbf{b}^l$  are weights and biases of this layer,  $\mathbf{a}^{l-1}$  is activation of its previous layer and  $f$  is the non-linear activation function. Therefore, after the CA processes its inside layers from 1 to  $l$ , it invokes a command to transfer the outputs (i.e., activation) of layer  $l$  (i.e., the last layer in the CA) to the secure memory through a *buffer* (in shared memory). The TA switches to the *forward\_net\_TA* function corresponding to the invoked command to receive parameters (i.e., outputs/activation) of layer  $l$  and processes the following forward pass of the network (from layer  $l + 1$  to layer  $L$ ) in the TEE. In the end, outputs of the last layer are first normalized as  $\hat{\mathbf{a}}^L$  to control the membership information leakage and are returned via shared memory as the prediction results.

The *backward pass* of fine-tuning computes gradients of the loss function  $\mathcal{L}(\mathbf{a}^L, y)$  with respect to each weight  $\mathbf{w}^l$  and bias  $\mathbf{b}^l$ , and updates the parameters of all layers,  $\{\mathbf{w}^l\}_{l=1}^L$  and  $\{\mathbf{b}^l\}_{l=1}^L$  as  $\mathbf{w}^l = \mathbf{w}^l - \eta \frac{\partial \mathcal{L}(\mathbf{a}^L, y)}{\partial \mathbf{w}^l}$  and  $\mathbf{b}^l = \mathbf{b}^l - \eta \frac{\partial \mathcal{L}(\mathbf{a}^L, y)}{\partial \mathbf{b}^l}$ , where  $\eta$  is a constant called the learning rate and  $y$  is the desired output (i.e., called label). The TA can compute the gradient of the loss function by receiving  $y$  from CA and backpropagate it to the CA in order to update all the parameters. In the end, to save the fine-tuned model on devices, all layers in the TA are encrypted and transferred back to the CA.

## 4.3 Experiment Settings

### 4.3.1 Models and datasets

This research first uses two popular DNNs, namely AlexNet and VGG-7, to measure the system’s performance. AlexNet has five convolutional layers (i.e., with kernel size 11, 5, 3, 3, and 3) followed by a fully-connected and a softmax layer, and VGG-7 has eight layers (i.e., seven convolutional layers with kernel size 3, followed by a fully-connected layer). Both AlexNet and VGG-7 use ReLU (Rectifier Linear Unit) activation functions for all convolutional layers. The number of neurons for AlexNet’s layers is 64, 192, 384, 256, and 256, while the number of neurons for VGG-7’s layers is 64, 64, 124, 124, 124, 124, and 124. The networks are trained and used to conduct inference on CIFAR-100 and ImageNet Tiny. Image classification datasets are used, as a recent empirical study shows that the majority of smartphone applications (70.6%) that use deep learning are for image processing [XLL<sup>+</sup>19]. Moreover, the state-of-the-art MIA being considered is demonstrated against such datasets [NSH18]. CIFAR-100 includes 50k training and 10k test images of size  $32 \times 32 \times 3$  belonging to 100 classes. ImageNet Tiny is a simplified ImageNet challenge that has 100k training and 10k test images of size  $64 \times 64 \times 3$  belonging to 200 classes.

In addition to this, this work uses six available DNNs (Tiny Darknet (4 megabytes (MB)), Darknet Reference (28MB), Extraction [SLJ<sup>+</sup>15] (90MB), Resnet-50 [HZRS16] (87MB), Densenet-201 [HLVDMW17] (66MB), and Darknet-53-448 (159MB)) pre-trained on the original ImageNet [DDS<sup>+</sup>09] dataset to measure DarkneTZ’s performance during *inference*. All pre-trained models can be found online<sup>1</sup>. ImageNet has 1000 classes, and consequently, these DNN models’ last layers occupy larger memory that can exceed the TEE’s limits, compared to models with 100/200 classes. Therefore, for these six models, Only their *last layer* in the TEE is evaluated.

To evaluate the defense’s effectiveness against MIAs, this research uses the same models as those used in the demonstration of the attack[NSH18] (AlexNet, VGG-7, and ResNet-110). This ResNet with 110 depth is an existing network architecture that has three blocks (each has

<sup>1</sup> <https://pjreddie.com/darknet/imagenet/>

36 convolutional layers) in the middle and another convolutional layer at the beginning and one fully connected layer at the end [HZRS16]. Published models trained (with 164 epochs) on CIFAR-100 [KNHb] online<sup>2</sup> are used. Three models on ImageNet Tiny<sup>3</sup> are trained with 300 epochs as target models (i.e., victim models during attacks). Models with the highest valid accuracy are used after training. this research follows [NSH18]’s methodology, and all training and test datasets are split into two parts with equal sizes randomly so that the MIA model learns both *Member* and *Non-member* images. For example, 25K of training images and 5K of test CIFAR-100 images are chosen to train the MIA model, and then the model’s test precision and recall are evaluated using 5K of training images and 5K of test images in the rest of CIFAR-100 images.

### 4.3.2 Implementation and evaluation setup

This work develops an implementation based on the Darknet [Red16] DNN library. This particular library is used because of its high computational performance and small library dependencies which fits within the limited secure memory of the TEE. I run the implementation on Open Portable TEE (OP-TEE), which provides the software (i.e., operating systems) for an REE and a TEE designed to run on top of Arm TrustZone-enabled hardware.

For TEE measurements, I focus on the performance of deep learning since secret provisioning only happens once for updating the model from servers. 128-bit AES-GCM is implemented for on-device secure storage of sensitive layers. This implementation is tested on a Hikey 960 board, a widely-used device [YAA<sup>+</sup>18, AAD18, DBJL18, BGJ<sup>+</sup>19] that is promising to be comparable with mobile phones (and other existing products) due to its Android open source project support. The board has four ARM Cortex-A73 cores and four ARM Cortex-A53 cores (pre-configured to 2362MHz and 533MHz, respectively, by the device OEM), 4GB LPDDR4 SDRAM, and provides 16MiB secure memory for trusted execution, which includes 14MiB for the TA and 2MiB for TEE run-time. Another 2MiB shared memory is allocated from non-secure memory. As the Hikey board adjusts the CPU frequency automatically according to the CPU

---

<sup>2</sup> <https://github.com/bearpaw/pytorch-classification>    <sup>3</sup> <https://tiny-imagenet.herokuapp.com/>

temperature, I decrease and fix the frequency of Cortex A73 to 903MHz and keep the frequency of Cortex A53 as 533Mhz. During experiments, a 120 seconds system sleep is introduced per trial to make sure that the CPU temperature begins under  $40^{\circ}C$  to avoid underclocking.

Edge devices suffer from limited computational resources, and as such, it is paramount to measure the efficiency of deep learning models when partitioned to be executed partly by the OS and partly by the TEE. In particular, I monitor and report CPU execution time (in seconds), memory usage (in megabytes), and power consumption (in watts) when the complete model runs in the REE (i.e., OS) and compare it with different partitioning configurations where more sensitive layers are kept within the TEE. CPU execution time is the amount of time that the CPU was used for deep learning operations (i.e., fine-tuning or inference). Memory usage is the amount of the mapping that is currently resident in the main memory (RAM) occupied by the process for deep learning-related operations. Power consumption is the electrical energy consumption per unit time that was required by the Hikey board.

More specifically, the REE's `/proc/self/status` is utilized for accessing the process information to measure the CPU execution time and memory usage of the implementation. CPU execution time is the amount of time for which the CPU was used for processing instructions of software (as opposed to wall-clock time which includes input/output operations) and is further split into (a) time in user mode and (b) time in kernel mode. The REE kernel time captures together (1) the time spent by the REEs kernel and (2) the time spent by the TEE (including both in user mode and kernel mode). This kernel time gives us a direct perception of the overhead when including TEEs for deep learning versus using the same REE without a TEE's involvement.

Memory usage is represented using *resident set size (RSS) memory* in the REE, but the memory occupied in the TEE is not counted by the RSS since the REE does not have access to gather memory usage information of the TEE. The TEE is designed to conceal this sensitive information (e.g., both CPU time and memory usage); otherwise, the confidentiality of TEE contents would be easily breached by utilizing side-channel attacks [WCP<sup>+</sup>17]. To overcome this, an abort is triggered from the TEE after the process runs stably (memory usage tends to

be fixed) to obtain the memory usage of the TEE.

To accurately measure the power consumption, a Monsoon High Voltage Power Monitor <sup>4</sup> is used, a high-precision power metering hardware capable of measuring the current consumed by a test device with a voltage range of 0.8V to 13.5V and up to 6A continuous current. It is configured to power the Hikey board using the required 12V voltage while recording the consumed current in a 50Hz sampling rate.

For conducting the MIA, I use a machine with 4 Intel(R) Xeon(R) E5-2620 CPUs (2.00GHz), an NVIDIA QUADRO RTX 6000 (24GB), and 24GB DDR4 RAM. Pytorch v1.0.1 [PGC<sup>+</sup>17] is used as the DNN library.

### 4.3.3 Measuring privacy in MIAs

The adversarial strategy in this setting is defined based on state-of-the-art white-box MIAs which observe the behavior of all components of the DNN model [NSH18]. White-box MIAs can achieve a higher accuracy in distinguishing whether one input sample is presented in the private training dataset compared to black-box MIAs since the latter only have access to the models' output [YGFJ18, SSSS17]. Besides, white-box MIAs are also highly possible in on-device deep learning, where a model user can not only observe the output, but also observe fine-grained information such as the values of the cost function, gradients, and activation of layers.

This research evaluates the membership information exposure of a set of the target model's layers by employing the white-box MIA [NSH18] on these layers. The attacker feeds the target data to the model and leverages all possible information in the white-box setting including activation of all layers, the model's output, loss function, and the gradients of the loss function with respect to the parameter of each layer. It then separately analyses each information source by extracting features from the activation of each layer, the model's output, and the loss function via fully connected neural networks with one hidden layer, while using convolutional neural networks for the gradients. All extracted features are combined in a global feature vector

---

<sup>4</sup> <https://www.monsoon.com/>



that is later used as an input for an inference attack model. The attack model predicts a single value (i.e., Member or Non-member) that represents the membership information of the target data (please refer the interested readers to [NSH18] for a detailed description of this MIA). I use the test accuracy of the MIA model trained on a set of layers to represent the advantage of adversaries as well as the sensitivity of these layers.

To measure the privacy risk when part of the model is in TEE, this MIA is conducted on the target model in two different settings: (i) starting from the first layer, the later layers are added one by one until the end of the network, and (ii) starting from the last layer the previous layers are added one by one until the beginning of the network. However, the available information of one specific layer during the *fine-tuning phase* and that during the *inference phase* are different when starting from the first layers. Inference only has a forward propagation phase which computes the activation of each layer. During fine-tuning and because of the backward propagation, in addition to the activation, gradients of layers are also visible. In contrast to that, attacks starting from the last layers can observe the same information in both the inference and fine-tuning since layers' gradients can be calculated based on the cost function. Therefore, in setting (i) activation, gradients, and outputs are utilized. In setting (ii), only the activation of each layer is used to evaluate inference and use both activation and gradients to evaluate fine-tuning, since the outputs of the model (e.g., confidence scores) are not accessible in this setup.

## 4.4 Evaluation Results

This Section first evaluates the efficiency of DarkneTZ when protecting a set of layers in the TrustZone to answer Question 2. To evaluate system efficiency, this research measures **CPU execution time**, **memory usage**, and **power consumption** of the implementation for both training and inference on AlexNet and VGG-7 trained on two datasets. The last layers are protected (starting from the output) since they are more vulnerable to attacks (e.g., MIAs) on models. The cost layer (i.e., the cost function) and the softmax layer are considered as a separate layer since they contain highly sensitive information (i.e., confidence scores and

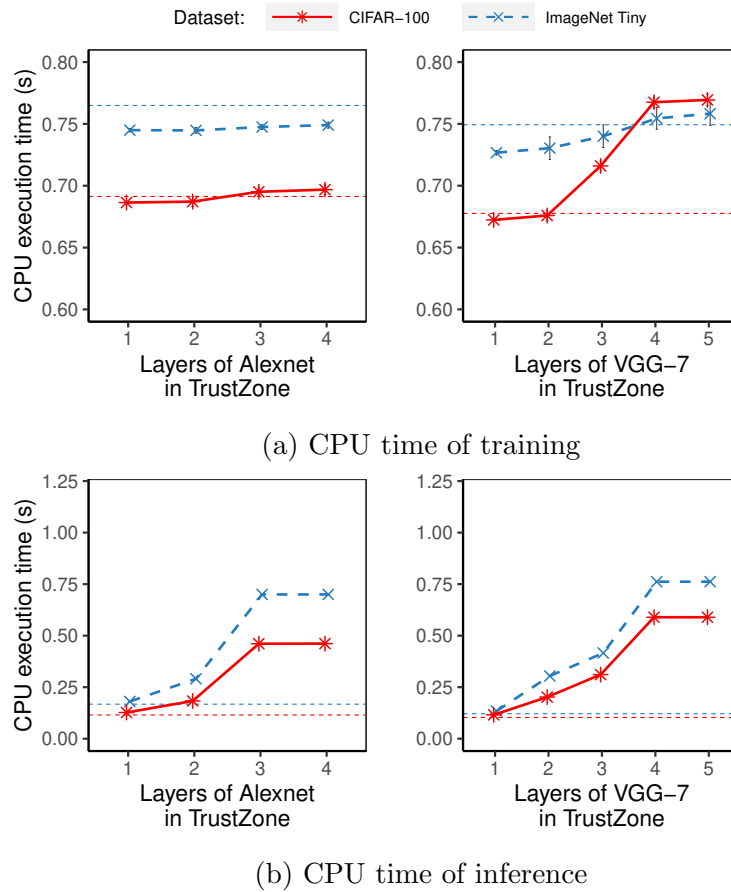


Figure 4.2: The CPU time of each step of training models or conducting inference on CIFAR-100 and ImageNet Tiny, protecting consecutive last layers using TrustZone.

Note: For example, when putting the last layers in the TrustZone, 1 refers to the cost function and the softmax layer, 2 includes 1 and the previous fully-connected layer, 3 includes 2 and the previous convolutional layers, etc. Horizontal dashed lines (  $\cdots$  and  $\cdots$  ) represent the baseline where all layers are out of the TrustZone. 20 times for each trial, and error bars are 95% CI. Several error bars of data points are invisible as they are too small to be shown in this figure as well as the following figures.

cost function). Starting from the last layer, I include the maximum number of layers that the TrustZone can hold. To answer Question 3, I use the **MIA success rate**, indicating the membership probability of target data (the more DarkneTZ limits this, the stronger the privacy guarantees are). I demonstrate the effect on performance and discuss the trade-off between performance and privacy using MIAs as one example.

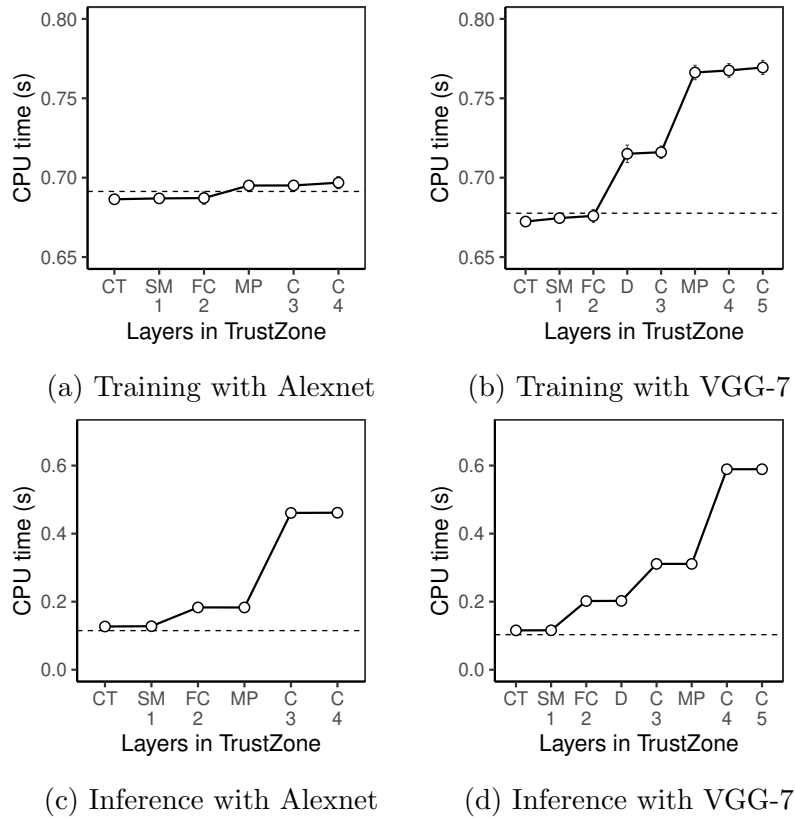
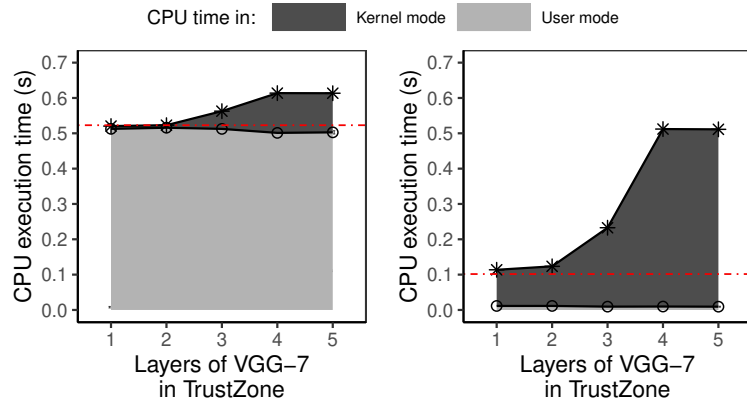


Figure 4.3: The CPU time of each step of training models or conducting inference on CIFAR-100, protecting consecutive last layers using TrustZone.

Note: The x-axis corresponds to several last layers included in the TrustZone. *CT*, *SM*, *FC*, *D*, *MP*, and *C* refer to the cost, softmax, fully connected, dropout, maxpooling, convolutional layers. 1, 2, 3, 4, and 5 in the x-axis are corresponding to the x-axis of Figure 4.2. Horizontal dashed lines ( --- ) represent the baseline where all layers are out of the TrustZone. 20 times for each trial, and error bars are 95% CI.

#### 4.4.1 CPU execution time

As shown in Figure 4.2, the results indicate that including more layers in the TrustZone results in an increasing CPU time for deep learning operations, where the most expensive addition is to put the maximum number of layers. Figure 4.2a shows the CPU time when *training* AlexNet and VGG-7 with TrustZone on CIFAR-100 and ImageNet Tiny dataset, respectively. This increasing trend is significant and consistent for both datasets (CIFAR-100:  $F_{(6,133)} = 29.37, p < 0.001$ ;  $F_{(8,171)} = 321.3, p < 0.001$ . ImageNet Tiny:  $F_{(6,133)} = 37.52, p < 0.001$ ;  $F_{(8,171)} = 28.5, p < 0.001$ ). It is also observed that protecting only the last layer in the TrustZone has a negligible effect on the CPU utilization while including more layers to fully utilize the TrustZone during training can increase CPU time (by 10%). For inference, the increasing trend



(a) Training on CIFAR-100 (b) Inference on CIFAR-100

Figure 4.4: The CPU execution time in user mode and kernel mode of each step of training the model or conducting inference on CIFAR-100, protecting consecutive last layers using TrustZone.

Note: Horizontal dot-dashed lines ( - - - ) represent the baseline where all layers are out of the TrustZone. 20 times for each trial. CPU time in user mode in Figure 4.4b is too small to be shown

is also significant (see Figure 4.2b). It only increases CPU time by around 3% when protecting only the last layer which can increase up to 10 $\times$  when the maximum possible number of layers is included in the TrustZone.

To further investigate the increasing CPU execution time effect, I analyzed all types of layers (both *trainable* and *non-trainable*) separately in the TrustZone. Trainable layers have parameters (e.g., weights and biases) that are updated (i.e., trainable) during the training phase. Fully connected layers and convolutional layers are trainable. Dropout, softmax, and maxpooling layers are non-trainable. As shown in Figure 4.3, different turning points exist where the CPU time significantly increases ( $p < 0.001$ ) compared to the previous configuration (i.e., one more layer is moved into the TrustZone) (Tukey HSD [AW10] was used for the post hoc pairwise comparison). When conducting *training*, the turning points appear when putting the maxpooling layer in the TrustZone for AlexNet (see Figure 4.3a) and when putting the dropout layer and the maxpooling layer for VGG-7 (see Figure 4.3b). All these layers are non-trainable. When conducting *inference*, the turning points appear when including the convolutional layers in TrustZone for both AlexNet (see Figure 4.3c) and VGG-7 (see Figure 4.3d), which are one step behind those points when conducting training.

One possible reason for the increased CPU time during *inference* is that the TrustZone needs to conduct extra operations (e.g., related secure memory allocation) for the trainable layer, as shown in Figure 4.3c and Figure 4.3d where all increases happen when one trainable layer is included in the TrustZone. Since I only conduct one-time inference during experiments, the operations of invoking TEE libraries, creating the TA, and allocating secure memory for the first time significantly increased the execution time compared to the next operations. Every subsequent inference attempt (continuously without rebuilding the model) does not include additional CPU time overhead. Figure 4.4 also shows that most of the increased CPU execution time (from  $\sim 0.1$ s to  $\sim 0.6$ s) is observed in the kernel mode—which includes the execution in TrustZone. The operation that needs to create the TA (to restart the TEE and load TEE libraries from scratch), such as one-time inference, should be taken care of by *preloading* the TA before conducting inference in practical applications.

During *training*, the main reason for the increased CPU time is that protecting non-trainable layers in the TrustZone results in an additional transmission of their previous trainable layers from the REE to the TrustZone. Non-trainable layers (i.e., dropout and max-pooling layers) are processed using a trainable layer as the base, and the non-trainable operation manipulates its previous layer (i.e., the trainable layer) directly. To hide the non-trainable layer and to prevent its next layer from being transferred to the REE during backward propagation (as mentioned in Section 4.2.4), I also move the previous convolutional layer to the TrustZone, which results in the turning points of the training that are one layer in front of the turning points during inference. Therefore, in practical applications, I should protect the trainable layer and its previous non-trainable layer together, since only protecting the non-trainable layer still requires moving its trainable layer into TrustZone and does not reduce the cost.

#### 4.4.2 Memory usage

*Training* with the TrustZone does not significantly influence the memory usage (in the REE) as it is similar to training without TrustZone (see Figure 4.5a). *Inference* with TrustZone uses less memory (in the REE) (see Figure 4.5b) but there is still no difference when more

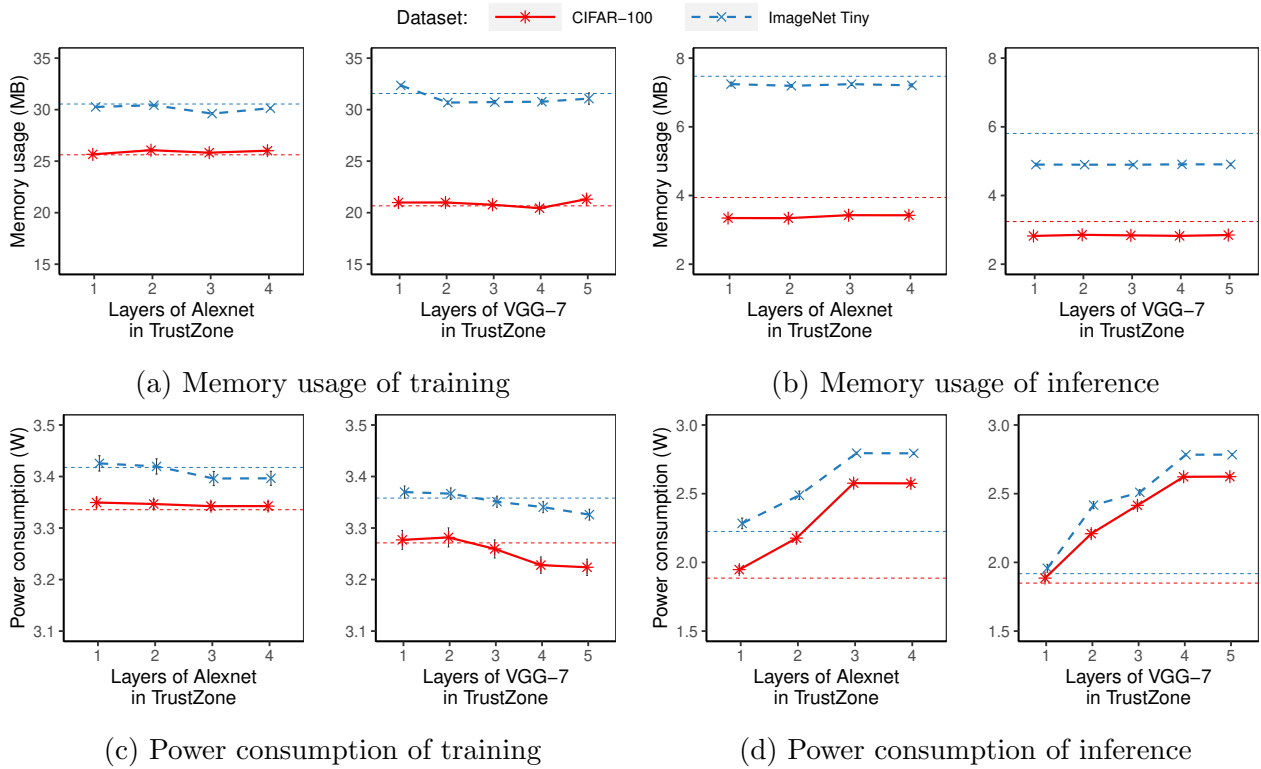


Figure 4.5: The memory usage and power consumption of training models, while conducting training or inference on CIFAR-100 and ImageNet Tiny, protecting consecutive last layers using TrustZone.

Note: Horizontal dashed lines (  $\cdots$  and  $\cdots$  ) represent the baseline where all layers are outside the TrustZone. 20 times for each trial, error bars are 95% CI.

layers are placed into TrustZone. Memory usage (in the REE) *decreases* since layers are moved to TrustZone and occupy secure memory instead. I measure the TA’s memory usage using all mapping sizes in secure memory based on the TA’s abort information. The TA uses five memory regions for sizes of 0x1000, 0x101000, 0x1e000, 0xa03000, and 0x1000 which is 11408KiB in total for all configurations. The mapping size of secure memory is fixed when the TEE runtime allocates memory for the TA, and it does not influence when moving more layers into the memory. Therefore, because of the different model sizes, a good setting is to maximize the TA’s memory mapping size in TrustZone in order to hold several layers of a possible large model.

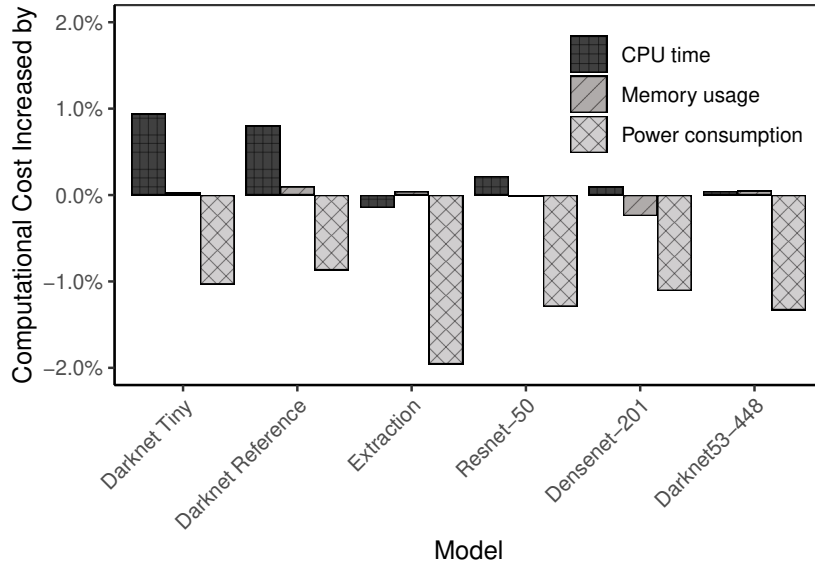


Figure 4.6: Performance on protecting the last layer of models trained on ImageNet in TrustZone for inference.

Note: 20 times per trial; error bars are too small to be visible in the plot.

### 4.4.3 Power consumption

For *training*, the power consumption significantly decreases ( $p < 0.001$ ) when more layers are moved inside TrustZone (see Figure 4.5c). In contrast, the power consumption during *inference* significantly increases ( $p < 0.001$ ) as shown in Figure 4.5d. In both training and inference settings, the trend of power consumption is likely related to the change in CPU time (see Figure 4.2). More specifically, trajectories of them in the figures have the same turning points (i.e., decreases or increases when moving the same layer to the TEE). One reason for the increased power consumption during inference is the significant increase in the number of CPU executions for invoking the required TEE libraries that consume additional power. When a large number of low-power operations (e.g., memory operations for mapping areas) are involved, the power consumption (i.e., energy consumed per unit time) could be lower compared to when a few CPU-bound computationally-intensive operations are running. This might be one of the reasons behind the decreased power consumption during training.

**System performance on large models.** This research also tests the performance of DarkneTZ on several models trained on ImageNet when protecting the last layer only, including the

Table 4.1: Training and testing accuracy (Acc.) and corresponding MIA precision (Pre.) with or without DarkneTZ (DTZ) of all models and datasets.

Dataset	Model	Train Acc.	Test Acc.	Attack Pre.	Attack Pre. (DTZ)
CIFAR-100	AlexNet	97.0%	43.9%	84.7%	51.1%
	VGG-7	83.8%	62.7%	71.5%	50.5%
	ResNet-100	99.6%	72.4%	88.3%	50.6%
ImageNet Tiny	AlexNet	40.3%	31.5%	56.7%	50.0%
	VGG-7	57.1%	48.6%	54.2%	50.8%
	ResNet-110	62.1%	54.2%	54.6%	50.2%

softmax layer (or the pooling layer) and the cost layer in TrustZone, in order to hide confidence scores and the calculation of cost. The results show that the overhead of protecting large models is negligible (see Figure 4.6): increases in CPU time, memory usage, and power consumption are lower than 2% for all models. Among these models, the smaller models (e.g., Tiny Darknet and Darknet Reference model) tend to have a higher rate of increase of CPU time compared to the larger models (e.g., Darknet-53 model), indicating that with larger models, the influence of TrustZone protection on resource consumption becomes relatively less.

**System performance summary.** In summary, *it is practical to process a sequence of sensitive DNN models layers inside the TEE of a mobile device.* Putting the last layer in the TrustZone does not increase CPU time and only slightly increases memory usage (by no more than 1%). The power consumption increase is also minor (no more than 0.5%) when fine-tuning the models. For inference, securing the last layer does not increase memory usage but increases CPU time and power consumption (by 3%). Including more layers to fully utilize the TrustZone during training can further increase CPU time (by 10%) but does not harm power consumption. One-time inference with multiple layers in the TrustZone still requires further development, such as utilizing the preliminary load of the TA, in practical applications. Note that in some cases utilizing TEEs gains better performance than the baseline; this may also be due to the pre-loading of some TEE driver and software stacks.



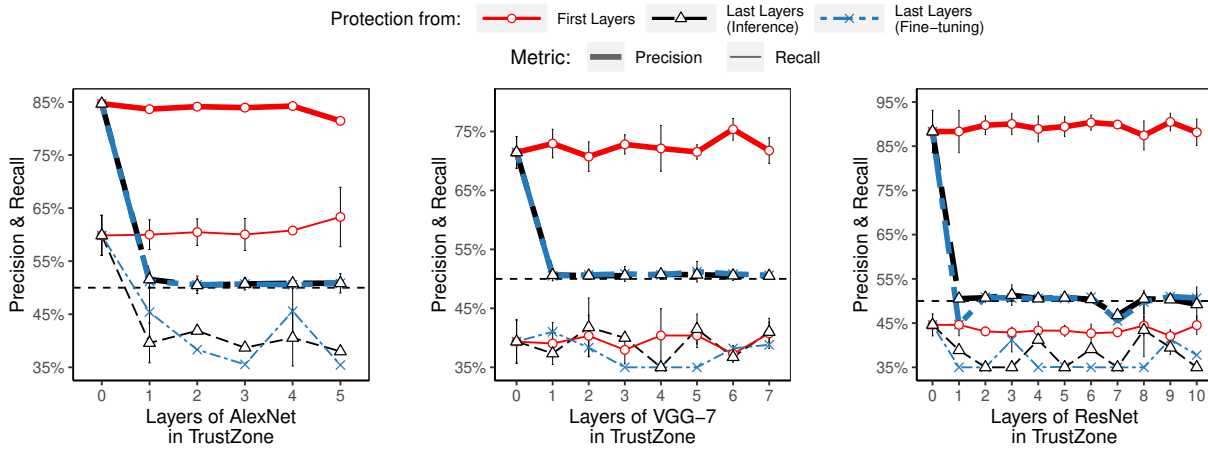


Figure 4.7: Precision and recall of white-box membership inference attacks when first or last layers of the model, trained on CIFAR-100, are protected using TrustZone.

Note: For first layer protection, 1 refers to the first layer, 2 refers to the first and the second layer, etc. For last layer protection, 1 refers to the last layer (i.e., the output layer), 2 refers to the last and second last layer, etc. 0 means that all layers are out of the TrustZone. Dashed lines at 50% represent baselines (i.e., random guess). Each trial has been repeated 5 times, and the error bars are 95% CI

#### 4.4.4 Privacy measurement

This research conducts the white-box MIA (Section 4.3.3) on all target models (see Section 4.3.1 for the choice of models) to analyze the privacy risk while protecting several layers in the TrustZone. I used the standard *precision* and *recall* metrics, similar to previous works [SSSS17]. In this context, precision is the fraction of records that an attacker infers as being members, that are indeed members in the training set. The recall is the fraction of training records that had been identified correctly as members. The performance for both models and MIAs is shown in Table 4.1. Figure 4.7 shows the attack success precision and recall for different configurations of DarkneTZ. In each configuration, a different number of layers is protected by TrustZone before I launch the attack. The configurations with zero layers protected correspond to DarkneTZ being disabled (i.e., with this defense disabled). In particular, I measure the MIA adversary’s success following two main configuration settings of DarkneTZ. In the first setting, I incrementally add consecutive layers in the TrustZone starting from the front layers and moving to the last layers until the complete model is protected. In the second setting, I do the opposite: I start from the last layer and keep adding previous layers in TrustZone for each configuration. The results show that when protecting the first layers in TrustZone, the attack success precision does not change

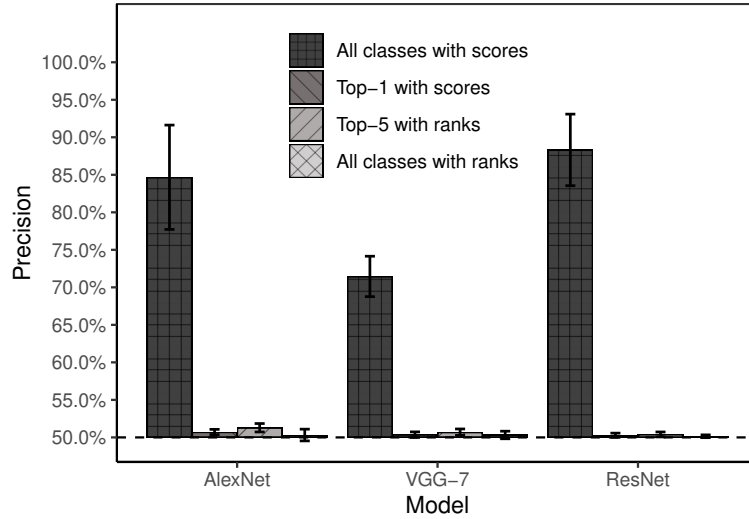


Figure 4.8: Precision of white-box membership inference attacks on models trained on CIFAR-100 when only outputs are protected using TrustZone.

Note: Dashed lines at 50% represent baselines (i.e., random guess). 5 times for each trial, and error bars are 95% CI.

significantly. In contrast, hiding the last layers can significantly decrease the attack success precision, even when only a single layer (i.e., the last layer) is protected by TrustZone. The precision decreases to  $\sim 50\%$  (random guessing) no matter how accurate the attack is before the defense. For example, for the AlexNet model trained on CIFAR-100, the precision drops from 85% to  $\sim 50\%$  when I only protect the *last* layer in TrustZone. Precision is much higher than recall since the number of *members* in the adversary’s training set is larger than that of *non-members*, so the MIA model predicts *member* images better. The results also show that the membership information that leaks during inference and fine-tuning is very similar. Moreover, according to [NSH18] and [SSSS17], the attack success precision is influenced by the size of the attackers’ training dataset. I used relatively large datasets (half of the target datasets) for training MIA models so that it is hard for the attacker to increase success precision significantly in the defense setting. Therefore, by hiding the last layer in TrustZone, the adversary’s attack precision degrades to 50% (random guess) while the overhead is under 3%.

This research also evaluated the privacy risk when DarkneTZ protects the model’s outputs in TrustZone by *normalizing* it before outputting prediction results. In this configuration, I conduct the white-box MIAs when all other layers (in the untrusted REE) are accessible by the

adversary. This means that the cost function is protected, and the confidence score’s outputs are controlled by TrustZone. Three combinations of models and datasets, including AlexNet, VGG-7, and ResNet on CIFAR-100 are selected as they were identified as more *vulnerable* (i.e., with high attack precision see Table 4.1) to MIAs [NSH18]. DarkneTZ is set to control the model’s outputs in three different ways: (a) top-1 class with its confidence score; (b) top-5 classes with their confidence scores; (c) all classes with their confidence scores. As shown in Figure 4.8 all three methods can significantly ( $p < 0.001$ ) decrease the attack success performance to around 50% (i.e., random guess). Therefore, *it is highly practical to use DarkneTZ to tackle MIAs: it incurs low resource consumption cost while achieving high privacy guarantees.*

## 4.5 Discussion and Summary

### 4.5.1 System performance

**Effects of the model size.** This chapter showed that protecting large models with TrustZone tends to have a lower rate of increase of CPU execution time than protecting small models (see Figure 4.6). One possible explanation is that the last layer of a larger model uses a lower proportion of computational resources in the whole model compared to that of a smaller model. This work has also examined the effect of different hardware: The implementation of DarkneTZ has been executed with similar model sizes on a Raspberry Pi 3 Model B (RPi3B), and it is found to have a lower rate of increase of cost (i.e., lower overhead) than when executed on the Hikey board [MSK<sup>+</sup>19]. This is because the Hikey board has much faster processors optimized for matrix calculations, which renders additional operations of utilizing TrustZone more noticeable compared to other normal executions (e.g., deep learning operations) in the REE. Moreover, results show that a typical configuration (16MiB secure memory) of the TrustZone is sufficient to hold at least the last layer of practical DNN models (e.g., trained on ImageNet). However, it is challenging to fit multiple layers of large models in a significantly smaller TEE. I tested a TEE with 5MiB secure memory on a Grapeboard<sup>5</sup>: only 1,000 neurons (corresponding

---

<sup>5</sup> <https://www.grapeboard.com/>

to 1,000 classes) in the output layer already occupy 4MiB memory when using floating-point arithmetic. In such environments, model compression, such as pruning [HMD15] and quantization [WLL<sup>+</sup>19, JKC<sup>+</sup>18], could be one way to facilitate including more layers in the TEE. Lastly, it is found that utilizing TEEs for protecting the last layer does not necessarily lead to resource consumption overhead, which deserves further investigation in future work. Overall, results show that utilizing TrustZone to protect outputs of large DNN models is effective and highly efficient.

**Extrapolating for other mobile-friendly models.** Tiny Darknet and Darknet Reference have been used for testing DarkneTZ’s performance on mobile-friendly models (for ImageNet classification). Another widely-used DNNs on mobile devices, Squeezenet [IHM<sup>+</sup>16] and Mobilenet [HZC<sup>+</sup>17], define new types of convolutional layers that are not supported in Darknet framework currently. It is expected that these have a similar privacy and TEE performance footprint because of the comparable size of model (4MB, 28MB, 4.8MB, 3.4MB for Tiny Darknet, Darknet Reference, Squeezenet, and Mobilenet, respectively), floating-point operations (980M, 810M, 837M, 579M), and model accuracy (58.7%, 61.1%, 59.1%, and 71.6% for Top-1)<sup>6</sup>.

**Improving performance.** Modern mobile devices usually are equipped with GPU or specialized processors for deep learning such as NPU. The current implementation only uses the CPU but can be extended to utilize faster chips (i.e., GPU) by moving the first layers of the DNN that is always in the REE to these chips. By processing several layers of a DNN in a TEE (SGX) and transferring all linear layers to a GPU, Tramer et al. [TB18] have obtained a 4x to 11x increase for verifiable and private inference in terms of VGG16, MobileNet, and ResNet. For edge devices, another way for expediting the deep learning process is to utilize TrustZone’s AXI bus or peripheral bus, which also has an additional secure bit on the address. Accessing a GPU (or NPU) through the secure bus enables the TrustZone to control the GPU so that the confidentiality of DNN models on the GPU cannot be breached and achieve faster executions for partitioned deep learning on devices.

---

<sup>6</sup> <https://github.com/albanie/convnet-burden> and <https://pjreddie.com/darknet/tiny-darknet/>

## 4.5.2 Models' privacy

**Defending against other adversaries.** DarkneTZ is not only capable of defending MIAs by controlling information from outputs but also capable of defending other types of attacks such as training-based model inversion attack [FJR15, YZCL19] or GAN attack [HAPC17] as they are also highly dependent on the model's outputs. In addition to that, by controlling the output information during inference, DarkneTZ can provide different privacy settings depending on different privacy policies to servers correspondingly. For example, options included in experiments are outputting Top-1 only with its confidence scores, outputting Top-5 with their ranks, or outputting all classes with their ranks which all achieve strong defense against MIAs. Recent research [JSB<sup>+</sup>19] also manipulates confidence scores (i.e., by adding noises) to defend against MIAs, but their protection can be broken easily if the noise addition process is visible to the adversaries of a compromised OS. DarkneTZ also protects layers while training models and conducting inference, so that it leaves broader use cases that deserve further investigations.

**Preserving model utility.** By "hiding" (instead of obfuscating) parts of a DNN model with TrustZone, DarkneTZ preserves a model's privacy without reducing the utility of the model. Partitioning the DNN and moving its more sensitive part into an isolated TEE maintains its prediction accuracy, as no obfuscating technique (e.g., noise addition) is applied to the model. As one example of obfuscation, applying differential privacy can decrease the prediction accuracy of the model [YLP<sup>+</sup>19]. Adding noises to a model with three layers trained on MNIST leads to the model accuracy drop by 5% for small noise levels ( $\epsilon = 8$ ) and by 10% for large noise levels ( $\epsilon = 2$ ) [ACP19, ACG<sup>+</sup>16]. The drop increases to around 20% for large-level noises when training on CIFAR-10 [ACG<sup>+</sup>16]. To obtain a higher accuracy when using differential privacy, one needs to train the model with more epochs, which is challenging for larger models since more computational resources are needed. In recent work, carefully crafted noise is added to confidence scores by applying adversarial examples [JSB<sup>+</sup>19]. Compared to the inevitable decreasing utility of adding noise, DarkneTZ achieves a better trade-off between privacy and utility compared to differential privacy.

### 4.5.3 Limitations & Next steps

The issue of private information leaked from layers' gradients becomes more serious considering that DNN models' gradients are shared and exchanged among devices in collaborated/federated learning. [MSDCS19]'s work successfully shows private (e.g., membership) information about participants' training data using their updated gradients. Recent research [ZLH19b] further reveals that it is possible to recover images and texts from gradients at pixel-level and token-level, respectively, and the last layers have a low loss for the recovery. Although DarkneTZ supports running on-device training and inference, this has not been tested on a federated learning setting. Indeed, by limiting information exposure of layers, these types of attacks could be weakened. Advanced protection for federated learning will be further investigated in later sections. More importantly, how and why private information leaks from intermediate gradients are unclear. Without such an understanding, it is also not reasonable to directly design the corresponding protections. Therefore, one theoretically founded measurement of privacy leakage from neural network gradients is further investigated in the next chapter.

## Chapter 5

# Privacy Measure of Neural Network Gradients

In this chapter, I further strengthen the privacy measure of neural network gradients since it is found that gradients, the intermediate results in collaborative/federated learning, complicate the privacy-related issues beyond sharing neural network weights. This is a natural step considering that on-device trained models are increasingly shared with third parties like servers or other devices in collaborative/federated learning. Privacy attacks on collaborative learning are usually conducted on the exposed elements such as the gradients [MSDCS19, ZLH19a] and intermediate representation [WSZ<sup>+</sup>19]. The attacker can even actively participate in the training stage [HZL19]. Thus, in this chapter, I aim to establish a formal definition of such private information leakage from gradients which answers Research Questions 4 and 5. Specifically, I establish a privacy measurement framework based on  $\mathcal{V}$ -information. This gives grounds for privacy leakage with an information theory basis. In addition, I also aim to provide a further justification on why such privacy leakage happens. Built on top of the generalization idea, I utilize the sensitivity of gradients with respect to private information as the indicator of privacy leakage. Below I first present the notation that will be intensively used and related work for privacy measures of neural network gradients.

## 5.1 Introduction and Related Work

**Notation.** This part uses lower-case italic (e.g.,  $x$ ), lower-case bold italic (e.g.,  $\mathbf{x}$ ), and upper-case bold italic (e.g.,  $\mathbf{X}$ ) for deterministic scalars, vectors, and matrices, respectively. Besides, roman-type upper-case (e.g.,  $X$ ) and lower-case (e.g.,  $x$ ) denote random variables (of any dimensions) and their instances, respectively. Table 5.1 summarises frequently used notations.

Table 5.1: Frequently used notation in this chapter for privacy measures in neural network gradients.

Symbol	Description
$\mathbf{X}$ , $X$ and $x$	Original input data or original information
$\mathbf{G}$ , $G$ and $g$	Gradient matrix or information as random variable
$\mathbf{y}$ , $Y$ and $y$	Label of input data or information as random variable
$\mathbf{p}$ , $P$ and $p$	Latent attribute or latent information
$A$ and $a$	Attack aim as random variables
$I(X; G)$	Shannon mutual information between $X$ and $G$
$f \in \mathcal{V}$	function $f$ in predictive family $\mathcal{V}$ of one attacker
$f[g](a)$	Probability of achieving attack aim $a$ with $f$ on $g$
$\hat{I}_{\mathcal{V}}(G \rightarrow A)$	Empirical usable information from $G$ to $A$ under $\mathcal{V}$
$\mathbf{J}_l^{(\mathbf{G})}(\mathbf{X})$	Input-gradient Jacobian of layer $l$
$\mathcal{R}_l^{(\mathbf{X})}$ , $\mathcal{R}_l^{(\mathbf{p})}$	Original and latent information risk of layer $l$
$\mathbb{G}_l^{(0)}$ , $\mathbb{G}_l^{(1)}$	Linear subspaces of $l$ 's gradients w/o and w/ an attribute
$d_{\text{Gr}}(\mathbb{G}_l^{(0)}, \mathbb{G}_l^{(1)})$	Grassmann distance between subspaces

### 5.1.1 Gradient sharing

In collaborative machine learning, multiple participants train a common model on their local private datasets [MMR<sup>+</sup>17, KMA<sup>+</sup>19]. The *gradients* computed over *one batch* of a participant's data is denoted by  $\mathbf{G}^1 = \frac{\partial \ell(\mathbf{X}, \mathbf{y}, \mathbf{W}^0)}{\partial \mathbf{W}^0}$ , where  $\mathbf{X}$  and  $\mathbf{y}$  denote the local training data and its corresponding labels,  $\mathbf{W}^0$  refers to the current model parameters, and  $\ell(\cdot)$  refers to the training loss function. The  $\mathbf{G}^1$  consists of the gradients of all model layers w.r.t.the training data; which in typical collaborative learning settings is what is sent to other participants or a central server. Conventionally, this process is called FedSGD [MMR<sup>+</sup>17]. To save some communication budgets, it is preferable to consecutively update the received model on *more than one batch*



of local data before sharing, called **FedAvg** [MMR<sup>+</sup>17]. After updating the model on the  $t$ -th batch, the updated parameters are  $\mathbf{W}^t = \mathbf{W}^0 + \sum_{i=1}^t \mathbf{G}^i$ , where  $\mathbf{G}^i$  is the gradients of  $\mathbf{W}^i$  computed based on the  $i$ -th batch of sampled data. When it is not necessary, I might remove the superscripts and refer  $\sum_{i=1}^t \mathbf{G}^i$  as  $\mathbf{G}$  and  $\mathbf{W}^t$  as  $\mathbf{W}$ , for simplicity. Although commonly a participant shares the last snapshot of the updated model that consists of the original model and computed gradients (i.e.,  $\mathbf{W}^0 + \mathbf{G}$ ), this practically amounts to *sharing*  $\mathbf{G}$  since all participants know  $\mathbf{W}^0$ . Private information can then be compromised by the attacker as this  $\mathbf{G}$  could contain original private information (e.g.,  $\mathbf{X}$ ) or some latent attribute information of it. The gradient sharing can be among participants or through a central server; depending on the specific settings [MHK<sup>+</sup>21, ZLH19a, KMA<sup>+</sup>19]. Gradients shared by the central server are usually aggregated across the participants, meaning that disclosing the private information of a specific participant from the aggregated gradients becomes harder. The considered attacks include both, but I do not distinguish them as fundamentally disclosing information from gradients on the central server and participants is the same and only differs in the level of gradient aggregation.

### 5.1.2 Attack and defense measurements

Here I focus on the gradient-based attacks that have been massively studied in recent three years because they are easier to conduct and harder to detect. These attacks are usually based on training some generative adversarial networks and optimizing an attack objective function [HAPC17, ZLH19a]. These gradient-based attacks are categorized into: i) the original information, to reconstruct the input data [HZRS16, ZB20, GBDM20, YMV<sup>+</sup>21], which I regard as data reconstruction attacks (DRA), and ii) the latent information, to infer attributes of the input data [MSDCS19], which I regard as attribute inference attacks (AIA).

Indeed, this type of attack is considered as a severer privacy violation because it does not require full or even any access to the participation and only needs one or a small number of snapshots of gradients. Besides, due to that the attack can be conducted on “static” gradients directly, investigating such attacks could potentially provide a fundamental understanding of

how the produced gradients are correlated to their inputs.

Along with the rising concern of sharing gradients, studies have also started to investigate the performance of current defenses against these attacks. Table 5.2 gives the works that are most related to ours. In addition to the measurement of (semi-standard) defense techniques such as differential privacy (DP) and dropout, [URPPK22] suggested applying simple model adaptation, such as increasing of models’ depth and the number of clients, which drops the attack success rates in general. [LWH<sup>+</sup>22] further developed a python-based software ML-doctor that can be reused partially for conducting attacks and measuring with some defense mechanisms. However, they have several key downsides as above mentioned; they are most likely coarse-grained and purely empirical and do not provide underlying justifications for these private information leakages.

Table 5.2: Related works of measurements on attacks and defenses on neural network gradients.

Ref.	Attack	Defenses	Evaluation and results
[HGS <sup>+</sup> 21]	DRA	Gradient pruning, MixUp, InstaHide	Discover two strong assumptions that current DRAs implicitly make; summarize defenses and their combinations on defending against DRAs (e.g., computation cost estimation)
[URPPK22]	DRA	Differential privacy, layer width and number, number of clients, batch size, dataset complexity (coarsely)	Generative encoder and Deep Leakage from Gradients (DLG) attacks for DRA; SSIM and PSNR as metrics for images; model hyperparameters have significant influences on existing attacks
[LWH <sup>+</sup> 22]	AIA	Differential privacy, knowledge distillation, dataset complexity, overfitting, epochs	Black-box and white-box access using shadow or partial dataset for AIAs; DP-SGD and Knowledge Distillation can only mitigate some of the inference attacks; a modular re-usable software ML-Doctor

### 5.1.3 Information flow via neural networks

An information-theoretically motivated approach to analyzing the information flow through DNN layers is using Shannon information theory. Some previous works have put efforts into analyzing intermediate representations of the input data in a layer-wise manner to understand

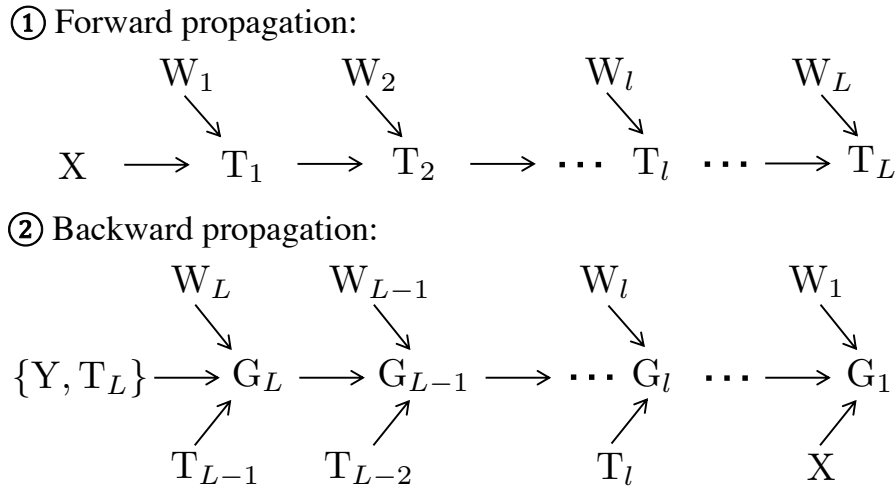


Figure 5.1: Markov chain in forward propagation (from layer 1 to  $L$ ) and backward propagation (from layer  $L$  to 1).  $W$  refers to weights;  $G$  refers to gradients;  $T$  refers to intermediate representations.

the flow of information in the forward propagation of a DNN [MMB10, CHM<sup>+</sup>15, HQL<sup>+</sup>20]. The visualization of data representations (i.e., layers' outputs) and model's parameters indicate that early layers mostly learn general information about data distribution (e.g., face color), while latter layers learn sample-specific information related to the underlying task (e.g., face identity) [ZF14, MV15, YCN<sup>+</sup>15]. For theoretical justifications of such forward propagation analyzes, the widely adopted method has been Shannon mutual information (MI or  $I(\cdot; \cdot)$ , hereinafter) [Sha48, Sha49]. Specifically, for an observed data sample  $X$ , labelled with  $Y$ , the layer-by-layer computations (from layer 1 to  $L$ ) form a Markov chain (see Figure 5.1-①). According to the data processing inequality (DPI) [TPB00],  $I(X; T)$  and  $I(Y; T)$  should *not* increase during the forward propagation, which is extensively studied for DNNs in [TPB00, SZT17, GVDBG<sup>+</sup>19], where authors show how to quantify information flow in forward propagation. Note that there is an implicit assumption that weights  $W$  do not have prior information about one specific  $X$  and the corresponding set of  $T$  in order to apply this estimation.

However, it is still challenging (and we are not aware of any properly designed method) to apply similar methods for analyzing backward propagation for reliable estimation. The computation of gradients  $G_l$  in the backward propagation form a more complicated Markov chain (see Figure 5.1-②), as one  $G$  is formed based on i) the gradients of its latter layer, ii) the intermediate

representations of its former layer extracted in the forward propagation, and iii) the weights of this layer. This does not form a DPI because one  $T$  contains information about  $X$  and  $Y$ . Also, due to that the weights keep updating during backward propagation, the above-mentioned implicit assumption is much unlikely to be held.

## 5.2 Problem Formulation

### 5.2.1 Usable information definition

The concept of *usable information* [XZS<sup>+</sup>20] has been recently introduced as a generalization of Shannon MI [Sha48] to account for both computational constraints and empirical evidence in ML. Usable information relies on a *predictive family*  $\mathcal{V}$ , the definition of which allows incorporating the computational power of a specific (set of) attackers. More importantly, usable information relaxes the famous data processing inequality condition and takes into account that more information can be empirically extracted more information from the data by processing it using some advanced ML models than only by observing the data.

This section provides an adaptation of usable information for measuring how much private information is contained in a DNN’s gradients that can be exploited by a targeted family of attackers. Specifically, the goal is to define a usable information-based quantity to measure which layer(s) in a DNN carry more original or latent information about the private data. This complies with the fact that the amount of private information leakage incurred by sharing the gradients can be measured through a customized framework that takes the attackers’ power into account. The setting considered is as follows: a family of computationally bounded adversaries  $\mathcal{V}$  aims to predict the outcome of a real-valued random variable, e.g., the ability to recover private information. To this end, the adversaries in  $\mathcal{V}$  can use any observed subset of the shared gradients as side information. I present below the general definition of measuring the probabilistic attacker’s aim; after this, I will show how to use it for specific attacks.

Let the random variable  $G$  denote the *shared gradients* and the random variable  $A$  denote the

*attacker's aim*, e.g., the information the attacker aims to successfully extract or the goal that the attacker's aims to reach in the reconstruction of the original data. Let  $\mathcal{V}$  denote the family of attack models that an attacker can use. For  $g \sim G$  and any attack model  $f \in \mathcal{V}$ ,  $f[g]$  denotes the probability distribution over  $A$  computed based on the received gradients  $g$ . Similarly,  $f[\emptyset]$  denotes the prior distribution given no gradient information  $\emptyset$ ; this thus represents attackers who make a random guess for the private information they aim to infer. Then, for some  $a \sim A$ ,  $f[g](a) \in \mathbb{R}$  is defined as the value of the density evaluated at  $a$ , i.e., the probability that an attacker can successfully reach its aim by observing an instance of gradients  $g$ .

**Definition 5.1 (Empirical usable private information in gradients)** *Let  $\mathcal{V}$ ,  $G$ , and  $A$  be defined as above. Given a dataset  $\mathcal{D} = (g_i, a_i)_{i=1}^D$  of the obtained gradients  $g_i$  and the corresponding attack aims  $a_i$ , the usable information from  $G$  to  $A$  given  $\mathcal{D}$  is (empirically) computed as*

$$\begin{aligned} \hat{I}_{\mathcal{V}}(G \rightarrow A; \mathcal{D}) &= \inf_{f \in \mathcal{V}} \frac{1}{|\mathcal{D}|} \sum_{a_i \in \mathcal{D}} -\log f[\emptyset](a_i) \\ &\quad - \inf_{f \in \mathcal{V}} \frac{1}{|\mathcal{D}|} \sum_{g_i, a_i \in \mathcal{D}} -\log f[g_i](a_i), \end{aligned} \tag{5.1}$$

It is worth remarking that the attack family  $\mathcal{V}$  can contain multiple attack models; the inf is then computed as the empirically best-performing model in that family. For an attack family consisting of only one attack model, computing usable information is a probabilistic generalization of the attack success rate<sup>1</sup> In order to apply the probabilistic measure of private information contained in gradients, a probability distribution, e.g.,  $f[g]$ , is required over the outcome of attacks. To achieve that one needs to i) define what is used to measure the outcome of an attack, e.g., based on one chosen metric, ii) define what outcome the attacker aims to achieve, e.g., a good enough success rate, iii) compute the probability of obtaining  $a$ .

In the following sections, I explain the privacy terminology, introduce the definition of specific attack models used to extract private information, and how to define probability distributions

<sup>1</sup> As a technicality, one needs to ensure that there exists an  $f' \in \mathcal{V}$  such that both  $f'[g] = Q$  and  $f'[\emptyset] = Q$  for some probability distribution  $Q$ , i.e., the model can ignore the side information so that the definition maintains non-negativity (see [XZS<sup>+</sup>20] for more details).

over them for  $\mathcal{V}$ -information.

### 5.2.2 Privacy terminology

Let us distinguish between two types of private information:

- **Original information**, i.e., the explicit data considered to contain private information about its owner, such as a complete private image used for model training.
- **Latent information**, i.e., implicit data contained in or related to the original information, such as attributes/properties or parts of the image.

The starting point of this distinction between two types of information is inspired by corresponding attacks, i.e., data reconstruction attacks (DRA) and attribute/property information attacks (AIA), as they exist in different conditions and have different attack aims. The original information leakage is then measured by how much information about a client’s private training data  $X$  an attacker can extract, and the latent information leakage risk is measured by how much information about latent information  $P$  an attacker can extract from the gradients.

It is obvious that  $P$  can be extracted from  $X$ , i.e., once the original data is recovered one can identify the attributes presented in the data. However, this does not imply that the two types of information are equivalent in terms of their cause and localization. Specifically, while the ability to extract latent information from original information holds true, the reverse does not. Knowing the attribute present in a subset of the data does not allow recovering the original information since  $P$  usually has lower entropy than  $X$ . When recovering original data, a unique global minimizer of (5.3) does not always exist or maybe computationally very hard to find [ZB20, FNJ<sup>+</sup>20]. This is in turn related to the fact that aggregation, i.e., the computation/aggregation of the gradients over larger batches of data, can protect original information, but still allows extracting whether the majority of the data had a certain attribute. Furthermore, the computational power of the attacker is crucial in determining what kind of information it is able to extract. According to DPI in the forward propagation (see Section 5.1.3), information of  $X$  is refined in the latter layers of a DNN; however, this may then result in an attacker

with smaller computational power still being able to extract  $P$ . Thus, original information and latent information are considered separately due to the differences in layer-wise localization and their required computational power. I explain the different adversaries on them in detail in the following sections.

### 5.2.3 Attacker: Original information

**Definition 5.2 (Data reconstruction attacker)** *Let  $\mathcal{V}_{DRA}$  present the attack family of the data reconstruction attacker. The attacker has access to some subset of gradients  $g$  and aims to infer the original data sample  $x \sim X$ :*

$$\mathcal{V}_{DRA}(g) \rightarrow \hat{x} \quad (5.2)$$

*The attackers in  $\mathcal{V}_{DRA}$  aim to minimize the distance between  $\hat{x}$  and  $x$  such that they achieve their target reconstruction quality.*

To reconstruct the original data, state-of-the-art DRAs [ZLH19a, ZMB20, GBDM20] usually start by randomly initializing dummy data and feeding it into the model to get dummy gradients. Then, the dummy data is optimized such that the dummy gradients get close to the real gradients. More specifically, the DRA tries to minimize the following objective by updating both dummy input  $\hat{x}$  and dummy label  $\hat{y}$ ,

$$\operatorname{argmin}_{\hat{x}, \hat{y}} \operatorname{dist}(\hat{g}, g), \quad (5.3)$$

where  $\hat{g}$  and  $g$  denote the dummy and observed gradients, respectively, and  $\operatorname{dist}(\cdot, \cdot)$  can be any suitably chosen distance metric for them. For example, [ZLH19a, ZMB20] used  $\|\hat{g} - g\|_p$  with  $p$ -norm, while [GBDM20] used cosine similarity between  $\hat{g}$  and  $g$ . The attack will be successful if minimizing the above distance will result in the dummy input data being close to the real private information, which can reach high accuracy (e.g., to a pixel-wise level for images) as shown in [ZLH19a, ZMB20, GBDM20].

Note that  $\hat{x}$  and  $\hat{y}$  can be one individual data point or a batch of data points. For a batch size  $B > 1$ , the optimizer can have  $B!$  different permutations, so that choosing the right gradient descent direction is complicated - as a solution a single training sample can be updated in each optimization step [ZLH19a]. It may fail when gradients have sufficient aggregation (e.g., a large batch size) because the optimization becomes hard to solve (i.e., more variables to optimize over than we have constraints) when the gradients of multiple images are aggregated together [ZLH19a, ZB20].

**Probabilistic measure for original information.** To derive  $f[g]$  for original information, we first need to define a metric for measuring the successfulness of one attack. This can be any suitable metric that measures the similarity between original information and reconstruction information, depending on the data types (e.g., images or audios) and the focus (e.g., better realization for machines or humans). As one example, both Peak signal-to-noise ratio (PSNR) and Structural Similarity Index (SSIM) can be measures of images; however, the latter is defined based on the human visual system and thus is more suitable for measuring human perception [WBSS04, HZ10] while the former is easier to compute on machines. Here I consider using SSIM as the metric. For one reconstruction data  $\hat{x}$  of one particular attack result based on  $g$ ,  $\text{SSIM}(\hat{x}, x)$  is a quantity which measures the perceptual similarity between reconstructed and original data, where  $\text{SSIM}(\hat{x}, x) = l(\hat{x}, x)c(\hat{x}, x)s(\hat{x}, x)$ , a combination of luminance ( $l$ ), contrast ( $c$ ), and structure ( $s$ ).  $\text{SSIM}$  lies within a range of  $[0, 1]$ , with the value of 1 being achieved for a close-to-perfect reconstruction.

Lets define a successful attack aim  $a$  as observing  $\text{SSIM} \geq a_T$  for some user-chosen threshold  $a_T$ . We then estimate  $f[g]$  as follows: i) fix a gradient subset  $g$ , ii) for  $R$  randomly initialized reconstructions we perform the reconstructions and compute  $\text{SSIM}$  to obtain a set of values  $\{g, \text{SSIM}_i\}_{i=1}^R$ , iii) using these  $R$  samples we compute the probability of obtaining  $a$  for those samples using an empirical estimate of the probability:

$$f[g](a) = \frac{\sum_{i=1}^R \mathbf{1}_{\text{SSIM}_i \geq a_T}}{R}, \quad (5.4)$$

where many reconstructions were successfully divided by the total number of reconstructions.



When having no side information, the best one could do is reconstruct a random image; similar to the above, one can compute in how many samples  $\text{SSIM}_i \geq a_T$ ,  $i = 1, \dots, R$ ; for  $a_T > 0$  this will typically result in a  $f[\emptyset](a)$  value close to 0. While computationally expensive due to repeating the attack for the  $R$  reconstruction initialization, such a framework allows us to compute the probabilistic success rate of reconstructing the data using a subset of gradients  $g$ . It is worth remarking that for other types of data one can adopt measures in similar ways (e.g., PSNR for audio).

#### 5.2.4 Attacker: Latent information

Extracting original information is not always feasible; especially when the gradients the attacker has access to are aggregated over many samples. In those cases extracting ‘high-level’ information (i.e., information that is more abstract and with lower entropy) may be more realistic. That is, instead of recovering the exact data point, an attacker [MSDCS19, GWY<sup>+</sup>18, NSH19] aims to recover latent information such as various attributes of data points.

**Definition 5.3 (Attribute inference attacker)** *Let  $\mathcal{V}_{AIA}$  be the attribute attacker. The attacker has access to some subset of gradients  $g$  and aims to infer the target sample’s attribute  $p \sim P$  is:*

$$\mathcal{V}_{AIA}(g) \rightarrow \hat{p}. \quad (5.5)$$

*The success rate or advantage of  $\mathcal{V}_{AIA}$  is defined as the ability to recover the correct attribute.*

It is assumed that in practice the attacker has access to *auxiliary* batches of data with an attribute and without an attribute. The attacker was able to collect a set of gradients  $\{g_i^0\}_{i=1}^T$  and  $\{g_i^1\}_{i=1}^T$ , without and with the attribute, respectively, where  $T$  refers to the iterations in which the attacker obtained the model and computed the gradients. Using these two sets of gradients the model then trains a binary classifier. Specifically, one AIA tries to minimize the following objective by updating parameters  $\theta$  of the classifier based on auxiliary set of gradients

$\{\mathbf{g}_i^0\}_{i=1}^T$  and  $\{\mathbf{g}_i^1\}_{i=1}^T$  and the corresponding labels 0 and 1,

$$\operatorname{argmin}_{\theta} \operatorname{dist}(\hat{\mathbf{p}}, \mathbf{p}), \quad (5.6)$$

where  $\hat{\mathbf{p}}$  is the predicted attribute, and  $\operatorname{dist}(\cdot, \cdot)$  can be a chosen distance metric such as the cross-entropy loss. Then given a new gradient sample one can classify it as coming from a dataset where the (majority of) data samples are with or without the chosen attribute.

The success of such an attack thus relies on how well the classifier can distinguish the samples from the gradients with the attributes from those without. Note that the membership inference attack (MIA) is another well-known attack that infers the existence of a particular client's data in the training dataset [NSH19, SSSS17]. This membership information can be inferred using the same method in Equation (5.6) and may be regarded as a kind of 'high-level' latent information reflecting the data point's existence.

**Probabilistic measure for latent information.** For latent information the attack aim is the successful inference of an attribute, e.g., infer the presence of an attribute  $a = 1$ . It is worth remarking that this can easily be generalized to a set of more than two attributes since we can convert the attack aim to be binary: i) with the attribute or ii) without the attribute. To derive  $f[\mathbf{g}]$ , I use the above-mentioned auxiliary data to compute the gradients and use these to train a binary classifier. Denote the output of the model by  $\hat{\mathbf{y}}$ , i.e., a two-dimensional vector. For  $a = 1$  we can transform it into a probability as follows (similar to [XZS<sup>+</sup>20] and Section 4.2 of [Bis06])

$$f[\mathbf{g}](1) = \frac{e^{\{\hat{\mathbf{y}}\}_1}}{e^{\{\hat{\mathbf{y}}\}_0} + e^{\{\hat{\mathbf{y}}\}_1}}, \quad (5.7)$$

where  $\{\hat{\mathbf{y}}\}_1$  represents the element of  $\mathbf{y}$  corresponding to  $a = 1$  and similarly one would derive the probability of  $a = 0$ . Then,  $f[\emptyset](a) = \frac{1}{2}$ , i.e., the value of the uniform distribution over the outputs ( $\{\hat{\mathbf{y}}\}_0 = \{\hat{\mathbf{y}}\}_1 = 0.5$ ).

### 5.2.5 The role of computational power

Shannon MI relies on the implicit assumption that “the attacker is computationally unbounded”. If this is indeed the case, then both original and latent information are equally well-extractable from the first layer, as having given  $X$  we can theoretically infer any  $Y$ . For a computationally unbounded attacker, the information in the consequent layers can only decrease, in accordance with the DPI. But in practice, the computational power and also the knowledge of the attacker are limited. One can then reasonably expect that the latter layers in a model are able to extract features that are more useful for the target task (i.e., the classification at hand), so that the usable information for the task  $Y$ , that is captured in the gradients, should be *increasing* throughout the layers. Consequently, one may also assume that other attributes besides the main task are also easier to extract from the latter layers. This would seemingly be in violation of the DPI; however, it can be reconciled under the framework of usable information [XZS<sup>+</sup>20] which allows for the usable information to increase throughout the network due to these computational constraints. This does not imply that less latent information is present in earlier layers, but earlier layers will require the attacker to perform more heavy computations on this data to infer  $P$ .

The computational constraints also imply bounds in terms of the computational power of the attacker. In fact, from [XZS<sup>+</sup>20] the following holds: if  $\mathcal{U} \subseteq \mathcal{V}$  and  $f_1[\emptyset](a) = f_2[\emptyset](a)$  for  $f_1 \in \mathcal{V}$  and  $f_2 \in \mathcal{U}$ , then

$$\hat{I}_{\mathcal{V}}(G \rightarrow A; \mathcal{D}) \geq \hat{I}_{\mathcal{U}}(G \rightarrow A; \mathcal{D}). \quad (5.8)$$

## 5.3 Sensitivity Measure on Private Information

Having defined the *usable information* measure for quantifying the amount of private information in the gradients in the previous section, I now explain how to reason about the leakage based on the *gradient sensitivity* measure and validate the efficacy of usable information. The sensitivity measure is motivated in two ways: i) it gives further insight into model character-

istics that facilitate private information leakages, and ii) it allows us to measure information leakage in a way that does not depend on the attack models and thus allows us to validate the proposed method without making any explicit assumption on the attackers' aims. In the following, I explain how *gradient sensitivity*, which previously has been studied for examining model robustness and generalization [NBA<sup>+</sup>18, DLJ<sup>+</sup>18], can be expanded into metrics to be used for studying both original and latent information.

### 5.3.1 Sensitivity justification

Consider again the objective in Equation (5.3). In cases where the objective function  $\text{dist}(\hat{g}, g)$  has multiple or very flat minima, solving the optimization and thus recovering the true original data reconstruction will be more challenging as multiple solutions with similar objective function values may exist. Therefore, the success of data reconstruction depends on the structure of the objective function. Previous work has introduced rank-based metrics to quantify the ability to find the unique optimizer [ZLH19a] relying on the fact that the optimization problem can be rewritten as a system of equations. If more parameters exist than equations, no unique solution will exist. Alternatively, one could choose a metric that accounts for the flatness of the loss function (and hence the gradients) in parameter or input space. As these flat minima are a common occurrence in the loss functions of DNNs [DVSH18], having a metric that is able to account for this is essential. This notion of sensitivity based on flatness has also been applied in measuring model robustness on adversarial example attacks [ZKS<sup>+</sup>18, DLJ<sup>+</sup>18]. A model that is non-sensitive to certain changes in inputs is expected to have high robustness. On the other hand, by bounding the sensitivity of outputs w.r.t. inputs leveraging differential privacy, one can certify model robustness to changes in input i.e., adversarial examples, proposed in [LAG<sup>+</sup>19].

### 5.3.2 Sensitivity of gradients w.r.t. original information

For quantifying the original information leakage, the *gradients sensitivity* metric is quite straightforward: I compute the sensitivity of the *gradient* w.r.t. the private *input* via the Jacobian matrix of the gradients. I use the following intuition to measure private information leakage: if the gradient sensitivity is low and thus gradients change insignificantly when altering the input, reconstructing the input will be more challenging as more possible reconstructions with similar accuracy will exist. This research will utilize the Jacobian matrix of the gradients w.r.t. the input (similar to input-output Jacobian [NBA<sup>+</sup>18, SGSR17]) to reflect how sensitive the gradients are when changing the input, i.e., a sensitivity measure on gradients. The input-gradient Jacobian is calculated by:

$$\mathbf{J}_l^{(\mathbf{G})}(\mathbf{X}) = \frac{\partial \mathbf{g}_l(\mathbf{X})}{\partial \mathbf{X}} = \frac{\partial}{\partial \mathbf{X}} \left( \frac{\partial \ell(\mathbf{X}, \mathbf{y}, \mathbf{W})}{\partial \mathbf{W}_l} \right), \quad (5.9)$$

where  $\mathbf{g}_l(\cdot)$  represents the function that produces layer  $l$ 's gradients  $\mathbf{G}_l$ . Besides,  $\ell(\cdot)$  is the loss function over  $\mathbf{X}$ , ground truth  $\mathbf{y}$ , and parameters of the complete model  $\mathbf{W}$ , so  $\mathbf{g}_l(\cdot)$  can be regarded as the partial derivative of  $\ell(\cdot)$  w.r.t. layer  $l$ 's parameters  $\mathbf{W}_l$  (i.e., backward propagation).

Then we compute the Frobenius norm (referred to  $F$ -norm) of the above input-gradient Jacobian matrix [NBA<sup>+</sup>18], which indicates the general original information risk. As in our case Jacobians are compared across layers with different sizes, I include two other norms, i.e., 1-norm, and the  $\infty$ -norm. Using different norms will help in capturing adversaries with different capabilities, similar to [CBG<sup>+</sup>17, LAG<sup>+</sup>19] where  $p$ -norm reflects how the attacker measures distance between two data samples (e.g., 1-norm reflects all dimensions of the data sample, and  $\infty$ -norm reflects on one dimension). Thus, given  $K$  data samples, I compute the Jacobian with  $p$ -norm (referred to as ‘Jacobian  $p$ -norm’ hereinafter) averaged over the data samples as the

leakage risk of the original information ( $\mathbf{X}$ ) in layer  $l$ 's gradients:

$$\begin{aligned}\mathcal{R}_l^{(\mathbf{X})} &= \mathbb{E}_{\Delta\mathbf{X}} \left[ \|\mathbf{g}_l(\mathbf{X}) - \mathbf{g}_l(\mathbf{X} + \Delta\mathbf{X})\|_p \right] \\ &= \frac{1}{K} \sum_{k=1}^K \left\| \mathbf{J}_l^{(\mathbf{G})}(\mathbf{X}_k) \right\|_p\end{aligned}\tag{5.10}$$

where  $p = F, 1,$  or  $\infty$ . Note that the size of  $\mathbf{G}_l$  may still have an impact on the computed sensitivity. A larger  $\mathbf{G}_l$  size can result in a larger sensitivity in terms of 1-norm and  $F$ -norm. When reconstructing high-dimensional input data, a large  $\mathbf{G}_l$  size is necessary, which can be reflected by these norms.

### 5.3.3 Sensitivity of gradients w.r.t. latent information

For attacks on latent information [MSDCS19], the success of the attack lies in the ability of the classifier to distinguish between whether certain gradients were computed over data that did or did not have a certain attribute. Specifically, it relies on how accurate the classifier trained on samples with or without an attribute can become (see Section 5.2.4).

For two reasons, computing the sensitivity of gradients w.r.t. latent information is not as straightforward as original information. First, the latent information (the private label) is not explicitly fed into the DNN, and thus the backpropagation computational graph cannot be used to compute  $\frac{\partial \mathbf{g}}{\partial p}$ . Second, in Equation (5.10), the sensitivity is computed separately over each sample input, but a large number of inputs have the same latent information and the attack's classifier only produces a binary prediction. Thus, the fine-grained sensitivity measure in (5.10) cannot be directly used to collect the amount of latent (i.e., coarse) data in the gradients.

Therefore, to quantify the changes in gradients, this work offers a more coarse-grained approach. First, the target dataset is separated into two parts based on the presence of one specific latent information, and then the subspace distance between gradients computed on these two parts of the dataset is compared. This still follows the same intuition as understanding how sensitive the gradients are w.r.t. latent information but considers a more 'high-level' measurement.

Let us assume that a dataset  $\mathcal{S}$  consists of two disjoint subsets:  $\mathcal{S}_0$ , whose samples do not have the target latent information, and  $\mathcal{S}_1$ , whose samples have the target latent information. Then one can obtain the corresponding gradients of layer  $l$  computed on these two subsets by:

$$\mathbf{G}_l^{(0)} = \mathbb{E}_{\mathbf{X} \in \mathcal{S}_0}[\mathbf{g}_l(\mathbf{X})] \text{ and } \mathbf{G}_l^{(1)} = \mathbb{E}_{\mathbf{X} \in \mathcal{S}_1}[\mathbf{g}_l(\mathbf{X})]. \quad (5.11)$$

For each pair of matrices  $\mathbf{G}_l^{(0)}$  and  $\mathbf{G}_l^{(1)}$  (in  $l \in \{1, \dots, L\}$ ), let  $\mathbb{G}_l^{(0)}$  and  $\mathbb{G}_l^{(1)}$  denote their corresponding linear subspaces, respectively. To measure the difference between the computed gradients with and without target latent information, the Grassmann geodesic distance [Drm00, YL16] between these two subspaces can be computed as the leakage risk of this latent information  $\mathbf{p}$  in layer  $l$ 's gradients:

$$\begin{aligned} \mathcal{R}_l^{(\mathbf{p})} &= \text{dist}(\mathbf{G}_l^{(0)}, \mathbf{G}_l^{(1)}) = d_{\text{Gr}(k,n)}(\mathbb{G}_l^{(0)}, \mathbb{G}_l^{(1)}) \\ &= \left( \sum_{i=1}^k \theta_i^2 \right)^{1/2}, \end{aligned} \quad (5.12)$$

where  $\text{Gr}(k, n)$  denotes the Grassmann manifold, and both  $\mathbb{G}_l^{(0)}$  and  $\mathbb{G}_l^{(1)}$  are elements of  $\text{Gr}(k, n)$  and are  $k$ -dimensional linear subspace in  $\mathbb{R}^n$ .  $k$  is the layer  $l$ 's gradient size  $N_l \times N_{l-1}$ .  $\theta_i$  for  $i \in \{1, \dots, k\}$  are principal angles between the two subspaces which can be computed using numerical methods [BG73]. Here the Grassmann distance is used; other common distances defined on Grassmannians (e.g., Asimov) [YL16] can be derived similarly.

## 5.4 Numerical Evaluation

In this section, to validate the proposed *usable information* (Section 5.2.1) measure, I first provide a comprehensive experimental evaluation using benchmark datasets and DNN architectures, and then I utilize the *sensitivity measure* (Section 5.3) to further verify the integrity of the proposed measure.

Table 5.3: Models and datasets used in experiments.

Model	Architecture	Datasets
LeNet [LBB <sup>+</sup> 98]	C6(5)-P(2)-C16(5)-P(2)-F120-F84-O	CIFAR-100,
AlexNet [KSH12]	C8(5)-C16(3)-P(2)-C32(3)×3-P(2)-F120-F84-O	LFW, CelebA,
VGG9 [SZ14]	C8(3)-C16(3)×2-P(2)-C32(3)×3-P(2)-F256-F128-O	& PubFig
TextClf [Kim14]	C16(3)-P(2)-C16(8)-P(2)-F50-O	IMDB & CSI

Notations.  $Ci(j)$ : a convolutional layer with  $i$  2D filters each of size  $j \times j$ .  $Fi$ : a fully connected layer with  $i$  neurons.  $P(j)$ : a max-pooling layer with a window size of  $j \times j$ .  $O$ : the output classification layer with  $Y$  classes (depending on the dataset).

### 5.4.1 Evaluation setup

**Model.** This research performs experiments on four DNN models that are mostly used by previous works on privacy threats of collaborative learning [ZLH19a, ZMB20, GBDM20, MSDCS19, NSH19]. See Table 5.3 for the details of each DNN’s architecture. The number of neurons in the output layer is adjusted based on the number of classes of the dataset used to train. All models use ReLU activation functions for Conv and FC layers (except the output layer with Softmax).

**Datasets.** I conduct evaluations on six datasets that have been used in previous works for DRA or AIA evaluations. LeNet, AlexNet, and VGG9 models are trained on CIFAR-100 [KH09] and *three other image datasets* with attributes, including Labeled Faces in the Wild (LFW) [HMBLM08], Large-scale CelebFaces Attributes (CelebA) [LLWT15], and Public Figures Face Database (PubFig) [KBBN09]. TextClf is trained on *two text datasets*: IMDB reviews [MDP<sup>+</sup>11] and CSI corpus [VD14]. LFW contains 13233 face images (cropped as  $62 \times 47$  RGB). All images are labeled with around 100 attributes such as gender, race, age, hair color, etc. I use a subset of the cropped version (i.e., 15000 images of  $64 \times 64$  RGB) of CelebA which contains face images of celebrities with 40 attribute annotations such as gender, hair color, eyeglasses, etc. I also use a cropped version ( $100 \times 100$  RGB) of PubFig which contains 8300 facial images made up of 100 images for each of 83 persons [PSZC11], marked with 73 attributes (e.g., gender, race, etc). In IMDB reviews [MDP<sup>+</sup>11] dataset, each review is labeled with sentiment and length, and in CSI corpus [VD14], each corpus review is labeled with sentiment, veracity, length, etc.



**Training setup.** I conduct experiments on a cluster with 50 nodes where each has 4 Intel(R) Xeon(R) E5-2620 CPUs (2.00GHz), one/two NVIDIA RTX 6000 GPU(s) (24GB), and 24GB/48GB DDR4 RAM. PyTorch v1.4.0 is used for both usable information (e.g., conducting DRAs and AIAs) and sensitivity computations. I refer to the previous research [ZMB20, MSDCS19] for attack settings.

### 5.4.2 Layer-wise localization of information leakages

**Attack families.** This research defines an attack family (with computational bounds) that covers the most recent attacks including [ZLH19a, ZMB20, GBDM20, MSDCS19] to compute the empirical usable information as in Equation (5.1). In particular, for original information, the attack family includes deep learning techniques with the same training hyperparameters used in the previous attacks [ZLH19a, ZMB20, GBDM20]. Moreover, for the latent information, random forests are included in the attack family with the hyperparameters tuned in a state-of-the-art manner based on previous successful attacks [MSDCS19]. In general, to empirically achieve the Infima over the attack family in Equation (5.1), one needs not only to include the state-of-the-art attacks in the literature but also to perform hyperparameter tuning for learning better attack models. I remark that any new-coming attacks can be added to the attack family and their results can be merged into current results without redoing the computation of previous attack models.

**Usable original information.** To measure layer-wise information leakage, I perform attacks on individual layers. First, I found that all existing DRAs fail in achieving any meaningful outcome when using gradients of only one layer. I thus measure the leakage on the set of at least two consecutive layers at a time. Specifically, in Figure 5.2, set  $\{1\}$  denotes layers 1 and 2, set  $\{2\}$  denotes layers 2 and 3, *etc.* Figure 5.2 (Left) shows the results of the results of the attack family (measured with  $\text{inf}$ ), which includes DLG [ZLH19a], iDLG [ZMB20], and Inverting Gradients [GBDM20]. In Figure 5.2 (right), I show the results of a single attack: iDLG attack model [ZMB20].

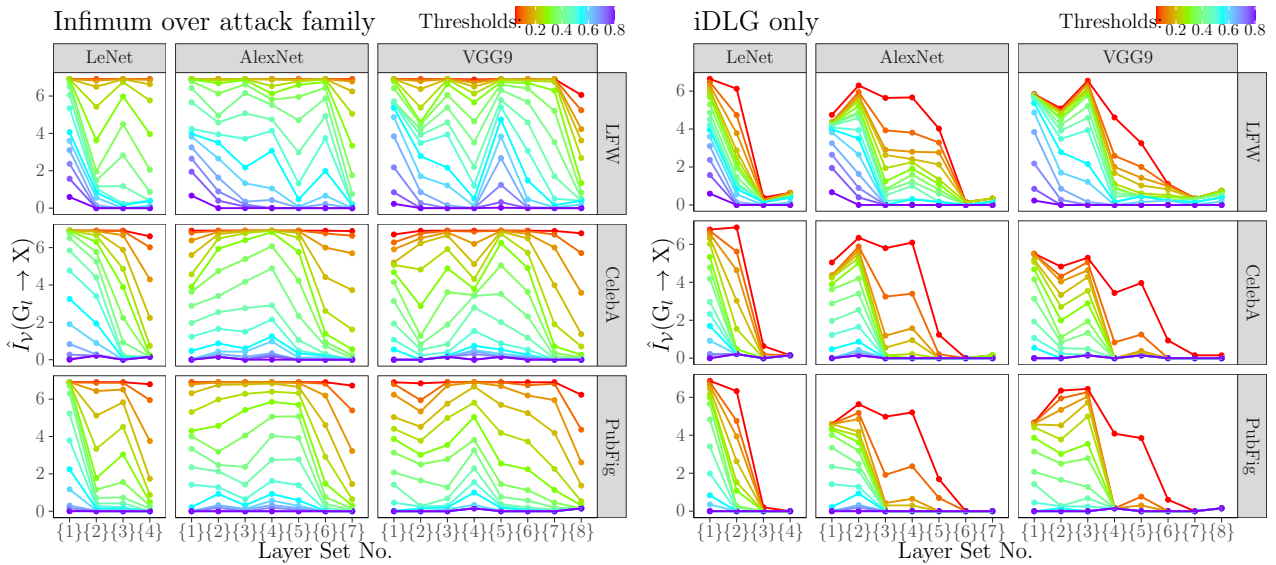


Figure 5.2: Original information leakage measured by usable information with thresholds from 0.05 to 0.8 (with a step size of 0.05) in each 2-layer set. The left figure shows the results of Infimum over attack family [ZLH19a, ZMB20, GBDM20]. The right figure shows the results of iDLG model only [ZMB20].

The results of the attack family, including three attacks, indicate that for LeNet the leakage is monotonically decreasing for thresholds larger than 0.5, but for deeper DNNs, i.e., AlexNet and VGG9, the leakage is highest in the middle layer-sets. The iDLG attack [ZMB20] model shows a little different pattern where the original information leakage risk generally decreases when moving from the first 2-layer set to the last 2-layer set. This could probably be due to that the cosine similarity used in Inverting Gradients [GBDM20] for the cost function could better capture gradient differences than the euclidean distance used by [ZMB20], especially for middle layers in more complicated DNN architectures. Thus, Inverting Gradients outperforms iDLG in most cases, so that the former will represent the final original information leakage over the defined attack family. Still, one common observation is that for both attack models the last layer-set has the lowest level of information leakage.

**Usable latent information.** For latent information, the work additionally trains the TextClf on the two text datasets and put CIFAR100 aside because it does not have multiple attributes. Also, since each dataset can have multiple types of attributes considered as private latent information, I compute usable information on all these (private) attributes and then fit smooth curves using nonparametric local weighted (i.e., LOESS) regression [CD88] to clarify the general trend across layers. I also normalize the computed usable information of one model with the

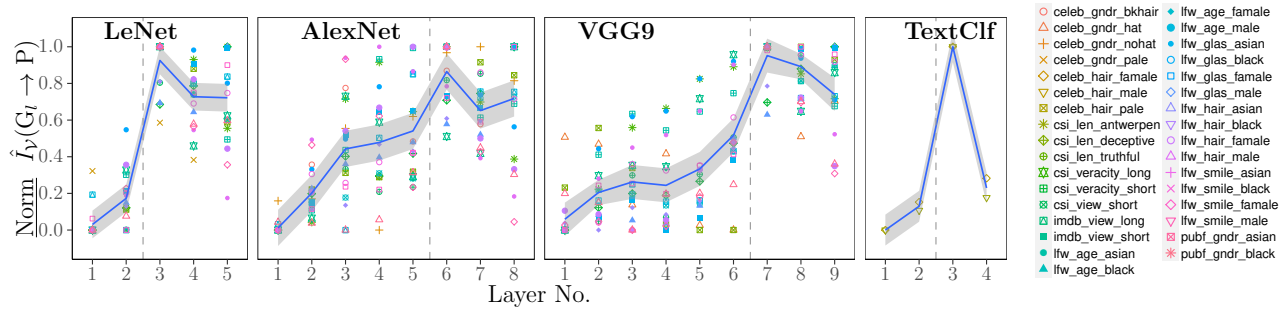


Figure 5.3: Latent information leakage, measured by usable information, in each layer's gradients. Scattered points refer to private attributes being measured for different datasets and classification tasks. Blue lines give the LOESS regression curve. Dashed lines (--) separate the feature extractor and the classifier of the model.

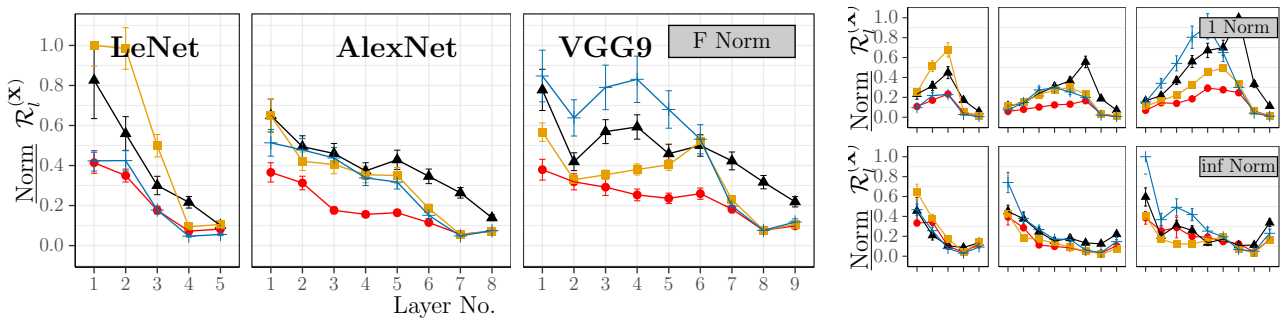


Figure 5.4: Original information leakage risks, measured by sensitivity ([Left]  $F$ -norm; [Right Bottom] 1-norm; [Right Top]  $\infty$ -norm), in each layer's gradients, based on CIFAR100 (—), LFW (—), CelebA (—), PubFig (—) dataset. Error bars are 95% confidence intervals.

maximum value of its layer for comparability across models and tasks. As in Figure 5.3, the results show that the latent information leakage risk follows a trend that increases when moving through the feature extractor layers, reaches its maximum at the first classifier layer, and then decreases. This tendency is similar across all models. In addition, the results based on usable information also confirm the analysis in Section 5.1.3 that the private information flow in gradients does not satisfy DPI.

### 5.4.3 Sensitivity analysis

This research further performs sensitivity analysis to evaluate information leakage by measuring the gradient changes w.r.t. inputs.

**Sensitivity of gradients w.r.t. original information.** This work computes Equation (5.10) to measure original information leakage risks. As shown in Figure 5.4 (Left) for the  $F$ -norm,

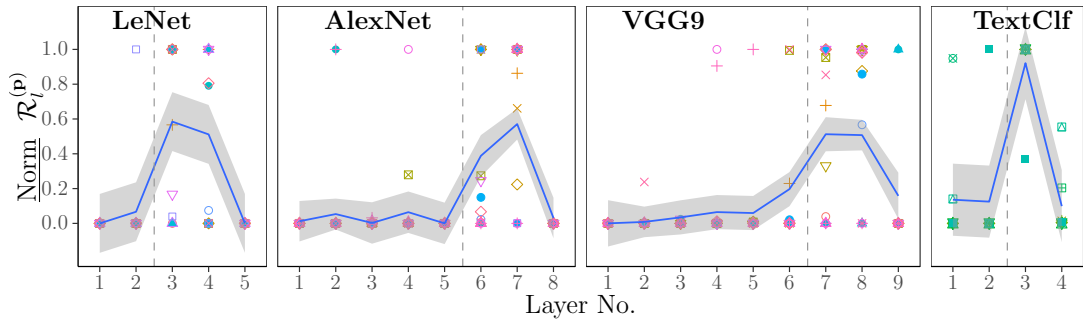


Figure 5.5: Latent information leakage risks, measured by subspace distances, in each layer’s gradients. Scattered points correspond to private attributes. Blue lines give the LOESS regression. Dashed lines (--) refer to the feature extractor-classifier connection.

the sensitivity in overall shows a similar pattern as the usable information with the attack model of [ZMB20] in Figure 5.2. Specifically, gradients of the first layers are more sensitive to changes in inputs compared to the latter layers, thus they could potentially carry more private original information that can be detected by adversaries easier. Besides, 1-norm sensitivity results, in Figure 5.4 (Right-Top), have a similar pattern with the attack model of [GBDM20], where DNNs’ middle parts, i.e., around the connection from feature extractor to classifier, get the highest information leakage. We know that the difference for attacks in usable information probably is due to the distance measure in their cost functions, i.e., iDLG [ZMB20] used 2-norm while Gradient Inverting attack model [GBDM20] used cosine similarity. This information leakage may be tied with the sensitivity of gradients w.r.t. the input (i.e., private information) with  $p$ -norm where  $p$  links to attack models with different capabilities.

**Sensitivity of gradients w.r.t. latent information.** This work computes Equation (5.12) to measure latent information leakage risks. Similar to usable information, I measure all possible attributes of used datasets and again plot the regression curves for each model. As Figure 5.5 shows,  $\mathcal{R}_l^{(p)}$  generally follows a similar trend as the latent information risks computed using usable information. That is, the layers related to the classifier have higher leakage risks than the feature extractor layers. Most risks are either in the classifier’s first layer or the second layer. While this is slightly different from the results in Figure 5.3 where the first layer always has the highest leakage, the overall trend remains the same. Notice that several scattered points closely coincide, and the last layer has a lower risk compared to that in usable information. One explanation is that this measure may tend to capture the extreme risks among layers so

the first and second layers of the classifier show significantly higher leakage risks than the other layers. Overall, the computation measuring gradient changes (i.e., sensitivity) depends only on the gradients is easy to compute and can capture the main pattern of layer-wise leakage risks measured by attack-based usable information.

#### 5.4.4 Impacts of training hyperparameters

We now analyze how some training hyperparameters in collaborative learning affect gradient-based information leakages. For computing original information leakage, I fix the threshold to evaluate the expected performance of attack models for different hyperparameter values. Here, instead of computing usable information for different thresholds, I fix the threshold to evaluate the expected performance of attack models. Specifically, I remark that a given threshold in Equation (5.1) provides a tunable parameter for measurement depending on whether the considered situation is severe or mild, but this could be troublesome for general measurement as suitable thresholds required in every setting are unknown. In particular, I fix the threshold to the expectation of attack outcomes' probability distribution; then for one individual attack on one sample we have  $a_T = E_{a \sim \mathcal{A}}(f[g])$ . In such a way, I evaluate the expected performance of one attack on  $g$ .

**Aggregation levels.** Performing aggregation before sharing gradients with others is a common strategy in collaborative learning. This could be the *local* aggregation (e.g., batch size in FedSGD or multiple steps in FedAvg) and also the *global* aggregation over updates of all clients. For the aggregation measurement of one target's (victim) original or latent information, I mix it with multiple gradients of non-target information, i.e., information irrelevant to the private information targeted by adversaries. The empirical results show that aggregation can significantly reduce gradients' information leakage risks (see Figure 5.6). Specifically, disclosing target original information from gradients that have been updated on non-target information with an amount 10 times larger than the amount of target information can be very difficult (e.g., a batch size of 10 in [ZLH19a, ZMB20]). When gradients are mixed with non-target information which is 30 times the amount of target information, the original information risk

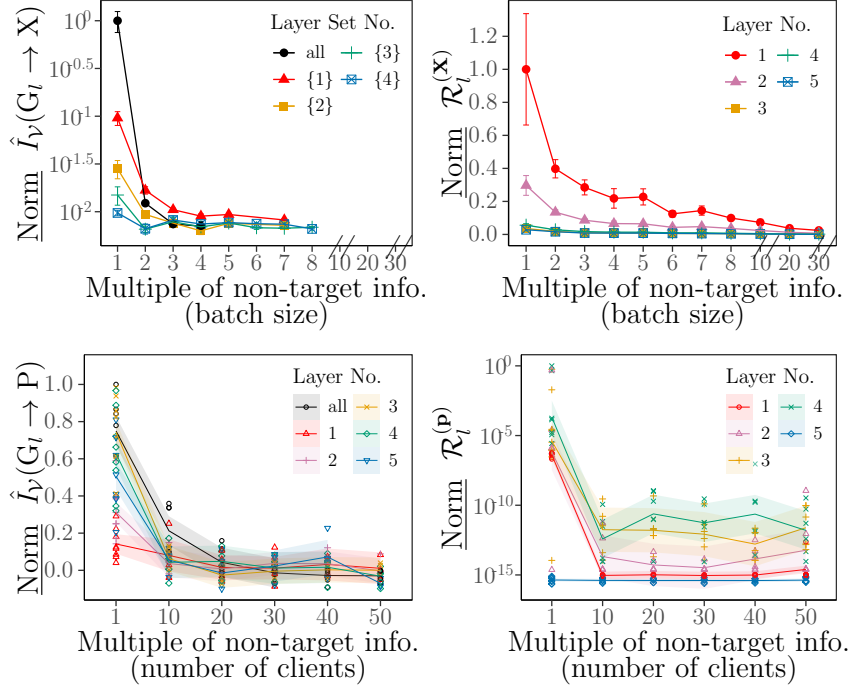


Figure 5.6: Influence of gradient aggregation on risks of original information [Left two] and latent information [Right two], measured on LeNet trained on CIFAR100 and LFW, respectively (Note: Points corresponding to failed trials of attacks are not plotted. Logarithmic scale is used in some plots for better visualization).

measured by sensitivity is extremely low (near zero). Also, extracting latent information from 30~50 times non-target information is hard (e.g., 30~50 participants in [MSDCS19]).

**Epochs.** This part measures the information leakage risks of gradients at specific epochs (from 1 to 100) during training. After epochs of training, the information leakage from gradients is calculated. The results show that overall the epoch only has a negligible impact on leakage risks (see Figure 5.7). Except for the leakage risk drop in the first several epochs (i.e., first 10 epochs for original information and first 20 epochs for latent information), epochs barely change gradients leakage risks. This may be due to the fact that the magnitude of the updated gradients does not change significantly throughout these epochs (for fixed learning rate). In such a sense, in the first epochs, the machine learning optimizer quickly converges somewhere near the optimum and increases the model accuracy very quickly; in later epochs, the optimization becomes slower, and consequently, the magnitude of gradient changes stabilizes. It is expected that with (much) more epochs of training, the information leakage may significantly change due to overfitting or convergence, but the analysis shows that with a reasonable number of epochs,

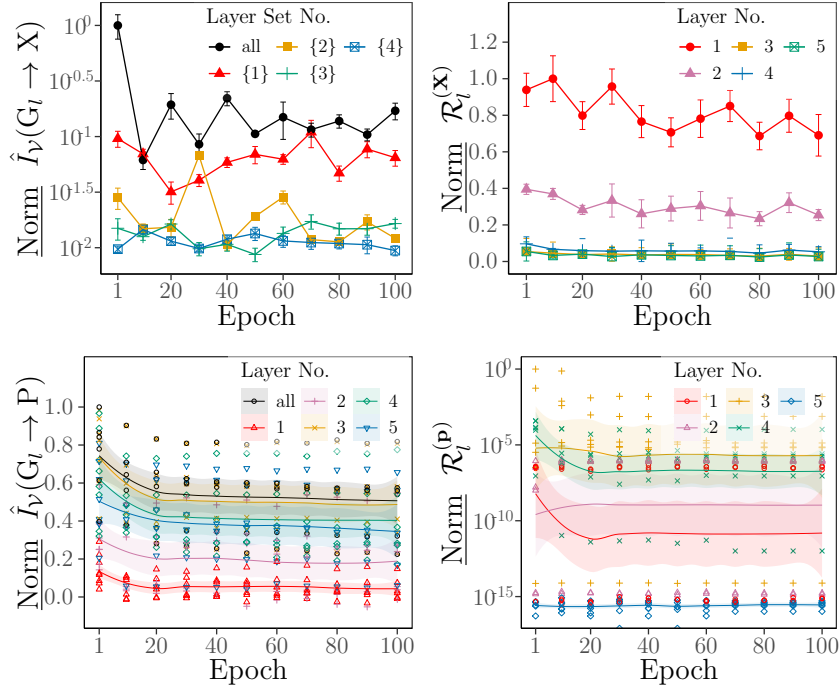


Figure 5.7: Influence of training epoch on risks of original information [Left] and latent information [Right], measured on LeNet trained on CIFAR100 and LFW, respectively.

it does not impact the leakage a lot.

### 5.4.5 Impact of defense mechanisms

**Dropping fraction or number of gradients.** In Figure 5.8, I present the usable information results when only parts of the gradients are available as side information. It is shown that with a particular dropping rate, the first layer of the classifier always has the highest usable information for the target property. Having only 5% of the gradients (or even 1% of the sensitive layers) available can still leak a great amount of private property information compared with having complete gradients. However, the way of keeping a *fixed fraction* of gradients can be influenced by the total number of gradients of the layer. I then measure the usable information from a *fixed number* of gradients from each layer. Interestingly, the results still show that classifiers' first layers contain the most private latent information.

**Differential privacy.** Adding DP noises before releasing gradients could be one way to reduce the privacy risks. Here I follow [SS15, JE19b], i) to clip gradients using  $l_2$ -norm and then ii) to

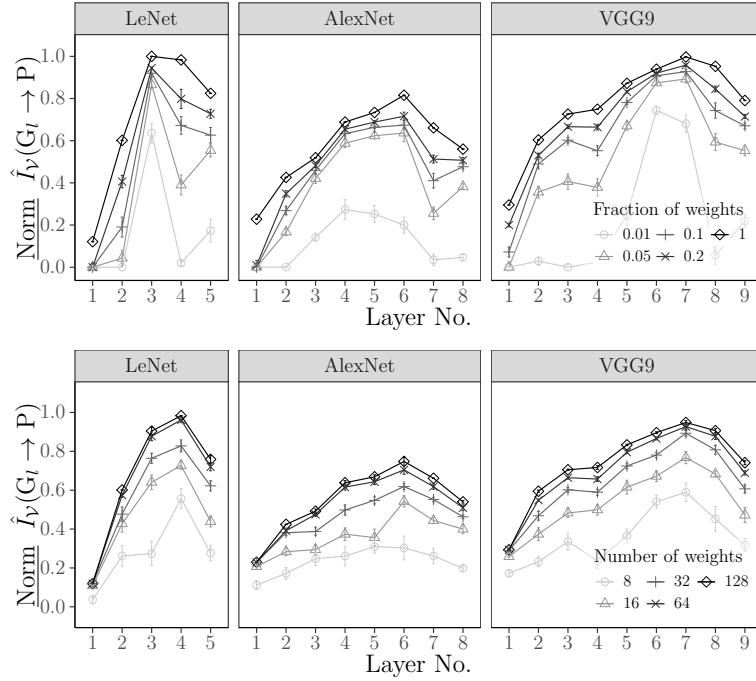


Figure 5.8: Usable information of the ‘race’ attribute in a fixed number of gradients (Top) and a fraction of gradients (Bottom) from each layer of the neural network.

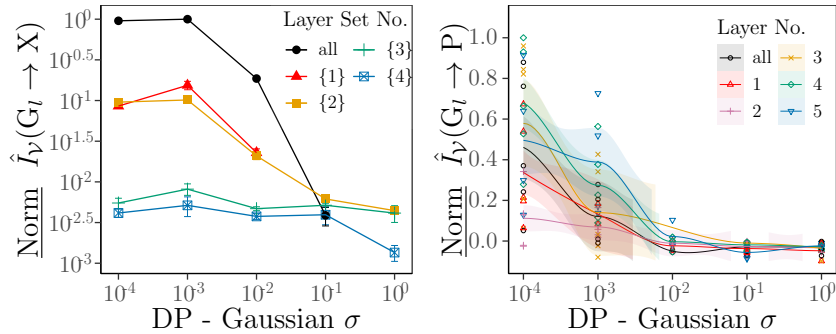


Figure 5.9: Influence of training epoch on risks of original information [Left] and latent information [Right], measured on LeNet trained on CIFAR100 and LFW, respectively.

add Gaussian noises. Specifically, I set the max norm as 1 for all cases and adjust the standard deviation  $\sigma$  of Gaussian noises from  $10^0$  to  $10^{-4}$ . Noises are only added every time before releasing gradients; that is, pre-sample noise addition for DRAs, and pre-batch (32 samples) noise addition for AIAs. I do not report the privacy budget because this is not a fully DP training (i.e., I do not test accuracy and focus on privacy analysis). Note that adding noises to gradients does not change the derivative results of  $\frac{\partial g_l(\mathbf{X})}{\partial \mathbf{X}}$ , i.e., sensitivity. The empirical results are given in Figure 5.9 showing that DP can reduce the privacy risks of both original and latent information because noises can perturb information in gradients. In addition, adding noises to the first layers tends to have higher effectiveness in alleviating original information



risks, as the risks do not change significantly in terms of the last layers. Similarly, noises could provide better protection in the last layers when it comes to latent information. This again highlights the locations of different types of information and the need for potential layer-wise DP protection.

## 5.5 Discussion and Summary

### 5.5.1 Localization of the most sensitive layers

The location of the most sensitive layers in a model can change depending on the type of information and the attack family I consider. More specifically, the analysis showed that the nodes in the first layer or nodes in the middle layers of some deeper neural networks can contain the most sensitive information of the original input [GBDM20, ZLH19a]. Regarding latent information, it is found that the first FC layers (typically placed after the last Conv layer) are the most sensitive layers. One explanation for these observations is that latent information is more specific than the original information,  $X$ , but still more general than output information,  $Y$ , thus it may be best captured in layers between them. Specifically, an attribute is a high-level feature; the FC layer classifies the feature maps from the previous Conv layers, and it can contain distinguishable high-level latent information such as the property. Similarly, the last Conv layer is likely to have more latent information than previous Conv layers, because initial Conv layers learn more general information (e.g., ambient colors or edge in images), while the latter Conv layers focus on high-level latent information (e.g., face identity) [ZF14, MV15].

### 5.5.2 Insights gained from gradient sensitivity

Sensitivity is a local measure that allows accounting for flatness in the objective function minimized in DRA attacks (Equation (5.3)). This flatness is known to occur in neural network loss functions [DVSH18], and therefore the proposed sensitivity metric can be a valuable tool in understanding sensitive information leakage. In addition, a low sensitivity has been linked to

increased robustness of trained models [SGSR17]. Adapting the tools that were consequently introduced to improve robustness can help design better defenses against original information leakage. The gradient subspace distances are a novel metric introduced with the aim of capturing more *coarse-grained* information. As sensitivity in itself is a local measure to capture latent information leakages, subspace distances are a valuable extension. Defined as the distance between gradients with and without target information, it helps inform in which trained models leaks can happen, or in other words whether the weights of the models are trained in such a way that they are able to distinguish between attributes.

### 5.5.3 Localization for the design of protection mechanisms

The characterization allows us to *localize* which layers/nodes in which layers leak the most sensitive information. This highlights opportunities for designing defenses that are flexible and practical at a layer-wise level. For example, fully homomorphic encryption (FHE)-based approaches fully respect the model privacy but it leads to high runtime overhead. The analysis can be used towards solutions utilizing FHE only on the sensitive part of the model during training. In addition, given the resource constraints on edge devices, TEE-based approaches with model partitioning have also become a promising approach. Specifically, during learning, one can deploy and run the most sensitive layers (e.g., the last Conv layer and the first FC layer) inside the TEE using model partitioned execution techniques across trusted and untrusted environments similar to [MSK<sup>+</sup>20]. The measures could also provide insights for layer-wise, federated training of models [MHK<sup>+</sup>21, WYS<sup>+</sup>19], which could further reduce the privacy risk by exposing only specific layers instead of the complete model.

### 5.5.4 Insights for defense mechanisms

There are privacy-preserving techniques proposing to add noise to gradients (i.e., DP), share fewer gradients, or use dimensionality reduction and regularization (e.g., Dropout). However, none of them can protect against latent or original information leakage without significantly

compromising model utility [MSDCS19, ZLH19a]. The ability to localize the most sensitive layers can allow adding noise only where necessary. As one example, clipping or adding noises differently for layers in DP [MAE<sup>+</sup>18, PSY<sup>+</sup>19] and directly hiding layers in TEEs [MSK<sup>+</sup>20, GHZ<sup>+</sup>18a] could be one promising protection without compromising the utility of the other layers and the quantification helps in understanding and designing these layer-wise protection mechanisms.

### 5.5.5 Limitations & Next steps

Overall there are several points to improve the work in this chapter: i) It is expected that DNNs with skip connections (e.g., ResNet [HZRS16]) could give similar results because they usually have a unidirectional gradient flow, but this needs further experiments. Information leakages in other widely used networks such as Recurrent [She20] or Graph [WPC<sup>+</sup>20] neural networks need investigating; ii) More defense mechanisms can be analyzed using the proposed metrics. For example, the influence of differential privacy can provide a further understanding of the linkage between information leakages and gradient changes; iii) Factors other than sensitivity deserve further exploration. Bayesian-based generalization analysis [WI20], distribution shift, or concept drift may provide another way to theoretically characterize information leakages from another perspective.

Based on the results of privacy measurement, it is expected we would need to protect all layers in order to defend against all private-related attacks. This will require the DarkneTZ system to be changed because it does not support fitting the complete training process inside TEEs. More advanced training techniques are needed to achieve such a goal. Therefore, in the next step, I aim to propose one privacy-preserving method to fully guarantee privacy in federated learning atop TEEs.

---

## Chapter 6

# Privacy-preserving Federated Learning System

Running ML inside TEEs can hide model parameters from REE adversaries and consequently preserve privacy, as already used for light data analytics on servers [SCF<sup>+</sup>15, OSF<sup>+</sup>16] and for heavy computations such as DNN training [HSS<sup>+</sup>18, TB18, GHZ<sup>+</sup>18b, MSK<sup>+</sup>20]. However, due to TEEs' limited memory size, the work in the previous chapter runs only part of the model (e.g., sensitive layers) inside the TEE [MSK<sup>+</sup>20, MH19]. Only the last layers are run with a Trusted Application inside TEEs to defend against MIAs, and it leaves the first layers unprotected. DarkneTZ's evaluation showed no more than 10% overhead in CPU, memory, and energy on edge-like devices, demonstrating its suitability for client-side model updates in FL, but some layers will be left outside. Due to the broad attack surface on neural network gradients, it will be necessary to protect all gradients in a proper way so that we can defend against all possible attackers and meanwhile do not introduce high system overhead.

In this chapter, I aim to provide a system that achieves privacy-preserving collaborative/federated learning, motivated by the privacy measurement results from the previous chapter, i.e., all layers can leak some types of private information. This helps to answer Research Question 6. More specifically, these privacy leakages include original data leakage, attribute information leakage, and membership inference leakage. However, protecting all layers at once is infeasible

due to the limited secure memory space provided by TEEs; consequently, the training process requires to be partitioned in some ways.

This chapter aims at client-server federated learning, i.e., one server orchestrates multiple clients to train one global model. Below I first revisit some related works including TEE and privacy-related attacks. I then present the framework, Privacy-preserving Federated Learning, a greedy layer-wise training method to run ML inside collaborated TEEs to defend against all the above-mentioned attacks.

## 6.1 Introduction and Related Work

This section provides the background needed to understand the way TEEs work (Sec. 6.1.1), existing privacy risks in FL (Sec. 6.1.2), privacy-preserving ML techniques using TEEs, as well as core ideas behind layer-wise DNN training for FL (Sec. 6.1.3).

### 6.1.1 Revisiting TEEs

A TEE enables the creation of a secure area on the main processor that provides strong confidentiality and integrity guarantees to any data and codes it stores or processes. TEEs realize strong isolation and attestation of secure compartments by enforcing a dual-world view where even compromised or malicious system (i.e., privileged) software in the normal world – also known as the Rich Operating System Execution Environment (REE) – cannot gain access to the secure world. This allows for a drastic reduction of the TCB since only the code running in the secure world needs to be trusted. Another key aspect of TEEs is that they allow arbitrary code to run inside almost at native speed. In order to keep the TCB as small as possible, current TEEs have limited memory; beyond this, TEEs are required to swap pages between secure and unprotected memory, which incurs a significant overhead and hence must be prevented.

Over the last few years, significant research and industry efforts have been devoted to devel-

oping secure and programmable TEEs for high-end devices (e.g., servers<sup>1</sup>) and mobile devices (e.g., smartphones). This work leverages Intel Software Guard Extensions (Intel SGX) [CD16] at the server-side, while in the client devices we rely on Open Portable Trusted Execution Environment (OP-TEE) [Lin20]. OP-TEE is a widely known open-source TEE framework that is supported by different boards equipped with Arm TrustZone. While some TEEs allow the creation of fixed-sized secure memory regions (e.g., of 128MB in Intel SGX), some others (e.g., ARM TrustZone) do not place any limit on the TEE size. However, creating large TEEs is considered to be bad practice since it has proven to significantly increase the attack surface. Therefore, the TEE size must always be kept as small as possible independently of the type of TEEs and devices being used. This principle has already been adopted by the industry, e.g., in the HiKey 960 board the TEE size is only 16MiB.

### 6.1.2 Privacy risks in FL

Below I give a brief overview of the three main categories of privacy-related attacks in FL: data reconstruction, attribute inference, and membership inference attacks.

**Data Reconstruction Attack (DRA).** The DRA aims at reconstructing original input data based on the observed model or its gradients. It works by inverting model gradients based on generative adversarial attack-similar techniques [AHW<sup>+</sup>17, ZLH19a, GBDM20], and consequently reconstructing the corresponding original data used to produce the gradients. DRAs are effective when attacking DNN’s early layers, and when gradients have been only updated on a small batch of data (i.e., less than 8) [ZLH19a, GBDM20, MSK<sup>+</sup>20]. As the server typically observes updated models of each client in plaintext, it is more likely for this type of leakage to exist at the server. By subtracting updated models with the global model, the server obtains gradients computed w.r.t. clients’ data during the local training.

**Attribute Inference Attack (AIA).** The goal of AIAs is to infer the value of private properties in the input data. This attack is achieved by building a binary classifier trained on

---

<sup>1</sup> Recently, cloud providers also offer TEE-enabled infrastructure-as-a-service solutions to their customers (e.g., Microsoft Azure Confidential).

model gradients updated with auxiliary data and can be conducted on both server and client sides [MSDCS19]. Specifically, attribute information, which also refers to the feature/latent information of the input data, is easier to be carried in stronger aggregation [MBM<sup>+</sup>21a]. Even though clients in FL only observe multiple snapshots of broadcast global models that have been linearly aggregated on participating clients' updates, attribute information can still be well preserved, providing attack points to client-side adversaries.

**Membership Inference Attack (MIA).** The purpose of MIAs is to learn whether specific data instances are present in the training dataset. One can follow a similar attack mechanism as AIAs to build a binary classifier when conducting MIAs [NSH19], although there are other methods, e.g., using shadow models [SSSS17]. The risk of MIAs can exist on both the server and client sides. Moreover, because membership is 'high-level' latent information, adversaries can perform MIAs on the final (well-trained) model and its last layer [NSH19, SSSS17, YGFJ18].

### 6.1.3 Layer-wise DNN training for FL

Instead of training the complete DNN model in an end-to-end fashion, one can train the model layer-by-layer from scratch, i.e., *greedy layer-wise training* [BLPL06, LBLL09]. This method starts by training a shallow model (e.g., one layer) until its convergence. Next, it appends one more layer to the converged model and trains only this new layer [BEO19]. Usually, for each greedily added layer, the model developer builds a new classifier on top of it in order to output predictions and compute training loss. Consequently, these classifiers provide multiple early exits, one per layer, during the forward pass in inference [KHD19]. Furthermore, recently this method was shown to scale for large datasets such as ImageNet and to achieve performance comparable to regular end-to-end ML [BEO19]. Notably, all previous studies on layer-wise training focused on generic ML.

**Contribution.** Our work is the first to build a DNN model in a FL setting with privacy-preserving guarantees using TEEs, by leveraging the greedy layer-wise training, and to train each DNN layer inside each FL client's TEE. Thus, PPFL satisfies the constraint of TEE's

limited memory while protecting the model from the aforementioned privacy attacks. Interestingly, the classifiers built atop each layer may also provide personalization opportunities for the participating FL clients.

## 6.2 Threat Model and Assumptions

**Threat model.** A standard FL context is considered where multiple client devices train a DNN locally and send their (local) model parameters to a remote, centralized server, which aggregates these parameters to create a global model [MMR<sup>+</sup>17, BEG<sup>+</sup>19, KMA<sup>+</sup>19]. The goal of adversaries is to obtain sensitive information embedded in the global model through data reconstruction [ZLH19a, GBDM20] or inference attacks [MSDCS19, NSH19]. Two types of (passive) adversaries are considered: (i) users of client devices who have access to distinct snapshots of the global model and (ii) the server’s owner (e.g., a cloud or edge provider) who has access to the updated model gradients. Adversaries are assumed to be *honest-but-curious*, meaning that they allow FL algorithms to run as intended while trying to infer as much information as possible from the global model or gradients. Adversaries can have full control (i.e., root privileges) of the server or the client device and can perform their attacks against any DNN layer. However, attacks against the TEE, such as side-channel attacks (e.g., Voltpilager [CVM<sup>+</sup>21]), physical attacks (e.g., Platypus [LKO<sup>+</sup>21]) and those that exploit weaknesses in TEEs (e.g., [LJJ<sup>+</sup>17]) and their SDKs (e.g., [Van19]) are out of scope here.

**Assumptions.** It is assumed that the server and enough participating FL client devices have a TEE whose memory size is larger than the largest layer of the DNN to be trained. This is the case in current FL DNNs. However, in the unlikely case that a layer does not fit in available TEEs, the network design needs to be adjusted with smaller, but more layer(s), or a smaller training batch size. It is also assumed that there is a secure way to bootstrap trust between the server TEE and each of the client device TEE (e.g., using a slightly modified version of the SIGMA key exchange protocol [Kra03, ZZQ<sup>+</sup>19], or attested TLS [KSC<sup>+</sup>18]), and that key management mechanisms exist to update and revoke keys when needed [PKE18].



Finally, it is assumed that the centralized server will forward data to/from its TEE. Yet, it is important to note that if the server was malicious and would not do this, this would only affect the availability of the system (i.e., the security and privacy properties of our solution remain intact). This type of Denial-of-Service (DoS) attack is hard to defend against and is not considered within the standard TEE threat model.

## 6.3 PPFL Framework

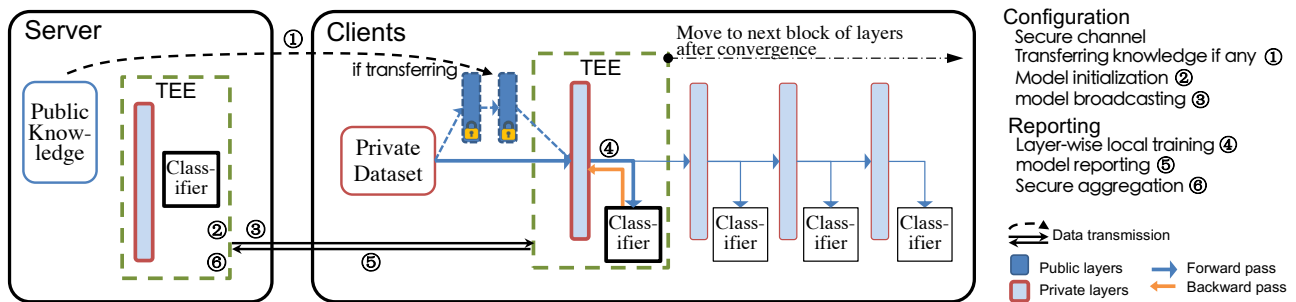


Figure 6.1: Schematic diagram of PPFL. Main phases follow the system design in [BEG<sup>+</sup>19].

This section first presents an overview of the proposed system and its functionalities (Sec. 6.3.1), and then details how the framework employs layer-wise training and aggregation in conjunction to TEEs in FL (Sec. 6.3.2).

### 6.3.1 System overview

This work proposes a Privacy-preserving Federated Learning framework that allows clients to collaboratively train a DNN model while keeping the model’s layers always inside TEEs during training. Figure 6.1 provides an overview of the framework and the various steps of the *greedy layer-wise training and aggregation*. In general, starting from the first layer, each layer is trained until convergence, before moving to the next layer. In this way, PPFL aims to achieve full privacy preservation without significantly increasing system costs. PPFL’s design provides the following functionalities:

**Privacy-by-design Guarantee.** PPFL ensures that layers are always protected from adver-

saries while they are being updated. Privacy risks depend on the aggregation level and frequency with which they happen, when exposing the model or its layers [MSDCS19, GBDM20, MBM<sup>+</sup>21a]. In PPFL, lower-level information (i.e., original data and attributes) is not exposed because updated gradients during training are not accessible from adversaries (they happen inside the TEEs). This protects against DRAs and AIAs. However, when one of such layers is exposed after convergence, there is a risk of MIAs. I follow a more practical approach based on the observation that membership-related information is only sensitive in the *last DNN layer*, making it vulnerable to MIAs as indicated in previous research [NSH19, SDS<sup>+</sup>19, MSK<sup>+</sup>20, MBM<sup>+</sup>21a]. To avoid this risk in the final model, PPFL can keep the last layer inside the clients' TEEs after training.

**Device Selection.** After the server and a set of TEE-enabled clients agree on the training of a DNN model via FL, clients inform the server about their TEE's memory constraints. The server then (re)constructs a DNN model suitable for this set of clients and selects the clients that can accommodate the model layers within their TEE. In each round, the server can select new clients and the device selection algorithm can follow existing FL approaches [NY19, HLW<sup>+</sup>20].

**Secure Communication Channels.** The server establishes two secure communication channels with each of its clients: (i) one from its REE to the client's REE (e.g., using TLS) to exchange data with clients and (ii) a logical one from its TEE to the client's TEE for securely exchanging private information (e.g., model layer training information). In the latter case, the transmitted data is encrypted using cryptographic keys known only to the server and client TEEs and is sent over the REE-REE channel. It is important to note that the secure REE-REE channel is only an additional security layer. All privacy guarantees offered by PPFL are based on the hardware-backed cryptographic keys stored inside TEEs.

**Model Initialization and Configuration.** The server configures the model architecture, decides the layers to be protected by TEEs, and then initializes model parameters inside the TEE (step ②, Fig. 6.1). The latter ensures clients' local training starts with the same weight distribution [MMR<sup>+</sup>17, WYS<sup>+</sup>20]. In addition, the server configures other training hyperparameters such as learning rate, batch size, and epochs, before transmitting such settings to

the clients (step ③, Fig. 6.1).

In cases of typical ML tasks such as image recognition where public knowledge is available such as pre-trained DNN models or public datasets with features similar to the client’s private data, the server can transfer this knowledge (especially in cross-device FL [KMA<sup>+</sup>19]) in order to bootstrap and speed up the training process. In both cases, this knowledge is contained in the first layers. Thus, the clients leave the first layers frozen and only train the last several layers of the global model. This training process is similar to the concept of transfer learning [PY09, Bro20, TS10], where, in this case, public knowledge is transferred in a federated manner.

In PPFL, the server can learn from *public models*. Thus, during initialization, the server first chooses a model pre-trained on public data that have a similar distribution to private data. The server keeps the first layers, removes the last layer(s), and assembles new layer(s) atop the reserved first ones. These first layers are transferred to clients and are always kept frozen (step ①, Fig. 6.1). New layers, attached to the reserved layers, are trained inside each client’s TEE, and then aggregated inside the server’s TEE (steps ②~⑥, Fig. 6.1). In learning from *public datasets*, the server first performs an initial training to build the model based on public datasets.

**Local Training.** After model transmission and configuration using secure channels, each client starts local training on their data on each layer via a model partitioned execution technique (step ④, Fig. 6.1). I detail this step in Sec. 6.3.2.

**Reporting and Aggregation.** Once local training of a layer is completed inside TEEs, all participating clients report the layer parameters to the server through secure channels (step ⑤, Fig. 6.1). Finally, the server securely aggregates the received parameters within its TEE and applies FedAvg [MMR<sup>+</sup>17], resulting in a new global model layer (step ⑥, Fig. 6.1).

### 6.3.2 Layer-wise training and aggregation

In order to address the problem of limited memory inside a TEE when training a DNN model, I modify the greedy layer-wise learning technique proposed in [BLPL06] for general DNN training [BEO19], to work in the FL setting. The procedure of layer-wise training and aggregation is detailed in the following Algorithms 1 and 2. Note that Algorithm 2 “ClientUpdate” is one sub-function to be called inside Algorithm 1.

**Algorithm 1.** This algorithm details the actions taken by PPFL on the server side. When not specified, operations are carried out outside the TEE (i.e., in the REE). First, the server initializes the global DNN model with random weights or public knowledge (steps ①-②, Fig. 6.1). Thus, each layer  $l$  to be trained is initialized ( $\theta_l$ ) and prepared for broadcast. The server checks all available devices and constructs a set of participating clients whose TEE is larger than the required memory usage of  $l$ . Then, it broadcasts the model’s layer to these participating clients (step ③, Fig. 6.1), via `ClientUpdate()` (see Algorithm 2). Upon receiving updates from all participating clients, the server decrypts the layer weights, performs secure layer aggregation and averaging inside its TEE (step ⑥), and broadcasts the new version of  $l$  to the clients for the next FL round. Steps ②~⑥ are repeated until the training of  $l$  converges, or a fixed number of rounds are completed. Then, this layer is considered fully trained ( $\theta_l^0$ ), it is passed to the REE, and is broadcasted to all clients to be used for training the next layer. Interestingly, PPFL also allows grouping *multiple layers into blocks* and training each block inside client TEEs in a similar fashion as the individual layers. This option allows for better utilization of the memory space available inside each TEE and reduces communication rounds for the convergence of more than one layer at the same time.

**Algorithm 2.** This algorithm details the actions taken by PPFL on the client side. Clients load the received model parameters from the server and decrypt and load the target training layer  $l$  inside their TEEs. More specifically, in the front, this new layer  $l$  connects to the previous pre-trained layer(s) that are frozen during training. In the back, the clients attach on  $l$  their *own derived classifier*, which consists of fully connected layers and a softmax layer as the model

**Algorithm 1: PPFL-Server with TEE**


---

```

1 Input:
  • Number of all clients:  $N$ 
  • TEE memory size of Client  $n$ :  $S^{(n)}$ 
  • Memory usage of layers  $\{1, \dots, L\}$  in training (forward and backward pass in total):
     $\{S_1, \dots, S_L\}$ 
  • Communication rounds:  $R$ 

2 Output: Aggregated final parameters:  $\{\theta_1^0, \dots, \theta_L^0\}$ 
  % Layer-wise client updates ;
3 for  $l \in \{1, \dots, L\}$  do
  | % Select clients with enough TEE memory
  | Initialize participating client list  $\mathbf{J} = \{\}$ 
  | for  $n \in \{1, \dots, N\}$  do
  | | if  $S^{(n)} > S_l$  then
  | | |  $\mathbf{J} \leftarrow \mathbf{J} \cup \{n\}$ 
  | Initialize  $\theta_l$  (parameters of layers  $l$ ) in TEE
  | for  $r \in \{1, \dots, R\}$  do
  | | for  $j \in \mathbf{J}$  do
  | | | % clients' local updating to get client  $j$ 's weight copy: see Algorithm 2
  | | |  $\theta_l^{(j)} = \text{ClientUpdate}(l, \theta_l)$ 
  | | | % FedAvg with Secure Aggregation
  | | |  $\theta_l = \frac{1}{\text{size}(\mathbf{J})} \sum_{j \in \mathbf{J}} \theta_l^{(j)}$  in TEE
  | Save  $\theta_l$  from TEE as  $\theta_l^0$  in REE
14 return  $\{\theta_1^0, \dots, \theta_L^0\}$ 

```

---

exit. Then, for each epoch, the training process iteratively goes through batches of data and performs both *forward and backward passes* [LBH15] to update both the layer under training and the classifier inside the TEE (step ④, Fig. 6.1). During this process, a *model partitioned execution* technique is utilized, where intermediate representations of the previously trained layers are passed from the REE to the TEE via shared memory in the forward pass. After local training is completed (i.e., all batches and epochs are done), each client sends via the secure channel the (encrypted) layer's weights from its TEE to the server's TEE (step ⑤).

**Model Partitioned Execution.** The above learning process is based on a technique that conducts model training (including both forward and backward passes) across REEs and TEEs,

**Algorithm 2: ClientUpdate**( $l, \theta_l$ ) with TEEs**1 Initialization:**

- Local dataset  $\mathcal{X}$ : data  $\{\mathbf{x}\}$  and labels  $\{\mathbf{y}\}$
- Trained final parameters of all previous layers, i.e.,  $\theta_1^0, \theta_2^0, \dots, \theta_{l-1}^0$
- Number of local training epochs:  $E$
- Activation function:  $\sigma()$  and loss function:  $\ell$
- Classifier:  $C()$

**2 Input:**

- Target layer:  $l$
- Broadcast parameters of layer  $l$ :  $\theta_l$

**3 Output:** Updated parameters of layer  $l$ :  $\theta_l$ 

```

% Weights and biases of layers 1, ..., (l - 1) and l
4 for  $i \in \{1, \dots, l - 1\}$  do
5   |  $\{\mathbf{W}_i, \mathbf{b}_i\} \leftarrow \theta_i^0$ 
6  $\{\mathbf{W}_l, \mathbf{b}_l\} \leftarrow \theta_l$  in TEE
% Training process
7 for  $e \in \{1, \dots, E\}$  do
8   | for  $\{\mathbf{x}, \mathbf{y}\} \in \mathcal{X}$  do
9     | % Forward pass
10    | Intermediate representation  $\mathbf{T}_0 = \mathbf{x}$ 
11    | for  $i \in \{1, \dots, l - 1\}$  do
12    |   |  $\mathbf{T}_i = \sigma(\mathbf{W}_i \mathbf{T}_{i-1} + \mathbf{b}_i)$ 
13    |   |  $\mathbf{T}_l = \sigma(\mathbf{W}_l \mathbf{T}_{l-1} + \mathbf{b}_l)$ 
14    |   |  $\ell \leftarrow \ell(C(\mathbf{T}_l), \mathbf{y})$ 
15    |   | % Backward pass
16    |   |  $\frac{\partial \ell}{\partial C}$  to update parameters of  $C$ 
17    |   | % Updating layer  $l$ 
18    |   |  $\mathbf{W}_l \leftarrow \mathbf{W}_l + \frac{\partial \ell}{\partial \mathbf{W}_l}; \mathbf{b}_l \leftarrow \mathbf{b}_l + \frac{\partial \ell}{\partial \mathbf{b}_l}$ 
19    |   | } in TEE
20   $\theta_l = \{\mathbf{W}_l, \mathbf{b}_l\}$  in TEE
21 return  $\theta_l$ 

```

namely model partitioned execution. The transmission of the forward activations (i.e., intermediate representation) and updated parameters happens between the REE and the TEE via *shared memory*. On a high level, when a set of layers is in the TEE, activations are transferred from the REE to the TEE (see Algorithm 2). Assuming global layer  $l$  is under training, the

layer with its classifier  $C(\cdot)$  is executed in the TEE, and the previous layers (i.e., 1 to  $l - 1$ ) are in the REE.

Before training, layer  $l$ 's parameters are loaded and decrypted securely within the TEE. During the *forward pass*, local data  $\mathbf{x}$  are inputted, and the REE processes the previous layers from 1 to  $l - 1$  and invokes a command to transfer the layer  $l - 1$ 's activations (i.e.,  $T_{l-1}$ ) to the secure memory through a buffer in shared memory. The TEE switches to the corresponding invoked command in order to receive layer  $l - 1$ 's activations, and then it processes the forward pass of layer  $l$  and classifier  $C(\cdot)$  in the TEE.

During the *backward pass*, the TEE computes the  $C(\cdot)$ 's gradients based on received labels  $\mathbf{y}$  and outputs of  $C(\cdot)$  (produced in the forward pass) and uses them to compute the gradients of the layer  $l$  in the TEE. The training of this batch of data (i.e.,  $\mathbf{x}$ ) finishes here, and there is no need to transfer  $l$ 's errors from the TEE to the REE via shared memory, as previous layers are frozen outside the TEE. After that, the parameters of layer  $l$  are encrypted and passed to the REE, ready to be uploaded to the server, corresponding to the *FedSGD* [CPM<sup>+</sup>16]. Further, *FedAvg* [MMR<sup>+</sup>17] which requires multiple batches to be processed before updating, repeats the same number of the forward and backward passes across the REE and the TEE for each batch of data.

**Algorithmic Complexity Analysis.** Next, this work analyzes the algorithmic complexity of PPFL and compares it to standard end-to-end FL. For the global model's layers  $l \in \{1, \dots, L\}$ , I denote the forward and backward pass cost on layer  $l$  as  $F_l$  and  $B_l$ , respectively. The corresponding cost on the classifier is denoted as  $F_c$  and  $B_c$ . Then, in end-to-end FL, the total training cost for one client is:

$$\left( \sum_{l=1}^L (F_l + B_l) + F_c + B_c \right) \cdot S \cdot E \quad (6.1)$$

where  $S$  is the number of steps in one epoch (i.e., number of samples inside local datasets divided by the batch size). As in PPFL all layers before the training layer  $l$  are kept frozen, the cost of training layer  $l$  is  $(\sum_{k=1}^l F_k + F_c + B_l + B_c) \cdot S \cdot E$ . Then, by summation, we can get

the total cost of all layers as:

$$\left( \sum_{l=1}^L \sum_{k=1}^l \mathbf{F}_k + \sum_{l=1}^L \mathbf{B}_l + L \cdot (\mathbf{F}_c + \mathbf{B}_c) \right) \cdot S \cdot E \quad (6.2)$$

By comparing Equations 6.1 and 6.2, we can see the overhead of PPFL comes from: (i) repeated forward pass in previous layers ( $l \in \{1, \dots, l-1\}$ ) when training layer  $l$ , and (ii) repeated forward and backward pass for the classifier atop layer  $l$ . Due to that backward pass usually triples the computation cost of the forward pass only, so most overhead will come from the classifier training.

## 6.4 Implementation & Evaluation Setup

This section first describes the implementation of the PPFL system (Sec. 6.4.1), and then detail how we assess its performance on various DNN models and datasets (Sec. 6.4.2) using different metrics (Sec. 6.4.3). This work follows common setups of past FL systems [MMR<sup>+</sup>17, WYS<sup>+</sup>20] and on-device TEE works [MSK<sup>+</sup>20, AS19].

### 6.4.1 PPFL prototype

I implement the *client-side* of PPFL by building on top of DarkneTZ [MSK<sup>+</sup>20], in order to support on-device FL with Arm TrustZone. In total, I changed 4075 lines of code of DarkneTZ in C to add functionalities including i) training any chosen layers (e.g., middle layers) inside TEE; ii) saving and loading any chosen layers; iii) communicating with a server. I run the client-side on a Hikey 960 Board, which has four ARM Cortex-A73 and four ARM Cortex-A53 cores configured at 2362MHz and 533MHz, respectively, as well as a 4GB LPDDR4 RAM with 16MiB TEE secure memory (i.e., TrustZone). Since the CPU power/frequency setting can impact the TrustZone’s performance [AS19], I execute the on-device FL training with full CPU frequency. In order to emulate multiple device clients and their participation in FL rounds, I use the HiKey board in a repeated, iterative fashion, one time per client device. I implement



the *server-side* of PPFL on generic Darknet ML framework [Red16] by adding 751 lines of C code based on Microsoft OpenEnclave [Mic20] with Intel SGX. For this, an Intel Next Unit of Computing (ver.NUC8BEK, i3-8109U CPU, 8GB DDR4-2400MHz) was used with SGX-enabled capabilities.

Besides, I developed a set of bash shell scripts to control the *FL process* and create the *communication channels*. For the communication channels between server and client to be secure, I employ standard cryptographic-based network protocols such as SSH and SCP. All data leaving the TEE are encrypted using the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode with random Initialization Values (IV) and 128-bit cryptographic keys. Without loss of generality, I opted for manually hardcoding the cryptographic keys inside the TEEs ourselves. Despite key management in TEE-to-TEE channels being an interesting research problem, I argue that establishing, updating, and revoking keys do not happen frequently and hence the overhead these tasks introduce is negligible compared to one from the DNN training.

The implementation of PPFL server and client is available for replication and extension: <https://github.com/mofanv/PPFL>.

## 6.4.2 Models and datasets

This work focuses on Convolutional Neural Networks (CNNs) since the privacy risks considered here (Sec. 6.2 and 6.3.1) have been extensively studied on such DNNs [MSDCS19, NSH19]. Also, layer-based learning methods mostly aim at CNN-like DNNs [BEO19]. Specifically, in this PPFL evaluation, I employ DNNs commonly used in the relevant literature (Table 6.1).

Experimental analysis used MNIST and CIFAR10, two datasets commonly employed by FL researchers. Note that in practice, FL training needs labeled data locally stored at the client’s side. Indeed, the number of labeled examples expected to be present in a real setting could be fewer than what these datasets may allocate per FL client. Nonetheless, using them allows comparison of results with state-of-art end-to-end FL methods [LSZ<sup>+</sup>18, WYS<sup>+</sup>20, GBDM20].

Table 6.1: DNNs used in the evaluation of PPFL.

DNN	Architecture
LeNet [LBB <sup>+</sup> 98, MMR <sup>+</sup> 17]	C20-MP-C50-MP-FC500-FC10
AlexNet [KSH17, BEO19]	C128×3-AP16-FC10
VGG9 [SZ14, WYS <sup>+</sup> 20]	C32-C64-MP-C128×2-MP-D0.05-C256×2 -MP-D0.1-FC512×2-FC10
VGG16 [SZ14]	C64×2-MP-C128×2-MP-C256×3-C512×3 -MP-FC4096×2-FC1000-FC10
MobileNetv2 [SHZ <sup>+</sup> 18]	68 layers, unmodified refer to [SHZ <sup>+</sup> 18] for details

Architecture notation: Convolution layer (C) with a given number of filters; filter size is  $5 \times 5$  in LeNet and  $3 \times 3$  in AlexNet, VGG9, and VGG16. Fully Connected (FC) with a given number of neurons. All C and FC layers are followed by ReLU activation functions. MaxPooling (MP). AveragePooling (AP) with a given stride size. Dropout layer (D) with a given dropping rate.

Specifically, LeNet is tested on MNIST [LBB<sup>+</sup>98] and all other models are tested on CIFAR10 [KH09]. The former is a handwritten digit image ( $28 \times 28$ ) dataset consisting of  $60k$  training samples and  $10k$  test samples with 10 classes. The latter is an object image ( $32 \times 32 \times 3$ ) dataset consisting of  $50k$  training samples and  $10k$  test samples with 10 classes. This work follows the setup in [MMR<sup>+</sup>17] to partition training datasets into 100 parts, one per client, in two versions: i) Independent and Identically Distributed (IID) where a client has samples of all classes; ii) Non-Independent and Identically Distributed (Non-IID) where a client has samples only from two random classes.

### 6.4.3 Performance metrics

The evaluation of PPFL prototype presented in the next section focuses on assessing the framework from the point of view of (i) privacy of data, (ii) ML model performance, and (iii) client-side system cost. Although ML computations (i.e., model training) have the same precision and accuracy no matter in REEs or TEEs, PPFL changes the FL model training process into a layer-based training. This affects ML accuracy and the number of communication rounds needed for the model to converge (among others). Thus, I devise several metrics and perform extensive measurements to assess overall PPFL performance. I conduct system cost measurements only on client devices since their computational resources are more limited compared to the server. All experiments are done with 10% of the total number of clients (i.e., 10 out of

100) participating in each communication round. I run FL experiments on PPFL prototype (Sec. 6.4.1) to measure the system cost. To measure privacy risks and ML model performance, I perform simulations on a cluster with multiple NVIDIA RTX6000 GPUs (24GB) nodes running PyTorch v1.4.0 under Python v3.6.0.

**Model Performance.** This research measures three metrics to assess the performance of the model and PPFL-related process:

1. *Test Accuracy*: ML accuracy of test data on a given FL model, for a fixed number of communication rounds.
2. *Communication Rounds*: Iterations of communication between server and clients needed to achieve a particular test accuracy.
3. *Amount of communication*: Total amount of data exchanged to reach a test accuracy. Transmitted data sizes may be different among communication rounds when considering different layers' sizes in layer-wise training.

**Privacy Assessment.** The research measures the privacy risk of PPFL by applying three FL-applicable, privacy-related attacks:

1. Data Reconstruction Attack (DRA) [ZLH19a]
2. Attribute Inference Attack (AIA) [MSDCS19]
3. Membership Inference Attack (MIA) [NSH19]

I follow the proposing papers and their settings to conduct each attack on the model trained in FL process.

**Client-side System Cost.** I monitor the efficiency of client on-device training, and measure the following device costs for PPFL-related process information:

1. *CPU Execution Time (s)*: Time the CPU was used for processing the on-device model training, including time spent in REE and the TEE’s user and kernel time, which is reported by using function `getrusage(RUSAGE_SELF)`.
2. *Memory Usage (MB)*: I add REE memory (the maximum resident set size in RAM, accessible by `getrusage()`) and allocated TEE memory (accessible by `mdbg_check(1)`) to get the total memory usage.
3. *Energy Consumption (J)*: Measured by all energy used to perform one on-device training step when the model runs with/without TEEs. For this, I use the *Monsoon High Voltage Power Monitor* [Mon20]. I configure the power to Hikey board as 12V voltage while recording the current in a 50Hz sampling rate. Training with a high-performance power setting can lead to high temperature and consequently under-clocking. Thus, I run each trial with 2000 steps continuously, starting with a 120s cooling time.





## 6.5 Evaluation Results

This section presents the experimental evaluation of PPFL aiming to answer a set of key questions.

### 6.5.1 Effectiveness of PPFL thwarting known privacy-related attacks

To measure the exposure of the model to known privacy risks, this work conducts data reconstruction, attribute inference, and membership inference attacks (i.e., DRAs, AIAs, and MIAs) on the PPFL model. While training AlexNet and VGG9 models on CIFAR10 in an IID setting. I compare the exposure of PPFL to these attacks against a standard, end-to-end FL-trained model. Table 6.2 shows the average performance of each attack in the same way it is measured in literature [ZLH19a, MSDCS19, NSH19]: Mean-Square-Error (MSE) for the DRA, Area-Under-Curve (AUC) for the AIA, and Precision for the MIA.

Table 6.2: Results of three privacy-related attacks (DRA, AIA, and MIA) on PPFL vs. end-to-end (E2E) FL. Average score reported with 95% confidence interval in parenthesis.

Learning Method	Model	Privacy-related Attack		
		DRA, in MSE <sup>α</sup>	AIA, in AUC <sup>δ</sup>	MIA, in Prec. <sup>ε</sup>
E2E	AlexNet 	0.017 (0.01)	0.930 (0.03)	0.874 (0.01)
	VGG9 	0.008 (0.01)	0.862 (0.05)	0.765 (0.04)
PPFL	AlexNet 	~1.3	~0.5	0.506 (0.01)
	VGG9 			0.507 (0.01)

<sup>α</sup>MSE (mean-square error) measures the difference between constructed images and target images (range is  $[0, \infty)$ , and the lower MSE is, the more privacy loss); <sup>δ</sup>AUC refers to the area under the receiver operating curve; <sup>ε</sup>Prec. refers to Precision. The range of both AUC and Prec. is  $[0.5, 1]$  (assuming 0.5 is for random guesses), and the higher AUC or Prec. is, the more privacy loss).

From the results, it becomes clear that, while these attacks can successfully disclose private information in regular end-to-end FL, they fail in PPFL. As DRAs and AIAs rely on intermediate training models (i.e., gradients) that remain protected, PPFL can fully defend against them. The DRA can only reconstruct a fully noised image for any target image (i.e., an MSE of  $\sim 1.3$  for the specific dataset), while the AIA always reports a random guess on private properties (i.e., an AUC of  $\sim 0.5$ ). Regarding the MIA on final trained models, as PPFL keeps the last layer and its outputs always protected inside the client’s TEE, it forces the adversary to access only previous layers, which significantly drops the MIA’s advantage (i.e., Precision  $\approx 0.5$ ). Thus, PPFL fully addresses privacy issues raised by such attacks.

## 6.5.2 Communication cost

**Predefined ML Performance.** Next, the research measures PPFL’s communication cost to complete the FL process, when a specific ML performance is desired. For this, I first execute the standard end-to-end FL without TEEs for 150 rounds and record the achieved ML performance. Subsequently, I set the same test accuracy as a requirement, and measure the number of communication rounds and amount of communication required by PPFL to achieve this ML performance.

In this experiment, I set the number of local epochs at clients as 10. I use SGD as the optimization algorithm and set the learning rate as 0.01, with a decay of 0.99 after each epoch.

Momentum is set to 0.5 and the batch size to 16. When training each layer locally, I build one classifier on top of it. The classifier’s architecture follows the last convolutional (Conv) layer and fully-connected (FC) layers of the target model (e.g., AlexNet or VGG9). Thus, the training of each global model’s layer progresses until all Conv layers are finished. I choose AlexNet and VGG9 on CIFAR10, because MNIST is too simple for testing. Then, the classifier atop all Conv layers is finally trained to provide outputs for the global model. Note that I also aggregate the client classifiers while training one global layer to provide the test accuracy after each communication round. I perform these experiments on IID and Non-IID data.

Overall, the results in Table 6.3 show that, while trying to reach the ML performance achieved by the standard end-to-end FL system, PPFL adds a small communication overhead, if any, to the FL process. In fact, in some cases, it can even reduce the communication cost, while preserving privacy when using TEEs. As expected, using Non-IID data leads to lower ML performance across the system, which also implies less communication cost for PPFL as well.

The reason why in many cases PPFL has reduced communication costs, while still achieving comparable ML performance, is that training these models on datasets such as CIFAR10 may not require training the complete model. Instead, during the early stage of PPFL’s layer-wise training (e.g., first global layer+classifier), it can already reach good ML performance, and in some cases even better than training the entire model. I explore this aspect further in the next subsection. Consequently, and due to the needed rounds being fewer, the amount of communication is also reduced.

The increased cost when training VGG9 is due to the large number of neurons in the classifier’s FC layer connected to the first Conv layer. Thus, even if the number of total layers considered (one global layer + classifier) is smaller compared to the latter stages (multiple global layers + classifier), the model size (i.e., number of parameters) can be larger.

Indeed, we usually are aware that by training any of these models on CIFAR10 [MMR<sup>+</sup>17] for more communication rounds, either the PPFL or the regular end-to-end FL can reach higher test accuracy such as 85% with standard *FedAvg*. However, the training rounds used here are sufficient for our needs, as the goal is to evaluate the performance of PPFL (i.e., what is the

Table 6.3: Communication overhead (rounds and amount) of PPFL to reach the same accuracy as end-to-end FL system.

Model	Data	Baseline Acc. <sup>α</sup>	Comm. Rounds	Comm. Amount
LeNet	IID	98.93%	56 (0.37×) <sup>δ</sup>	0.38 ×
	Non-IID	97.06% <sup>ε</sup>	-	-
AlexNet	IID	68.50%	97 (0.65×)	0.63 ×
	Non-IID	49.49%	79 (0.53×)	0.53 ×
VGG9	IID	63.09%	171 (1.14×)	2.87 ×
	Non-IID	46.70%	36 (0.24×)	0.60 ×

<sup>α</sup>Acc.: Test accuracy of 150 communication rounds in end-to-end FL;

<sup>δ</sup>1× refers to no overhead; <sup>ε</sup>PPFL reaches a maximum of 95.99%.

Table 6.4: Time duration of FL phases in *one communication round*, when training LeNet, AlexNet and VGG9 models with PPFL and end-to-end (E2E) FL.

Model	Method	Duration of FL phases (s)				Total
		B.cast <sup>α</sup>	Training	Upload	Aggr. <sup>δ</sup>	
LeNet	E2E	4.520	2691.0	6.645	0.064	2702.3
	PPFL	18.96	6466.2	7.535	1.887	6496.5
	- <i>layer 1</i>	4.117	1063.3	1.488	0.426	1069.8
	- <i>layer 2</i>	4.670	2130.6	1.627	0.692	2138.3
	- <i>layer 3</i>	5.332	2315.2	1.745	0.676	2323.6
	- <i>clf.</i> <sup>ε</sup>	4.845	957.16	2.675	0.093	964.87
AlexNet	E2E	14.58	3772.0	6.122	0.061	3792.8
	PPFL	57.24	14236	16.89	3.290	14316
	- <i>layer 1</i>	16.20	2301.8	4.690	0.129	2322.9
	- <i>layer 2</i>	12.56	4041.1	4.777	0.174	4058.8
	- <i>layer 3</i>	10.31	4609.4	5.388	0.243	4625.6
	- <i>clf.</i>	18.17	3283.8	2.033	2.744	3309.5
VGG9	E2E	14.10	2867.1	8.883	0.067	2890.2
	PPFL	353.5	21389	173.8	4.066	21924
	- <i>layer 1</i>	127.5	4245.7	95.58	0.375	4469.5
	- <i>layer 2</i>	77.22	2900.6	24.82	0.207	3003.1
	- <i>layer 3</i>	79.18	3703.1	24.84	0.223	3807.6
	- <i>layer 4</i>	27.05	2987.9	12.15	0.235	3027.6
	- <i>layer 5</i>	21.47	2404.4	9.137	0.347	2435.7
	- <i>layer 6</i>	10.95	2671.0	4.768	0.571	2687.9
	- <i>clf.</i>	10.11	2476.4	2.478	2.108	2493.2

<sup>α</sup>B.cast: Broadcast; <sup>δ</sup>Aggr.: Aggregation; <sup>ε</sup>clf.: Classifier.

cost for reaching the same accuracy), and not to achieve the best possible accuracy on this classification task.

**Communication Duration of FL Phases.** The next experiment investigates the wall-clock

time needed for running PPFL’s phases in one communication round: broadcast of the layer from server to clients, training of the layer at the client device, upload the layer to the server, aggregate all updates from clients and apply *FedAvg*. Depending on each layer’s size and TEE memory size, batch size can start from 1 and go as high as the TEE allows. However, since the models are uneven in layer sizes (with VGG9 being the largest), I set the batch size to 1 to allow comparison, and also capture an upper bound on the possible duration of each phase in each model training. Indeed, it is confirmed that increasing batch size for small models that allow it (e.g., AlexNet with batch size=16), incrementally reduces the duration of phases.

Table 6.4 shows the breakdown of time taken for each phase, for three models and two datasets (LeNet on MNIST; AlexNet and VGG9 on CIFAR10) and IID data. As expected, layer-wise FL increases the total time compared to end-to-end FL because each layer is trained separately, but the previously trained and finalized layers still need to be processed in the forward pass. In fact, these results are in line with the complexity analysis shown earlier in Sec. 6.3.2, i.e., to finish the training of all layers, layer-wise training introduces a  $3\times$  or higher delay, similar to the number of layers. On the one hand, we could argue that applications can tolerate this additional delay if they are to be protected from privacy-related attacks, despite the execution time increase being non-negligible and up to a few hours of training. Indeed, models can be (re)trained on longer timescales (e.g., weekly, monthly), and rounds can have a duration of 10s of minutes, while being executed in an asynchronous manner. On the other hand, training one layer in PPFL costs similar time to the end-to-end FL training of the complete model. This highlights that the minimum client contribution time is the same as end-to-end FL: clients can choose to participate in portions of an FL round, and in just a few FL rounds. For example, a client may contribute to the model training for only a few layers in any given FL round.

Among all FL phases, local training costs the most, while the time spent in server aggregation and averaging is trivial, regardless if it is non-secure (i.e., end-to-end FL) or secure (PPFL). Regarding VGG9, layer-wise training of early layers significantly increases the communication time in broadcast and upload, because the Conv layers are with a small number of filters and consequently the following classifier’s FC layer has a large size. This finding hints that selecting suitable DNNs to be trained in PPFL (e.g., AlexNet vs. VGG9) is crucial for practical



performance. Moreover, and according to the earlier FL performance results (also see Table 6.2), it may not be necessary to train all layers to reach the desired ML utility.

### 6.5.3 PPFL performance comparable to state-of-art FL

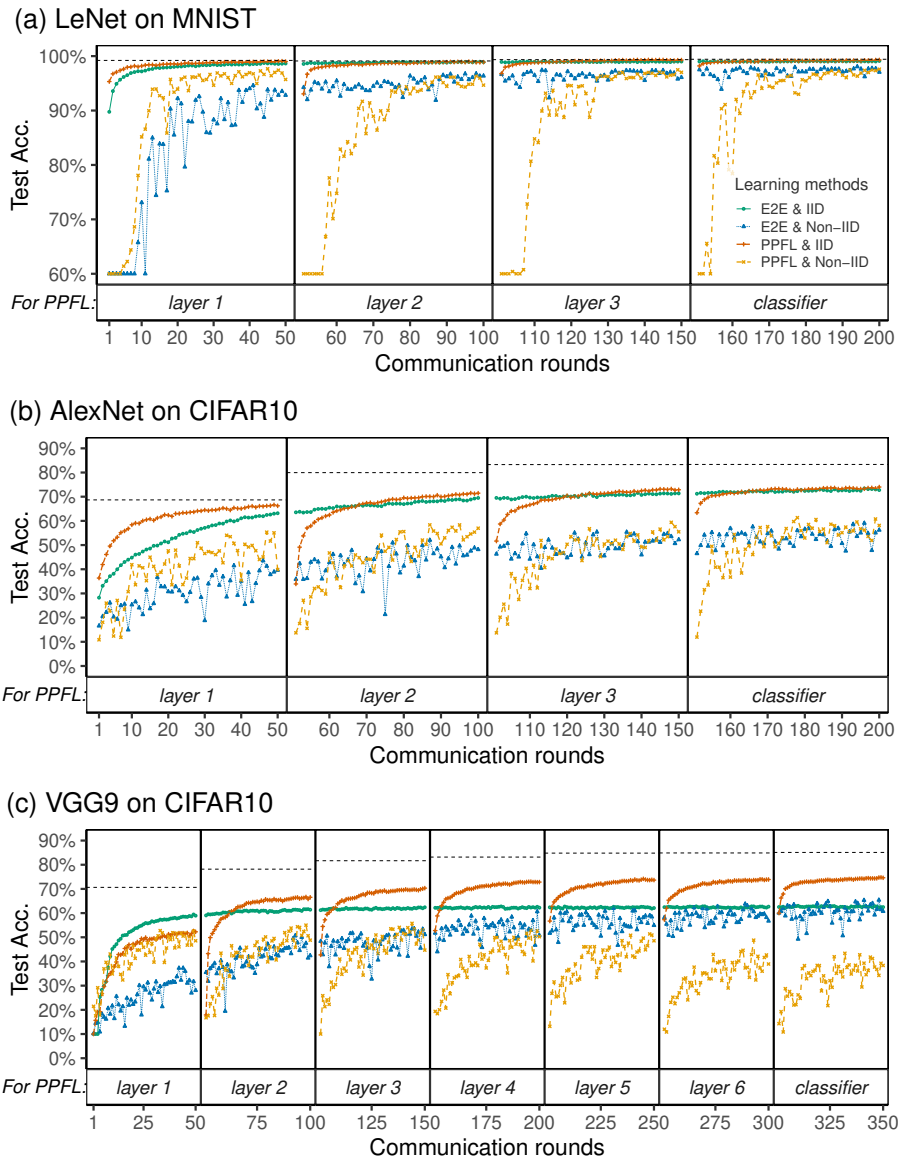


Figure 6.2: Test accuracy of training LeNet, AlexNet, and VGG9 models on IID and Non-IID datasets when using PPFL.

Note: Horizontal dashed lines refer to the accuracy that the centralized training reaches after every 50 epochs. End-to-end (E2E) FL trains the complete model rather than each layer, and the ‘Layer No.’ at x-axis are *only* applicable to PPFL.

In these experiments, I reduce the number of communication rounds that each layer in PPFL is

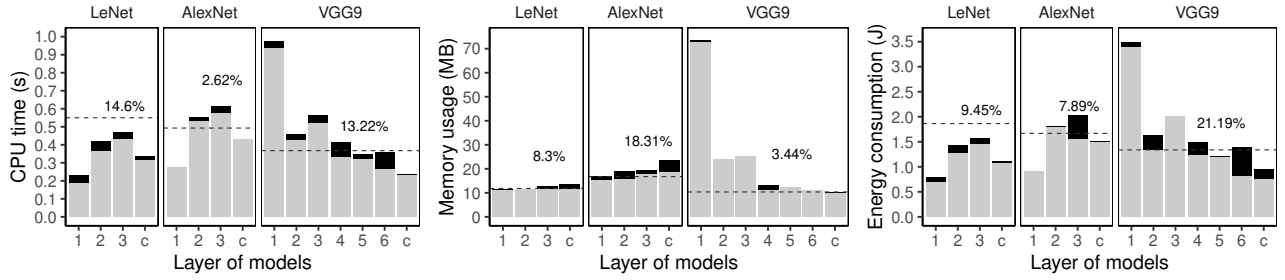


Figure 6.3: System performance of the client devices when training LeNet, AlexNet, and VGG9 using PPFL, measured on *one step of training* (i.e., one batch of data).

Note: The light grey bar (■) refers to learning without TEEs, and the black bar (■) refers to overhead when the layer under training is inside the TEE. Percentage (%) of the overhead (averaged on one model) is shown above these bars. Horizontal dashed lines signify the cost of end-to-end FL. In x-axis, ‘c’ refers to ‘classifier’.

trained to 50, finish the training process per layer, and compare its performance with centralized layer-wise training, as well as regular end-to-end FL. The latter trains the full model for all rounds up to that point. For example, if PPFL trains the first layer for 50 rounds, and then the second layer for 50 rounds, the end-to-end FL will train all the model (end-to-end) for 100 rounds.

As shown in Figure 6.2, training LeNet on the “easy” task of MNIST data (IID or not) leads quickly to high ML performance, regardless of the FL system used. Training AlexNet on IID and Non-IID CIFAR10 data can lead to test accuracy of 74% and 60.78%, respectively, while centralized training reaches 83.34%. Training VGG9, which is a more complex model on IID and Non-IID CIFAR10 data leads to lower performances of 74.60% and 38.35%, respectively, while centralized training reaches 85.09%. Note that the drop of performance in PPFL when every new layer is considered into training. This is to be expected, since PPFL starts from scratch with the new layer, leading to a significant performance drop in the first FL rounds. Of course, towards the end of the 50 rounds, PPFL performance matches and in some cases surpasses that of end-to-end FL.

In general, with more layers being included in the training, the test accuracy increases. Interestingly, in more complex models (e.g., VGG9) with Non-IID data, PPFL can lead to a drop in ML performance when the number of layers keeps increasing. In fact, in these experiments, it only reaches  $\sim 55\%$  after finishing the second layer and drops. One possible reason for this

degradation is that the first layers of VGG9 are small and may be not capable of capturing heterogeneous features among Non-IID data, which consequently has a negative influence on the training of the latter layers. On the other hand, this reminds us that we can have early exits for greedy layer-wise PPFL on Non-IID data. For example, clients that do not have enough data, or already have high test accuracy after training the first layers can quit before participating in further communication rounds. Overall, the layer-wise training outperforms end-to-end FL during the training of the first or second layer.

This part further discusses possible reasons for PPFL’s better ML performance compared to end-to-end FL. On the one hand, this could be due to some DNN architectures (e.g., VGG9) being more suitable for layer-wise FL. For example, training each layer separately may allow PPFL to overcome possible local optima at which the backward propagation can “get stuck” in end-to-end FL. On the other hand, hyper-parameter tuning may help improve performance in both layer-wise and end-to-end FL, always with the risk of overfitting the data. Indeed, achieving the best ML performance possible was not the focus, and more in-depth studying is needed in the future, to understand under what setups layer-wise can perform better than end-to-end FL.

#### 6.5.4 Client-side system cost

This research further investigates the system performance and costs on the client devices with respect to CPU execution time, memory usage, and energy consumption. Figure 6.3 shows the results for all three metrics, when training LeNet on MNIST, AlexNet, and VGG9 on CIFAR10, on IID data. The metrics are computed for one step of training (i.e., one batch of data). More training steps require analogously more CPU time and energy but do not influence memory usage since the memory allocated for the model is reused for all subsequent steps. Here, I compare PPFL with layer-wise training without TEEs, to measure the overhead of using the TEE. Among the trained models, the maximum overhead is 14.6% for CPU time, 18.31% for memory usage, and 21.19% for energy consumption. In addition, when training each layer, PPFL has comparable results with end-to-end training (i.e., horizontal dashed lines

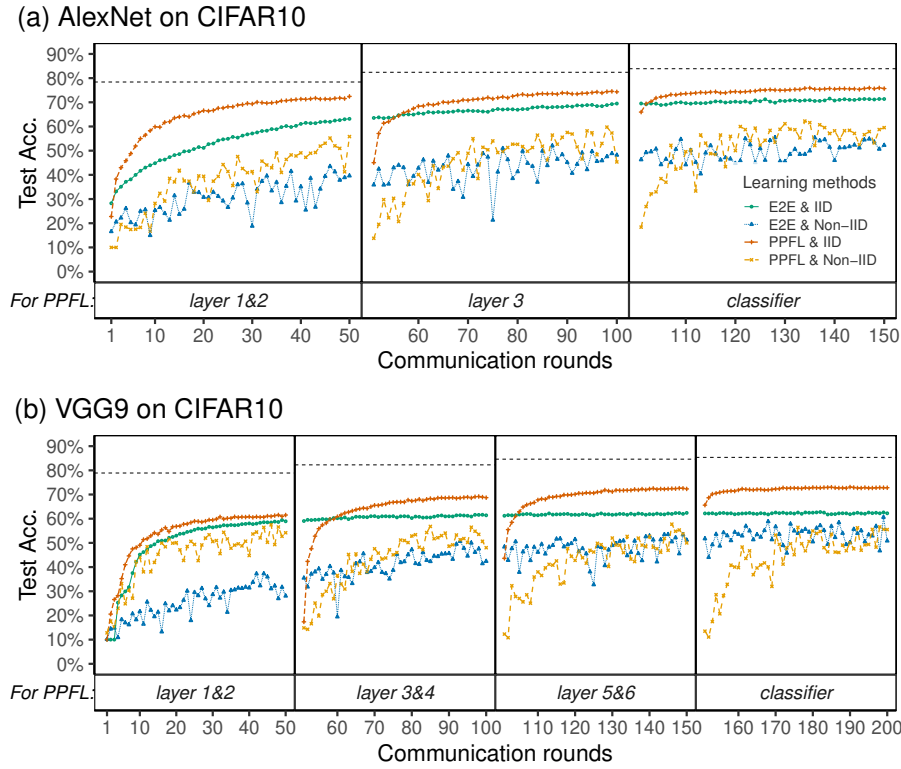


Figure 6.4: Test accuracy of training AlexNet and VGG9 models on CIFAR10 (IID and Non-IID) when using PPFL with blocks of two layers in TEE.

Note: horizontal dashed lines refer to the accuracy that the end-to-end (E2E) FL reaches after 50 communication rounds.

in Figure 6.3).

### 6.5.5 PPFL ML and system costs when blocks of layers were trained in clients

As explained in Algorithm 1 of Sec. 6.3.2, if the TEEs can hold more than one layer, it is also possible to put a block of layers inside the TEE for training. Indeed, heterogeneous devices and TEEs can have different memory sizes, thus supporting a wide range of block sizes. For these experiments, it is assumed that all devices have the same TEE size and construct 2-layer blocks, and measure the system’s test accuracy and ML performance on CIFAR10. The performance of three or more layers inside TEEs could be measured in a similar fashion (if the TEE’s memory can fit them). I do not test LeNet on MNIST because it can easily reach high accuracy (around 99%) as shown earlier and in previous studies [MMR<sup>+</sup>17, WYS<sup>+</sup>20].

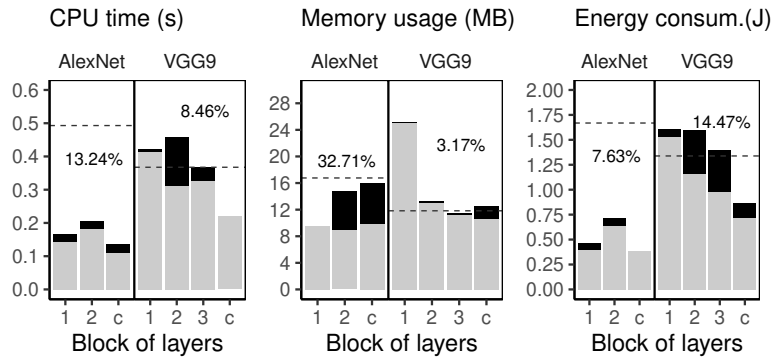


Figure 6.5: System performance of the client devices when training AlexNet and VGG9 models on CIFAR10 when using PPFL with blocks of two layers in TEE.

Note: This uses the same settings as in Figure 6.4, measured on one step of training. The light grey bar (■) refers to learning without TEEs, and the black bar (■) refers to overhead when the block’s layers under training are inside the TEE. Percentage (%) of the overhead is shown above these bars. Horizontal dashed lines refer to the cost of end-to-end FL. ‘c’ refers to ‘classifier’.

Table 6.5: Reduction of communication rounds and amount when training 2-layer instead of 1-layer blocks.

Model	Data	Comm. Rounds	Comm. Amount
AlexNet	IID	$0.65\times \rightarrow 0.18\times$	$0.63\times \rightarrow 0.27\times$
	Non-IID	$0.53\times \rightarrow 0.29\times$	$0.53\times \rightarrow 0.44\times$
VGG9	IID	$1.14\times \rightarrow 0.43\times$	$2.87\times \rightarrow 1.07\times$
	Non-IID	$0.24\times \rightarrow 0.11\times$	$0.60\times \rightarrow 0.27\times$

Results in Figure 6.4 indicate that training blocks of layers can reach similar or even better ML performance compared to training each layer separately (i.e., see Fig. 6.2). It can also improve the test accuracy of complex models such as VGG9, for which I noted a degradation of ML performance caused by the first layer’s small size and incapacity to model the data (see Fig. 6.2). In addition, compared to training one layer at a time, training 2-layer blocks reduces the total required communication to reach the desired ML performance. In fact, while aiming to reach the same baseline accuracy as in Table 6.3, training 2-layer blocks requires half or less communication cost than 1-layer blocks see Table 6.5. Also, layer-wise training outperforms end-to-end FL for similar reasons as outlined in Figure 6.2.

Regarding the system cost, results across models show that the maximum overhead is 13.24% in CPU time, 32.71% in memory usage, and 14.47% in energy consumption (see Fig. 6.5). Compared to training one layer at a time, training layer blocks does not always increase the overhead. For example, overhead when running VGG9 drops from 13.22% to 8.46% in CPU,

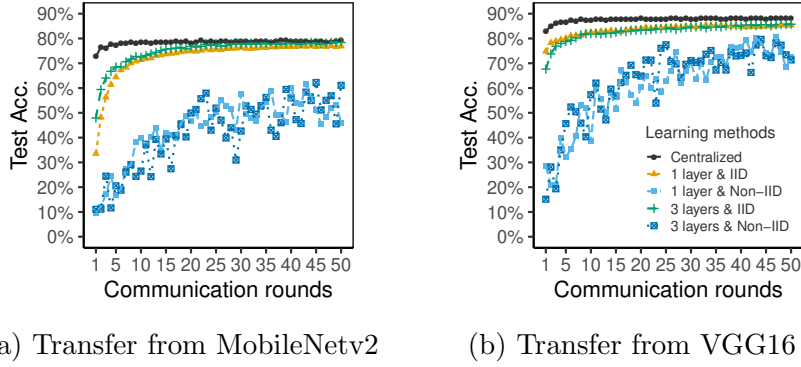


Figure 6.6: Test accuracy of training on CIFAR10 (IID and Non-IID) with public models MobileNetv2 and VGG16, pre-trained on ImageNet.

Note: Both models are trained and tested with 1 and 3 FC layers attached at the end of each model.

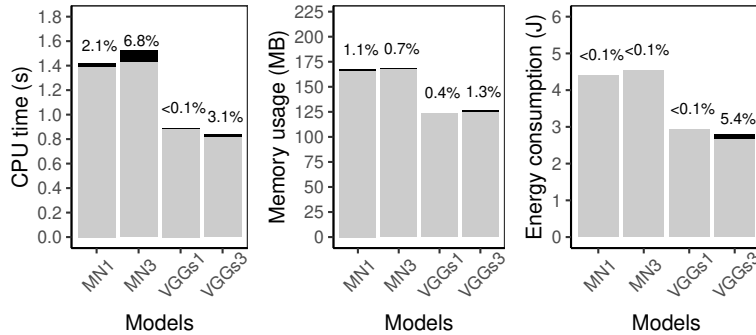


Figure 6.7: System performance of client devices when training with transferred public models on CIFAR10, measured on 1 step of training.

Note: Light grey bar (■): learning without TEEs; Black bar (■): overhead when layers under training are in TEE. Percentage (%) of overhead shown above bars. MN1: MobileNetv2 with one layer for training (i.e., ‘1 layer’ in Figure 6.6a). VGGs: a small size of VGG16.

from 2.44% to 3.17% in memory usage, and from 21.19% to 14.47% in energy consumption. One explanation is that combining layers into blocks amortizes the cost of “expensive” with “cheap” layers. Interestingly, PPFL still has a comparable cost with end-to-end FL training.

### 6.5.6 Bootstrapping the PPFL with public models help

This research investigates how the backend server of PPFL can use existing, public models to bootstrap the training process for a given task. For this purpose, I leverage two models (MobileNetv2 and VGG16) pre-trained on ImageNet to the classification task on CIFAR10. Because these pre-trained models contain sufficient knowledge relevant to the target task, training the

last few layers is already adequate for a good ML performance. Consequently, I can freeze all Conv layers and train the last FC layers within TEEs, thus protecting them as well. By default, MobileNetv2 has one FC layer, and VGG16 has three FC layers at the end. I test both cases that one and three FC layers are attached and re-trained for these two models, respectively. CIFAR10 is resized to  $224 \times 224$  in order to fit with the input size of these pre-trained models. I start with a smaller learning rate of 0.001 to avoid divergence and a momentum of 0.9 because the feature extractors are well-trained.

**Test Accuracy.** Figure 6.6 shows that the use of pre-trained first layers (i.e., feature extractors) to bootstrap the learning process can help the final PPFL models reach test accuracy similar to centralized training. Interestingly, transferring pre-trained layers from VGG16 can reach higher test accuracy than MobileNetv2. This is expected because VGG16 contains many more DNN parameters than MobileNetv2, which provides better feature extraction capabilities. Surprisingly, attaching and training more FC layers at the end of any of the models does not improve test accuracy. This can be due to the bottleneck of the transferred feature extractors, which, since they are frozen, they do not allow the model to *fully* capture the variability of the new data.

**Client-side System Cost.** In order to measure client-side cost under this setting, we need to do some experimental adjustments. The VGG16 (even the last FC layers) is too large to fit in TEEs. Thus, I reduce the batch size to 1 and proportionally scale down all layers (e.g., from 4096 to 1024 neurons for one FC layer). Indeed, scaling layers may lead to biases in results, but the actual performance cannot be worse than this estimation. As shown in [MSK<sup>+</sup>20], larger models have less overhead because the last layers are relatively smaller compared to the complete size of the model.

Interestingly, results shown in Figure 6.7 indicate that when we train and keep the last FC layers inside the client’s on-device TEEs, there is only a small overhead incurred in terms of CPU time (6.9%), memory usage (1.3%), and energy consumption (5.5%) in either model. These results highlight that transferring knowledge can be a good alternative for bootstrapping PPFL training and keeping system overhead low. In addition, note that when the server does

not have suitable public models, it is possible to first train a model on public datasets that have a similar distribution to local datasets.

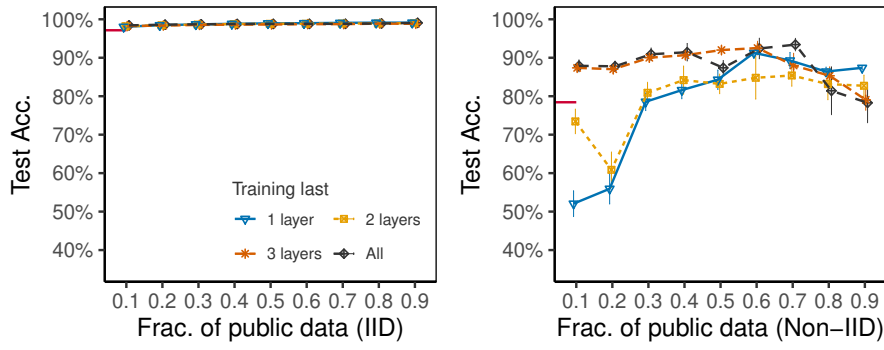
### 6.5.7 Bootstrapping the PPFL with public datasets help

The server can potentially gather data that have a similar distribution to clients' private data. In initialization, the server trains a global model based on the gathered data rather than using one existing model. Then, the server broadcasts the trained model to clients' devices. Clients feed their private data into the model but update only the last layers inside the TEE during local training. Also, only the last layers being trained are uploaded to the server for secure aggregation. Because the server holds public data, it is expected that it retrains the complete model before each communication round in order to keep fine-tuning the first layers. Here, I fix the communication rounds to 20 and measure only the test accuracy. It is expected the system cost to be similar to transferring from models because, similarly, only the last layers are trained at the client-side.

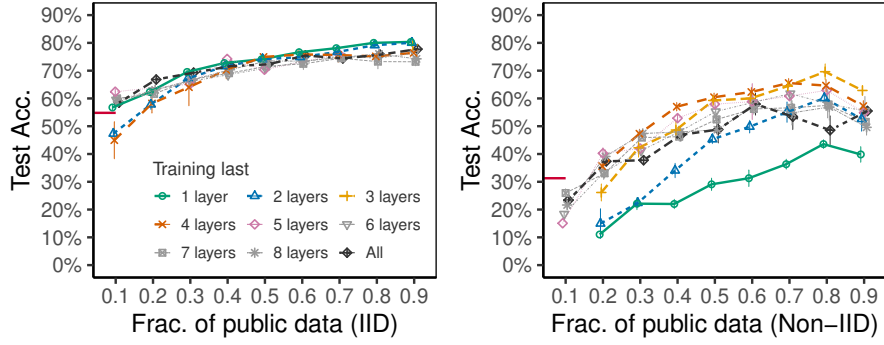
Test accuracy results are shown in Figure 6.8. It is indicated that in general when the server holds more public data, the final global model can reach a higher test accuracy. This is as expected since the server gathers a larger part of the training datasets. With complete training datasets, this process will finally become centralized training. Nevertheless, this indication is not always held. For example, in the IID case (see the two left plots in Figure 6.8), when training all layers, servers with public data of 0.1 fraction outperform servers without public data, i.e., the end-to-end FL, while regarding Non-IID of CIFAR10, servers with 0.1 fraction cannot outperform that without public data (see right plots in Figure 6.8b). One reason for it is that the first layers, which are trained on public datasets, cannot represent all features of privacy datasets. It is also observed that when the server does not have enough public data (e.g., 0.1 fraction), training only the last 1 or 2 layers can lead to extremely low performance or even failure. Still, this is because the first layers cannot represent sufficiently the clients' datasets.

Another observation is that the number of training last layers does not have a significant





(a) Transfer from public MNIST, to train LeNet.



(b) Transfer from public CIFAR10, to train VGG9.

Figure 6.8: Test accuracy when learning with public datasets. The short red line (—) starting from y-axis refers to end-to-end FL.

Note: Each trail runs for 10 times, and error bars refer to 95% confidence interval. In the top left figure, test accuracy is very high and almost the same, as the range of y-axis is set as the same for the same dataset (i.e., MNIST here). In the bottom right figure (i.e., for CIFAR10), several trails fail to train and thus corresponding points are not plotted.

influence on test accuracy in terms of IID cases, especially when the server holds more public data. This is because learning from IID public data is able to represent the feature space of the complete (private) training datasets. However, the results change when it comes to the Non-IID case. The number of training last layers has a significant influence on test accuracy. For instance, regarding VGG9, training only the last 1 or 2 layers at the client-side performs much worse compared to training 3 or 4 layers (see right plots in Figure 6.8b). Moreover, training 3 or 4 layers tends to have better test accuracy than training more layers (e.g., all layers). One explanation is that the feature extraction capability of the first layers is good enough when the server has many public data, so fine-tuning these first layers at the client (e.g., training all layers) may destroy the model and consequently drop the accuracy.

Overall, by training only the last several layers at the client-side, PPFL with public datasets

can guarantee privacy, and in the meanwhile, achieve better performance than that of training all layers.

## 6.6 Discussion and Summary

PPFL's experimental evaluation showed that:

- Protecting the training process (i.e., gradient updates) inside TEEs, and exposing layers only after convergence can thwart data reconstruction and attribute inference attacks. Also, keeping a model's last layer inside TEEs mitigates membership inference attacks.
- Greedy layer-wise FL can achieve comparable ML utility with end-to-end FL. While layer-wise FL increases the total of communication rounds needed to finish all layers, it can reach the same test accuracy as end-to-end FL with fewer rounds ( $0.538\times$ ) and amount of communication ( $1.002\times$ ).
- Most PPFL system cost comes from clients' local training: up to  $\sim 15\%$  CPU time,  $\sim 18\%$  memory usage, and  $\sim 21\%$  energy consumption in client cost when training different models and data, compared to training without TEEs.
- Training 2-layer blocks decreases communication cost by at least half and slightly increases system overhead (i.e., CPU time, memory usage, energy consumption) in cases of small models.
- Bootstrapping PPFL training process with pre-trained models can significantly increase ML utility, and reduce overall cost in communications and system overhead.

**Use cases.** I here present two use cases that can incorporate the technique I develop in this chapter along with the privacy measurement given in previous chapters.

Use case 1: Speech recognition is the key component in voice control where a ML model has been the dominant approach to achieving it. The ML model usually requires updates based

on persons who use it, in order to personalize the model and to achieve a higher recognition accuracy in a federated learning way (similar to Google Keyboard’s Next Word Prediction model trained in the federated learning way). However, giving end users access permission to edit the model would cause privacy leakage about other participants as we discussed such as data reconstruction and attribute information. Such privacy leakage will disclose the speech and voice that the end users have, which may be extremely private, so that end users are not willing to share their model. The Privacy-preserving Federated Learning framework prevents these potential privacy leakages, and end users would be more likely to share their model to improve the model accuracy. The provided privacy measurement tool can give end users a sense that how much privacy they are keeping while improving the their model accuracy, in order to further promote them to share their locally trained model.

Use case 2. More ML-based modules have been increasingly deployed on the vehicle for state-of-the-art functions like auto-driving. Obtaining performant ML-based modules is extremely hard because vehicular environments are multifarious. One approach is that the vehicle trains such a ML model locally during its daily operation and then shares the model with the server for deriving one highly performant global ML model. However, this will leak private information about other uses because the shared model may be attacked. Such private information could contain detailed geographic information, location information, human speech in the car, driving behaviors, etc. More importantly, this information can be used in turn to attack the vehicle which probably causes life-threatening conditions. By utilizing the Privacy-preserving Federated Learning method to protect the model training, all vehicles will benefit from performance gains by sharing the trained model and still keep their private information. This further pushes forwards the development of V2X (vehicles-to-everything) considering that all entities and interactions can cause fatal problems to the vehicle ML system without protection.

**Limitations.** The attacks tested in this thesis assume the classic ‘honest-but-curious’ adversary [PMB14]. In ML, however, there are also *dishonest attacks* or active attacks such as backdoor [SKSM19, BVH<sup>+</sup>20] or poisoning attacks [FCJG20], whose goal is to actively change the global model behavior, e.g., for surreptitious unauthorized access to the global model [JFK20].

---

In the future, one meaningful investigation can be how TEEs' security properties defend against such attacks. I further discuss all other limitations and potential future works in the next chapter.

# Chapter 7

## Conclusion and Future Outlook

### 7.1 Thesis Achievement

Protecting Machine Learning with the Confidential Computing technique is still challenging nowadays. Although many studies have been dedicated to running the training and inference processes inside TEEs, it is still facing the limitation of trust resources in CC. What is severer is that the current protection only provides the confidentiality and integrity of that specific stage in the complete ML pipeline. Malicious behaviors at other stages, especially at the upstream stage like data preparation, can cause huge negative impacts on the ML pipeline but are almost impossible to detect and defend against.

Confidential Computing achieves a hardware-based root-of-trust that can establish a more trustworthy execution environment for ML activities. However, we should rethink whether “hiding” the training/inference process inside such enclaves is the optimal solution, considering the conflict between the large scale of the complete ML pipeline and the need for small TCB. One promising way is to use TEEs wisely and to focus on the key or the most sensitive components, e.g., putting modules like a data signature verifier, a training integrity checker, or an output nosier inside TEEs. Therefore, the questions of utmost importance are how to choose modules to be run inside TEEs and how to design them in efficient and effective ways, which deserve further exploration.

This thesis first demonstrated a technique to improve model privacy for a deployed, pre-trained DNN model using an on-device Trusted Execution Environment (TrustZone). I have applied the protection to individual sensitive layers of the model (i.e., the last layers), which encode a large amount of private information on training data with respect to Membership Inference Attacks. This work analyzed the performance of the protection on two small models trained on the CIFAR-100 and ImageNet Tiny datasets, and six large models trained on the ImageNet dataset, during training and inference. The evaluation indicates that, despite memory limitations, the proposed framework, *DarkneTZ*, is effective in improving models' privacy at a relatively low performance cost. Using *DarkneTZ* adds a minor overhead of under 3% for CPU time, memory usage, and power consumption for protecting the last layer, and of 10% for fully utilizing a TEE's available secure memory to protect the maximum number of layers (depending on the model size and configuration) that the TEE can hold. It is believed that *DarkneTZ* is a step towards stronger privacy protection and high model utility, without significant overhead in local computing resources.

Furthermore, quantifying the information flow in backward propagation and information leakages associated with the computed gradients is still a conundrum, as it remains unclear in what part of the model, and for what kind of attacks, leakages happen. This research presented a framework that encompasses usable information on attack models and generalizes it by measuring the information loss from gradients over a certain *family* of attack models. I also presented *gradient-based* metrics. These metrics work directly on the trained model and are motivated by the mathematical formulation of successful attacks. Empirical results show that the layer-wise analysis provides a better understanding of the memorization of information in neural networks and facilitates the design of flexible layer-level defenses for establishing better trade-offs between privacy and costs. Specifically, mathematically-grounded tools are introduced to better quantify information leakages and applied these tools to localize sensitive information in several models over different datasets and using different training hyperparameters.

In line with the privacy analysis, this thesis proposed PPFL, a practical, privacy-preserving federated learning framework, which protects clients' private information against known privacy-related attacks. This system adopts greedy layer-wise FL training and updates layers always

inside Trusted Execution Environments (TEEs) at both server and clients. I implemented PPFL with mobile-like TEE (i.e., TrustZone) and server-like TEE (i.e., Intel SGX) and empirically tested its performance. For the first time, the possibility is shown to fully guarantee the privacy and achieve comparable ML model utility with regular end-to-end FL, without significant communication and system overhead.

## 7.2 Open Challenges and Future Outlook

There are open challenges that cannot be fully addressed in this thesis. In this section, I detail these open challenges and future outlooks.

**Privacy and Cost Trade-off.** The proposed frameworks guarantee ‘full’ privacy by keeping layers inside TEEs. However, executing computations in secure environments inevitably leads to system costs. To reduce such costs, one can relax their privacy requirements, potentially increasing privacy risks due to inference attacks with higher “advantage” [ZKK20]. For example, clients who do not care about high-level information leakages (i.e., learned model parameters), but want to protect the original local data, can choose to hide only the first layers of the model in TEEs. It is expected that by dropping clients already achieving good performance when training latter layers, one could gain better performance. This may further benefit personalization and achieve better privacy, utility, and cost trade-offs.

**Model Architectures.** The models tested in the layer-wise ML framework are linear links cross consecutive layers. However, the framework can be easily extended to other model architectures that have been studied in standard layer-wise training. For example, one can perform layer-wise training on (i) Graph Neural Networks by disentangling feature aggregation and feature transformation [YCWS20], and (ii) Long Short-Term Memory networks (LSTMs), by adding hidden layers [SK19a]. There are other architectures that contain skipping connections to jump over some layers such as ResNet [HZRS16]. No layer-wise training has been investigated for ResNets, but training a block of layers could be attempted by including the jumping shortcut inside a block.

**Accelerating Local Training.** The frameworks use only the CPU of client devices for local training. Training each layer does not introduce parallel processing on a device. Indeed, more effective ways to perform this compute load can be devised. One way is that clients could use specialized processors (i.e., GPUs) to accelerate training. The design can integrate such advances mainly in two ways. First, the client can outsource the first, well-trained, but non-sensitive layers, to specialized processors that can share computation and speed-up local training. Second, recently proposed GPU-based TEEs can support intensive deep learning-like computation in high-end servers [HJM<sup>+</sup>20, JTK<sup>+</sup>19]. Thus, such TEEs on client devices can greatly speed-up local training. However, as GPU-TEE still requires a small TCB to restrict the attack surface, PPFL’s design can provide a way to leverage limited TEE space for privacy-preserving local training.

**Federated Learning Paradigms.** The framework was tested with *FedAvg*, but there are other state-of-art FL paradigms that are compatible with PPFL. The system leverages greedy layer-wise learning but does not modify the hyper-parameter determination and loss function (which have been improved in *FedProx* [LSZ<sup>+</sup>18]) or aggregation (which is neuron matching-based in *FedMA* [WYS<sup>+</sup>20]). Compared with the system that trains one layer until convergence, *FedMA*, which also uses layer-wise learning, trains each layer for one round and then moves to the next layer. After finishing all layers, it starts again from the first. Thus, *FedMA* is still vulnerable because gradients of one layer are accessible to adversaries. The system could leverage *FedMA*’s neuron-matching technique when dealing with heterogeneous data [KBW<sup>+</sup>21]. Besides, the framework is compatible with other privacy-preserving techniques (e.g., differential privacy) in FL. This is useful during the model usage phase where some users may not have TEEs. Such a system can also be useful to systems such as FLaaS [KKP20] that enable third-party applications to build collaborative ML models on the device shared by said applications. In addition, there exists decentralized FL where clients communicate pair-to-pair [KKU<sup>+</sup>20, TGZ<sup>+</sup>18, MXNV20]. Our design fits into these decentralized FL paradigms without difficulty by leveraging all clients’ TEEs for layer-wise training, and there is no need to change other FL workflow.



## **Licensed Content Re-use Permissions**

### **ACM**

ACM grants gratis permission for individual digital or hard copies made without fee for use in academic classrooms and for use by individuals in personal research and study.

The original Owner/Author permanently holds rights including i) All other proprietary rights not granted to ACM, including patent or trademark rights. ii) Reuse of any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.



# Bibliography

- [AAD18] Francis Akowuah, Amit Ahlawat, and Wenliang Du. Protecting sensitive data in Android SQLite databases using TrustZone. In *Proceedings of the International Conference on Security and Management (SAM)*, pages 227–233. The Steering Committee of The World Congress in Computer Science, 2018.
- [AAUC18] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 51(4):79, 2018.
- [ABC<sup>+</sup>16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
- [AC10] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.
- [ACG<sup>+</sup>16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.
- [ACP19] Galen Andrew, Steve Chien, and Nicolas Papernot. Tensorflow privacy, 2019.

- [AHW<sup>+</sup>17] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.
- [AHW<sup>+</sup>18] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.
- [APS19] Alessandro Achille, Giovanni Paolini, and Stefano Soatto. Where is the information in a deep neural network? *arXiv preprint arXiv:1905.12213*, 2019.
- [Arm09] A Arm. Security technology-building a secure system using TrustZone technology. *ARM Technical White Paper*, 2009.
- [AS19] Julien Amacher and Valerio Schiavoni. On the performance of arm trustzone. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 133–151. Springer, 2019.
- [Asa] Ryo Asakura. rust-autograd.
- [asy] Asylo: An open and flexible framework for enclave applications. <https://asylo.dev/>. Accessed: 2021-10-21.
- [ATG<sup>+</sup>16] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O’keeffe, Mark Stillwell, et al. Scone: Secure linux containers with intel sgx. In *OSDI*, volume 16, pages 689–703, 2016.
- [AW10] Hervé Abdi and Lynne J Williams. Tukeys honestly significant difference (HSD) test. *Encyclopedia of Research Design. Thousand Oaks, CA: Sage*, pages 1–5, 2010.
- [BBR<sup>+</sup>18] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural

estimation. In *International Conference on Machine Learning*, pages 531–540, 2018.

- [BCG<sup>+</sup>19] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.
- [BD17] Jerome R Bellegarda and Jannes G Dolfing. Unified language modeling framework for word prediction, auto-completion and auto-correction, March 30 2017. US Patent App. 15/141,645.
- [BEG<sup>+</sup>19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. In *Conference on Machine Learning and Systems*, 2019.
- [BEO19] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pages 583–593. PMLR, 2019.
- [BF99] Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. *Journal of artificial intelligence research*, 11:131–167, 1999.
- [BFJ<sup>+</sup>20] Sebastian P Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. Offline Model Guard: Secure and private ML on mobile devices. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 460–465. IEEE, 2020.
- [BG73] Ake Björck and Gene H Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of Computation*, 27(123):579–594, 1973.
- [BGJ<sup>+</sup>19] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. SANCTUARY: ARMing TrustZone with user-space enclaves. In *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2019.

- [BIK<sup>+</sup>17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.
- [Bis06] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.
- [BJG20] Dara Bahri, Heinrich Jiang, and Maya Gupta. Deep k-nn for noisy labels. In *International Conference on Machine Learning*, pages 540–550. PMLR, 2020.
- [BLPL06] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153–160, 2006.
- [Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010*, pages 177–186. Springer, 2010.
- [BPH15] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with Haven. *ACM Transactions on Computer Systems (TOCS)*, 33(3):8, 2015.
- [BPS19] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. Differential privacy has disparate impact on model accuracy. In *Advances in Neural Information Processing Systems*, pages 15479–15488, 2019.
- [Bro20] Jason Brownlee. *A Gentle Introduction to Transfer Learning for Deep Learning*, 2019 (accessed November 11, 2020).
- [BVH<sup>+</sup>20] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

- [BWM20] Marcel Busch, Johannes Westphal, and Tilo Mueller. Unearthing the trust-core: A critical review on huaweis trusted execution environment. In *14th {USENIX} Workshop on Offensive Technologies ({WOOT} 20)*, 2020.
- [CBG<sup>+</sup>17] Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, 2017.
- [CBL<sup>+</sup>18] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.
- [cca] Azure confidential computing. <https://azure.microsoft.com/en-gb/solutions/confidential-compute/#overview>. Accessed: 2022-02-03.
- [ccg] Google Cloud Confidential Computing. <https://cloud.google.com/confidential-computing>. Accessed: 2022-02-03.
- [ccn] AWS Nitro Enclaves. <https://aws.amazon.com/ec2/nitro/nitro-enclaves/>. Accessed: 2022-02-03.
- [CD88] William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.
- [CD16] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.
- [CGJvdM21] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. *arXiv preprint arXiv:2112.12727*, 2021.
- [CGL<sup>+</sup>11] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen.

Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, 2011.

- [CHM<sup>+</sup>15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204. PMLR, 2015.
- [CIKW16] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.
- [CLG01] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems*, pages 402–408, 2001.
- [CLL<sup>+</sup>17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.
- [Coh87] Fred Cohen. A cryptographic checksum for integrity protection. *Computers & Security*, 6(6):505–510, 1987.
- [Cona] Confidential Computing Consortium. Confidential computing: Hardware-based trusted execution for applications and data. [https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/confidentialcomputing\\_outreach\\_whitepaper-8-5x11-1.pdf](https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/confidentialcomputing_outreach_whitepaper-8-5x11-1.pdf).
- [Conb] Confidential Computing Consortium. A technical analysis of confidential computing v1.2. <https://confidentialcomputing.io/wp-content/uploads/sites/85/2022/01/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.2.pdf>.
- [Cor19a] Intel Corporation. Intel(r) software guard extensions for linux os. <https://github.com/intel/linux-sgx>, 2019. Access Date :2019-03-01.



- [Cor19b] Microsoft Corporation. Open enclave sdk. <https://github.com/openenclave/openenclave>, 2019. Access Date :2019-08-12.
- [CPM<sup>+</sup>16] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. In *ICLR Workshop Track*, 2016.
- [CSFP20] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. SoK: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1416–1432. IEEE, 2020.
- [CVM<sup>+</sup>21] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D. Garcia. Voltpillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface. In *30th USENIX Security Symposium*, Vancouver, B.C., August 2021.
- [DB16] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4829–4837, 2016.
- [DBJL18] Pan Dong, Alan Burns, Zhe Jiang, and Xiangke Liao. TZDKS: A new TrustZone-based dual-criticality system with balanced performance. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 59–64. IEEE, 2018.
- [DDS<sup>+</sup>09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.
- [DLJ<sup>+</sup>18] Gavin Weiguang Ding, Kry Yik Chau Lui, Xiaomeng Jin, Luyu Wang, and Ruitong Huang. On the sensitivity of adversarial robustness to input data distributions. In *International Conference on Learning Representations*, 2018.

- [DR<sup>+</sup>14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends<sup>®</sup> in Theoretical Computer Science*, 9(3–4):211–407, 2014.
- [Drm00] Zlatko Drmac. On principal angles between subspaces of Euclidean space. *SIAM Journal on Matrix Analysis and Applications*, 22(1):173–194, 2000.
- [DVSH18] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pages 1309–1318. PMLR, 2018.
- [EKA14] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy*, 12(4):29–37, 2014.
- [ena] Enarx. <https://enarx.dev/>. Accessed: 2021-10-21.
- [EUd] 2018 reform of eu data protection rules.
- [FCJG20] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX Security Symposium*, pages 1605–1622, 2020.
- [FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.
- [FNJ<sup>+</sup>20] Lixin Fan, Kam Woh Ng, Ce Ju, Tianyu Zhang, Chang Liu, Chee Seng Chan, and Qiang Yang. Rethinking privacy preserving deep learning: How to evaluate and thwart privacy attacks. In *Federated Learning*, pages 32–50. Springer, 2020.
- [GBDM20] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients—how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

- [GHZ<sup>+</sup>18a] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. Yerbabuena: Securing deep learning inference data via enclave-based ternary model partitioning. *arXiv preprint arXiv:1807.00969*, 2018.
- [GHZ<sup>+</sup>18b] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. Securing input data of deep learning inference systems via partitioned enclave execution. *arXiv preprint arXiv:1807.00969*, 2018.
- [GHZ<sup>+</sup>19] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. Yerbabuena: Securing deep learning inference data via enclave-based ternary model partitioning. 2019.
- [GKN17] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.
- [Glo15] GlobalPlatform. *White Paper on the Trusted Execution Environment*. GlobalPlatform, 2015.
- [GLX<sup>+</sup>17] Le Guan, Peng Liu, Xinyu Xing, Xinyang Ge, Shengzhi Zhang, Meng Yu, and Trent Jaeger. Trustshadow: Secure execution of unmodified applications with arm trustzone. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 488–501. ACM, 2017.
- [Goo18] Google. Asylo - an open and flexible framework for enclave applications. <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm>, 2018.
- [gra] Gramine - a Library OS for Unmodified Applications. <https://grapheneproject.io/>. Accessed: 2021-10-21.
- [GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

- [GTS<sup>+</sup>18] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and secure dnn inference with enclaves. *arXiv preprint arXiv:1810.00602*, 2018.
- [GVDBG<sup>+</sup>19] Ziv Goldfeld, Ewout Van Den Berg, Kristjan Greenewald, Igor Melnyk, Nam Nguyen, Brian Kingsbury, and Yury Polyanskiy. Estimating information flow in deep neural networks. In *International Conference on Machine Learning*, pages 2299–2308, 2019.
- [GWY<sup>+</sup>18] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–633, 2018.
- [HAPC17] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618. ACM, 2017.
- [HCF04] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: A virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*, volume 2004, 2004.
- [HCS18] Nick Hynes, Raymond Cheng, and Dawn Song. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689*, 2018.
- [HCY21] Yaowei Han, Yang Cao, and Masatoshi Yoshikawa. Understanding the interplay between privacy and robustness in federated learning. *arXiv preprint arXiv:2106.07033*, 2021.
- [HGS<sup>+</sup>21] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

- [HJM<sup>+</sup>20] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J Rossbach, and Emmett Witchel. Telekine: Secure computing with cloud gpus. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*, pages 817–833, 2020.
- [HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.
- [HLW<sup>+</sup>20] Tiansheng Huang, Weiwei Lin, Wentai Wu, Ligang He, Keqin Li, and Albert Y Zomaya. An efficiency-boosting client selection scheme for federated learning with fairness guarantee. *arXiv preprint arXiv:2011.01783*, 2020.
- [HMBLM08] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *Workshop on faces in 'Real-Life' Images: detection, alignment, and recognition*, 2008.
- [HMD15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2015.
- [HQL<sup>+</sup>20] Lei Huang, Jie Qin, Li Liu, Fan Zhu, and Ling Shao. Layer-wise conditioning analysis in exploring the learning dynamics of dnns. In *European Conference on Computer Vision*, pages 384–401. Springer, 2020.
- [HRM<sup>+</sup>18] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.
- [HSS<sup>+</sup>18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

- [HTGW18] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 3:123–142, 2018.
- [HZ10] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.
- [HZC<sup>+</sup>17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [HZG<sup>+</sup>18] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. Mlcapsule: Guarded offline deployment of machine learning as a service. *arXiv preprint arXiv:1808.00590*, 2018.
- [HZG<sup>+</sup>21] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Maximilian Augustin, Michael Backes, and Mario Fritz. MLcapsule: Guarded offline deployment of machine learning as a service. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3300–3309, 2021.
- [HZL19] Zecheng He, Tianwei Zhang, and Ruby B Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, 2019.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [IHM<sup>+</sup>16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

- [int] Intel Software Guard Extensions SDK. <https://01.org/intel-software-guard-extensions>. Accessed: 2021-10-21.
- [JE19a] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1895–1912, Santa Clara, CA, August 2019. USENIX Association.
- [JE19b] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In *28th {USENIX} Security Symposium*, pages 1895–1912, 2019.
- [JFK20] M. S. Jere, T. Farnan, and F. Koushanfar. A taxonomy of attacks on federated learning. *IEEE Security & Privacy*, pages 0–0, 2020.
- [JG18] Jinyuan Jia and Neil Zhenqiang Gong. Attriguard: A practical defense against attribute inference attacks via adversarial machine learning. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 513–529, 2018.
- [JKC<sup>+</sup>18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.
- [JM15] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.
- [JSB<sup>+</sup>19] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. MemGuard: Defending against black-box membership inference attacks via adversarial examples. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, page 259274, 2019.
- [JSR<sup>+</sup>16] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel software guard extensions: Epid provisioning and attestation services. *White Paper*, 1(1-10):119, 2016.

- [JTK<sup>+</sup>19] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. Heterogeneous isolated execution for commodity gpus. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 455–468, 2019.
- [K<sup>+</sup>95] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Montreal, Canada, 1995.
- [KBBN09] Neeraj Kumar, Alexander C Berg, Peter N Belhumeur, and Shree K Nayar. Attribute and simile classifiers for face verification. In *2009 IEEE 12th International Conference on Computer Vision*, pages 365–372. IEEE, 2009.
- [KBW<sup>+</sup>21] Kleomenis Katevas, Eugene Bagdasaryan, Jason Waterman, Mohamad Mounir Safadieh, Eleanor Birrell, Hamed Haddadi, and Deborah Estrin. Policy-based federated learning. *arXiv preprint arXiv:2003.06612*, 2021.
- [key] Keystone: An Open Framework for Architecting Trusted Execution Environments. <https://keystone-enclave.org/>. Accessed: 2021-10-21.
- [KFPC16] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. Verena: End-to-end integrity protection for web applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 895–913. IEEE, 2016.
- [KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- [KHD19] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, pages 3301–3310. PMLR, 2019.
- [Kim14] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.



- [KKP20] Nicolas Kourtellis, Kleomenis Katevas, and Diego Perino. Flaas: Federated learning as a service. In *Workshop on Distributed ML*. ACM CoNEXT, 2020.
- [KKU<sup>+</sup>20] Caner Korkmaz, Halil Eralp Kocas, Ahmet Uysal, Ahmed Masry, Oznur Ozkasap, and Baris Akgun. Chain fl: decentralized federated machine learning via blockchain. In *2020 Second international conference on blockchain computing and applications (BCCA)*, pages 140–146. IEEE, 2020.
- [KMA<sup>+</sup>19] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.
- [KNHa] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).
- [KNHb] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).
- [KQG<sup>+</sup>19] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. TensorSCONE: A secure tensorflow framework using Intel SGX. *arXiv preprint arXiv:1902.04413*, 2019.
- [Kra03] Hugo Krawczyk. SIGMA: The SIGn-and-MAcapproach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Annual International Cryptology Conference*, pages 400–425. Springer, 2003.
- [KSC<sup>+</sup>18] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863*, 2018.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

- [KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [LAG<sup>+</sup>19] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.
- [LAP<sup>+</sup>14] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.
- [LBB<sup>+</sup>98] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [LBLE09] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of machine learning research*, 10(1), 2009.
- [LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2:18, 2010.
- [Le13] Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

- [Lin20] Linaro.org. *Open Portable Trusted Execution Environment*, 2020 (accessed September 3, 2020).
- [LJJ<sup>+</sup>17] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent Byunghoon Kang. Hacking in Darkness: Return-Oriented Programming against Secure Enclaves. In *Proceedings of the 26th USENIX Conference on Security Symposium*, pages 523–539, 2017.
- [LKO<sup>+</sup>21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Eason, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.
- [LKS<sup>+</sup>20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys 20*, 2020.
- [LKX<sup>+</sup>20] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 2020.
- [LLP<sup>+</sup>19] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2019.
- [LLWT15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [LLX<sup>+</sup>21] Zhuang Liu, Ye Lu, Xueshuo Xie, Yaozheng Fang, Zhaolong Jian, and Tao Li. Trusted-DNN: A trustzone-based adaptive isolation strategy for deep neural

networks. In *ACM Turing Award Celebration Conference-China (ACM TURC 2021)*, pages 67–71, 2021.

- [LMBL20] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pages 182–199. Springer, 2020.
- [LPM<sup>+</sup>17] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O’Keeffe, P Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rüdiger Kapitza, et al. Glamdring: Automatic application partitioning for intel sgx. *USENIX*, 2017.
- [LQS<sup>+</sup>13] Ninghui Li, Wahbeh Qardaji, Dong Su, Yi Wu, and Weining Yang. Membership privacy: a unifying framework for privacy definitions. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, pages 889–900. ACM, 2013.
- [LSTS20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.
- [LSZ<sup>+</sup>18] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.
- [LTJZ20] Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking. In *Federated learning*, pages 240–254. Springer, 2020.
- [LWH<sup>+</sup>22] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. ML-Doctor: Holistic risk assessment of inference attacks against machine learning models. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

- [LXL<sup>+</sup>21] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *arXiv preprint arXiv:2103.10689*, 2021.
- [LZC<sup>+</sup>22] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *arXiv preprint arXiv:2206.15472*, 2022.
- [MAE<sup>+</sup>18] H Brendan McMahan, Galen Andrew, Ulfar Erlingsson, Steve Chien, Ilya Mironov, Nicolas Papernot, and Peter Kairouz. A general approach to adding differential privacy to iterative training procedures. *arXiv preprint arXiv:1812.06210*, 2018.
- [Mar18] Pedro Marcelino. Transfer learning from pre-trained models. *Towards Data Science*, 2018.
- [MBG19] Muhammad Asim Mukhtar, Muhammad Khurram Bhatti, and Guy Gogniat. Architectures for security: A comparative analysis of hardware security features in intel sgx and arm trustzone. In *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, pages 299–304. IEEE, 2019.
- [MBM<sup>+</sup>21a] Fan Mo, Anastasia Borovykh, Mohammad Malekzadeh, Hamed Haddadi, and Soteris Demetriou. Layer-wise characterization of latent information leakage in federated learning. *ICLR Distributed and Private Machine Learning workshop*, 2021.
- [MBM<sup>+</sup>21b] Fan Mo, Anastasia Borovykh, Mohammad Malekzadeh, Hamed Haddadi, and Soteris Demetriou. Quantifying information leakage from gradients. *arXiv preprint arXiv:2105.13929*, 2021.

- [MDFFF17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.
- [MDP<sup>+</sup>11] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.
- [Mer80] Ralph C Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122. IEEE, 1980.
- [Mer89] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.
- [MH19] Fan Mo and Hamed Haddadi. Efficient and private federated learning using tee. In *EuroSys*, 2019.
- [MHK<sup>+</sup>21] Fan Mo, Haddadin Hamed, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. PPFL: Privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th International Conference on Mobile Systems, Applications, and Services*, 2021.
- [Mic20] Microsoft. *Open Enclave SDK*, 2020 (accessed Decemenber 4, 2020).
- [Mir17] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017.
- [MMB10] Grégoire Montavon, Klaus-Robert Müller, and Mikio Braun. Layer-wise analysis of deep networks with Gaussian kernels. *Advances in Neural Information Processing Systems*, 23:1678–1686, 2010.
- [MMR<sup>+</sup>17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from

decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

- [Mon20] Monsoon. Monsoon solutions inc. home page. <https://www.msoon.com/>, 2020 (accessed November 12, 2020).
- [MPA<sup>+</sup>16] Ian McGraw, Rohit Prabhavalkar, Raziq Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Haşim Sak, Alexander Gruenstein, Françoise Beaufays, et al. Personalized speech recognition on mobile devices. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5955–5959. IEEE, 2016.
- [MR96] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Computing Surveys (CSUR)*, 28(1):33–37, 1996.
- [MRTZ18] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.
- [MSDCS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. IEEE, 2019.
- [MSK<sup>+</sup>19] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Andrea Cavallaro, and Hamed Haddadi. Poster: Towards characterizing and limiting information exposure in dnn layers. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2653–2655. ACM, 2019.
- [MSK<sup>+</sup>20] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.
- [mul] MultiZone Security for RISC-V. <https://hex-five.com/multizone-security-sdk/>. Accessed: 2021-10-21.

- [MV15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.
- [MXNV20] Othmane Marfoq, Chuan Xu, Giovanni Neglia, and Richard Vidal. Throughput-optimal topology design for cross-silo federated learning. *Advances in Neural Information Processing Systems*, 33:19478–19487, 2020.
- [NBA<sup>+</sup>18] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations (ICLR)*, 2018.
- [NCW<sup>+</sup>21] Lucien KL Ng, Sherman SM Chow, Anna PY Woo, Donald PH Wong, and Yongjun Zhao. Goten: Gpu-outsourcing trusted execution of neural network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14876–14883, 2021.
- [NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.
- [NMB<sup>+</sup>16] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. Trustzone explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 445–451. IEEE, 2016.
- [NSH18] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910*, 2018.



- [NSH19] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753. IEEE, 2019.
- [NY19] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.
- [occ] Occlum - A library OS empowering everyone to run every application in secure enclaves. <https://occlum.io/>. Accessed: 2021-10-21.
- [ope] Open Enclave SDK. <https://openenclave.io/sdk/>. Accessed: 2021-10-21.
- [opt] Open Portable Trusted Execution Environment. <https://www.op-tee.org/>. Accessed: 2021-10-21.
- [OSF<sup>+</sup>16] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, pages 619–636, 2016.
- [OSF19] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019.
- [OST<sup>+</sup>17] Seyed Ali Osia, Ali Shahin Shamsabadi, Ali Taheri, Kleomenis Katevas, Sina Sajadmanesh, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. A hybrid deep learning architecture for privacy-preserving mobile analytics. *arXiv preprint arXiv:1703.02952*, 2017.
- [OST<sup>+</sup>18] Seyed Ali Osia, Ali Shahin Shamsabadi, Ali Taheri, Hamid R Rabiee, and Hamed Haddadi. Private and scalable personal data analytics using hybrid edge-to-cloud deep learning. *Computer*, 51(5):42–49, 2018.

- [Pan03] Liam Paninski. Estimation of entropy and mutual information. *Neural computation*, 15(6):1191–1253, 2003.
- [PBWH<sup>+</sup>11] Donald E Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C Hunt. Rethinking the library os from the top down. In *ACM SIGPLAN Notices*, volume 46, pages 291–304. ACM, 2011.
- [PGC<sup>+</sup>17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.
- [PGM<sup>+</sup>19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.
- [PKE18] Nicolae Paladi, Linus Karlsson, and Khalid Elbashir. Trust Anchors in Software Defined Networks. In *Computer Security*, pages 485–504, 2018.
- [PMB14] AJ Paverd, Andrew Martin, and Ian Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep.*, 2014.
- [PMSW18] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. SoK: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.
- [Pro] The Linux Foundation Project. Confidential computing consortium.
- [PSY<sup>+</sup>19] Venkatadheeraj Pichapati, Ananda Theertha Suresh, Felix X Yu, Sashank J Reddi, and Sanjiv Kumar. Adaclip: Adaptive clipping for private sgd. *arXiv preprint arXiv:1908.07643*, 2019.

- [PSZC11] Nicolas Pinto, Zak Stone, Todd Zickler, and David Cox. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. In *CVPR 2011 WORKSHOPS*, pages 35–42. IEEE, 2011.
- [PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.
- [PZLL19] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. StreamBox-TZ: secure stream analytics at the edge with TrustZone. In *2019 {USENIX} Annual Technical Conference 19*, pages 537–554, 2019.
- [QBL18] Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5822–5830, 2018.
- [Qua19] Qualcomm. *Guard Your Data with the Qualcomm Snapdragon Mobile Platform*. Qualcomm, 2019.
- [RBU18] Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. Down-sampling leads to image memorization in convolutional autoencoders. *arXiv preprint arXiv:1810.10333*, 2018.
- [Red16] Joseph Redmon. Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>, 2013–2016.
- [RGC15] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902. IEEE, 2015.
- [RHL<sup>+</sup>20] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

- [RRL<sup>+</sup>18] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Transactions on Data Privacy*, 11(1):61–79, 2018.
- [SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [SBD<sup>+</sup>19] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.
- [SCF<sup>+</sup>15] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54. IEEE, 2015.
- [sco] SCONE - A secure Container Environment. <https://scontain.com/>. Accessed: 2021-10-21.
- [SDS<sup>+</sup>19] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, Yann Ollivier, and Hervé Jégou. White-box vs black-box: Bayes optimal strategies for membership inference. In *International Conference on Machine Learning*, pages 5558–5567, 2019.
- [SEA20] Microsoft SEAL (release 3.5). <https://github.com/Microsoft/SEAL>, April 2020. Microsoft Research, Redmond, WA.
- [SGSR17] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.
- [Sha48] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

- [Sha49] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.
- [She20] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.
- [SHZ<sup>+</sup>18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [SK19a] Alaa Sagheer and Mostafa Kotb. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports*, 9(1):1–16, 2019.
- [SK19b] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.
- [SKSM19] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.
- [SLJ<sup>+</sup>15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [SRSW14] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. *ACM SIGARCH Computer Architecture News*, 42(1):67–80, 2014.
- [SS15] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

- [SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.
- [SSSS10] Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability, stability and uniform convergence. *Journal of Machine Learning Research*, 11(Oct):2635–2670, 2010.
- [SVK20] Pranav Subramani, Nicholas Vadivelu, and Gautam Kamath. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *arXiv preprint arXiv:2010.09063*, 2020.
- [SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [SZH<sup>+</sup>18] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. MI-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018.
- [SZT17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.
- [TB18] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.
- [TCCS20] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.
- [tea] Apache Teaclave (incubating). <https://teaclave.apache.org/>. Accessed: 2021-10-21.

- [TGGW21] Jean-Baptiste Truong, William Gallagher, Tian Guo, and Robert J Walls. Memory-efficient deep learning inference in trusted execution environments. *arXiv preprint arXiv:2104.15109*, 2021.
- [TGS<sup>+</sup>18] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and secure dnn inference. *arXiv preprint arXiv:1810.00602*, 2018.
- [TGZ<sup>+</sup>18] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. Communication compression for decentralized training. *Advances in Neural Information Processing Systems*, 31, 2018.
- [TKP<sup>+</sup>18] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018.
- [TM21] Davide Testuggine and Ilya Mironov. *Introducing Opacus: A high-speed library for training PyTorch models with differential privacy*, 2020 (accessed January 1, 2021).
- [TPB00] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.
- [TPV17] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, page 8, 2017.
- [Tru] Trustnic. Trustonic: World leading embedded cybersecurity technology trustonic.
- [TS10] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

- [TSJ<sup>+</sup>20] Chia-Che Tsai, Jeongseok Son, Bhushan Jain, John McAvey, Raluca Ada Popa, and Donald E Porter. Civet: An efficient java partitioning framework for hardware enclaves. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.
- [TTGL20] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*, pages 480–501. Springer, 2020.
- [TZJ<sup>+</sup>16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.
- [URPPK22] Dmitrii Usynin, Daniel Rueckert, Jonathan Passerat-Palmbach, and Georgios Kaissis. Zen and the art of model adaptation: Low-utility-cost attack mitigations in collaborative machine learning. *Proceedings on Privacy Enhancing Technologies*, 2022(1):274–290, 2022.
- [Van19] Van Bulck, Jo and Oswald, David and Marin, Eduard and Aldoseri, Abdulla and Garcia, Flavio D. and Piessens, Frank. A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1741–1758, 2019.
- [VD14] Ben Verhoeven and Walter Daelemans. CLiPS Stylometry Investigation (CSI) corpus: A Dutch corpus for the detection of age, gender, personality, sentiment and deception in text. In *LREC*, pages 3081–3085, 2014.
- [ver] Veracruz: Privacy-Preserving Collaborative Compute. <https://veracruz-project.com/>. Accessed: 2021-10-21.
- [VFGJ16] Esteban Vazquez-Fernandez and Daniel Gonzalez-Jimenez. Face recognition for authentication on mobile devices. *Image and Vision Computing*, 55:31–33, 2016.



- [VVB18] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on GPUs. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 681–696, 2018.
- [WBSS04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [WCP<sup>+</sup>17] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2421–2434. ACM, 2017.
- [WI20] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in Neural Information Processing Systems*, 2020.
- [WKPK16] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. Async-shock: Exploiting synchronisation bugs in intel sgx enclaves. In *European Symposium on Research in Computer Security*, pages 440–457. Springer, 2016.
- [WLL<sup>+</sup>19] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.
- [WPC<sup>+</sup>20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.
- [WSZ<sup>+</sup>19] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from feder-

ated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019.

- [WYS<sup>+</sup>19] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2019.
- [WYS<sup>+</sup>20] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*, 2020.
- [XDH<sup>+</sup>20] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, 33, 2020.
- [XLL<sup>+</sup>19] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, pages 2125–2136, 2019.
- [XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [XRZ<sup>+</sup>19] Chugui Xu, Ju Ren, Deyu Zhang, Yaoxue Zhang, Zhan Qin, and Kui Ren. GANobfuscator: Mitigating information leakage under GAN via differential privacy. *IEEE Transactions on Information Forensics and Security*, 14(9):2358–2371, 2019.
- [XZS<sup>+</sup>20] Yilun Xu, Shengjia Zhao, Jiaming Song, Russell Stewart, and Stefano Ermon. A theory of usable information under computational constraints. In *International Conference on Learning Representations (ICLR)*, 2020.
- [YAA<sup>+</sup>18] Kailiang Ying, Amit Ahlawat, Bilal Alsharifi, Yuexin Jiang, Priyank Thavai, and Wenliang Du. TruZ-Droid: Integrating TrustZone with mobile operating

- system. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 14–27. ACM, 2018.
- [YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.
- [YCN<sup>+</sup>15] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *Deep Learning Workshop in International Conference on Machine Learning*, 2015.
- [YCWS20] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2127–2135, 2020.
- [YGFJ18] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282. IEEE, 2018.
- [YL16] Ke Ye and Lek-Heng Lim. Schubert varieties and distances between subspaces of different dimensions. *SIAM Journal on Matrix Analysis and Applications*, 37(3):1176–1197, 2016.
- [YLCT19] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.
- [YLP<sup>+</sup>19] Lei Yu, Ling Liu, Calton Pu, Mehmet Emre Gursoy, and Stacey Truex. Differentially private model publishing for deep learning. In *Proceedings of 40th IEEE Symposium on Security & Privacy*, pages 332–349. IEEE, 2019.
- [YMV<sup>+</sup>21] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradInversion. *arXiv preprint arXiv:2104.07586*, 2021.

- [YXW<sup>+</sup>17] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Complexity vs. performance: empirical analysis of machine learning as a service. In *Proceedings of the 2017 Internet Measurement Conference*, pages 384–397, 2017.
- [YYZ<sup>+</sup>20] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. Cloudleak: Large-scale deep learning models stealing through adversarial examples. In *Network and Distributed System Security Symposium (NDSS)*, 2020.
- [YZCL19] Ziqi Yang, Jiyi Zhang, Ee-Chien Chang, and Zhenkai Liang. Neural network inversion in adversarial setting via background knowledge alignment. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 225–240. ACM, 2019.
- [ZB20] Junyi Zhu and Matthew B Blaschko. R-gap: Recursive gradient attack on privacy. In *International Conference on Learning Representations*, 2020.
- [ZBH<sup>+</sup>17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Toulon, France, April 2017.
- [ZF14] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
- [ZG09] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.
- [ZKK20] Benjamin Zi Hao Zhao, Mohamed Ali Kaafar, and Nicolas Kourtellis. Not one but many tradeoffs: Privacy vs. utility in differentially private machine learning. In *Cloud Computing Security Workshop*. ACM CCS, 2020.

- [ZKS<sup>+</sup>18] Tom Zahavy, Bingyi Kang, Alex Sivak, Jiashi Feng, Huan Xu, and Shie Mannor. Ensemble robustness and generalization of stochastic deep learning algorithms. In *International Conference on Learning Representations Workshop*, 2018.
- [ZLH19a] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14774–14784, 2019.
- [ZLH19b] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14747–14756, 2019.
- [ZLZ<sup>+</sup>20] Xiaoli Zhang, Fengting Li, Zeyu Zhang, Qi Li, Cong Wang, and Jianping Wu. Enabling execution assurance of federated learning at untrusted participants. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1877–1886. IEEE, 2020.
- [ZMB20] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.
- [ZSE17] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Learning hierarchical features from deep generative models. In *International Conference on Machine Learning*, pages 4091–4099, 2017.
- [ZXY<sup>+</sup>21] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. Citadel: Protecting data privacy and model confidentiality for collaborative learning. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 546–561, 2021.
- [ZZQ<sup>+</sup>19] Shijun Zhao, Qianying Zhang, Yu Qin, Wei Feng, and Dengguo Feng. SecTEE: A software-based approach to secure enclave architecture using TEE. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1723–1740. ACM, 2019.
- [ZZS<sup>+</sup>22] Zhaohua Zheng, Yize Zhou, Yilong Sun, Zhang Wang, Boyi Liu, and Keqiu Li. Applications of federated learning in smart cities: recent advances, taxonomy, and open challenges. *Connection Science*, 34(1):1–28, 2022.

- [EPK14] Irfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale, Arizona, 2014.