DarkneTZ: Towards Model Privacy at the Edge using Trusted Execution Environments

Fan Mo
Imperial College London

Ali Shahin Shamsabadi
Queen Mary University of London

Kleomenis Katevas
Telefónica Research

Soteris Demetriou
Imperial College London

Ilias Leontiadis
Samsung AI

Andrea Cavallaro
Queen Mary University of London

Hamed Haddadi
Imperial College London

ABSTRACT

We present *DarkneTZ*, a framework that uses an edge device's Trusted Execution Environment (TEE) in conjunction with model partitioning to limit the attack surface against Deep Neural Networks (DNNs). Increasingly, edge devices (smartphones and consumer IoT devices) are equipped with pre-trained DNNs for a variety of applications. This trend comes with privacy risks as models can leak information about their training data through effective membership inference attacks (MIAs).

We evaluate the performance of *DarkneTZ*, 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, we partition model layers 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. Our results show that even if a single layer is hidden, we can provide reliable model privacy and defend against state of the art MIAs, with only 3% performance overhead. When fully utilizing the TEE, *DarkneTZ* provides model protections with up to 10% overhead.

CCS CONCEPTS

• **Security and privacy** → **Embedded systems security**; • **Computing methodologies** → **Machine learning**.

ACM Reference Format:
Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: Towards Model Privacy at the Edge using Trusted Execution Environments. In *The 18th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '20), June 15–19, 2020, Toronto, ON, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3386901.3388946

1 INTRODUCTION

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). 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 [41]. Nonetheless, recent work demonstrated that local models still leak private information [21, 33, 34, 47, 57, 61–63]. This can be used by adversaries aiming to compromise the confidentiality of the model itself or that of the participants in training the model [48, 57]. 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 [33, 47] motivating a number of research works to focus on tackling them [1, 28, 35]. Predominantly, such mitigation approaches rely on differential privacy [14, 68], whose improvement in privacy preservation comes with an adverse effect on the model's prediction accuracy.

We observe, 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 can enforce memory isolation between an untrusted part of the system operating in a *Rich Execution Environment* (REE), and a smaller trusted component operating in hardware-isolated *Trusted Execution Environment* (TEE), responsible for security critical operations. If we 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 [17, 19, 24, 50, 51]. 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 (e.g., [17] and [24]) 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 we 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 in an edge device's TEE might not be computationally practical or even possible. We need a systematic measurement of the effects of such designs on edge-like environments.

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. We utilize the TEE (i.e., Arm TrustZone) and perform a unique layer-wise analysis to illustrate the privacy repercussions of an adversary on relevant neural network models on edge devices with the corresponding performance effects. To the best of our knowledge, we are the first to embark on examining to what extent this is feasible on resource-constrained mobile devices. Specifically, we lay out the following research question:

RQ1: *Is it practical to store and execute a sequence of sensitive DNN's layers inside the TEE of an edge device?*

To answer this question we design 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. We use DarkneTZ to measure, for a given DNN—we evaluate 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. We demonstrate our prototype of DarkneTZ using the Open Portable TEE (OP-TEE)[1] software stack running on a Hikey 960 board.[2] OP-TEE is compatible with the mobile-popular Arm TrustZone-enabled hardware, while our choice of hardware closely resembles common edge devices' capabilities [42, 58]. Our 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. Given this, we next ask the following question:

RQ2: *Are such partitions useful to both effectively and efficiently tackle realistic attacks against DNNs on mobile devices?*

To answer this question, we develop a threat model considering state of the art MIAs against DNNs. We implement the respective attacks and use DarkneTZ to measure their effectiveness (adversary's success rate) for different model partition choices. We 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. We 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.

Paper Organisation. The rest of the paper is organized as follows: Section 2 discusses background and related work and Section 3 presents the design and main components of DarkneTZ. Section 4 provides implementation details and describes our evaluation setup (our implementation is available online[3]), while Section 5 presents our system performance and privacy evaluation results. Lastly, Section 6 discusses further performance and privacy implications that can be drawn from our systematic evaluation and we conclude on Section 7.

2 BACKGROUND AND RELATED WORK

2.1 Privacy risks of Deep Neural Networks

Model privacy risks. With successful training (i.e., the model converging to an optimal solution), a DNN model "memorizes" features of the input training data [44, 57] (see [32, 64] for more details on deep learning), which it can then use to recognize unseen data exhibiting similar patterns. However, models have the tendency to include more specific information of the training dataset unrelated to the target patterns (i.e., the classes that the model aims to classify) [9, 57].

Moreover, each layer of the model memorizes different information about the input. Yosinki et al. [59] 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 [60, 62] and in generative models [65]. 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.

Membership inference attack (MIA). MIAs form a possible attack on devices which leverage memorized information on a models' layers to determine whether a given data record was part of the model's training dataset [48]. 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 [48, 57]. 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 [38]. It is shown that the last layer (model output) has the highest membership information about the training data [38]. We consider a white-box adversary as our threat model, as DNNs are fully accessible after being transferred from the server to edge devices [55]. 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.

[1]https://www.op-tee.org/
[2]https://www.96boards.org/product/hikey960/
[3]https://github.com/mofanv/darknetz

2.2 Deep learning in the TEE

Trusted execution environment (TEE). A TEE is a trusted component which 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 [7, 10].

In addition to strong security guarantees, TEEs also provide better computational performance than existing software protections, making it 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 [3, 37]. 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 we consider, these also differ with our approach in one more aspect: they depend on an underlying computer architecture which is more suitable for cloud environments. Recent work has suggested executing a complete deep learning model in a TEE [10], where during training, users' private data is transferred to the trusted environment using trusted paths. This prevents the host Cloud form eavesdropping on the data [39]. Several other studies improved the efficiency of TEE-resident models using Graphics Processing Units (GPU) [51], multiple memory blocks [24], and high-performance ML frameworks [25]. More similar to our approach, Gu et al. [17] 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 we consider, 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) [15]. 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 developments and investigations because of its different features compared to SGX.

2.3 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 [48]. *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 [1, 61]. However, DP may negatively affect the utility (i.e., the prediction accuracy) if the noise is not carefully designed [45]. In order to obfuscate private information only, one could apply methods such as generative neural networks [54] or adversarial examples [29] 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 [59] during 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 [6, 13]. 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 [40]. 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. [17] 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 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 [51]. 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.

3 DARKNETZ

We now describe DarkneTZ, a framework for preserving DNN models' privacy on edge devices. We start with the threat model which we focus on in this paper.

3.1 Threat Model

We consider an adversary with full access to the REE of an edge device (e.g., the OS) on edge devices: this could be the actual user, malicious third-party software installed on the devices, or a malicious or compromised OS. We only trust the TEE of an edge device to guarantee the integrity and confidentiality of the data and software in it. In particular, we assume that a DNN model is pre-trained using private data from the server or other participating nodes. We assume the model providers can fully guarantee the model privacy during training on their servers by utilizing existing protection

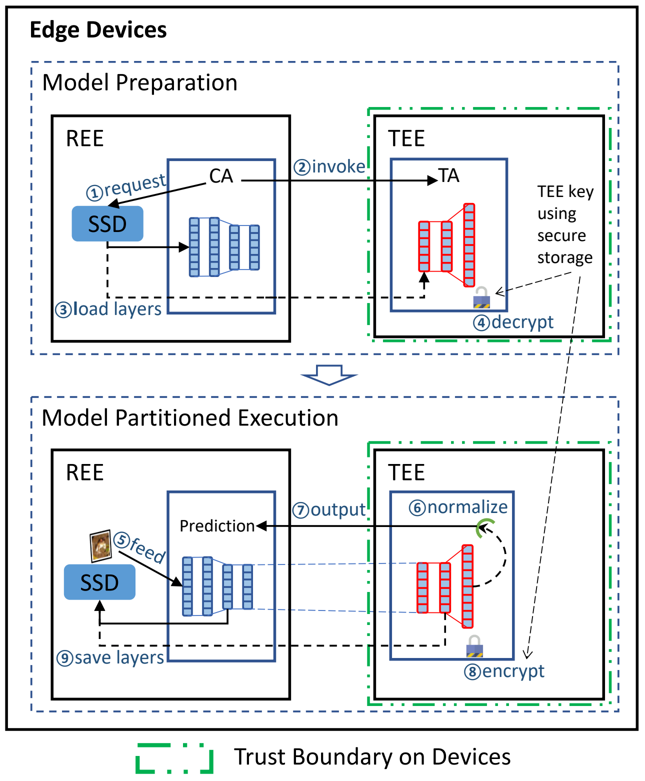


Figure 1: DarkneTZ uses on-device TEE to protect a set of layers of a deep neural network for both inference and fine-tuning. (Note: The trusted compute base—or trust boundary—for the model owner on edge devices is the TEE of the device).

methods [39] or even by training the model offline, so the model can be secret provisioned to the user devices without other privacy issues.

3.2 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), use standard TEE system architectures and corresponding APIs.

We propose DarkneTZ, illustrated in Figure 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 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 [36, 38], as the last layers have a higher probability of leaking private information about training data (see Section 2).

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 [30], modified to include remote attestation, similar to [66]. 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.

3.3 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 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 [7]. At this point, the model is ready for fine-tuning and inference.

3.4 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

$\mathbf{w}^l = \mathbf{w}^l - \eta \frac{\partial \mathcal{L}(\mathrm{a}^L, y)}{\partial \mathbf{w}^l}$ and $\mathbf{b}^l = \mathbf{b}^l - \eta \frac{\partial \mathcal{L}(\mathrm{a}^L, y)}{\partial \mathbf{b}^l}$, where η 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 back propagate 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 EXPERIMENT SETTINGS

4.1 Models and Datasets

We first use 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. We train the networks and conduct inference on CIFAR-100 and ImageNet Tiny. We use image classification datasets, as a recent empirical study shows that the majority of smartphone applications (70.6%) that use deep learning are for image processing [55]. Moreover, the state of the art MIA we are considering is demonstrated against such datasets [38]. 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, we use six available DNNs (Tiny Darknet (4 megabytes (MB)), Darknet Reference (28MB), Extraction [49] (90MB), Resnet-50 [20] (87MB), Densenet-201 [23] (66MB), and Darknet-53-448 (159MB)) pre-trained on the original ImageNet [11] dataset to measure DarkneTZ's performance during *inference*. All pre-trained models can be found online[4]. 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, we only evaluate the condition that their *last layer* is in the TEE.

To evaluate the defence's effectiveness against MIAs, we use the same models as those used in the demonstration of the attack[38] (AlexNet, VGG-7, and ResNet-110). This ResNet with 110 depth is an existing network architecture that has three blocks (each has 36 convolutional layers) in the middle and another one convolutional layer at the beginning and one fully connected layer at the end [20]. We use published models trained (with 164 epochs) on CIFAR-100 [31] online[5]. We also train three models on ImageNet Tiny[6] with 300 epochs as target models (i.e., victim models during attacks). Models with the highest valid accuracy are used after training. We follow [38]'s methodology, and all training and test datasets are split to 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]https://pjreddie.com/darknet/imagenet/
[5]https://github.com/bearpaw/pytorch-classification
[6]https://tiny-imagenet.herokuapp.com/

4.2 Implementation and Evaluation Setup

We develop an implementation based on the Darknet [46] DNN library. We chose this particular library because of its high computational performance and small library dependencies which fits within the limited secure memory of the TEE. We 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, we focus on the performance of deep learning since secret provisioning only happens once for updating the model from severs. We implement 128-bit AES-GCM for on-device secure storage of sensitive layers. We test our implementation on a Hikey 960 board, a widely-used device [4, 8, 12, 58] 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 temperature, we decrease and fix the frequency of Cortex A73 to 903MHz and keep the frequency of Cortex A53 as 533Mhz. During experiments we introduce a 120 seconds system sleep per trial to make sure that the CPU temperature begins under $40°C$ 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 we 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 our 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, we utilized the REE's `/proc/self/status` for accessing the process information to measure the CPU execution time and memory usage of our 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 REE's kernel and (2) the time spent by the TEE (including both while 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 [53]. To overcome this, we trigger an abort 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, we used Monsoon High Voltage Power Monitor,[7] 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. We configured it to power the Hikey board using the required 12V voltage while recording the consumed current in a $50Hz$ sampling rate.

For conducting the MIA, we 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 [43] is used as the DNN library.

4.3 Measuring Privacy in MIAs

We define the adversarial strategy in our setting based on state-of-the-art white-box MIAs which observe the behavior of all components of the DNN model [38]. White-box MIAs can achieve higher accuracy of 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 [48, 57]. 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.

We evaluate the membership information exposure of a set of the target model's layers by employing the white-box MIA [38] 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, 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 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 (we refer the interested readers to [38] for a detailed description of this MIA). We 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, we conduct this MIA on our target model in two different settings: (i) starting from the first layer, we add the later layers one by one until the end of the network, and (ii) starting from the last layer we add the previous layers 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 inference and fine-tuning since layers' gradients can be calculated based on the cost function. Therefore, in setting (i), we utilize activation, gradients, and outputs. In setting (ii), we only use the activation of each layer 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.

[7]https://www.msoon.com/

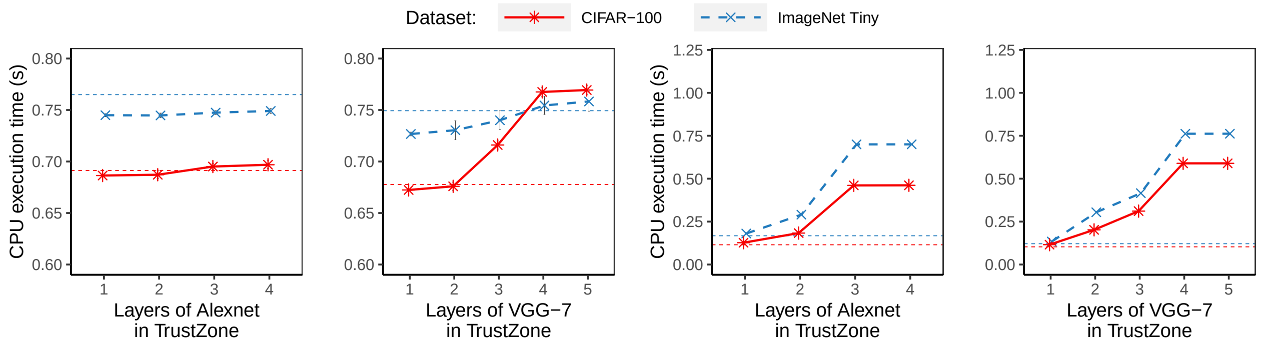
5 EVALUATION RESULTS

In this Section we first evaluate the efficiency of DarkneTZ when protecting a set of layers in the TrustZone to answer **RQ1**. To evaluate system efficiency, we measure **CPU execution time**, **memory usage**, and **power consumption** of our implementation for both training and inference on AlexNet and VGG-7 trained on two datasets. We protect the last layers (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 cost function). Starting from the last layer, we include the maximum number of layers that the TrustZone can hold. To answer **RQ2**, we use the **MIA success rate**, indicating the membership probability of target data (the more DarkneTZ limits this, the stronger the privacy guarantees are). We demonstrate the effect on performance and discuss the trade-off between performance and privacy using MIAs as one example.

5.1 CPU Execution Time

As shown in Figure 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 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: $F_{(6,133)} = 37.52, p < 0.001$; $F_{(8,171)} = 28.5, p < 0.001$). We also observe that protecting only the last layer in the TrustZone has 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 is also significant (see Figure 2b). It only increases CPU time by around 3% when protecting only the last layer which can increase up to 10× when the maximum possible number of layers is included in the TrustZone.

To further investigate the increasing CPU execution time effect, we 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.



(a) CPU time of training **(b) CPU time of inference**

Figure 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 (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 (⋯⋯ and ⋯⋯) 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).

As shown in Figure 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 [2] 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 3a) and when putting the dropout layer and the maxpooling layer for VGG-7 (see Figure 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 3c) and VGG-7 (see Figure 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 3c and Figure 3d where all increases happen when one trainable layer is included in the TrustZone. Since we 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 also shows that most of the increased CPU execution time (from ∼0.1s to ∼0.6s) 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.

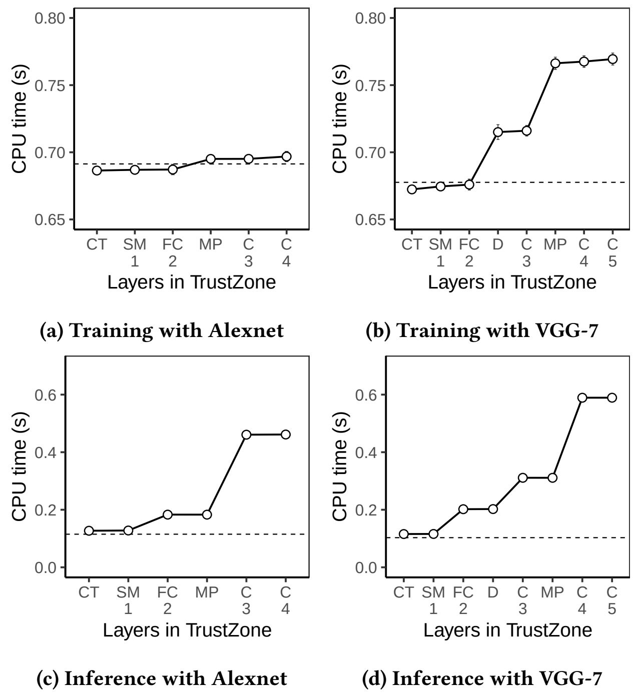
During *training*, the main reason for the increased CPU time is that protecting non-trainable layers in the TrustZone results in an additional transmission of their previous trainable layers from the REE to the TrustZone. Non-trainable layers (i.e., dropout and max-pooling layers) are processed using 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 3.4), we also move the previous convolutional layer to the TrustZone, which results in the turning points of the training are one layer in front of the turning points during inference. Therefore, in practical applications, we 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.

5.2 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 5a). *Inference* with TrustZone uses less memory (in the REE) (see Figure 5b) but there is still no difference when more layers are placed into TrustZone. Memory usage (in the REE) *decreases* since layers are moved to TrustZone and occupy secure memory instead. We 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 run-time 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.

5.3 Power Consumption

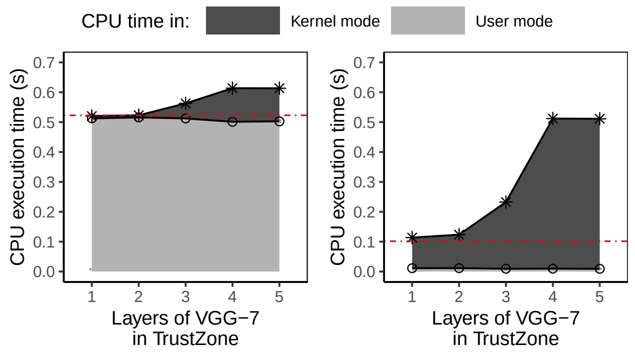
For *training*, the power consumption significantly decreases ($p < 0.001$) when more layers are moved inside TrustZone (see Figure 5c). In contrast, the power consumption during *inference* significantly increases ($p < 0.001$) as shown in Figure 5d. In both training and



(a) Training with Alexnet (b) Training with VGG-7

(c) Inference with Alexnet (d) Inference with VGG-7

Figure 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 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).



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

Figure 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 4b is too small to be shown).

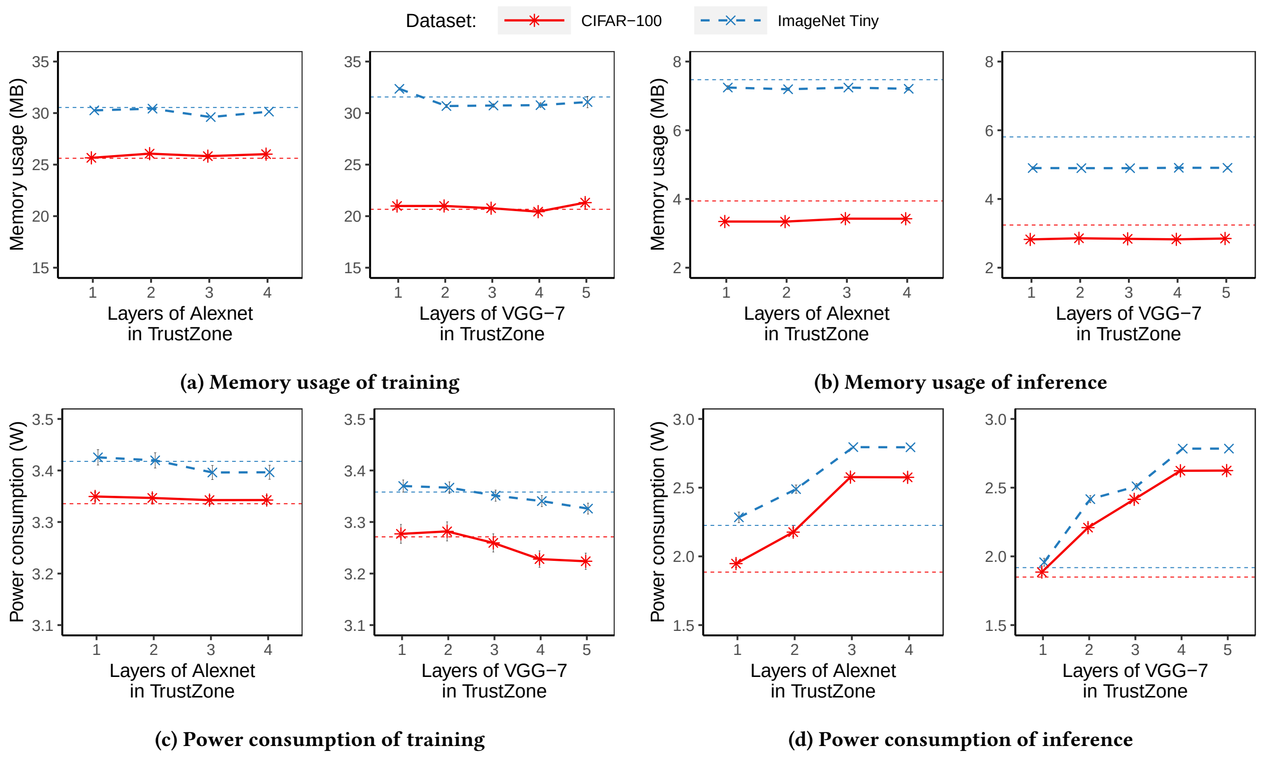
inference settings, the trend of power consumption is likely related to the change of CPU time (see Figure 2). More specifically, trajectories of them in figures have the same turning points (i.e., decreases or increases when moving the same layer to the TEE). One reason for the increased power consumption 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. We also test the performance of DarkneTZ on several models trained on ImageNet when protecting the last layer only, including the 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 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 model's 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 preliminary load of the TA, in practical applications.

5.4 Privacy

We conduct the white-box MIA (Section 4.3) on all target models (see Section 4.1 for the choice of models) to analyze the privacy risk while protecting several layers in the TrustZone. We used the standard *precision* and *recall* metrics, similar to previous works [48]. In our context, precision is the fraction of records that an attacker infers as being members, that are indeed members in the training set. Recall is the fraction of training records that had been identified correctly as members. The performance for both models and MIAs are shown in Table 1. Figure 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 we launch the attack. The configurations with zero layers



(a) Memory usage of training

(b) Memory usage of inference

(c) Power consumption of training

(d) Power consumption of inference

Figure 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 (······ and ······) represent the baseline where all layers are outside the TrustZone. 20 times for each trial, error bars are 95% CI).

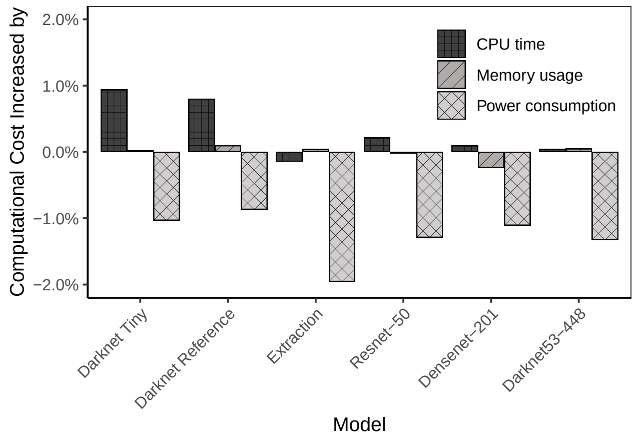


Figure 6: Performance on protecting the last layer of models trained on ImageNet in TrustZone for inference (Note: 20 times per trial; error bars are too small to be visible in the plot).

Table 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%

protected correspond to DarkneTZ being disabled (i.e., with our defense disabled). In particular, we measure the MIA adversary's success following two main configuration settings of DarkneTZ. In the first setting, we 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 we do the opposite: we start from the last layer and keep adding previous layers in TrustZone for each configuration. Our results show that when protecting the first layers in TrustZone, the attack success precision does not change 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 ~50% (random guessing)

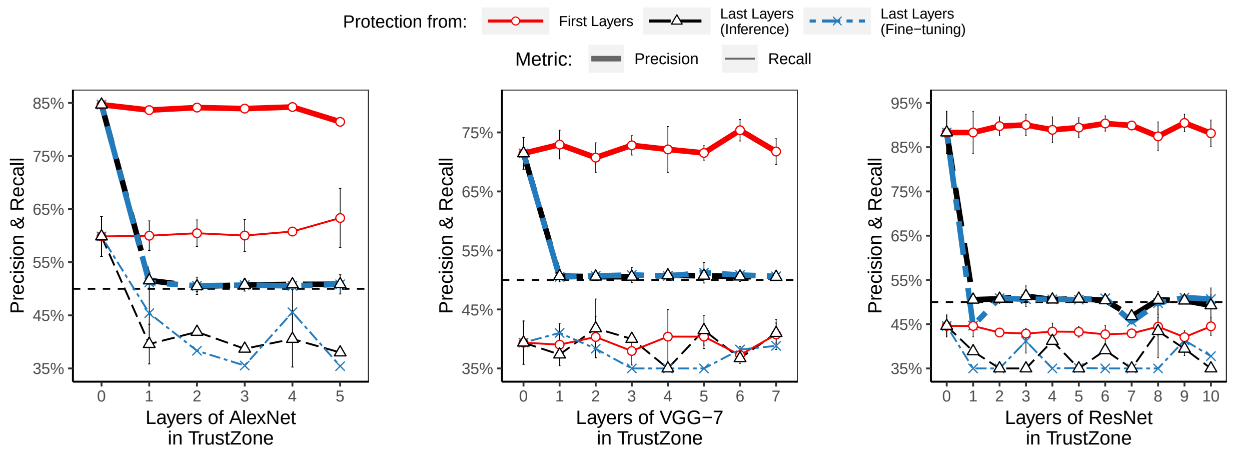


Figure 7: Precision and recall of white-box membership inference attacks when first or last layers of the model, trained on CIFAR-100, are protected using TrustZone. (Note: For first layer protection, 1 refers to the first layer, 2 refers to the first and the second layer, etc. For last layer protection, 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 error bars are 95% CI).

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 ~50% when we 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 [38] and [48], the attack success precision is influenced by the size of the attackers' training dataset. We 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 our 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%.

We also evaluated the privacy risk when DarkneTZ protects the model's outputs in TrustZone by *normalizing* it before outputting prediction results. In this configuration we 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 1) to MIAs [38]. 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 8 all three methods can significantly ($p < 0.001$) decrease the attack success performance to around 50% (i.e., random guess). Therefore, *we found that it is highly practical to use DarkneTZ to tackle MIAs: it incurs low resource consumption cost while achieving high privacy guarantees.*

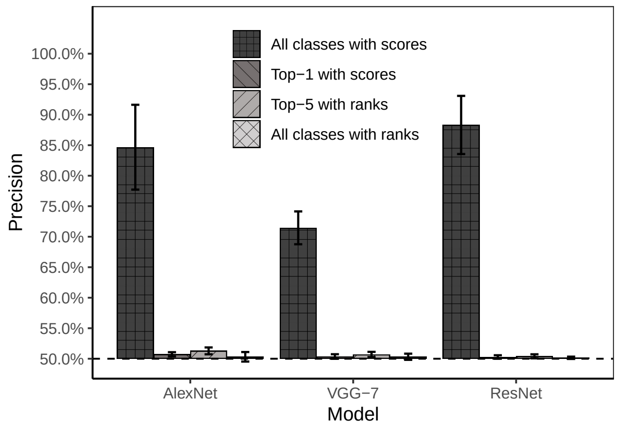


Figure 8: Precision of white-box membership inference attacks on models trained on CIFAR-100 when only outputs are protected using TrustZone (Dashed lines at 50% represent baselines (i.e., random guess). 5 times for each trial, and error bars are 95% CI).

6 DISCUSSION

6.1 System Performance

Effects of the model size. We 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 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. We have also examined the effect of different hardware: we executed our implementation of DarkneTZ with similar model sizes on a Raspberry Pi 3 Model B (RPi3B) and found it to have a lower rate of increase of cost (i.e., lower overhead) than when executed on the Hikey board [36]. 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, our 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. We tested a TEE with 5MiB secure memory on a Grapeboard[8]: only 1,000 neurons (corresponding 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 [18] and quantization [27, 52], could be one way to facilitate including more layers in the TEE. Lastly, we 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, our results show that utilizing TrustZone to protect outputs of large DNN models is effective and highly efficient.

Extrapolating for other mobile-friendly models. We have used Tiny Darknet and Darknet Reference for testing DarkneTZ's performance on mobile-friendly models (for ImageNet classification). Another widely-used DNNs on mobile devices, Squeezenet [26] and Mobilenet [22], define new types of convolutional layers are not supported in Darknet framework currently. We expect these to 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)[9].

Improving performance. Modern mobile devices usually are equipped with GPU or specialized processors for deep learning such as NPU. Our current implementation only uses the CPU but can be extended to utilizing 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 transfer all linear layers to a GPU, Tramer et al. [51] have obtained 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.

6.2 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 [16, 56] or GAN attack [21] as they also highly depend 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 our 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 [29] 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 for a compromised OS. DarkneTZ also protects layers while training models and conducting inference. 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. [34]'s work successfully shows private (e.g., membership) information about participants' training data using their updated gradients. Recent research [67] further reveals that it is possible to recover images and texts from gradients in pixel-level and token-level, respectively, and the last layers have a low loss for the recovery. By using DarkneTZ to limit information exposure of layers, this type of attack could be weakened.

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 [61]. 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 ($\epsilon = 2$) [1, 5]. The drop increases to around 20% for large level noises when training on CIFAR-10 [1]. To obtain considerable 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 [29]. Compared to the inevitable decreasing utility of adding noise, DarkneTZ achieves a better trade-off between privacy and utility compared to differential privacy.

7 CONCLUSION

We demonstrated a technique to improve model privacy for a deployed, pre-trained DNN model using on-device Trusted Execution Environment (TrustZone). We 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. We analyzed the performance of our 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. Our 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

[8]https://www.grapeboard.com/
[9]https://github.com/albanie/convnet-burden and https://pjreddie.com/darknet/tiny-darknet/

available secure memory to protect the maximum number of layers (depending on the model size and configuration) that the TEE can hold. We believe that *DarkneTZ* is a step towards stronger privacy protection and high model utility, without significant overhead in local computing resources.

ACKNOWLEDGMENTS

We acknowledge the constructive feedback from the anonymous reviewers. Katevas and Haddadi were partially supported by the EPSRC Databox and DADA grants (EP/N028260/1, EP/R03351X/1). This research was also funded by a gift from Huawei Technologies, a generous scholarship from the Chinese Scholarship Council, and a hardware gift from Arm.

REFERENCES

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

[2] Hervé Abdi and Lynne J Williams. 2010. Tukey's honestly significant difference (HSD) test. *Encyclopedia of Research Design. Thousand Oaks, CA: Sage* (2010), 1–5.

[3] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 79.

[4] Francis Akowuah, Amit Ahlawat, and Wenliang Du. 2018. Protecting Sensitive Data in Android SQLite Databases Using TrustZone. In *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, 227–233.

[5] Galen Andrew, Steve Chien, and Nicolas Papernot. 2019. TensorFlow Privacy. https://github.com/tensorflow/privacy

[6] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345.

[7] A Arm. 2009. Security technology-building a secure system using TrustZone technology. *ARM Technical White Paper* (2009).

[8] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with user-space enclaves.. In *Network and Distributed Systems Security (NDSS) Symposium 2019*.

[9] Rich Caruana, Steve Lawrence, and C Lee Giles. 2001. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems*. 402–408.

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. Ieee, 248–255.

[12] Pan Dong, Alan Burns, Zhe Jiang, and Xiangke Liao. 2018. 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)*. IEEE, 59–64.

[13] Alexey Dosovitskiy and Thomas Brox. 2016. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4829–4837.

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

[15] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. 2014. The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy* 12, 4 (2014), 29–37.

[16] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1322–1333.

[17] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. YerbaBuena: Securing Deep Learning Inference Data via Enclave-based Ternary Model Partitioning. *arXiv preprint arXiv:1807.00969* (2018).

[18] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1510.00149

[19] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. 2018. Mlcapsule: Guarded offline deployment of machine learning as a service. *arXiv preprint arXiv:1808.00590* (2018).

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 770–778.

[21] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 603–618.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 4700–4708.

[24] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving Machine Learning as a Service. *arXiv preprint arXiv:1803.05961* (2018).

[25] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689* (2018).

[26] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[27] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[28] Bargav Jayaraman and David Evans. 2019. Evaluating Differentially Private Machine Learning in Practice. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1895–1912. https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman

[29] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. 2019. MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 259–274.

[30] Hugo Krawczyk. 2003. SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Annual International Cryptology Conference*. Springer, 400–425.

[31] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n.d.]. CIFAR-100 (Canadian Institute for Advanced Research). http://www.cs.toronto.edu/~kriz/cifar.html

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[33] Ninghui Li, Wahbeh Qardaji, Dong Su, Yi Wu, and Weining Yang. 2013. Membership privacy: a unifying framework for privacy definitions. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*. ACM, 889–900.

[34] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *Proceedings of 40th IEEE Symposium on Security & Privacy*. IEEE, 480–495.

[35] Ilya Mironov. 2017. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 263–275.

[36] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Andrea Cavallaro, and Hamed Haddadi. 2019. Poster: Towards Characterizing and Limiting Information Exposure in DNN Layers. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2653–2655.

[37] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 113–124.

[38] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks. In *Proceedings of 40th IEEE Symposium on Security & Privacy*. IEEE.

[39] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[40] Seyed Ali Osia, Ali Shahin Shamsabadi, Ali Taheri, Kleomenis Katevas, Sina Sajadmanesh, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. 2020. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of Things Journal* (2020).

[41] Seyed Ali Osia, Ali Shahin Shamsabadi, Ali Taheri, Hamid R Rabiee, and Hamed Haddadi. 2018. Private and Scalable Personal Data Analytics Using Hybrid Edge-to-Cloud Deep Learning. *Computer* 51, 5 (2018), 42–49.

[42] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. StreamBox-TZ: secure stream analytics at the edge with TrustZone. In *2019 {USENIX} Annual Technical Conference 19*. 537–554.

[43] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[44] Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. 2018. Downsampling leads to Image Memorization in Convolutional Autoencoders. *arXiv preprint arXiv:1810.10333* (2018).

[45] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. 2018. Membership Inference Attack against Differentially Private Deep Learning Model. *Transactions on Data Privacy* 11, 1 (2018), 61–79.

[46] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[47] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. 2018. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. arXiv preprint arXiv:1806.01246. In *Network and Distributed Systems Security (NDSS) Symposium 2018*. https://arxiv.org/abs/1806.01246

[48] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proceedings of 38th IEEE Symposium on Security & Privacy*. IEEE, 3–18.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1–9.

[50] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. 2018. Privado: Practical and secure DNN inference. *arXiv preprint arXiv:1810.00602* (2018).

[51] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. arXiv preprint arXiv:1806.03287. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1806.03287

[52] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 8612–8620.

[53] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. 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*. ACM, 2421–2434.

[54] Chugui Xu, Ju Ren, Deyu Zhang, Yaoxue Zhang, Zhan Qin, and Kui Ren. 2019. GANobfuscator: Mitigating information leakage under GAN via differential privacy. *IEEE Transactions on Information Forensics and Security* 14, 9 (2019), 2358–2371.

[55] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*. 2125–2136.

[56] Ziqi Yang, Jiyi Zhang, Ee-Chien Chang, and Zhenkai Liang. 2019. Neural Network Inversion in Adversarial Setting via Background Knowledge Alignment. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 225–240.

[57] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 268–282.

[58] Kailiang Ying, Amit Ahlawat, Bilal Alsharifi, Yuexin Jiang, Priyank Thavai, and Wenliang Du. 2018. TruZ-Droid: Integrating TrustZone with mobile operating system. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 14–27.

[59] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in Neural Information Processing Systems*. 3320–3328.

[60] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579. In *Deep Learning Workshop in International Conference on Machine Learning*. https://arxiv.org/abs/1506.06579

[61] Lei Yu, Ling Liu, Calton Pu, Mehmet Emre Gursoy, and Stacey Truex. 2019. Differentially private model publishing for deep learning. In *Proceedings of 40th IEEE Symposium on Security & Privacy*. IEEE, 332–349.

[62] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.

[63] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1611.03530

[64] C. Zhang, P. Patras, and H. Haddadi. 2019. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys Tutorials* 21, 3 (2019), 2224–2287.

[65] Shengjia Zhao, Jiaming Song, and Stefano Ermon. 2017. Learning hierarchical features from deep generative models. In *International Conference on Machine Learning*. 4091–4099.

[66] Shijun Zhao, Qianying Zhang, Yu Qin, Wei Feng, and Dengguo Feng. 2019. SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1723–1740.

[67] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*. 14747–14756.

[68] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. Scottsdale, Arizona. https://arxiv.org/abs/1407.6981