HYBRIDTEE: SECURE AND PRIVACY-PRESERVING MOBILE DNN EXECUTION USING HYBRID TRUSTED EXECUTION ENVIRONMENT

by

AKSHAY GANGAL

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Sheng Wei

and approved by

New Brunswick, New Jersey

October, 2020

ABSTRACT OF THE THESIS

HybridTEE: Secure and Privacy-Preserving Mobile DNN Execution Using Hybrid Trusted Execution Environment

By Akshay Gangal

Thesis Director:

Sheng Wei

Deep neural networks (DNNs) have been increasingly adopted in many mobile applications involving security/privacy sensitive data and inference models. Therefore, there is an urgent demand for security and privacy protection of DNN execution on mobile devices. Catering to this demand, hardware-based trusted execution environments (TEEs), such as ARM TrustZone, have recently been considered for secure mobile DNN execution. However, none of the existing attempts of running DNN in TrustZone have been successful due to the stringent resource and performance limitations posed by the mobile TEE. We develop *HybridTEE*, a novel hardware-based security framework to securely execute DNN in the resource-constrained local TEE (i.e., ARM TrustZone), by offloading a part of the DNN model to a resource-rich remote TEE (i.e., Intel SGX). The key design of *HybridTEE* is two-fold. First, it strategically divides the DNN model into privacy-aware local (TrustZone) and remote (SGX) partitions by employing two privacy-oriented metrics based on object recognition and Scale Invariant Feature Transform (SIFT). Second, it builds a trustworthy communication channel bridging TrustZone and SGX to enable secure offloading of the DNN model between the two TEEs. Our evaluations based on a prototype implementation of HybridTEE and 4 popular DNN models indicate enhanced security and a 1.75x - 3.5x speedup compared to mobile-only DNN execution without TEE.

Acknowledgements

I would like to take this opportunity to thank my advisor Dr. Sheng Wei for his guidance, motivation and insightful solutions that paved the way for an innovative idea. I also appreciate Dr. Wei for being a constructive critique of my research. I would also like to express my gratitude towards Mengmei Ye, Ke Xia and Xianglong Feng for their valuable suggestions leading to productive discussions. A special thanks for Dr. Yingying Chen and Dr. Saman Zonouz for taking time out from their busy schedule for being part of my thesis committee. Last but not the least, I would like to thank my family and friends for their continued support and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep neural networks (DNNs) have been widely adopted in various mobile applications to accomplish critical inference tasks, such as healthcare [16], authentication [32, 9], and computer vision [13, 27]. Many of these applications interact with security or privacy sensitive data that require protection, such as the biometrics used by an authentication app and the medical information involved in a healthcare app. In addition, the confidentiality and integrity of the mobile DNN model itself are subject to a variety of security threats [17, 33, 10] that must be addressed.

Recently, hardware-based trusted execution environments (TEEs), such as ARM TrustZone [1], have been developed to address mobile security challenges in general. ARM TrustZone can create a hardware-isolated secure world to seal the sensitive data, which remains secure even if the operating system (OS) has been compromised by an attacker. Such exclusion of OS from the trusted computing base (TCB) makes TrustZone a strong hardware security primitive and a viable option for securing mobile DNN executions.

However, in reality, it is very challenging to execute DNN in TrustZone due to the huge gap between the limited computing resources (e.g., memory space or hardware/software accelerators) in the TrustZone secure world and the high demand of the DNN models that are both data and computation intensive. Although it is physically feasible to deploy more resources into the secure world, doing so is directly against the security principle of maintaining a small TCB and would result in an increased level of exploitable security vulnerabilities. In fact, due to this fundamental challenge, the existing efforts in the community attempting to execute DNN in TrustZone have not been able to reach satisfactory solutions. Figure 1.1 summarizes the existing categories of research efforts targeting mobile DNN execution. For example, several works [31, 22] proposed to partition the DNN model and run all the partitions sequentially within the limited memory space of TrustZone. Although they could achieve the desired security, the resulting timing overhead is huge even for simple DNN models (i.e., the *high-security, low-performance* category). Another approach [7] proposed to partition the DNN model into a small secure component which runs in the local device TEE and a large non-secure component that runs on a remote non-TEE server. However, the non-secure portion of the model often leaves traceable information for the attacker to reverse engineer and compromise the security of the entire DNN model (i.e., the *medium-security, high-performance* category). In addition, the conventional approaches run the DNN model in non-TEE environments either with (i.e., [14, 26]) or without offloading (i.e., the *baseline* approach). These approaches would achieve *low security* without the protection provided by TEE.

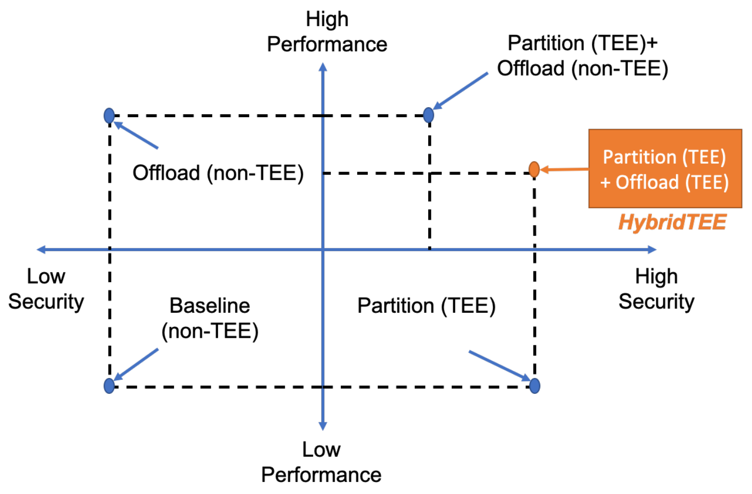


Figure 1.1: Illustration of the novel contribution made by *HybridTEE* compared to the existing works on mobile DNN execution. The existing works contain 4 main categories: (1) Baseline (non-TEE); (2) Offload (non-TEE) [14, 26]; (3) Partition (TEE) [31, 22]; and (4) Partition (TEE) + Offload (non-TEE) [7].

To address the security and performance challenges in the existing approaches of DNN execution in TrustZone, we propose to adhere to two design principles for an effective defense approach. First, for the consideration of security, *the entire DNN model must remain in TEE.* Second, to boost the performance of DNN execution in TEE, a second heterogeneous TEE (e.g., Intel SGX [20]) may be leveraged to compensate for the limited computation resources in TrustZone. The two design principles combined lead to our proposed solution *HybridTEE*, which offloads a strategically partitioned part of the DNN to a remote SGX enclave. In this way, *HybridTEE* ensures the security of the DNN model as it is entirely executed in TEEs. Also, it has the potential of significantly improving the performance given the higher amount of computation resources and capacity in the remote SGX server. Figure 1.1 illustrates the novel contribution made by *HybridTEE* compared to the existing approaches. In particular, *HybridTEE* creates a new category of approach for secure mobile DNN execution, namely *high security, medium performance*, which aims to meet the high security requirement in the sensitive applications while providing acceptable performance.

HybridTEE involves two key technical components with novel research contributions to realize the aforementioned design objectives. *First*, we develop a novel privacy-preserving partitioning algorithm to eliminate the potential privacy concerns while offloading the data and model from the local mobile device to the remote server. *Second*, we develop a *HybridTEE* framework by bridging the two singular TEEs, namely TrustZone and SGX, via a cryptography-based secure handshake. *HybridTEE* ensures that there is no data or model exposed in the clear during the secure offloading and execution of the DNN model across the two TEEs.

The key design challenge in the partitioning algorithm is to determine the particular DNN layer, namely the *partition point*, after which the data involved in the DNN model would contain least amount of identifiable information about the original inputs and thus can be offloaded to the remote server without privacy concerns. We develop two partitioning algorithms to determine such *partition point* for a given DNN model: (1) We employ an auxiliary DNN model to determine the confidence level of the current layer output being successfully identified as the original input (i.e., a confidence score), and we select the layer that has a sufficiently low confidence score as the *partition point* to prevent the adversary from directly identifying privacy-sensitive information. (2) We adopt a scale invariant feature transform (SIFT)-based method to match the keypoints between the input and output images at each layer of the DNN model, and we select the layer that causes significant mismatch in the key points as the *partition point*, to prevent the adversary from inferring and reconstructing the original input.

We evaluate *HybridTEE* on real system implementations of the two singular TEEs, including the TrustZone using OP-TEE [15] on a Raspberry Pi and the SGX on a server. Our evaluation results on 4 popular DNN models indicate that *HybridTEE* enables TEE-based secure DNN execution with a 1.75x-3.5x performance speedup compared to mobile-only DNN execution without TEE.

Chapter 2

Background

2.1 Deep Neural Networks

A deep neural network (DNN) belongs to the broad category of artificial neural networks, which are used for representation learning. DNN uses multiple layers to predict several high-level features from the raw input data [2, 25]. It has multiple hidden layers between the input and output. Each layer contains multiple neurons, and the connections between the neurons are associated with weight vectors. The main component of DNN inference phase is the feed forward network, which maps a given input to its respective prediction label or class. Figure 2.1 illustrates a sample DNN for image classification [2].

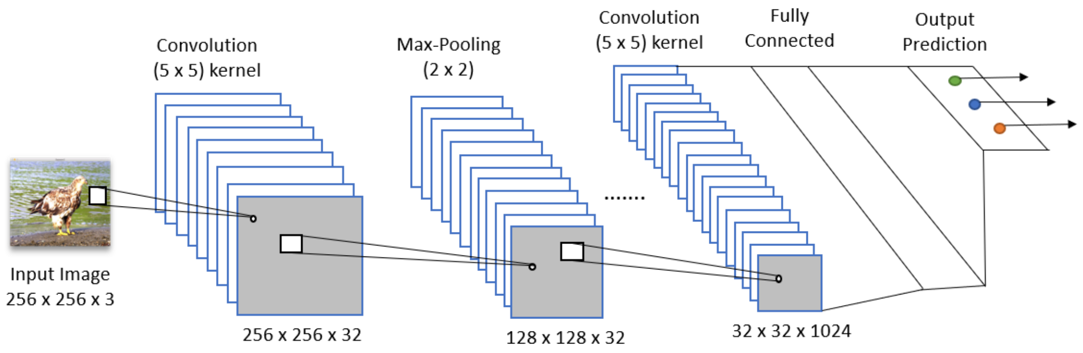


Figure 2.1: Deep Neural Network for image classification [2].

A feed forward network can be represented as a function $f_\theta : X \rightarrow Y$, where X is a high dimensional vector and Y is a set of class labels for prediction. For a DNN network f_θ with input X_k, the feed forward network predicts the output Y_k after a series of sequential computations $f_n f_{n-1} ... f_1(X_k)$, across multiple hidden layers with weight vectors $W_{i=1}^{N}$ and activation function A_k, where k is the sample input and N is the number of neurons in the corresponding hidden layer. The top k entries are extracted from Y_k and mapped to a set of class labels L.

2.2 Trusted Execution Environment

The security community has developed and deployed hardware-based *trusted execution environment* (TEE) to protect sensitive computations and data [1, 20]. A TEE provides isolated execution of the application along with confidentiality and integrity for the code and data, which cannot be exploited even if the OS has been compromised. This excludes the OS from the trusted computing base (TCB) of the system and significantly reduces the possible attack surface. A TEE's ability to provide safe execution of authorized software is known as trusted application (TA). The contents of the TA are isolated from the untrusted components of the system, and also from the other TAs running within the TEE. Any modification to a TA can be performed only by an authenticated entity. The TEE also provides secure storage where all the TAs code/data will be stored in a protected memory block. In our proposed *HybridTEE* framework, we primarily focus on the combination of two TEEs: ARM TrustZone [1] for mobile devices and Intel SGX [20] for server/PC platforms.

2.2.1 ARM TrustZone

ARM TrustZone [1] is a hardware isolation technology present in ARM based devices. TrustZone provides a TEE which contains three main components: secure world, normal world, and secure monitor, on a single processor core. The secure world can access the resources in the normal world, but the normal world cannot directly access any resources in the secure world. The secure monitor takes responsibility of context switching between the two worlds. Access to the secure monitor is triggered by executing a dedicated instruction, the Secure Monitor Call (SMC) instruction, or by a subset of the hardware exception mechanisms. TrustZone provides isolation using an extended AXI bus design with an additional NS (non-secure) bit, which is a control signal for read and write accesses on the main system bus. Figure 2.2 shows the architecture of ARM TrustZone with secure and unsecure regions [1].

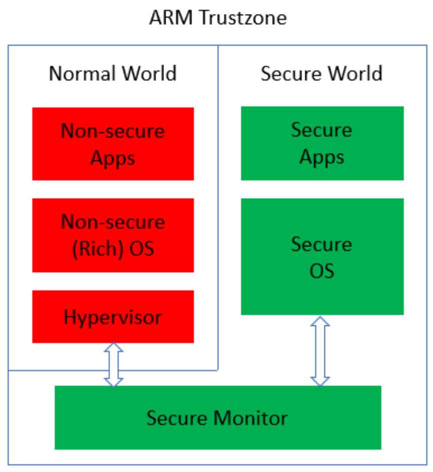


Figure 2.2: ARM TrustZone architecture [1].

2.2.2 Intel SGX

Intel SGX [20] is a security extension on the traditional Intel architecture that provides confidentiality and integrity protection for the sensitive user data. It provides security by encryption of a portion of the memory by the CPU. In particular, SGX protects the confidentiality and integrity of the data in the secure container, also known as the *enclave*, by isolating the enclave's code and data from the untrusted components, including the system OS and hypervisor. The processor sets aside a region of memory called *Protected Reserved Memory (PRM)* which contains the Enclave Page Cache (EPC) to store the enclave code and data. Each page in the EPC belongs to exactly one enclave. The state of all EPC pages is maintained in the Enclave Page Cache Map (EPCM). When an enclave is loaded, the CPU computes a cryptographic hash of its contents. During the enclave initialization and before running any applications, this hash acts as a verification or attestation metric to validate the status of the enclave. The communication between the enclaves and the untrusted applications is performed using enclave calls (i.e., ecalls) and outside calls (i.e., ocalls). The communication channels are also encrypted to guarantee the confidentiality of the data. SGX also involves remote attestation, where the *enclave* establishes a trust relationship with a remote third party service provider. Figure 2.3 describes the Intel SGX architecture [20].

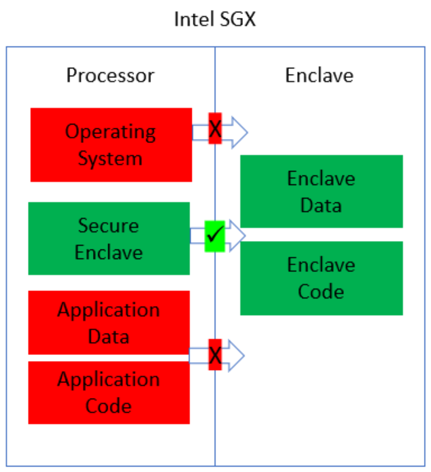


Figure 2.3: Intel SGX architecture [20].

2.3 DNN Offloading

DNNs have been widely deployed in mobile or IoT-based devices for computation-intensive AI applications [8, 29, 32, 9, 13, 27, 16], which often overwhelm the limited computation capacities of the mobile devices, significantly affecting the performance. To address this issue at the system level, recent studies have adopted DNN offloading as a means to execute the mobile DNN applications under the resource constraints [14, 26].

Consider the case that the user needs to run N applications on a mobile device. Each of these applications needs to perform a prediction task using a DNN inference model. Let the total amount of resources on the device be R and each model needs r resources for running the inference. If $\sum_{i=1}^{k} r_i \geq R$, then the device cannot run all inference models at the same time, causing a performance bottleneck. The solution that is widely deployed to address this problem is to find an application f_k in which resource usage $r_k \geq \mathcal{T}$, where $\mathcal{T}$ is the threshold, and offload the computation-intensive part of f_k to a resource-rich cloud platform [34]. Some of the previously suggested methods use a cost-driven strategy to offload the DNN computation at run time [14], where the profiling on the mobile device is performed by monitoring the CPU utilization rate. While offloading is a plausible way to enable efficient execution of DNNs on mobile devices, various design challenges must be addressed such as the content and timing of the offloading, as well as the potential performance degradation caused by the round-trip communication. In the design of *HybridTEE*, we explore and address these design challenges under the unique security context.

2.4 Remote Attestation

Remote attestation is a method used to establish handshake between the service provider and an enclave in the remote server [28, 5, 11]. This process enables the service provider to authenticate and build the trust relationship with the remote enclave, which can be used to address various trust problems including guaranteed invocation of software on the server, delivery of privacy sensitive content to trusted clients, detection of malicious unauthorized access to the service provider, etc.

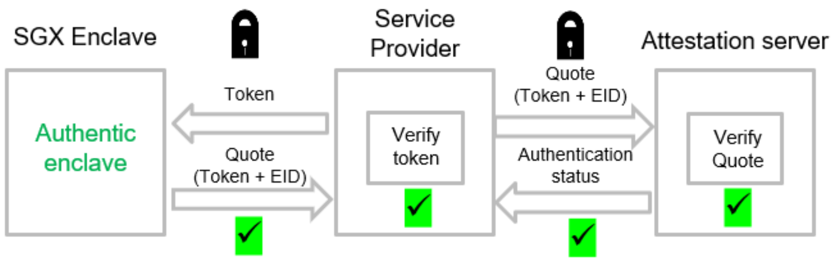


Figure 2.4: Remote attestation handshake between a service provider and an authentic enclave. A Quote is a combination of token and Enclave ID signed by the enclave [11].

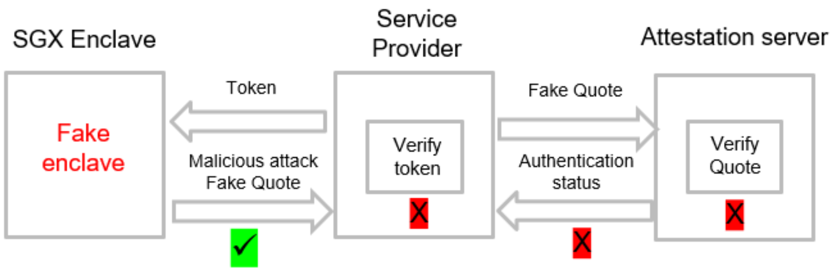


Figure 2.5: Remote attestation blocking a fake enclave from getting unauthorized access to the service provider [11].

In our work, we consider Intel SGX as the remote server and ARM TrustZone as the service provider. Figure 2.4 shows the normal scenario where an authentic enclave is attested by the service provider. A quote signed using a private key by the enclave is verified by the attestation server [11]. Also, a random token is used by the service provider to prevent replay attacks [19]. Figure 2.5 illustrates the case of a fake enclave or adversary trying to establish connection with the service provider [11]. As part of the remote attestation process, the attestation server will detect this malicious quote and notify the service provider about the discrepancy. If the attestation is unsuccessful, the handshake process is terminated.

Chapter 3

Threat Models

We assume the goals of the adversary are to (1) gain access to the sensitive input data that is fed into the DNN, (2) reconstruct the model by using confidential information obtained during the model execution, (3) uncover the final prediction results generated after the classification process, and (4) modify the input or model proprietary information, that can alter the final prediction outcome. In this work, we aim to leverage the strong security features and performance improvements brought by *HybridTEE* to address all the above confidentiality and integrity threat models.

On the other hand, we assume that both ARM TrustZone and Intel SGX are not compromised by the attackers. In other words, we do not consider side channel attacks or hardware physical attacks that could compromise the trusted execution environments. Also, we assume that the DNN training for the model has been done in a secure environment and the adversary does not have any information about the training process or the model proprietary information prior to the application deployment.

In addition to the security threat models, we also consider a privacy threat model, in which the users are concerned about their private images (e.g., the input images of the DNN model) being offloaded to a remote server and accessed by the non-malicious service provider. In the scenario of offloading, the privacy threat model determines that the input images, either in the original form or with downgraded but still identifiable quality, should not be offloaded to the remote SGX enclave, even if the enclave remains intact without being compromised. This creates a significant challenge to the design of the offloading mechanism in *HybridTEE*, which we aim to address with the privacy-aware partitioning algorithm presented in Chapter 5.

Chapter 4

Problem Definition: Mobile DNN Execution

A pre-trained DNN model contains proprietary information such as model runtime configuration parameters and weights. Exposure of this information to an adversary can leak sensitive information about the intellectual property of the model. Also, input data that is fed into the model and the final prediction results are security sensitive. In order to protect the intellectual property, input data and prediction results, it is desirable to execute the DNN models in a hardware-based TEE [31, 22].

However, the design principle of TEEs is to protect the sensitive code/data using small Trusted Computing Base (TCB) sizes [1, 20]. Such design principle minimizes the potential security vulnerabilities of the TEE, thus making it more challenging for attackers to compromise. This indicates that the amount of data or code that can be securely stored or run within the TEE is limited. On the other hand, the size of a DNN model is typically huge containing 3 major components: neural network layers, weight vectors, and input data. In particular, the weight vectors of popular DNNs are can be up to a few hundred megabytes, as summarized in Table 4.1. As a comparison, the memory capacity of a typical TrustZone implementation, such as OP-TEE [15] on Raspberry pi 3, is in the range of a few megabytes. The huge gap between the demand and supply creates the research problem we must address with the design of *HybridTEE*.

DNN model	Number of layers	Total weight vector size in MB
Darknet19	26	80
VGG-16	25	528
Resnet152	205	220
GoogLeNet	27	90

Table 4.1: Typical DNN model weight vector sizes in megabytes [24].

Let us consider the mathematical representation of the problem definition. Let $D(\theta)$ be the DNN model with total N layers. Let S_{l_i} be the size of the i^{th} layer. Let w_i be the weight vector at layer i with size S_{w_i}. Let $\mathbb{R}^{w \times h \times c}$ be the high dimensional input vector with length $S_{\mathbb{R}}$, and T be the TCB size of the TEE for the device. The entire DNN model can be run in the TrustZone secure world if and only if -

$$\sum_{i=1}^{N} S_{l_i} + \sum_{i=1}^{N} S_{w_i} + S_{\mathbb{R}} \leq T \tag{4.1}$$

However, the total size of the layers, weights and input data $S_t >> T$, since the DNN model's memory requirement is very high, and T is expected to be as small as possible. This creates a challenging problem that we aim to address with the proposed *HybridTEE*.

In a nutshell, *HybridTEE* offloads a part of the DNN model from the local TEE (TrustZone) to a remote TEE (SGX) to address the aforementioned demand & supply problem. The realization of *HybridTEE* involves two major components. (1) Algorithmic component: a privacy-aware DNN partitioning algorithm (discussed in Chapter 5) to determine the *partition point* between the sub-models at local and remote TEEs; and (2) System component: a system-level *HybridTEE* framework (discussed in Chapter 6) to securely bridge the two TEEs and jointly accomplish the secure DNN execution.

Chapter 5
Privacy-Aware DNN Partitioning

In our *HybridTEE* system we deploy a three-way sequential partitioning technique. The DNN model is divided into 3 partitions, namely *LocalNet(L1), RemoteNet(L2), and PredNet(L3)*, as shown in Figure 5.1. First, *LocalNet* or local partition contains the most sensitive layers of the network that are vulnerable to privacy exposure about the input data. The code and data of *LocalNet* and its feed forward function should reside in the *local TEE* for privacy protection. Second, *RemoteNet* or remote partition contains the next set of sequential layers in the network. It is stored and executed in the *remote TEE.* In order to maximize the overall performance of the application, *RemoteNet* should contain the most number of layers in the network, since the SGX server is a faster and resource rich device. However, due to privacy considerations, there is critical information in the DNN model that must stay in *LocalNet*, and the *partition point* for the transition from *LocalNet* to *RemoteNet* is determined by our privacy-aware partitioning algorithm that will be discussed next. Finally, *PredNet* or prediction partition contains the final layer of the DNN that performs the final prediction of the labels for the given input. This layer runs in the *local TEE*, considering the privacy sensitive nature of the prediction result for the user.

The mathematical representation of the partitioning approach is as follows. Consider a DNN model with feed forward function $f_\theta : \mathcal{X} \to \mathcal{Y}$. The network partitions for *LocalNet*, *RemoteNet* and *PredNet* can be formulated as $f_\theta = f_{\theta_l} \oplus f_{\theta_r} \oplus f_{\theta_p}$, where f_{θ_l}, f_{θ_r} and f_{θ_p} are the respective feed forward functions of *LocalNet, RemoteNet and PredNet*, and n be the total number of layers in the network. *LocalNet* obtains the high dimensional input vector $\mathbb{R}^{w \times h \times c}$ and generates the partial outputs in the form of $f_{\theta_l}(pp-1)$, where pp represents the *partition point.* These outputs are then encrypted using symmetric encryption before being sent to *RemoteNet*.

$$f_{\theta_l} = f(\mathbb{R}^{w \times h \times c}) \tag{5.1}$$

$$f^*_{\theta_l} = Enc(f_{\theta_l}(pp - 1)) \tag{5.2}$$

The encrypted outputs of *LocalNet* act as inputs to *RemoteNet*, which decrypts and computes the results up to the $(n-1)^{th}$ layer of the network.

$$f_{\theta_r} = Dec(f^*_{\theta_l}) \tag{5.3}$$

$$f^*_{\theta_r} = Enc(f_{\theta_r}(n - 1)) \tag{5.4}$$

The encrypted outputs from *RemoteNet* act as inputs to *PredNet* partition, which decrypts the partial outputs and performs the final prediction $\mathcal{Y}$ based on the class labels.

$$f_{\theta_p} = Dec(f^*_{\theta_r}) \tag{5.5}$$

$$\mathcal{Y} = f_\theta = Pred_{labels}(f_{\theta_p}) \tag{5.6}$$

While designing the partitioning algorithm, our key idea is to automatically detect the privacy exposure in each layer of the DNN execution and identify the last layer that still exposes privacy-sensitive information as the *partition point pp*. While there is no common standard of privacy defined for this domain-specific application, we consider the privacy of the user by regarding all the original input images as privacy-sensitive, and the intermediate inputs/outputs that could reveal the original input images should remain in *LocalNet* without being offloaded. Following this privacy model, we develop two systematic methods to find the optimal *partition point*. First, We conduct object detection at the intermediate layers to identify potential exposure of meaningful, identifiable information of the original input images (discussed in Chapter 5.1). Second, we employ the Scale Invariant Feature Transform (SIFT) method to detect localized keypoints in the intermediate inputs/outputs and quantify the potential privacy exposure (discussed in Chapter 5.2). In each method, the optimal *partition point* can be determined as the DNN layer that results in significantly low exposure of the original

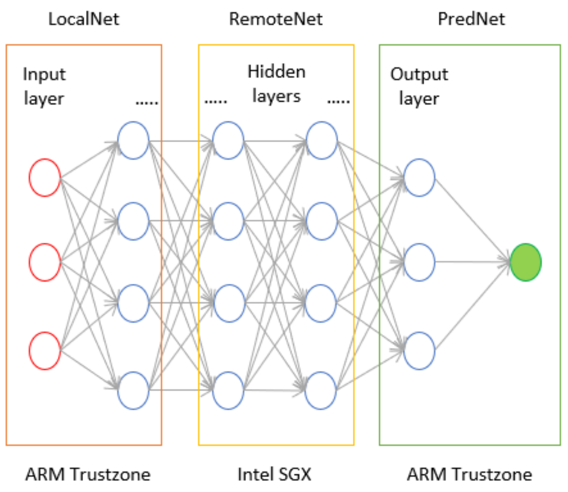


Figure 5.1: Three-way DNN partitioning with *LocalNet* and *PredNet* partitions running in ARM TrustZone and *RemoteNet* partition running in Intel SGX.

input images. Algorithm 1 describes the steps involved in determining the *partition point* using these two methods. Both partitioning methods are executed offline prior to deployment of the application on the user device.

5.1 Object Detection Using an Auxiliary DNN

The objective of the partition point detection algorithm is to find the last layer in the DNN model that is still susceptible to information exposure about the original input images. If the unencrypted input image is recovered, the adversary can gain critical and sensitive information about the user, thus invading the user's privacy. Based on this point of view, we develop an algorithm to determine the partition point by employing an auxiliary DNN to detect the objects in the intermediate input/output images. The idea is based on the assumption that the adversary is able to identify any object that is distinctly visible. Specifically, the auxiliary DNN acts as an adversary attempting to classify the key object in the intermediate images. We argue that if the auxiliary DNN is not able to classify the image with a certain degree of confidence, then the image is considered as free of privacy exposure for offloading.

In particular, we use YOLO [24] as the auxiliary DNN model, which is a state-of-the-art, real-time object detection mechanism. The YOLO model predicts the bounding boxes using dimension clusters, with 4 coordinates for each box. Then, each box predicts the classes that are related to the objects present in the box with a confidence score. We run the target DNN model offline and save the images of each channel in every layer. Then, we feed each image into the YOLO model to find its confidence score of object detection. We select the maximum confidence score among all channels as the confidence score in an individual layer, and we repeat this process over all the layers. Finally, we select the partition point based on one of the two criteria: (1) the layer achieves sufficiently low confidence score, or (2) the layer reaches the cumulative memory requirement for the local TEE. The first layer that fulfills either standard is then treated as the *partition point* for the DNN model.

Let f_θ be the DNN model function under consideration. Let $\{l_i\}_{i=1}^N$ be the layers of the DNN and $\{c_i\}_{i=1}^C$ be the number of channels in each layer. Therefore, $\{\{X_{cl}\}_{c=1}^C\}_{l=1}^N$ are the images of respective layers and channels. Let $\{\phi\}_{l=1}^N$ be the maximum confidence scores generated at each layer and l_{TCB} be the cutoff layer with memory usage below TCB threshold. Let η be the partition point of *localNet*. With $g_{\theta_l}(c) = f_\theta(l)(c)$ for all $1 \leq c \leq C$, we obtain the maximum confidence score of object detection among all channels in every layer i.e. ϕ_i.

$$\{\phi_i\}_{i=1}^N = \max_{1\leq c\leq C} g_{\theta_l}(X_c) \tag{5.7}$$

Then, we find the layer with the minimum confidence score among all the layers, and compare this value to the TCB cutoff layer l_{TCB}. We then select the minimum value between the two as the partition point η.

$$\eta = \min(\min_{1\leq i\leq N} \phi_i, l_{TCB}) \tag{5.8}$$

5.2 Scale Invariant Feature Transform

In addition to object detection, we also employ SIFT [18] as another means of identifying the privacy exposure at intermediate DNN layers. SIFT is a feature detection algorithm to detect local features or keypoints in images, which goes through four key steps: *feature detection, feature matching and indexing, cluster identification* and *model verification*. In *feature detection*, an image is transformed into a large collection of feature vectors that are agnostic to image scaling or rotation. The features are detected

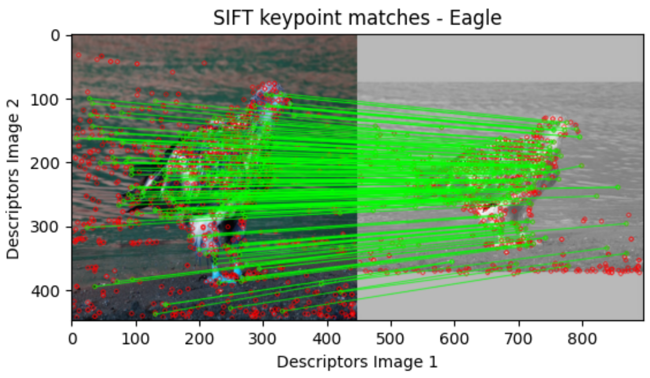


Figure 5.2: SIFT keypoint matching between two images. The image in the left half is the DNN input image, and the image in the right half is generated by the feed forward network at layer 0 channel 7. The green lines represent the keypoints matched.

using a staged filtering process to identify stable points. Each point is used to generate a feature vector that describes the local image region [18]. In order to index a new object in the image, the SIFT keys for similar images are stored in a database. In *cluster identification*, an object is recognized in the new image by comparing to the features in the original database. Finally, *model verification* uses least-squares method to determine the closest parameters from the projected model locations to the corresponding image locations [18].

In order to determine the similarity between two images, the keypoints of both images are computed using a keypoint descriptor. We now remove the unnecessary or approximate matches using the ratio test given by [18] and only select the matches with distance less than 0.75x of the feature distance in the original image. The number of matches obtained indicates the degree of similarity between the images. Figure 5.2 demonstrates the keypoints matched between the input image in the left half of the figure, and the output generated at layer 0 channel 7 in the right half. The red dots indicate the features in the respective images, and the green lines indicate the matches detected. The images have 221 matching keypoints.

Mathematically, let $\mathbb{R}1$ and $\mathbb{R}2$ be the high dimensional vectors representing the

Algorithm 1 Partition Point Detection

```
1: function GenerateImages(void)    ▷ Save the images of every channel and layer
2:     for $layer = 1, 2, \ldots, N$ do
3:         for $channel = 1, 2, \ldots, C$ do
4:             $image \leftarrow GetNetworkImage(layer, channel)$
5:             Save network image
6:         end for
7:     end for
8: end function
9: function Partition_AuxDnn(void)    ▷ Compute partition point using auxiliary DNN
10:     for $layer = 1, 2, \ldots, N$ do
11:         for $channel = 1, 2, \ldots, C$ do
12:             $conf\_score, label \leftarrow Yolo\_detectobject(image)$
13:         end for
14:         $M_l \leftarrow \max_{1 \le c \le C} conf\_score$
15:     end for
16:     $\eta \leftarrow f_{thresh}(M_l)$
17: return $\eta$
18: end function
19: function Partition_SIFT($image1$)    ▷ Compute partition point using SIFT keypoint matching
20:     for $layer = 1, 2, \ldots, N$ do
21:         for $channel = 1, 2, \ldots, C$ do
22:             $kp1, kp2 \leftarrow detect\_keypoints(image1, image2)$
23:             $matches \leftarrow Ratiotest(SIFT\_match(kp1, kp2))$
24:         end for
25:     end for
26:     $\eta \leftarrow f_{thresh}(matches)$
27: return $\eta$
28: end function
```

two input images. Let f_θ be the function to compute and detect the keypoints in the image, and $\mathcal{Y}$ be the keypoint descriptor. Let g_θ be the function to detect the number of matches between the images using k-nearest neighbours, and *len* represents the length of the vector. We compute the keypoint descriptors of the input image $\mathcal{Y}1$ and the layer output image $\mathcal{Y}2$ using the SIFT keypoint detection function.

$$\mathcal{Y}1 = f_\theta(\mathbb{R}1) \tag{5.9}$$

$$\mathcal{Y}2 = f_\theta(\mathbb{R}2) \tag{5.10}$$

Then, we determine the similarity between features by comparing the k-nearest neighbours of $\mathcal{Y}1$ and $\mathcal{Y}2$.

$$\mathcal{Y}^* = g_\theta(\mathcal{Y}1, \mathcal{Y}2, k) \tag{5.11}$$

Finally, we use the ratio test to remove approximate matches and determine the number of features matched, which represents the degree of similarity ϵ.

$$\epsilon = len(\mathcal{Y}^*) \tag{5.12}$$

In order to determine the partition point, we set a threshold on the degree of similarity $\mathcal{T}$. The layer at which $\epsilon_l < \mathcal{T}$ is selected as the partition point η.

Chapter 6
HybridTEE Framework

In this chapter, we describe the system-level design of *HybridTEE*, where the key goal is to bridge the two TEEs and jointly accomplish the offloading of DNN execution in a secure manner. Figure 6.1 shows the architecture and system workflow of *HybridTEE*, which consists of the *Local TEE* (TrustZone) and the *Remote TEE* (SGX) with a secure communication channel. we adopt two solutions, namely *remote attestation* and *symmetric encryption* to conduct a secure handshake between the local and remote TEEs and establish the secure communication channel for DNN offloading. First, we employ *remote attestation* [20] to authenticate the SGX *enclave* to eliminate the possibility of *local TEE* communicating with a fake enclave staged by an adversary. The offloading of the DNN execution can be initiated if and only if the remote attestation procedure has been accomplished successfully. Also, the communications between the two TEEs is encrypted to ensure confidentiality.

6.1 System Workflow

The application developer runs the DNN model offline and determines the optimum partition point for each model using the partitioning algorithms presented in Chapter 5. Then, based on the partition point, the developer deploys the model on the *local TEE* with the capability of executing layers 1 to *pp*, i.e., *LocalNet* and the final prediction layer N, i.e., *PredNet*, where *pp* represents the partition point. Next, the developer deploys the model on the *remote TEE* with the capability of executing layers $pp + 1$ to $N - 1$, i.e., the *RemoteNet*.

Step 1. Before the DNN model is executed, the *local TEE* verifies the *remote TEE*

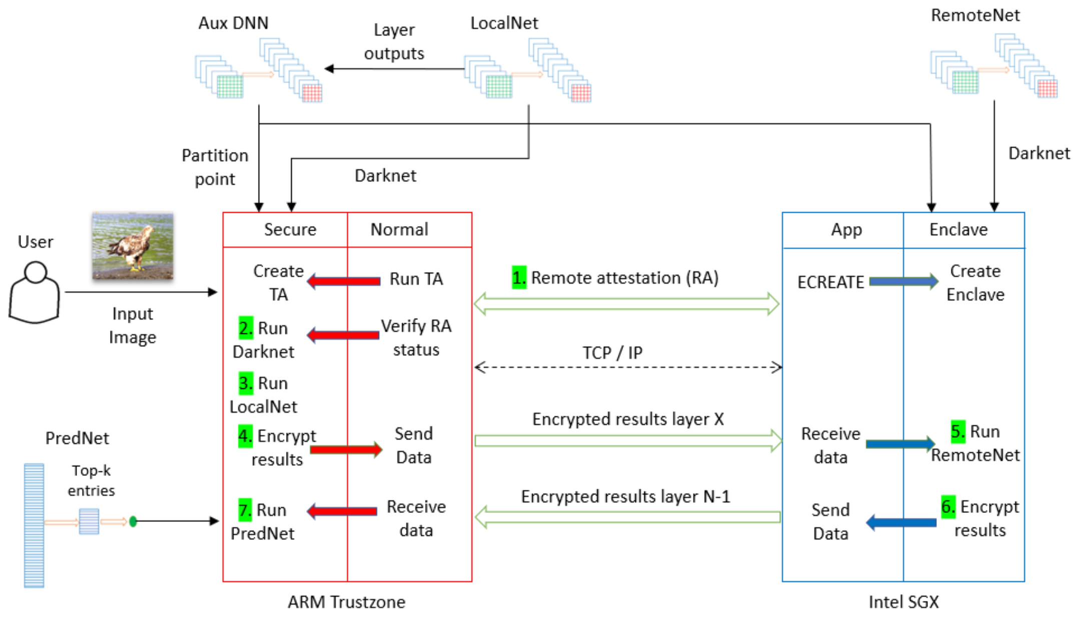


Figure 6.1: System architecture and workflow of *HybridTEE*.

via remote attestation, and a shared secret key between the two TEEs is established.

Step 2: If the attestation process is successful, the *local TEE* creates the DNN layers in the secure world based on the model configuration file.

Step 3. The application runs the first *pp* layers of the model (i.e., *LocalNet*) in the *local TEE* for the input provided by the user.

Step 4. The intermediate outputs of the *LocalNet* are then encrypted using AES-GCM and sent to the *remote TEE*.

Step 5. The *remote TEE* decrypts the partial results and executes layers $pp + 1$ to $N - 1$ of the DNN model, i.e., *RemoteNet*.

Step 6. The intermediate outputs of the *RemoteNet* are encrypted using the shared symmetric key and sent back to the *local TEE*.

Step 7. The *local TEE* decrypts the results obtained from the *remote TEE* and runs the *PredNet* partition, i.e., the N^{th} layer, to generate the final inference results.

6.2 System Implementation

In our HybridTEE prototype, we adopt OP-TEE [15] as the *Local TEE*. OP-TEE is an open source TEE which is designed as companion to a non-secure Linux kernel running on ARM [15]. It implements the TEE Internal Core API which acts as the secure world for trusted applications, and the TEE client API which acts as the secure monitor.

We use Darknet [23] as the DNN implementation. We create a new trusted application (TA) with a universally unique identifier (UUID) for the Darknet code/data. The normal world OS can only access the application using the UUID of the TA, which is equivalent to the NS (non-secure) bit in TrustZone. The entire layer creation and computation code resides in the TA, and it is encrypted before being shared with the rich OS.

We use an open source cryptographic library for AES-GCM encryption with tag validation [4] and a 128-bit symmetric key. The key has been shared between the two TEEs using SIGMA [12] key exchange protocol before the offloading session begins. Once shared, the key is hard coded in the secure world of TrustZone and the enclave of SGX, which cannot be accessed by attackers. Also, we implement a lightweight attestation procedure in our prototype considering the limited computation resources in TrustZone. In the remote attestation, an enclave ID is hardcoded in the application running in both TrustZone and SGX. A pseudorandom token is generated by TrustZone and sent to SGX. SGX appends its enclave ID to the token, encrypts the combined string using the previously shared key, and sends it back to TrustZone. TrustZone decrypts the data and verifies the token and Enclave ID. The remote attestation process is completed if the verification is successful.

Chapter 7

Experimental Results

7.1 Experimental Setup

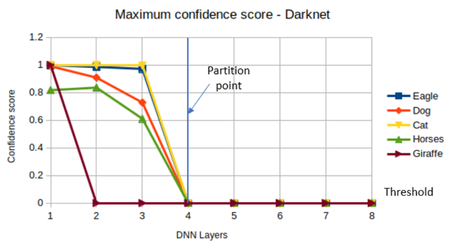
In the experiments, we use a Raspberry pi 3 Model B board with Quad Core 1.2GHz Broadcom BCM2837 64bit CPU and 1 GB RAM as the mobile device. It runs OP-TEE [15] with a Linux OS as the *local TEE*. Also, the *remote TEE* is deployed on a server with 4-core Intel Core i7-6700 3.4GHz CPU, where SGX is enabled in the hardware mode. We use Darknet [23] as the reference DNN implementation. We perform our experiments using 4 DNN models (i.e., Darknet19, VGG-16, Resnet152, and GoogLeNet) with 5 images (i.e., Eagle, Dog, Cat, Horse, and Giraffe).

7.2 Security and Privacy Evaluation

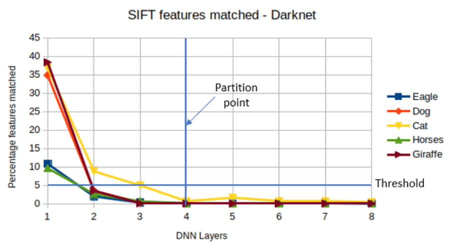
We consider two representative cases in the security and privacy evaluation of *HybridTEE*:

- *Reconstruction (Security).* An adversary attempts to reconstruct the model by accessing the unsecured regions of TrustZone and SGX.
- *Object Identification (Privacy).* An adversary attempts to identify the object in the input image based on the results sent by the local device.

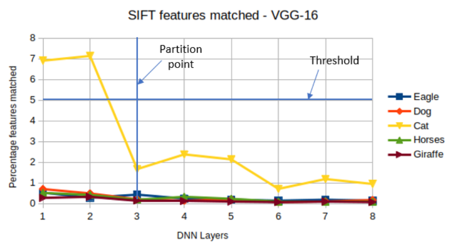
For the reconstruction-based security attack, the model configuration and weight vectors are stored in the local and remote TEEs and, therefore, the hardware-based isolation provided by TEEs ensures that the attacker does not have access to such information to reconstruct the network. For the object identification-based privacy attack, the partitioning algorithm in *HybridTEE* ensure that the input image sent to



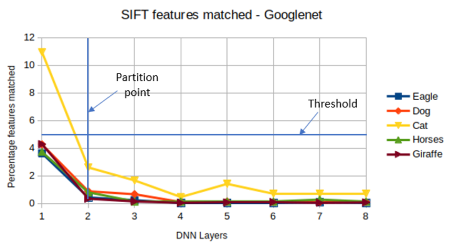
(a) Darknet19 confidence scores



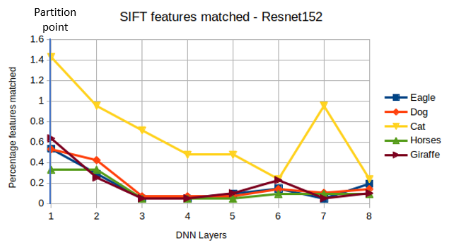
(b) Darknet19 SIFT similarity scores



(c) VGG-16 SIFT similarity scores



(d) GoogLeNet SIFT similarity scores



(e) Resnet152 SIFT similarity scores

Figure 7.1: (a) represents the confidence scores of YOLO object detection for 5 images for the Darknet19 DNN model; and (b)(c)(d)(e) represent the SIFT similarity scores for 4 DNN models.

the *remote TEE* has minimum information exposure about the privacy-sensitive input data from the user. We define the degree of information exposure based on a) the relative ability of an auxiliary DNN model to detect the object in the data sent to the server and b) the degree of similarity between the input image and the layer output image, determined by the number of SIFT features matched between these images.

Figure 7.1(a) shows the confidence scores for object detection using the auxiliary DNN model. We assume that with a non-zero confidence score, the adversary can recover the original input using this model. In other words, we set the confidence score threshold at 0 for partitioning, to provide the strongest privacy guarantee. As illustrated in Table 7.1, for the Darknet19 model, the confidence score drops to 0 at the output of layer 4, indicating that the object detection algorithm was not able to detect any image from all channels of layer 4 for all the 5 images under evaluation. Therefore, we select layer 4 as the *partition point*. We perform the similar procedure for VGG-16, Resnet152 and GoogLeNet. Interestingly, for these models, the object detection algorithm was not able to detect the image from the output of any layer. This indicates that the configuration of these models makes them inherently immune to information exposure, as compared to the Darknet19 model. For these models, the attacker needs an advanced object recognition method to recover the input image. In order to corroborate our observation for this method, we used the second technique to find the partition points for these models, which is SIFT [18].

Image	**L0**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**
Eagle	0.9988	0.986	0.9716	0	0	0	0	0
Dog	0.9907	0.9083	0.7279	0	0	0	0	0
Cat	0.9971	0.9975	0.9957	0	0	0	0	0
Horses	0.8172	0.8362	0.6104	0	0	0	0	0
Giraffe	0.9992	0	0	0	0	0	0	0

Table 7.1: Confidence scores of the Darknet19 model for 5 images and first 8 layers (L0 - L7).

Image	**L0**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**
Eagle	10.86	2.03	0.33	0.09	0.09	0.09	0.09	0.04
Dog	34.86	3.53	0.32	0.11	0.11	0.07	0.11	0.07
Cat	36.91	8.81	5	0.71	1.67	0.71	0.71	0.47
Horses	9.62	2.68	0.67	0.09	0.14	0.09	0.14	0.09
Giraffe	38.37	3.53	0.12	0.05	0.05	0.07	0.05	0.05

Table 7.2: Matched SIFT features of the Darknet19 model for 5 images and first 8 layers (L0 - L7). All values are in terms of % match.

Figure 7.1(b)(c)(d)(e) show the SIFT percentage features matched between the input image and the images generated at the output of each layer. Let $good\_features_{layer\_img}$ be the features of the layer output image with distance less than 0.75x of the feature distance in the original image, and $features_{input\_img}$ be the total number of features in the input image. The percentage match is the ratio of

$$percent\_match \leftarrow \frac{length(good\_features_{layer\_img})}{length(features_{input\_img})} \times 100$$

Table 7.2 demonstrates the percentage match of the SIFT scores for the layer output image with respect to the input image for the Darknet19 model. We set the threshold on the percentage features matched as **5%**. Figure 7.1(b) indicates the features matched for the Darknet19 model. We observe that the features matched value dropped below the threshold after layer 3. Therefore, we select layer 4 as the *partition point*. Similarly, we select layer 3 for VGG-16, layer 2 for GoogLeNet, and layer 1 for Resnet152 as the *partition points*.

7.3 Performance Evaluation

We evaluate the performance of *HybridTEE* by measuring the overall prediction time as compared to the baseline version, which is to execute the DNN model on the raspberry pi in a non-TEE environment. In *HybridTEE*, the overall execution time comprises of 5 components: *LocalNet*, *RemoteNet*, *PredNet*, *Encryption/Decryption* and *Network Send/Receive*.

DNN model	Baseline (sec)	HybridTEE (sec)	Speedup
Darknet19	21.55	12.27	1.75x
GoogLeNet	94.14	26.86	3.5x

Table 7.3: Performance evaluation of HybridTEE compared to the baseline non-TEE system.

For the Darknet19 model, the partition point is at layer 4, so we run 4 layers in *LocalNet* and 21 layers in *RemoteNet*. Similarly, for the GoogLeNet model, the partition point is at layer 2, so we run 2 layers in *LocalNet* and 24 layers in *RemoteNet*. The *PredNet* partition only runs the last layer. Table 7.3 shows the timing measurements, which indicate a significant speedup (1.75x to 3.5x) of our HybridTEE prototype as compared to the non-TEE baseline. Note that we were not able to run VGG-16 and Resnet152 in the baseline version due to their high memory requirement, which further promotes our idea of using a hybrid TEE environment. In our prototype, we use

Ethernet as the communication medium between local and remote partitions due to the lack of WiFi support in OP-TEE. The actual speedup on the system deployed on a mobile device may be slightly lower than our prototype, given the wireless network latency.

Chapter 8

Limitations

We acknowledge the limitations in our *HybridTEE* framework, namely *a) Small sample size used for partition point computation* and *b) Algorithmically computed partition point may be greater than TCB size limit of the local device TEE.*

We test our partition point computation algorithm using a small subset of images. The possibility of the partition point being dependent on the image semantics makes it vulnerable to change as per the type of image under consideration. The partition point computed using our offline approach provides a static value for each DNN model. This value acts as a configuration parameter while running the DNN inference on the local device. Since this value is predetermined, it does not take into account the characteristics of the input image for partitioning. Partitioning the DNN based on an inaccurate partition point may cause privacy sensitive information being available to the remote SGX server. The solution to address this issue is to convert this offline method to online. This will ensure that the partition point is calculated for each input image separately. However, the challenge to deploy such an online approach is the complexity of the partition point detection algorithms, versus the TCB size limit of the TEE.

The second limitation of our design is handling the case where the partition point computed by the offline process is greater than the TCB size threshold of the TEE. Currently, we choose the minimum value between the two as the final partition point. This leaves the possibility of privacy exposure if TCB size is the lesser value. The solution to address this problem is to degrade the image quality at the TCB size threshold to trade-off accuracy for privacy. For example, *Shredder* [21] suggests an inference privacy protection method. It targets at finding additive noise distributions that can be added to the intermediate results on the local device, to preserve the privacy of input data.

It aims to strike a balance between accuracy and inference privacy. A similar solution can be deployed in *HybridTEE*, where we can add noise to the intermediate layer-image at the TCB size threshold. The goal of this method would be to reach the same level of privacy that is guaranteed by the partitioning algorithm with an acceptable level of accuracy.

Chapter 9

Related Work

Since the execution of deep learning applications requires substantial computational resources and involves sensitive information, security researchers have been working on the model/data protection by leveraging TEE in the edge/cloud computing platforms. For example, HETEE is a heterogeneous TEE framework where the deep-learning applications are offloaded from singular CPU based TEE to accelerators [34]. However, the size of TCB in a TEE must be minimized to avoid potential security vulnerabilities. In this case, offloading entire DNN computation to the accelerators might not be practical. To minimize the workload of TEE and prevent sensitive data from leaking to accelerators, researchers have used several DNN partitioning techniques [6][22][30][31]. Darknet executes a set of sensitive layers sequentially to minimize resource usage in the TEE [22]. Slalom allocates the integrity-verifiable linear layers into an untrusted GPU, and protects the remaining layers in SGX [30]. To prevent the data from being tampered, Chen et al. propose DeepAttest [3], a TEE-based attestation method to verify the model integrity. Different from the previous work, *HybridTEE* employs a combination of two TEEs, i.e., a *local TEE* and a *remote TEE* to reduce the workload of singular TEE for secure DNN computations, together with the privacy consideration.

Chapter 10

Conclusion

We have developed *HybridTEE*, a novel hardware-based security framework that uses a combination of ARM TrustZone and Intel SGX TEE platforms to securely execute DNN inference. We systematically studied the layer-wise privacy exposure of the DNN about input data and devised an offline DNN partitioning strategy using image similarity metrics based on object detection and SIFT. We implemented a prototype of *HybridTEE* on real hardware systems. Our security and performance evaluations with 4 DNN models demonstrated that *HybridTEE* can successfully defend against model reconstruction and object identification attacks and achieve 1.75-3.5x speedup compared to the non-TEE baseline system.

References

[1] *ARM Security Technology: Building a Secure System using TrustZone Technology.* 2005.

[2] Y. Bengio, A. Courville, and P. Vincent. “Representation Learning: A Review and New Perspectives”. In: *arXiv:1206.5538.* 2012.

[3] H. Chen, C. Fu, B. Rouhani, J. Zhao, and F. Koushanfar. “DeepAttest: An End-to-End Attestation Framework for Deep Neural Networks”. In: *International Symposium on Computer Architecture.* 2019.

[4] M. Clark. *AES-GCM.* `https://github.com/michaeljclark/aes-gcm`. 2016.

[5] K. Goldman, R. Perez, and R. Sailer. “Linking Remote Attestation to Secure Tunnel Endpoints”. In: *Proceedings of the First ACM Workshop on Scalable Trusted Computing.* 2006.

[6] Z. Gu, H. Huang, J. Zhang, D. Su, H. Jamjoom, A. Lamba, D. Pendarakis, and I. Molloy. “Securing Input Data of Deep Learning Inference Systems via Partitioned Enclave Execution”. In: *arXiv:1807.00969.* 2018.

[7] Z. Gu, H. Huang, J. Zhang, D. Su, H. Jamjoom, A. Lamba, D. Pendarakis, and I. Molloy. “YerbaBuena: Securing Deep Learning Inference Data via Enclave-based Ternary Model Partitioning”. In: *arXiv:1807.00969v2.* 2019.

[8] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, A. Coates, and A. Ng. “Deep Speech: Scaling up end-to-end speech recognition”. In: *arXiv:1412.5567.* 2014.

[9] W. Hao, Z. Yitong, J. Zheng, G. Xing, Z. Dihong, L. Jingchao, and W. Zhifeng. “Cosface: Large margin cosine loss for deep face recognition”. In: *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 5265–5274.

[10] W. Hua, Z. Zhang, and G. Suh. “Reverse Engineering Convolutional Neural Networks through Side-Channel Information Leaks”. In: *Design Automation Conference*. 2018.

[11] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen. “Intel Software Guard Extensions: EPID Provisioning and Attestation Services”. In: 2016.

[12] H. Krawczyk. “SIGMA: the ‘SIGn-and-MAc’ Approach to AuthenticatedDiffie-Hellman and its Use in the IKE Protocols”. In: *International Cryptology Conference*. 2003.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2012.

[14] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li. “Cost-Driven Offloading for DNN-based Applications over Cloud, Edge and End Devices”. In: *arXiv:1907.13306*. 2019.

[15] Linaro. *Open Portable Trusted Execution Environment*. `https://www.op-tee.org`. Accessed on 2019.

[16] G. Litjens, T. Kooi, B. Bejnordi, A. Setio, F. Ciompi, M. Ghafoorian, B. Ginneken J. Laak, and C. Sánchez. “A survey on deep learning in medical image analysis”. In: *Medical image analysis*. 2017.

[17] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang. “Trojaning attack on neural networks”. In: *Network and Distributed System Security Symposium*. 2017.

[18] D. Lowe. “Object Recognition from Local Scale-Invariant Features”. In: *International Conference on Computer Vision*. 1999.

[19] S. Malladi, J. Alves-Foss, and R. Heckendorn. “On Preventing Replay Attacks on Security Protocols”. In: *International Conference on Security and Management.* 2002.

[20] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar. “Innovative instructions and software model for isolated execution”. In: *International Workshop on Hardware and Architectural Support for Security and Privacy.* 2013.

[21] F. Mireshghallah, M. Taram, P. Ramrakhyani, D. Tullsen, and H. Esmaeilzadeh. “Shredder: Learning Noise Distributions to Protect Inference Privacy”. In: *Architectural Support for Programming Languages and Operating Systems.* 2020.

[22] F. Mo, A. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi. “DarkneTZ: Towards Model Privacy at the Edge using Trusted Execution Environments”. In: *International Conference on Mobile Systems, Applications, and Services.* 2020.

[23] J. Redmon. *Darknet: Open Source Neural Networks in C.* `http://pjreddie.com/darknet/`. 2013–2016.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *arXiv:1506.02640.* 2016.

[25] J. Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *arXiv:1404.7828.* 2014.

[26] K. Shin, H. Jeong, and S. Moon. “Enhanced Partitioning of DNN Layers for Uploading from Mobile Devices to Edge Servers”. In: *International Workshop on Embedded and Mobile Deep Learning.* 2019, pp. 35–40.

[27] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: 2014. URL: `http://arxiv.org/abs/1409.1556`.

[28] F. Stumpf, O. Tafreschi, P. Roder, and C. Eckert. “A robust integrity reporting protocol for remote attestation”. In: *Second Workshop on Advances in Trusted Computing.* 2006.

[29] C. Szegedy, A. Toshev, and D. Erhan. “Deep neural networks for object detection”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems*. 2013.

[30] F. Tramer and D. Boneh. “Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware”. In: *International Conference on Learning Representations*. 2019.

[31] P. VanNostrand, I. Kyriazis, M. Cheng, T. Guo, and R. Walls. “Confidential Deep Learning: Executing Proprietary Models on Untrusted Devices”. In: *arXiv:1908.10730*. 2019.

[32] J. Yang, P. Ren, D. Zhang, D. Chen, F. Wen, H. Li, and G. Hua. “Neural aggregation network for video face recognition”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 4362–4371.

[33] Y. Zhao, X. Hu, S. Li, J. Ye, L. Deng, Y. Ji, J. Xu, D. Wu, and Y. Xie. “Memory Trojan Attack on Neural Network Accelerators”. In: *Design, Automation Test in Europe Conference Exhibition*. 2019, pp. 1415–1420.

[34] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, L. Zhao, F. Yuan, P. Li, B. Zhao, and D. Meng. “Enabling Privacy-Preserving, Compute-and Data-Intensive Computing using Heterogeneous Trusted Execution Environment”. In: *arXiv:1904.04782*. 2019.