Confidential Execution of Deep Learning Inference at the Untrusted Edge with ARM TrustZone

Md Shihabul Islam
md.shihabul.islam@utdallas.edu
The University of Texas at Dallas
Richardson, Texas, USA

Mahmoud Zamani
mxz173130@utdallas.edu
The University of Texas at Dallas
Richardson, Texas, USA

Chung Hwan Kim
chungkim@utdallas.edu
The University of Texas at Dallas
Richardson, Texas, USA

Latifur Khan
lkhan@utdallas.edu
The University of Texas at Dallas
Richardson, Texas, USA

Kevin W. Hamlen
hamlen@utdallas.edu
The University of Texas at Dallas
Richardson, Texas, USA

ABSTRACT

This paper proposes a new confidential deep learning (DL) inference system with ARM TrustZone to provide confidentiality and integrity of DL models and data in an untrusted edge device with limited memory. Although ARM TrustZone supplies a strong, hardware-supported trusted execution environment for protecting sensitive code and data in an edge device against adversaries, resource limitations in typical edge devices have raised significant challenges for protecting on-device DL requiring large memory consumption without sacrificing the security and accuracy of the model. The proposed solution addresses this challenge without modifying the protected DL model, thereby preserving the original prediction accuracy. Comprehensive experiments using different DL architectures and datasets demonstrate that inference services for large and complex DL models can be deployed in edge devices with TrustZone with limited trusted memory, ensuring data confidentiality and preserving the original model's prediction exactness.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Software and application security**.

KEYWORDS

Deep Learning, Embedded Device, Trusted Execution Environment

ACM Reference Format:
Md Shihabul Islam, Mahmoud Zamani, Chung Hwan Kim, Latifur Khan, and Kevin W. Hamlen. 2023. Confidential Execution of Deep Learning Inference at the Untrusted Edge with ARM TrustZone. In *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy (CODASPY '23), April 24–26, 2023, Charlotte, NC, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3577923.3583648

CODASPY '23, April 24–26, 2023, Charlotte, NC, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0067-5/23/04.
https://doi.org/10.1145/3577923.3583648

1 INTRODUCTION

The ubiquity and proliferation of the Internet of Things (IoT) and edge devices in various domains, such as smart homes, smart cities, smart transportation, industrial automation, agriculture, healthcare, and so forth, have instigated the need for *on-device machine learning*. While traditional edge devices merely perform simple tasks upon receiving specific requests, advances in computing power and hardware efficiency have allowed artificial intelligence to move towards the edge devices, and thus effectively automate processes with minimal human intervention [33]. Therefore, researchers are progressively deploying more complex deep learning (DL) models such as Deep Neural Networks (DNN) [4, 39, 41] at the edge devices rather than in the cloud to improve the performance of edge/IoT applications by reducing dependency on the network (i.e., bandwidth), minimizing communication cost and latency of the application.

One of the major reasons behind the demand for on-device learning is the privacy of the user data, as the data may never need to leave the device [5]. Since most interconnected edge devices collect sensitive private information from users (e.g., medical information, location, etc.), data leakage from the learning process on the user data may raise serious privacy concerns. For instance, adversaries may leverage DL model information (e.g., parameters, intermediate results, and final output) to devise attacks such as membership inference attacks [38] and input reconstruction attacks [8, 27] to gain insights on valuable yet private user data. Some solutions have been proposed throughout the years to mitigate the information leakage, such as differential privacy [49] and homomorphic encryption [31]; however, they tend to affect the inference prediction accuracy of the learning models and increase the computational cost significantly, respectively.

The research community has recently leveraged hardware-based security features, called *trusted execution environment* (Tee), to massively escalate data protection and information leakage mitigation [15, 16, 30]. Although most of the solutions are targeted for server systems (e.g., *Intel SGX* [6]), ARM has developed TrustZone [46] for their processors to target mobile and edge devices. However, the *trusted memory*, which is isolated from untrusted memory, is costly and hence usually restricted in size on the IoT/edge devices. For instance, IoT devices are provided with approximately 3–5 MB of trusted memory [1]. The trusted computing base (TCB) of the Tee must be kept as small as possible for the edge devices to minimize the attack surface [30]. Since DL algorithms are highly resource-intensive, it is therefore challenging to deploy security-sensitive DL applications on the edge protected by TrustZone [53].

Prior studies mainly focus on different methodologies to optimize the expensive DNN models to reduce their overall resource requirements. *Quantization* [17, 43, 47] and *pruning* [13, 43] techniques have been extensively studied. However, because of information loss, these techniques affect the inference accuracy of the model [18, 23]. Another approach is to statically partition a DNN model into protected and unprotected segments and only execute the protected segment in a Tee of an edge device in isolation from the unprotected segment [28, 45]. Other approaches propose a similar strategy to protect the entire DNN model, but for the cloud environment with the help of SGX [12, 18, 23].

Table 1 compares these methodologies to our work. *Static model partitioning* is unsuitable for small IoT devices with limited memory, as it does not allow executing a model with a protected segment larger than the trusted memory where the protected DL code, model, and data must be loaded for confidential execution. Many edge devices do not have such a large memory, and only a fraction of the memory can be used as the trusted memory in ARM TrustZone, forcing the user to only protect a small segment of a large DNN model. For instance, Alexnet requires at least 272*MB* memory for the prediction procedure, which is unfeasible for an edge device to allocate in the limited trusted memory. Therefore, in traditional setting, a large segment of Alexnet is executed outside of trusted memory. The unprotected segments may disclose private and sensitive information, such as the training data of the pre-trained DL model [12, 18, 23, 45]. These limitations have motivated us to develop a new technique to enable the confidential execution of an entire DL model without sacrificing its inference exactness, or having to decide which segment of the model execution is more privacy-sensitive.

This paper hence proposes a new DL inference framework that enables protected execution of an entire DNN model in an untrusted edge device using ARM TrustZone with limited trusted memory. Our approach does not modify the protected DL model or retrain it, and thus does not suffer any sacrifice to the prediction accuracy. More specifically, we propose a framework named T-Slices that *dynamically* converts an unmodified DL model into units called *slices*, and executes them entirely on the trusted memory. Our study of convolutional neural networks (CNN) shows that the required memory of some convolution and connected layers exceeds the usual memory limit of the trusted memory [28]. We therefore devise a mechanism to transform the layers into slices where each slice is small enough to be loaded in the secure memory of TrustZone. We dynamically calculate the slice for each convolution and connected layer based on convolution layer's input channels and connected layer's output neurons, respectively, so that each slice executes independently and remains within the trusted memory limit.

T-Slices sequentially loads as many slices as possible to fit into the trusted memory at a time, and the next slices that have a dependency with the previous slices wait for their turn in the untrusted memory in an encrypted form. This protects the entire DL inference execution (e.g., parameters, intermediate results, and final output) utilizing ARM TrustZone's security features, as all the slices time-share the limited trusted memory throughout the execution, leaving no units unprotected.

Our approach avoids redesigning or modifying any component of the model, retaining the original inference's full accuracy. We conduct comprehensive experiments using different CNN architectures and datasets for image classification, then study the performance trade-off to evaluate T-Slices with insufficient trusted memory in TrustZone. Our results show that T-Slices can reduce layer peak memory requirements by 72% and improve the execution time by 29% on average compared to the baseline approach. We empirically show that T-Slices can deploy inference services for moderately large and complex pre-trained CNN models on ARM TrustZone processors with limited trusted memory, ensuring data confidentiality and preserving the original model prediction.

To summarize, we make the following contributions:

- We propose and develop a framework called T-Slices to convert DNNs into a set of slices so that each slice can fit into a given limited memory, thereby executing the entire model within the memory limit without sacrificing inference prediction accuracy.
- Our system enables protected inference of pre-trained DNN models utilizing ARM TrustZone in an edge device with limited trusted memory while preserving the confidentiality of the entire model and data.
- Detailed case studies against prevalent privacy attacks assess the effectiveness of the framework.
- We perform thorough experiments on a real device to assess the proposed framework with different CNN architectures and datasets for image classification, and evaluate the performance tradeoffs of DL methods on edge Tees.

The rest of the paper is organized as follows. Section 2 presents some background on ARM TrustZone and on-device learning challenges. Section 3 explains the threat model. Section 4 describes the state-of-the-art on-device learning methodologies with TrustZone. Section 5 introduces our proposed approach and its components. Section 6 details the security analysis for the framework. Section 7 describes the experiments and evaluation of the framework. Sections 8 and 9 describe limitations with future work and related work, respectively. Finally, Section 10 concludes.

2 BACKGROUND

2.1 ARM TrustZone

ARM is the pioneer in developing processors for the embedded devices globally [48], powering over 60% of all embedded devices, while 4.5 billion mobile devices also have ARM processors [32]. ARM TrustZone consists of hardware-level security features that separate the physical environment into two parts: Rich Execution Environments (Rees) and Trusted Execution Environments (Tees). A Ree, also known as the *normal world*, contains the non-secure operating system (OS) and other privileged software where security is not the main concern because of its vast size and complexity. A Tee, also known as the *secure world*, provides a restricted execution environment where sensitive and private data can be processed; everything outside the Tee is untrusted. More specifically, TrustZone's secure world ensures the integrity and confidentiality of an

Table 1: Comparison with different methodologies supporting DL execution

Publication	Resource Optimization Technique	Security Features	Preserves Accuracy	Edge Device Support	Full Model Protection on Edge
T-Slices (our method)	DNN dynamic fragmentation	TrustZone	✓	✓	✓
DarkneTZ [28]	DNN static layer-wise partition	TrustZone	✓	✓	✗
Confidential DL [45]	DNN static layer-wise partition	TrustZone	✓	✓	✗
Vessels [18]	DNN layer-wise partition	SGX	✓	✗	✗
Infenclave [12]	DNN layer-wise partition	SGX	✓	✗	✗
Occlumency [23]	DNN convolution layer partition	SGX	✓	✗	✗
TensorFlow Lite [43]	Quantization	✗	✗	✓	✗

application's security-sensitive computation and data on a device, where all the privileged software such as the OS is potentially malicious, by constraining the OS to operate within the boundaries of the normal world.

Our work targets Op-Tee [25] as our Tee for the ARM TrustZone. It is designed to depend on the TrustZone technology for the underlying hardware-enforced isolation mechanism. The main goals of Op-Tee are the isolation from the non-secure OS and protection of the Trusted Applications (TA), small-footprint to reside in limited on-chip memory as found on ARM-based systems, and portability to support different architectures and hardware. Op-Tee conforms to GlobalPlatform API specifications by developing Tee Internal Core API for implementing TAs and the Tee Client API to communicate with the Tee [24].

2.2 Challenges of Secure On-device Learning

On-device learning with edge devices represents a potential paradigm shift in that it protects the privacy of the user data by bringing the computation to the data instead of bringing the data to the computation, which leaves the data in place instead of shipping it to cloud services. But its realization raises several new security challenges, including protection of the user data and the compute model on the untrusted device from the adversaries, and protection of the computation and resource-intensive compute models on resource-constrained edge devices.

As with cloud services, adversaries target embedded devices to steal valuable information [32]. A pre-trained DNN model may leak information about the original inputs, which could violate user privacy. For example, in a *membership inference attack* (MIA), adversaries discover whether a person's health record was used to build a ML model that predicts a disease to infer whether that person has the disease with high probability [38]. *Input reconstruction attacks* enable adversaries to reconstruct the input from the observed model (i.e., parameters) and predicted outcomes [12]. Adversaries can utilize the leaked confidential information to take targeted actions against the user (e.g., for monetary purposes). Although traditional cryptographic approaches can preserve the model and data at rest, they cannot safeguard them when they are used in computation, since they need to be decrypted before any execution. More robust cryptographic techniques, such as fully homomorphic encryption (FHE) and secure multi-party computation (SMC), can mitigate this complication; however they come with heavy computational and communication costs, respectively, which is not appropriate for edge devices [19, 29]. Prior works have therefore leveraged ARM TrustZone security features to ensure the confidentiality and integrity of the DL models in edge devices.

Generally, machine learning models work in two stages: training and inference. It is the standard practice to execute the training process in a secure cloud environment, as it requires more computational power, time, and memory, and transfer the inference process to the edge, as it is performed frequently to test instances required by the application. However, it still may be a challenge for the resource-constrained edge device to carry out an inference task of a substantial pre-trained DL model securely. In fact, Tees such as ARM TrustZone offer very limited trusted memory to keep the memory footprint as small as possible, which is a serious obstacle to executing the entire DL tasks. In general, the amount of trusted memory allocated by the Op-Tee for a TA in mobile devices containing TrustZone is roughly 16 MB [28], and even less for low-end devices (approximately 3–5 MB) [1]. Some works have introduced quantization and model pruning techniques to fit in the TrustZone; however, these approaches affect the accuracy of the classification because of the loss of information in the original model.

3 THREAT MODEL

In this paper, we consider an adversary that seeks to surreptitiously gain insight into sensitive user information from the edge devices. We assume that the adversary has full control of the OS, the privileged software components, and other applications in the edge device and the DL models are securely trained in a cloud environment with private user data before being deployed in the edge device without leaking any information during and after the deployment. In addition, we assume that any modification to the application or any model information can be detected in the TrustZone component of the application. The adversary must therefore try to access the parameters, intermediate results, and prediction outcomes of the deployed DL models to acquire confidential information. The adversary's goal is to expose these contents to infer secrets, such as the training data of the pre-trained DL model, without interfering with the inference task.

Our focus in this work is not on protecting against side-channel attacks, which are well studied against Tee in prior work [32]. Rather, we focus on improving the performance of DL models for low-powered and memory-constrained edge devices utilizing TrustZone security features. Existing defence techniques against side-channel attacks are applicable to our system and should be employed alongside our defense for comprehensive protection. Denial-of-Service (DoS) attacks [14] are also out of our scope.

4 ANALYSIS OF STATE-OF-THE-ART ON-DEVICE LEARNING SCHEME WITH TRUSTZONE

Numerous recent research works have proposed layer-based partitioning of CNNs to fit the model into restricted memory [12, 18, 28, 45]. More specifically, a CNN is partitioned into layers so that each layer is executed one at a time with the TrustZone's trusted memory. Generally, each layer in a CNN depends on its parameters (e.g., weights, biases) and the output of the previous layer, except for the first layer where it is dependent on the input values. Since there is no cross-dependency between any two layers, each layer can be computed sequentially in an independent way. Thus, with this scheme it is possible to execute CNNs with limited memory.

For a CNN, the most computationally demanding and memory-intensive operation is the convolution operation. Most of the memory in a convolution operation is typically occupied by the input matrix; the output matrix, kernel, or weight matrix; and a special memory transformation matrix called *im2col*. The *im2col* transformation is an essential component of the General Matrix Multiplication (Gemm) algorithm, which is used as the convolution operator in most of the major DL frameworks [9]. The *im2col* replicates a patch of input pixels that affect the value of an output pixel to a separate matrix. The product of this matrix of input patches and the corresponding convolution kernel produces the output matrix. The *im2col* matrix requires substantial additional memory proportional to the kernel size and number of inputs. Therefore, this transformation may cause nontrivial overhead in memory storage and bandwidth, and reduce data locality [2].

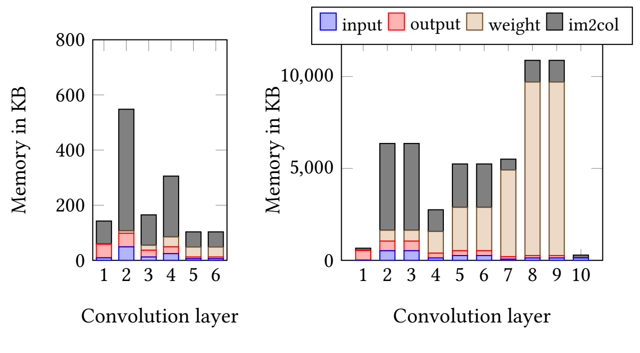
We assume that a convolution layer l includes the following properties: input height H_{in}, input width W_{in}, number of input channels C_{in}, output height H_{out}, output width W_{out}, number of output channels C_{out}, kernel height H_k, and kernel width W_k. The sizes of input $\mathcal{I}$, output $\mathcal{O}$, kernel $\mathcal{K}$, and *im2col* $\mathcal{B}$ matrices for the layer are:

$$\mathcal{I} = H_{in} \times W_{in} \times C_{in} \qquad \mathcal{O} = H_{out} \times W_{out} \times C_{out}$$
$$\mathcal{K} = H_k \times W_k \times C_{in} \times C_{out} \qquad \mathcal{B} = (H_{out} \times W_{out}) \times (H_k \times W_k) \times C_{in}$$
$$\mathcal{M}_l \approx \mathcal{I} + \mathcal{O} + \mathcal{K} + \mathcal{B} + \theta$$

where $\mathcal{M}$ is the total size of the input, output, kernel, and *im2col* matrices for layer l; and θ denotes the size of other parameters, such as bias.

Figure 1 illustrates how different components of a convolution layer allocate memory for CNN architectures. From the figure, it is evident that the memory overhead of the *im2col* transformation is significant, especially at the beginning of the network. Moreover, layers at the end of the network have significant filter weight sizes, because of the increasing number of filters and channels. Thus, the total layer size grows notably for these layers.

Figure 2 demonstrates how the required peak memory to execute any convolution or connected layer following layer-wise partition strategy varies for different CNN architectures for inference. We can see from the figure that if the TA size limit is 16 MB, which is the trusted memory size allocated for mobile devices (as discussed in §2.2), some resource-hungry CNNs fail to execute at least one layer in the secure world of TrustZone. Therefore, partitioning the network by layers may only be enough to execute the whole model



(a) Vgg-7 (b) Cifar

Figure 1: Convolution layer sizes for (1a) Vgg-7 and (1b) Cifar models. Each layer size is comprised of the input, output, weight, and *im2col* memory buffer sizes.

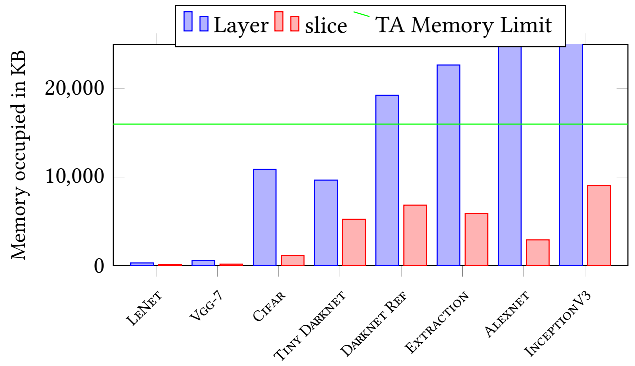


Figure 2: Peak memory required to execute any convolution/connected layer in different CNN architectures. ▯ denotes CNN is partitioned layer-wise, and ▯ denotes when CNN execution is *sliced* (we discuss *slice* in §5). The green line indicates the TA size limit; here it is considered as 16 MB.

for simple CNNs, such as LeNet and Vgg-7; it will be insufficient for more complex CNNs, such as Alexnet and Darknet Ref, in an edge setting with TrustZone.

To overcome the trusted memory constraint, some approaches statically limit the number of layers in the TrustZone while the other layers are performed in the untrusted normal world, exposing secret data and model information [28, 45]. Our objective is to ensure the confidentiality of data and model by computing all the layers in the TrustZone trusted memory without redesigning the network.

Further investigation reveals the lack of an optimized memory management plan in contemporary DL implementations and designs. At the start of the process, the application usually assigns the necessary memory buffers for parameters (e.g., weights, biases) and other intermediate activations (e.g., *im2col*, outputs). The buffer sizes can be easily calculated from the hyperparameters before the execution, and are allocated for all layers and kept in memory until

Table 2: Peak memory consumption of different DL predictors

Model	# Layers	Pre-trained Model Size (MB)	Peak Mem. Usage (MB)
LeNet	10	0.2	7
VGG-7	13	0.3	7
CIFAR	18	30.7	45
Tiny	22	4.2	71
Darknet	16	29.3	88
Extraction	27	93.8	163
Alexnet	14	249.5	272
Darknet53	78	159	273
Inception-v3	145	95.5	448
Yolov3	107	237	840
VGG-16	24	528	923

the entire execution is concluded. Yet, once a layer is finished executing, the parameters and other intermediate outputs are never reused in the later layer executions unless there exists a dependency between the layers.

Table 2 shows the peak memory consumption of different DL predictors with its pre-trained sizes. It indicates that a considerable memory is allocated and retained for the entire execution. Therefore, retaining the memory buffers without any dependency not only engenders redundancy but also wastes critical memory usage. This is a significant issue for memory-constraint devices, especially for ARM TrustZone, where abundant memory is not available.

5 PROPOSED FRAMEWORK

Our goal is to mitigate the information leakage of the DNN models in untrusted embedded devices using TrustZone from widely used ARM processors for the embedded devices without relinquishing prediction accuracy. To tackle the limited resource constraint of trusted memory of the TrustZone as discussed in §4, we design a scheme called T-Slices that dynamically transforms the traditional layer-based DNN models into smaller fragments called slices, and sequentially executes sets of slices in the trusted memory, thereby protecting the entire model with TrustZone. This section presents our framework starting with the design preliminaries, deployment strategy, and T-Slices description.

5.1 Design Preliminaries

We assume that an edge device closer to the data source or even the source device itself is tasked with a prediction or classification problem. For simplicity, the edge device here denotes the source device, end-user device, or any device that is closer to the data source where we perform the on-device learning. In addition, we assume that the edge device contains an ARM processor and is equipped with TrustZone security features. As a running case study, we consider a prediction problem consisting of image classification. Particularly, we consider CNN as the base algorithm, since it is the breakthrough image classification model among many DNNs.

5.2 Deployment Strategy

Since the TrustZone for the edge devices suffer from limited memory constraints, and since CNN execution is computationally expensive (as discussed in §2.2), the edge device is tasked with only the inference operation of CNN, where the training operation will be executed in a cloud environment securely. The pre-trained model contains the parameter values as well as the hyperparameter values (i.e., structure or properties of the particular CNN). This model is then deployed to the edge device either via a secure communication channel from the cloud or by loading it to the device beforehand. The parameters of the pre-trained model are always kept encrypted in the local storage and could be updated intermittently depending on the applications. However, the hyperparameters of the model are kept unencrypted in the normal world since those are usually already well-known to the public and do not disclose any sensitive information of the input or training data [18].

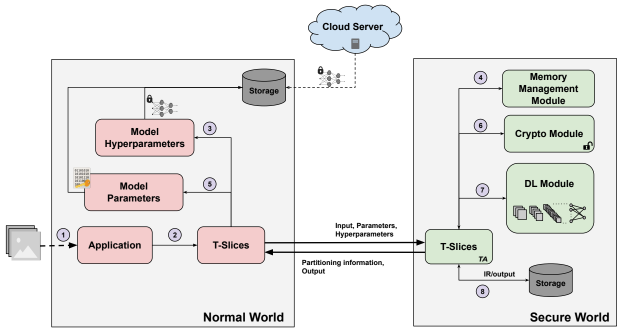


Figure 3: Workflow of T-Slices with ARM TrustZone.

Once an inference task is required, the input data (in this case an image) and model parameters are loaded into the TrustZone's trusted memory, and the prediction task is carried out after the decryption of the model parameters. Depending on the application, the inference result could be utilized on the device or dispatched to the cloud in an encrypted form. The following subsections address the problem of guaranteeing the confidentiality of the model and data when the input data, CNN model (i.e., parameters), and the produced intermediate representations of CNN's inference operations do not fit the available trusted memory of TrustZone.

5.3 T-SLICES

Our proposed technique partitions each traditional DNN layer into segments called slices such that each slice is small enough to be loaded and executed in the trusted memory of ARM TrustZone. The framework dynamically determines a set of slices depending on the memory buffer available in the secure world. Predominantly, all the memory allocation for a particular CNN depends on the predetermined hyperparameters [18]. As the hyperparameters never change during executions, we can easily derive memory requirements and dependencies of each layer for the execution. As a result, we can choose a number of slices such that it can fit into the available trusted memory in TrustZone. The set of slices are then executed in the TrustZone with the next set of slices dependent on the current set waiting for their turn in the unsecure memory in an encrypted form. This executes the whole network in chunks without breaking the memory constraint of TrustZone and without affecting the final prediction result.

Figure 3 illustrates the workflow of T-Slices with ARM TrustZone for an edge device. When an application receives instruction for an inference task (`step-1`), it triggers our framework (`step-2`). T-Slices reads the hyperparameters of the prediction model (`step-3`) and communicates with the TA for initializing the memory buffers in the TrustZone (`step-4`). Then, T-Slices apply slicing technique on model parameters (`step-5`) and sends the slices to TrustZone for decryption (`step-6`). Next, the decrypted slices are feed to the forward pass of the DL model (`step-7`), where any outputs or required intermediate results are stored in the trusted memory (`step-8`). Steps 4 to 8 are repeated for all the slices in a layer and all the layers in the network until the final prediction is achieved.

For instance, in case of a convolution layer, we define a slice as the execution of one *input channel* of a layer at a time in the secure world. The intuition behind this approach is that we can reduce the memory requirement of *im2col* transformation and the kernel weights by taking only a single input channel instead of all the input channels. With this approach, we can rewrite the equation outlined in §4 for the memory buffer of *im2col* matrix as follows:

$$\mathcal{B} \approx (H_{out} \times W_{out}) \times (H_k \times W_k) \tag{1}$$

Figure 4 illustrates how this slicing strategy can significantly reduce peak size of the required memory for each convolution layer of Darknet Ref model. Figure 4a depicts peak sizes of each convolutional layer, while Figure 4b shows peak size of a slice for each convolutional layer. For example, in the figures, the $7th$ convolution layer occupies more than $18MB$ of buffers, whereas a slice occupies less than $0.5MB$. Therefore, slicing can drastically reduce the sizes of *im2col* and kernel weight matrices, which makes it straightforward to load the slices into the trusted memory sequentially and compute the convolution operations in multiple steps.

Taking only one slice at a time for the computation does not fully utilize the available memory in the secure world. Therefore, we can utilize more slices in batches to optimize the unoccupied memory and speed up the process. The full process is carried out as follows.

For a particular CNN, we first derive the peak memory buffer size $\mathcal{P}$ required for a layer to compute in the TrustZone. For the case of convolution layers, we can compute size of input $\hat{\mathcal{I}}$, kernel $\hat{\mathcal{K}}$, and *im2col* $\hat{\mathcal{B}}$ matrices, and total memory buffer size $\hat{\mathcal{M}}$ for a slice as:

$$\hat{\mathcal{I}} = \begin{cases} H_{in} \times W_{in} & \text{if } l = 1 \\ \mathcal{I} & \text{if } l > 1 \end{cases} \qquad \hat{\mathcal{K}} = H_k \times W_k \times C_{out}$$

$$\hat{\mathcal{B}} = (H_{out} \times W_{out}) \times (H_k \times W_k) \qquad \hat{\mathcal{M}}_l \approx \hat{\mathcal{I}} + \mathcal{O} + \hat{\mathcal{K}} + \hat{\mathcal{B}} + \theta$$

where θ is other parameters whose size does not depend on the input channels (e.g., bias), and l is the layer number in the network. Note that the output matrix size is the same since slicing does not affect the output matrix size, and slicing affects the size of the input, kernel weight, and *im2col*. Moreover, for the first layer only we slice the input since it is supplied from the normal world. The inputs to later layers are the outputs of previous layers, which already persists in the secure world.

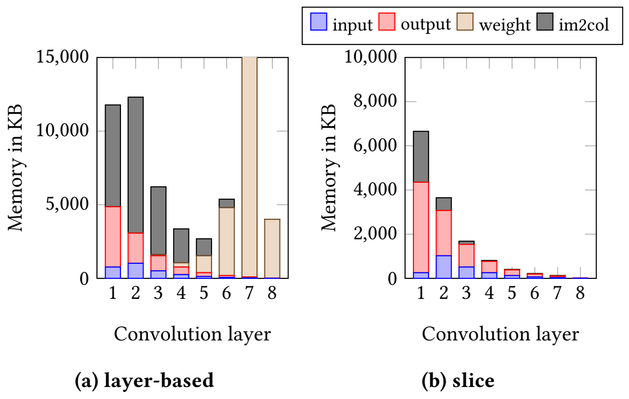


Figure 4: Comparison of the input, output, weight, and *im2col* memory buffer sizes for each convolution layer of the Darknet Ref model for (4a) layer-based partitioning and (4b) slice techniques.

In this scheme we can define a number of slices/channels n, where $n \leq C_{in}$, such that

$$n = \begin{cases} \lfloor \mathcal{P} / \hat{\mathcal{M}} \rfloor & \text{if } \mathcal{P} \leq \mathcal{M} \\ C_{in} & \text{otherwise} \end{cases} \tag{2}$$

This means that we can take at most n slices to the secure world to compute the convolution operation without breaking the memory constraint. After the operation at each step, the output values are aggregated to the output matrix, so that after the final step we have the actual output matrix, which is used as the input to the next layer.

For the case of connected layers, there is no concept of kernels and *im2col* transformation. Moreover, the number of output neurons is specified in the CNN property as a hyperparameter, and the input is transformed to a one-dimensional vector. Hence, we create slices by output neurons for each connected layer. We can modify the above equations to get the buffer sizes of different components of a slice for connected layers as follows:

$$\mathcal{I} = 1 \times 1 \times O_{l-1} \qquad \mathcal{K} = \mathcal{I} \times \mathcal{O} \qquad \hat{\mathcal{K}} = \mathcal{I}$$

$$\mathcal{M}_l \approx \mathcal{I} + \mathcal{O} + \mathcal{K} + \theta \qquad \hat{\mathcal{M}}_l \approx \mathcal{I} + \mathcal{O} + \hat{\mathcal{K}} + \theta$$

Connected layers are usually at the ends of the CNNs whose total memory buffer sizes are dominated by the sizes of the weights, as seen in Figure 5. We can see that slicing reduces the required weights for each layer. So in that case, for each slice we compute only n output neurons following equation 2, where $n \leq \mathcal{O}$. The other layers of CNNs, such as the pooling (i.e., average pool, max pool), dropout, the softmax, etc., do not need to be further fragmented as the buffer sizes of these layers are insignificant. Therefore, when it comes to layers other than the convolution and connected in the CNN, they are treated as a single layer in the trusted memory.

T-Slices defines the set of slice dynamically such that at any point of computation the required memory buffer never overflows the TA memory limit. Therefore, at any layer execution, the peak memory would be the TA memory limit itself. This ensures the execution of the entire DL model in the limited memory of TrustZone.

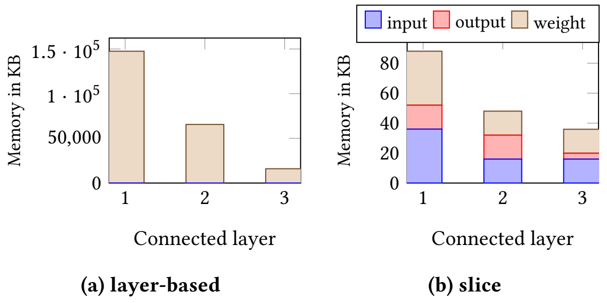


Figure 5: Comparison of the input, output, and weight memory buffer sizes for each connected layer of the ALEXNET model for (5a) layer-based partitioning and (5b) slice techniques.

However, as discussed in §4, the existing static layer-wise partitioning schemes lack a proper memory management plan by following a traditional approach of allocating the necessary data buffers before the start of the execution, severely limiting the unoccupied memory in the TrustZone. As a result, more memory-intensive models fail to run in the TrustZone for lack of available trusted memory.

Therefore, T-Slices follows an *on-demand* parameter loading scheme as an optimized memory management plan. More specifically, instead of allocating trusted memory buffers for all parameters beforehand, our framework occupies trusted memory buffers for only the segment of parameters needed for processing the current batch of slices. Furthermore, after the current batch of slices are processed, it replaces the memory buffers containing parameters with the new parameters for the next batch and any intermediate results with new intermediate results of subsequent operations. As a result, by reusing the memory buffers, T-Slices optimizes the trusted memory consumption for memory-constrained devices.

Algorithm 1 sketches how the T-Slices framework processes the prediction task. Utilizing the hyperparameters we first build a *network graph* in the secure world that incorporates all the layer dependencies as well as the memory requirements. When the prediction procedure starts, we go through each layer and calculate the slice properties, which includes slice size, number of slices per batch, and number of slices per layer. These properties are measured based on the available trusted memory in the TrustZone for that particular layer's data using the network graph. We next extract the parameters according to the number of slices per batch and load it into the trusted memory before decryption. For the first layer only, the input is also loaded in a similar way. For the other layers, the input slices are considered from the previous layer's output that is already placed in the trusted memory. This *on-demand* data loading strategy helps optimize the memory usage in the TrustZone.

After the set of slices are dispatched to the trusted memory, the forward layer operation is executed and the intermediate output is stored in the output matrix. Then, we fetch the next batch of slices, replace the previous ones, and perform the next operation in a similar way, except that the intermediate output is appended with the previous output in the the output matrix. This procedure continues until all the slices are processed for that particular layer.

Algorithm 1 DL Inference with T-Slices

```
1: procedure ForwardNetworkSW($idx, n$)
2:     $l \leftarrow g.layers[idx]$
3:     $l.forward(g)$
4:     if all slices are executed then
5:         $g.input = l.output$
6:         $l.clearBuffers()$
7:     if final layer then
8:         return $encrypt(l.output)$
9: procedure ForwardNetwork($g, W, I$)
10:    for each layer $l \in g.layers$ do
11:        if $l$ contains parameters then
12:            $s \leftarrow getSliceProperties(l.index)$                ▷ from SW
13:            $n \leftarrow s.numSlicePerBatch()$
14:            while $n \leq s.totalSlices()$ do
15:                $p \leftarrow readParameters(W, s, n)$
16:                $loadParamsSW(p, l.index)$                   ▷ decrypt in SW
17:                if first layer then
18:                    $i \leftarrow readInput(I, s, n)$
19:                    $loadInputSW(i, l.index)$                ▷ decrypt in SW
20:                $forwardNetworkSW(l.index, n)$
21:                $n \leftarrow n + s.numSlicePerBatch()$          ▷ process next batch
22:        else
23:            $forwardNetworkSW(l.index, n = -1)$
24: procedure startPredict($H, W, I$)           ▷ $H$=hyper, $W$=param, $I$=input
25:    load $H$ and encrypted $W$ in NW                       ▷ NW=Normal World
26:    $g \leftarrow constructNetworkGraphSW(H)$                 ▷ SW=Secure World
27:    $forwardNetwork(g, W, I)$
```

If there is no matrix multiplication involved for a layer operation (e.g., *pool* layers), then we process the layer in a conventional way, since memory requirement of these operations is trivial. To reuse available trusted memory, we replace the memory buffers containing the parameters, intermediate results, and input with new data after each batch of slice execution unless there's a dependency with an upcoming layer, which we can easily find out from the network graph built from the hyperparameters. After the final layer, the output probability scores are encrypted and returned to the application.

6 INFORMATION LEAKAGE ANALYSIS

In this section we analyze how T-Slices ensures data confidentiality. Before execution, the DL model is kept encrypted in the local storage of the device maintained by the untrusted OS. Since the model is encrypted, it cannot leak any information to the untrusted parties. Then, the input is received, which could be either encrypted or unencrypted depending on the device or application. As our framework does not produce the input, it provides options to handle both versions of input data. Securing the input data before it enters the framework is beyond our scope.

When the model begins the prediction procedure, it creates slices as explained in §5.3 and dispatch the data and parameters of slices to the secure world via the shared memory buffer. As the shared memory buffer of ARM TrustZone is not secure and controlled by untrusted OS, we copy the data from the shared buffer to the trusted memory before decryption. Then, the computation for the respective slice operation takes place, where the generated output and any intermediate data are stored in the trusted memory. As all the computation occurs inside the TrustZone and the results are always kept in the secure world, this process does not leak any information.

Finally, the softmax layer generates the output probability values from 0 to 1 for each of the classification labels. We return a true boolean value only for the class label of maximum probability and a false boolean value for the other class labels. To further restrict the information leakage, we encrypt the prediction output as well before passing it to the normal world. As a result, only the encrypted model and prediction results are exposed to the untrusted environment, which makes it difficult to obtain any insights about the original data.

The following two case studies validate the effectiveness of our framework against two prevalent privacy attacks: model inversion (MI) [11] and membership inference attacks (MIA) [38].

6.1 Model Inversion Attack

In an MI attack, the adversary tries to reconstruct or recover the training data or any sensitive attributes from the trained ML model. The attacker access could be either *white-box* or *black-box*. In the white-box setting, attackers have access to and knowledge of the entire model, including the parameters, intermediate results, and classification output. In the black-box setting, attackers can only make prediction queries to the model to get the prediction results, but they do not have any knowledge of the model.

In our case study, the attacker's goal is to invert the full vector of pixel intensities that corresponds to an image in the training data. In other words, the attacker tries to reconstruct an image from the training set by observing various details of the model. To do so, the attacker leverages a gradient descent-based approach [11] that defines a cost function based on the output label of the model and some auxiliary information (e.g., error statistics, number of instances in the training set, etc.) and iteratively transforms randomly chosen candidate vectors to a reconstructed image from the training set.

In our design, the black-box attacker can only perceive the encrypted DL classification scores through the queries, which significantly limits the efficacy of estimating pixel values of an image from the training set. Since the model parameters are kept encrypted in the normal world and all other intermediate results are processed in the secure world of TrustZone, the white-box MI attacker can only perceive the encrypted data and prediction output. In addition, the attacker does not have access to the training data information. Therefore, the lack of useful and adequate knowledge severely restricts the effectiveness of this attack with T-Slices.

6.2 Membership Inference Attack

In MIA, the adversary tries to discover whether a given data sample or instance is a part of the training dataset for the particular trained ML model. Like the MI attack, MIA also exhibits *white-box* and *black-box* settings. The adversary proceeds using a shadow training technique [38] that trains a set of attack models to distinguish the target model's behavior from the prediction output depending on whether the input instance is a member of the training set. In the black-box setting, the attackers only have access to the model output, whereas the white-box attack utilizes prior knowledge, such as statistics of the training dataset or a noisy training dataset.

In our design, the black-box MIA cannot succeed since the attacker must decipher the encrypted prediction outputs first to train the shadow models. Moreover, the white-box attacker does not have access to the prior knowledge about the training data and cannot gain information from the trained model since the parameters and other data are only deciphered in the secure memory of the TrustZone. Thus, T-Slices defends against both attacks.

Table 3: Evaluation models and datasets

Model	# Layers	# Conv. Layers	Dataset	Pre-trained Model Size (MB)
LeNet	10	2	MNIST	0.2
Cifar_Small	12	7	CIFAR10	0.08
Vgg-7	13	6	CIFAR10	0.26
Vgg-7	13	6	CIFAR100	0.3
Cifar	18	10	CIFAR10	30.7
Tiny Darknet	22	16	ImageNet1k	4.2
Extraction	27	21	ImageNet1k	93.8
Darknet Ref	16	8	ImageNet1k	29.3
Alexnet	14	5	ImageNet1k	249.5
InceptionV3	145	94	ImageNet1k	95.5

7 EVALUATION

The proposed framework is next evaluated by executing different CNN models in the secure world with restricted trusted memory. Recall that we execute all operations of CNN in the secure world without modifying or redesigning any portion of the network to conserve the original output.

7.1 Experimental Setting

We begin with a description of the dataset and CNN models we use for the training and prediction. Also, we describe the devices used to run the experiments.

Dataset and Models. Table 3 lists the architectural details of the CNNs used in the evaluation, along with their pre-trained model sizes. We use a total of three image datasets for the experiments: Mnist [22], Cifar [21], and ImageNet [7]. These are used to train different CNNs, including LeNet [22], Vgg-7 [39], Cifar [21], Tiny Darknet [36], Extraction [41], Darknet Ref [35], Alexnet [20], and InceptionV3 [42]. Particularly, we train LeNet with Mnist, Vgg-7 and Cifar with CIFAR-10 and CIFAR-100, and the others with ImageNet1k.

Device Configuration. We use Op-Tee [25] as our Tee that allows the development and integration of secure services and applications under trusted execution environments. To evaluate soundness of our system under realistic conditions, we perform experiments with a physical STM32MP157C-DK2 board [40] from a popular STM32MP1 series of microprocessors and a Raspberry Pi 3 Model B (RPi3B) [34]. The STM32MP157C-DK2 contains ARM-based dual Cortex-A7 32-bit and Cortex-M4 32-bit MPUs with 4-Gb DDR3L memory, where the Cortex-A7 processor provides the TrustZone security features on this board. Additionally, we utilize RPi3B to test the prototype and evaluate the performance of T-Slices on embedded devices. Although, we implemented and experimented T-Slices with Cortex-A series of ARM TrustZone, the design of our framework is not specific to Cortex-A and is applicable to any platform that supports ARM TrustZone including Cortex-M.

Generally, 32 MB of memory is assigned to Op-Tee, where 1 MB is for Tee RAM, 1 MB for non-secure RAM, and the remaining 30 MB are for Trusted Applications (TA) RAM [1]. Every TA instance owns its private heap, stack, code, and data memory within the TA RAM, where the stack and heap sizes are defined at build time by macros `TA_STACK_SIZE` and `TA_DATA_SIZE`, respectively [24].

For training the CNN models, we use a system containing an 8-core i7-6700 (Skylake) processor operating at 3.4GHz, running Ubuntu 18.04 with 64GB RAM. Our implementation is based on both Darknet [35] and DarkneTZ [28] frameworks. Although our T-Slices framework is independent of these two frameworks and can be used with other DL systems as well.

7.2 Trusted Memory Consumption

We evaluate T-Slices by measuring the memory usage of the CNN models' inference processes entirely executed with TrustZone and compare with the state-of-the-art static layer-wise partitioning scheme DarkneTZ. To conduct comprehensive experiments, we modify the baseline DarkneTZ for layer-wise execution in TrustZone, but with *on-demand* parameter loading strategy [1]. With this optimized approach, we could successfully run more memory-intensive CNNs entirely in TrustZone with DarkneTZ* since any layer's peak memory is within the TA memory limit.

Secure World Memory Usage. For the experiments, we set TA heap size as the optimal heap memory, which is the *peak memory* required to execute a single layer (or a slice for T-Slices) of a particular CNN. That way, static layer-wise partition methods can execute all the layers in the secure world sequentially. Similarly, T-Slices can execute all the slices sequentially when TA heap size is set as the largest slice size. All TAs share the TA RAM, hence this strategy helps developers to define and allocate more trusted memory for other TAs. Table 4 compares the peak memory sizes required to execute a layer in DarkneTZ (with and without *on-demand* parameter loading scheme) and a slice in T-Slices. For simplicity, we take the ceiling of the numbers in the table as the TA heap size. Figure 2 shows a graphical comparison of peak memory usage between layer-wise partitioning following only *on-demand* parameter loading approach and slicing.

It is evident from the table that our *on-demand* parameter loading strategy notably reduces the memory requirement for each layer in the secure world. For DarkneTZ*, it shows a 47% (for Alexnet) to 90% (for InceptionV3) and 96% (for LeNet) decrease in required TA memory relative to DarkneTZ with on average 81% reduction. Without this technique, DarkneTZ* fails to execute the whole DL model in the secure world that occupies more than 30MB of peak memory. For instance, form Table 4, DarkneTZ can execute only LeNet and Vgg-7 entirely in the trusted memory of TrustZone. On the other hand, the *on-demand* memory management allows DarkneTZ* to process more memory-intensive models in the limited untrusted memory of TrustZone by minimizing the required peak memory to less than TA RAM.

Table 4 demonstrates that T-Slices overall reduces the peak memory consumption by 72% on average, and achieves a 47% (for Tiny Darknet) to 98% (for Alexnet) decrease in required TA memory

[1]Symbol * herein refers to modified DarkneTZ with *on-demand* parameter loading strategy for each layer

Table 4: Peak Memory Usage (MB) Comparison between Layer and Slice in TrustZone

Model	DarkneTZ per Layer	DarkneTZ* per Layer	T-Slices per Slice	% Decrease†
LeNet	7	0.25	0.1	60
Vgg-7	7	0.7	0.2	71
Cifar	45	10.5	1.25	88
Tiny Darknet	71	9.5	5	47
Darknet Ref	88	18.5	6.5	65
Extraction	163	22.6	5.6	75
Alexnet	272	144	2.75	98
InceptionV3	337	33	9	73

* with *on-demand* parameter loading scheme
† decrease from DarkneTZ* to T-Slices

relative to DarkneTZ*. For instance, the third convolution layer of InceptionV3 needs little more than 33MB of memory, which is the highest required memory to execute a layer for the model. Since it exceeds the trusted RAM size, that layer will not be allowed to execute in the secure world. On the other hand, T-Slices is able to run all the layers of these models since slice is small-scaled and can fit into the available limited trusted memory, and thus dynamically execute the layer operation in multiple steps. In fact, our framework can execute all the models in Table 4 with only 9MB of memory, which is a 70% reduction in required TA RAM. This reduction in TA size enables T-Slices to run on-device learning in low-end devices, as discussed in §2.2. Moreover, it allows to keep the TA size in TrustZone as small as possible to minimize the attack surface [30].

The above results therefore confirm that the traditional layer-wise static DNN partitioning schemes fail to run at least a single layer for intricate and resource-hungry DNNs (i.e., Alexnet, InceptionV3) since the peak layer size exceeds the TA size limit. We conclude that T-Slices can significantly reduce TA memory requirement and dynamically load memory-intensive DNNs to limited trusted memory of TrustZone without altering the network model or original prediction accuracy.

7.3 Prediction Time Overhead

We want to discover how the integration of slices alters the time overhead of a CNN model's inference process without affecting the accuracy of the original result, and study the trade-off with trusted memory size. We measure the time overhead and compare it with DarkneTZ. We try to run all the layers of the CNN in the trusted memory for DarkneTZ as well following the *on-demand* parameter loading approach discussed before. We run each experiment 10 times with a predefined set of test images from the corresponding dataset and report the average.

Table 5 presents the execution time measured in seconds on STM32MP157C-DK2. The results show that T-Slices improves the inference execution time compared to DarkneTZ* when the entire model is executed in the secure world. T-Slices achieves at most 53% (for Cifar) and at least 2% (for Tiny Darknet) improvement in execution time relative to DarkneTZ* with on average 29% improvement. The modified baseline could not execute more intricate models (e.g., Alexnet and InceptionV3) entirely in the TrustZone, since the layer peak memory surpasses the the TA RAM limit.

Op-Tee runs and executes the TAs in an on-chip memory (OCM) by default, as ARM TrustZone recommends storing all the code and data in the secure world in OCM [3]. OCM is faster and more secure because it blocks physical attacks [51]. Consequently, it is expensive to use in the system-on-chip platforms, and generally its size is kept small (e.g., a few hundred KB). Our STM32MP157C-DK2 board uses an internal 708KB SRAM to load the Op-Tee and the TAs.

Typically, Op-Tee leverages the *paging* mechanism to run the secure world OS and the TAs on the limited OCM, and uses less secure DRAM as the backing store where 4KB pages are encrypted and integrity protected [24]. Op-Tee's pager swaps memory pages on-demand from/to OCM to/from DRAM with encryption/decryption and integrity check [52]. The pager efficiency depends on OCM size and TA trusted memory size. Therefore, the expensive paging mechanism slows the TA execution in TrustZone considerably in favor of code and data security, especially when more memory is allocated for the TA in the secure world. As a result, the inference procedure of large CNN models with TrustZone incurs notable execution time overhead with STM32MP157C-DK2, as shown in Table 5. Nevertheless, T-Slices performs significantly faster than the layer-based partitioning scheme since, as discussed in §7.2, it allocates significantly less trusted memory to run the TA.

However, the TA execution time on RPi3B is notably accelerated as shown in Table 6, since Op-Tee runs and executes the TAs in the DRAM and the expensive paging is disabled. In fact, the results show that the execution time of DarkneTZ* and T-Slices are similar on RPi3B, which indicates that our framework does not incur any prediction time overhead. As a result, T-Slices can be deployed to an edge device in realistic setting and achieve similar performance as the baseline.

Figure 6 shows a graphical comparison of prediction time for DarkneTZ, DarkneTZ*, and T-Slices. Recall that, because of the memory constraint, DarkneTZ could execute only a few layers of the CNN model in TrustZone. We denote these range of layers as $x \longrightarrow y$ in the figure, where x and y are the starting and ending layer numbers of the range that are operated within the trusted memory, respectively. For selecting the layers we follow baseline's approach in the original paper, which is choosing the last layers first (i.e., from rightmost to leftmost) in the CNN. Consequently, more compute-intensive and memory-hungry layers are executed in the normal world. As operations performed in the normal world is faster than the operations performed in the secure world [44, 45], DarkneTZ performs slightly faster than T-Slices, although, exposing sensitive information to the adversary in the process [12].

Micro-Benchmarks. To investigate how different components of the prediction procedure contribute to overall execution time for T-Slices, we conduct benchmark operations. Figure 7 depicts the benchmark results for STM32MP157C-DK2, which reports the time for three crucial sub-operations as a percentage of total execution time for 6 CNN architectures. The three sub-operations are: the context switching between the normal world and secure world, the execution of all the slices in TrustZone (i.e., forward pass), and initializing the network in TrustZone that includes allocating/de-allocating memory buffers, loading the parameters, and decryption

Table 5: Execution time (seconds) of different CNN models on STM32MP157C-DK2

CNN	Dataset	DarkneTZ*	T-Slices	% Improvement
LeNet	MNIST	2.44	2.10	14
Cifar_Small	CIFAR10	3.49	3.24	7
Vgg-7	CIFAR10	11.93	6.38	47
Cifar	CIFAR10	608.04	285.07	53
Tiny Darknet	ImageNet1k	874.58	859.34	2
Extraction	ImageNet1k	1244.84	615.56	51
Darknet Ref	ImageNet1k	1175.69	815.55	31
Alexnet	ImageNet1k	✗	1219.31	✗
InceptionV3	ImageNet1k	✗	1928.41	✗

*with *on-demand* parameter loading
✗: Unable to execute due to not enough trusted memory

Table 6: Execution time (seconds) of different CNN models on RPi3B

CNN	Dataset	DarkneTZ*	T-Slices	% Improvement
LeNet	MNIST	0.092	0.092	0
Cifar_Small	CIFAR10	0.19	0.19	0
Vgg-7	CIFAR10	0.309	0.307	1
Cifar	CIFAR10	30.43	30.26	1
Tiny Darknet	ImageNet1k	14.72	14.71	0
Extraction	ImageNet1k	116.57	116.24	0
Darknet Ref	ImageNet1k	18.81	18.78	0
Alexnet	ImageNet1k	✗	44.19	✗
InceptionV3	ImageNet1k	✗	468.1	✗

*with *on-demand* parameter loading
✗: Unable to execute due to not enough trusted memory

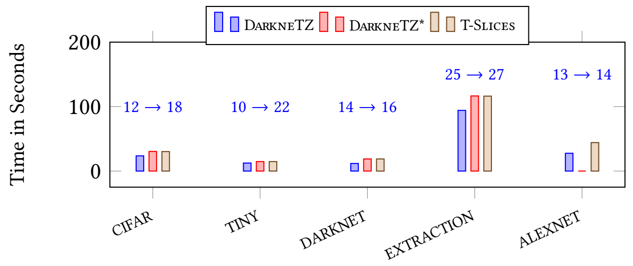


Figure 6: Execution Time Comparison on RPi3B for DarkneTZ, DarkneTZ*, and T-Slices. $x \longrightarrow y$ indicates the ranges of layers executed in TrustZone for DarkneTZ.

of parameters. The other parts of the execution (e.g., file I/O) are defined as the miscellaneous operations.

The figure shows that the operations of the slice (i.e., the forward executions) dominates the overall execution time. To discover why the forward execution takes the majority of the time for the prediction process, we further investigate its different components. More specifically, we observe the time overhead of Gemm operations and the *im2col* operations in the forward pass. Figure 8 shows that the Gemm operation is accountable for a substantial part of the time overhead, especially for more complex models. As a result, we can deduce that the design of T-Slices does not significantly contribute to the time overhead of the inference process in the device; rather, the major portion of the overhead comes from the conventional DL operators and Op-Tee's secure paging mechanism with STM32MP157C-DK2.

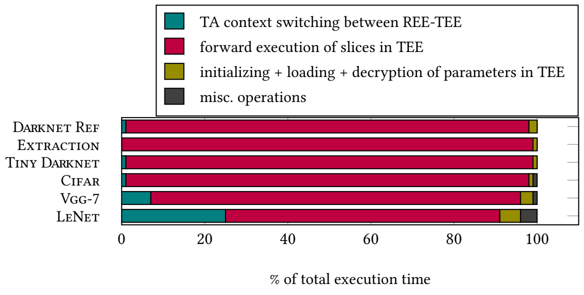


Figure 7: Benchmark results showing the % of total execution time occupied by different components

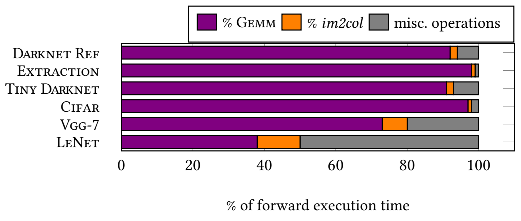


Figure 8: Benchmark results showing the % of total forward execution time occupied by Gemm and *im2col*

8 LIMITATIONS & FUTURE WORK

Although our approach facilitates secure execution of some resource-intensive CNN models in memory-constrained TrustZone of ARM edge devices, there exist other CNNs that are prepared for systems with vast resources and that remain unsuitable for embedded devices. For example, vgg-16 [39] has a pre-trained model size of 528MB and peak memory of approximately 923MB, and Yolov3 [37] has a peak memory of approximately 840MB. In future work, we plan to investigate the deployment of these complex models on edge/IoT devices. Moreover, in this paper, we specifically target and experiment with the CNN architectures from DL models. However, we aim to apply our method to other DL architectures, such as Recurrent Neural Networks (RNNs) in the future.

The susceptibility of Tees to side-channel attacks is well established by prior studies. These attacks, based on power consumption, operation timing, or memory cache usage, may leak sensitive information and endanger data privacy [26, 50]. For this particular case, these attacks could leak sensitive DL information such as model parameters and intermediate representations of learning operations. Therefore, in the future we plan to investigate the capability of these attacks on ARM TrustZone and our framework.

9 RELATED WORK

Recent research has focused on secure on-device learning without sacrificing prediction accuracy, especially on resource-constraint embedded devices. A preliminary study on the limited capacity of trusted memory in TrustZone introduced theoretical models to partition CNN execution to cope with this problem [45]. More recently, DarkneTZ [28] proposed a layer-based partitioning mechanism with TrustZone to securely execute CNN layers in the secure world. Although it executes simple CNN models (e.g., Vgg-7) entirely in the TrustZone's limited memory, more complex CNNs (e.g., Alexnet) only execute the last few layers in the secure world, since the TrustZone's limited memory cannot handle the memory-intensive layers of these models. However, layers that execute outside of the secure world expose information to the untrusted normal world, raising data privacy concerns. In contrast, T-Slices executes the whole CNN model in the resource-constraint TrustZone to protect model parameters, intermediate results, and even the outputs from untrusted access, and thus ensure data privacy with preserving model prediction accuracy.

In recent years, cryptographic approaches, such as homomorphic encryption [31] and SMC [29], have gained considerable attention for designing privacy-preserving ML solutions. FHE tends to be costly due to inevitable requirements of time and space, which may not be a feasible solution for real-time systems and edge devices [19]. Moreover, SMC-based solutions to ensure data privacy in ML-based approaches incur unwanted additional communication costs, which escalate the latency of edge applications [29]. Another popular technique to mitigate the private data leakage of ML models is differential privacy [49]. Nevertheless, it usually suffers from prediction accuracy loss and always induces a trade-off between statistical accuracy and data privacy [10]. In our approach, we leverage the Tees for edge devices to protect DL models and their entire execution without affecting the accuracy, and thus effectively preserve privacy.

10 CONCLUSION

In this paper, we address the memory restriction of DL models for embedded devices and propose a framework called T-Slices to fit every layer of memory-intensive DL models into the secure world of an ARM TrustZone-based embedded device that has limited trusted memory. We perform protected inference of pre-trained DL models utilizing the security features of TrustZone that help to protect the confidentiality of data and the model parameters. Our framework does not change or redesign any component of the original DL model, thereby retaining the model's original prediction accuracy without any information loss. A series of case studies demonstrate the effectiveness of the framework against different privacy attacks. We evaluate T-Slices with various CNN models on real embedded devices containing TrustZone and study the trade-off between execution time and memory consumption. The results show that T-Slices on average achieves 72% reduction in peak memory consumption and 29% improvement in execution time.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful feedback. This work was supported in part by the Texas A&M Engineering Experiment Station on behalf of its SecureAmerica Institute, DARPA awards FA8750-19-C-0006 and N6600121C4024, NSF awards DMS-1737978, DGE-2039542, OAC-1828467, OAC-1931541, and DGE-1906630, ONR awards N00014-21-1-2654, N00014-17-1-2995, and N00014-20-1-2738, ARO award W911NF2110032, and IBM faculty award (Research).

REFERENCES

[1] Julien Amacher and Valerio Schiavoni. 2019. On the Performance of ARM TrustZone. In *Proc. IFIP Int. Conf. Distributed Applications and Interoperable Systems.* 133–151.

[2] Andrew Anderson, Aravind Vasudevan, Cormac Keane, and David Gregg. 2017. Low-memory Gemm-based Convolution Algorithms for Deep Neural Networks. *arXiv Preprint* 1709.03395 (2017).

[3] ARM. 2009. *ARM Security Technology: Building a Secure System using TrustZone Technology.* White paper PRD29-GENC-009492C. ARM.

[4] Mamoun A Awad and Latifur R Khan. 2007. Web navigation prediction using multiple evidence combination and domain knowledge. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 37, 6 (2007), 1054–1062.

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Proc. Machine Learning and Systems.*

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

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-scale Hierarchical Image Database. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition.*

[8] Alexey Dosovitskiy and Thomas Brox. 2016. Inverting Visual Representations with Convolutional Networks. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 4829–4837.

[9] Marat Dukhan. 2019. The Indirect Convolution Algorithm. *arXiv Preprint* 1907.02129 (2019).

[10] Vitaly Feldman, Konstantin Kakaes, Katrina Ligett, Kobbi Nissim, Aleksandra Slavkovic, and Adam Smith. 2020. *Differential Privacy: Issues for Policymakers.* White paper. Simons Institute Theory of Computing, University of California at Berkeley.

[11] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proc. ACM Conf. Computer and Communications Security.* 1322–1333.

[12] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Confidential Inference Via Ternary Model Partitioning. *arXiv Preprint* 1807.00969 (2018).

[13] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *Proc. Int. Conf. Learning Representations.*

[14] IBM X-Force® Research. 2017. The Weaponization of IoT Devices. www.ibm.com/downloads/cas/6MLEALKV.

[15] Md Shihabul Islam, Mustafa Safa Ozdayi, Latifur Khan, and Murat Kantarcioglu. 2020. Secure IoT data analytics in cloud via Intel SGX. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD).* IEEE, 43–52.

[16] Md Shihabul Islam, Harsh Verma, Latifur Khan, and Murat Kantarcioglu. 2019. Secure real-time heterogeneous iot data management system. In *2019 first IEEE international conference on trust, privacy and security in intelligent systems and applications (TPS-ISA).* IEEE, 228–235.

[17] 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 *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 2704–2713.

[18] Kyungtae Kim, Chung Hwan Kim, Junghwan "John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. 2020. Vessels: Efficient and Scalable Deep Learning Prediction on Trusted Processors. In *Proc. ACM Sym. Cloud Computing.* 462–476.

[19] Çetin Kaya Koç. 2020. Formidable Challenges in Hardware Implementations of Fully Homomorphic Encryption Functions for Applications in Machine Learning. In *Proc. ACM Workshop Attacks and Solutions in Hardware Security.*

[20] Alex Krizhevsky. 2014. One Weird Trick for Parallelizing Convolutional Neural Networks. *arXiv Preprint* 1404.5997 (2014).

[21] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images.* Technical Report. University of Toronto.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[23] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. 2019. Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In *Proc. Annual Int. Conf. Mobile Computing and Networking.*

[24] Linaro. 2022. OP-TEE Architecture. https://optee.readthedocs.io/en/latest/architecture/index.html.

[25] Linaro. 2022. Open Portable Trusted Execution Environment. www.op-tee.org.

[26] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. Armageddon: Cache Attacks on Mobile Devices. In *Proc. USENIX Security Sym.* 549–564.

[27] Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding Deep Image Representations By Inverting Them. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 5188–5196.

[28] 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 *Proc. Int. Conf. Mobile Systems, Applications, and Services.* 161–174.

[29] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-party Machine Learning on Trusted Processors. In *Proc. USENIX Security Sym.* 619–636.

[30] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone. In *Proc. USENIX Annual Technical Conf..* 537–554.

[31] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2017. Privacy-preserving Deep Learning Via Additively Homomorphic Encryption. *IEEE Trans. Information Forensics and Security* 13, 5 (2017), 1333–1345.

[32] Sandro Pinto and Nuno Santos. 2019. Demystifying ARM TrustZone: A Comprehensive Survey. *ACM Computing Surveys* 51, 6 (2019), 1–36.

[33] Qualcomm. 2020. We Are Making AI Ubiquitous. www.qualcomm.com/news/onq/2020/06/we-are-making-ai-ubiquitous.

[34] Raspberry Pi. 2022. Raspberry Pi 3 Model B. https://www.raspberrypi.com/products/raspberry-pi-3-model-b/.

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

[36] Joseph Redmon. 2022. Tiny Darknet. pjreddie.com/darknet/tiny-darknet.

[37] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv Preprint* 1804.02767 (2018).

[38] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP).* IEEE, 3–18.

[39] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-scale Image Recognition. In *Proc. Int. Conf. Learning Representations.*

[40] ST Microelectronics. 2022. Discovery Kit with STM32MP157C MPU. www.st.com/en/evaluation-tools/stm32mp157c-dk2.html.

[41] 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 *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 1–9.

[42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 2818–2826.

[43] TensorFlow. 2022. TensorFlow Lite. www.tensorflow.org/lite.

[44] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287* (2018).

[45] Peter M. VanNostrand, Ioannis Kyriazis, Michelle Cheng, Tian Guo, and Robert J. Walls. 2019. Confidential Deep Learning: Executing Proprietary Models on Untrusted Devices. *arXiv Preprint* 1908.10730 (2019).

[46] Johannes Winter. 2008. Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms. In *Proc. ACM Workshop Scalable Trusted Computing.* 21–30.

[47] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized Convolutional Neural Networks for Mobile Devices. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 4820–4828.

[48] I-Ling Yen, Jayabharath Goluguri, Farokh Bastani, Latifur Khan, and John Linn. 2002. A component-based approach for embedded software development. In *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002.* IEEE, 402–410.

[49] Lei Yu, Ling Liu, Calton Pu, Mehmet Emre Gursoy, and Stacey Truex. 2019. Differentially Private Model Publishing for Deep Learning. In *Proc. IEEE Sym. Security & Privacy.* 332–349.

[50] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Y. Thomas Hou. 2016. TruSpy: Cache Side-channel Information Leakage From the Secure World on ARM Devices. *IACR Cryptology ePrint Archive* 2016 (2016).

[51] Shijun Zhao, Qianying Zhang, Yu Qin, Wei Feng, and Dengguo Feng. 2019. Minimal Kernel: An Operating System Architecture for TEE to Resist Board Level Physical Attacks. In *Proc. Int. Sym. Recent Advances in Intrusion Detection.* 105–120.

[52] Shijun Zhao, Qianying Zhang, Yu Qin, Wei Feng, and Dengguo Feng. 2019. SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE. In *Proc. ACM Conf. Computer and Communications Security.* 1723–1740.

[53] Li Zhou, Hao Wen, Radu Teodorescu, and David HC Du. 2019. Distributing Deep Neural Networks with Containerized Partitions at the Edge. In *Proc. USENIX Workshop Hot Topics in Edge Computing.*