MirrorNet: A TEE-Friendly Framework for Secure On-device DNN Inference

Ziyu Liu, Yukui Luo, Shijin Duan, Tong Zhou, and Xiaolin Xu
Northeastern University, Boston, MA, USA
{liu.ziyu4, luo.yuk, duan.s, zhou.tong1, x.xu}@northeastern.edu

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323746

***Abstract*—Deep neural network (DNN) models have become prevalent in edge devices for real-time inference. However, they are vulnerable to model extraction attacks and require protection. Existing defense approaches either fail to fully safeguard model confidentiality or result in significant latency issues. To overcome these challenges, this paper presents MirrorNet, which leverages Trusted Execution Environment (TEE) to enable secure on-device DNN inference. It generates a TEE-friendly implementation for any given DNN model to protect the model confidentiality, while meeting the stringent computation and storage constraints of TEE. The framework consists of two key components: the backbone model (*BackboneNet*), which is stored in the normal world but achieves lower inference accuracy, and the Companion Partial Monitor (CPM), a lightweight mirrored branch stored in the secure world, preserving model confidentiality. During inference, the CPM monitors the intermediate results from the BackboneNet and rectifies the classification output to achieve higher accuracy. To enhance flexibility, MirrorNet incorporates two modules: the CPM Strategy Generator, which generates various protection strategies, and the Performance Emulator, which estimates the performance of each strategy and selects the most optimal one. Extensive experiments demonstrate the effectiveness of MirrorNet in providing security guarantees while maintaining low computation latency, making MirrorNet a practical and promising solution for secure on-device DNN inference. For the evaluation, MirrorNet can achieve a 18.6% accuracy gap between authenticated and illegal use, while only introducing 0.99% hardware overhead.**

***Index Terms*—Machine Learning, Security, Trusted Execution Environment**

I. Introduction

Deep neural networks (DNN) are designed to automatically learn the complex pattern and feature representation of the input data, which has been successfully used for various applications, like facial recognition [1], autonomic driving [2], and health care monitoring [3]. However, given concern about the data privacy, many users of the DNN model prefer not to share their private data with the online server. As a result, there is a growing trend in implementing DNN models on edge devices [4]. By deploying DNN models directly on edge devices, such as smartphones, the model inference can be performed locally to protect user privacy, since most sensitive data remains in device. Moreover, DNN inference on edge can achieve lower latency compared to cloud computing.

Despite its encouraging performance, deploying high-performance DNN models on edge devices is vulnerable to model extraction [5]. Specifically, attackers can extract the model architecture and weights, then transplant it to the unauthorized device without claiming ownership or paying the patent fee. However, training a high-performance DNN model requires a large number of labeled data and substantial computational resources [6]. Therefore, these high-performance models are the intellectual property (IP) of the model owners and should be well-protected [7].

TABLE I: The comparison with previous works

	Privacy	Latency	Flexibility	Accuracy
DarkneTZ [8]	○	◐	●	●
eNNclave [9]	●	◐	○	○
Confidential DL [10]	●	○	●	●
ShadowNet [11]	◐	○	●	●
MirrorNet (ours)	●	●	●	●

○ Not covered; ● Covered; ◐ Partially covered

Several methods have been proposed for model protection, such as watermarking [12], non-transferable learning [13], and obfuscation [14]. Among the existing methods, we believe that Trusted Execution Environment (TEE)-based solutions are particularly well-suited for safeguarding DNN models on edge devices. TEE is commonly available in modern edge devices, such as ARM processors, and refers to an area (secure world) inside the main processor of the device that is separated from the system's main operating system (normal world). TEE ensures the confidentiality and integrity of data processing, making it a widely utilized technology for user authentication and key management on edge devices [15]. However, fully deploying DNN models inside TEE to achieve model protection is impractical, due to their increasing model size and the limited computational resources and storage memory within TEEs. For example, the Raspberry Pi 3B with ARM Cortex-A chip offers a maximum of 16MB memory for the TEE [16], which is incompatible with the state-of-the-art DNN models, e.g., over 100MB for ResNet-101 [17].

Although several previous works employed TEE to secure the DNN inference by uploading certain layers to TEE [8] or introducing extra masking and linear transformation in TEE between the execution of layers [11], these methods either cannot fully protect the model or incur a large overhead, as summarized in Tab.I and detailed in Sec. II-C. More importantly, all these existing methods fail to address a significant vulnerability associated with layer-wise dependency in DNN computation. For example, if the first layer is executed in the secure world and the second in the normal world, the intermediate feature maps might be leaked during the communication,

and the attacker could retrain the model with less effort. i.e., merely isolating a few layers within the secure world of TEE does not provide sufficient protection for maintaining model confidentiality (see results in Sec. III-A).

To overcome these design challenges, we propose a framework, namely MirrorNet, which aims to protect the functionality of the input DNN model by generating a TEE-friendly deployment for the given model. Additionally, it addresses the vulnerability of model extraction attacks by limiting attackers to only extract a poorly performing model. We take the architecture of the input DNN model as a backbone network, which is deployed in the normal world. Besides, we develop a lightweight *mirror network* based on the backbone network, namely Companion Partial Monitor (CPM), which is connected with the backbone but stored in the secure world. Through model training, we enable the entire model to achieve high accuracy, while intentionally degrading the accuracy of the backbone model in the normal world. For clarity and conciseness, we refer to both our framework and the combined model as MirrorNet. In summary, MirrorNet addresses the vulnerability of model extraction by safeguarding the lightweight mirror network within secure world. This lightweight network is designed to meet the computational and storage limitations of the TEE. It serves as a crucial component of the entire model, enabling authorized users to perform high-performance inference in conjunction with the backbone network.

The contributions of this work are summarized as follows:

- We identify and validate a neglected vulnerability in the existing DNN model protection methods, see Sec. III-A. For the first time, we take the re-training vulnerability into consideration in developing a TEE-friendly framework MirrorNet, for secure DNN inference on edge devices.
- We leverage the layer-wise and channel-wise dependency of DNN models to generate a mirrored network architecture named Companion Partial Monitor (CPM) for a given victim DNN model. As a lightweight network, CPM can be deployed in the limited secure memory of TEE but determines the performance of the overall inference. MirrorNet can be optimized with arbitrary strategies and can even be integrated with other purpose training.
- We explore numbers of potential mirrored model architectures and equip the MirrorNet framework with two components, namely *CPM Strategy Generator* and *Performance Emulator*, to selectively generate the optimal protection scheme.
- By deploying MirrorNet on a Raspberry Pi 3 Model B board, we validate its performance with input networks including LeNet-5 [18] and VGG networks [19]. We also train models on three datasets for inference, including MNIST [20], FashionMNIST [21], and CIFAR-10 [22], to evaluate the security level of our method. Experimental results demonstrate that our proposed MirrorNet framework achieves comparable accuracy to the input networks with low hardware overhead, e.g., 0.99% for VGG-7. On the other hand, illegal model extraction from the normal world can only obtain a compromised model, with an accuracy gap as high as 18.6%.

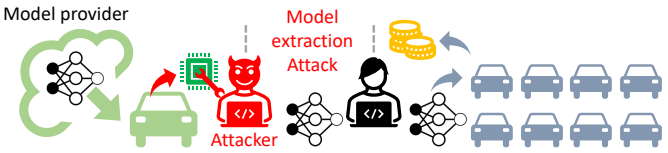

Fig. 1: The illustration of our threat model.

II. Background and Related Work

A. Threat Model

In our thread model, as shown in Fig. 1, we focus on real-time scenarios when executing security-sensitive applications, such as a self-driving system or an AI system used for face recognition. We assume a "knowledgeable" attacker whose object is to extract a well-performed model. The attacker can directly acquire sensitive information like the model architecture and weights of a DNN model in the normal world, or correctly infer them from some approaches, such as side-channels [23]. Besides, we assume the secure world, i.e., the TEE, is isolated and remains as a black box for the attackers, who are not able to infer the computation performed inside the secure world. Yet, the attacker can figure out when the secure world computation is invoked and monitor the data communication between the secure world and the normal world through the shared buffers. Since we focus on the model confidentiality issue during the usage of the TEE-friendly secured model, we assume our MirrorNet framework is trained offline and the normal world + secure world implementation is only conducted for the model inference procedure.

B. Trusted Execution Environment (TEE)

TEE is a secure area within a processor that provides a secure environment for executing code and processing data, guaranteeing confidentiality and integrity inside [24]. It helps isolate sensitive operations from the rest of the device and protect it from potential attackers or malware. Taking the ARM TrustZone as an example, it creates two virtual environments, namely *Normal World* and *Secure World*. The normal world is the normal operating environment for most applications. Differently, the secure world has its own isolated memory and peripherals, and provides an environment for running trusted applications with the help of security mechanisms. Although there are some security issues challenging the TEE execution, such as TrustZone [25], it is still the cornerstone of modern secure applications. Moreover, researchers have been dedicated to implementing DNN with the TEE because of its protection of confidentiality and integrity in recent years [10].

C. Related Works on DNN Protection

The DNN model execution on CPU has been proven vulnerable to information leakages, such as training data and model parameters [26]. In this context, one prominent concern is the model extraction attack, in which the attacker aims to duplicate or "steal" the DNN model through different approaches, such

as shadow training [27], constructing meta-models [28], and exploiting memory and timing side-channels [29].

Recent efforts have attempted to utilize TEE for securing DNN inference. However, the limited computation resources and memory in TEE make the implementation rather challenging. DarknetZ [8] partitions the model and put the last few dense layers inside TEE. However, the convolution layers of the victim model are still running in the normal world in plaintext format, presenting a vulnerability. Similarly, another work eNNclave [9] uses a few pre-trained and publicly available convolution layers as the feature extractor in the normal world and a followed user-defined dense layer as the classifier in TEE. However, it suffers from performance loss since the parameters of the feature extractor are fixed during retraining. Another approach is to protect the whole model with sequential executions. For instance, one work divides the model into partitions and executes each separately inside TEE [10], which causes an extremely long execution time. ShadowNet obfuscates the weights by masking them with linear transformations [11]. However, the transformation is insecure against "strong" attacker who can monitor the memory access pattern and extract the pattern of weights [30].

III. Challenges of DNN Protection with TEE

A. Inadequate Confidentiality Protection Against Retraining

In previous works like DarknetZ [8] and eNNclave [9], although the last few dense layers are protected inside the TEE, the architecture, and parameters of the previous convolution layers are exposed to the attacker in plaintext format. In this situation, an experienced attacker can easily retrain the model with high accuracy and break the protection schemes. To be more specific, the attacker can freeze the previous convolution layers which are exactly part of the high-accuracy model, and randomly initializes or customizes a dense layer that is unknown to the attacker. The retraining process will take less effort since the intermediate result after the convolution layers are totally correct. A few more computations are needed to obtain the final prediction result. In order to verify this vulnerability, we trained a LeNet-5 model on the CIAFR-10 dataset for image classification and remove the final dense layer. The result shows that even with random initialization for weights parameter inside the final dense layer, the extracted model was able to be recovered to its original accuracy within just 20 epochs using 1% of the dataset.

B. Large Hardware Overhead for TEE-assisted Protection

The DNN model typically consists of multiple layers and they are arranged in a sequential manner. During inference, the input data is passed through the model layer by layer and the model forward propagates the result until the final output layer. Previous methods try to split the whole architecture into different parts and run some of them inside TEE. However, computation resources are limited inside the TEE and the operation in the TEE runs orders of magnitude slower than in the normal world [31]. For the example in ShadowNet [11], the TEE-assisted model inference time for one image can be ranging from tens to thousands of milliseconds, in terms of different models, and the memory requirement in TEE ranges from several to tens of MegaBytes. This is highly inefficient for resource-limited edge devices.

C. Other Challenges

The layer-wise feature of a DNN model determines that when leveraging TEE to protect some intermediate layers, the communication between the normal world and the secure world is bidirectional, i.e., the input to the layer in the TEE will return the output to the normal world for further computation. Thus, the intermediate result will be exposed to the attacker who can monitor everything in the normal world, favoring an attacker to infer the function inside the TEE. Accordingly, ShadowNet [11] adds a mask and linear transformation before sending the intermediate result back to the normal world, which however, increases the computation complexity and data transmission overhead. More severely, the lightweight encryption and masking are prone to be broken [32].

Regarding the limitations of previous works, a good framework should comprehensively protect the model confidentiality so that the attacker can not extract and transplant the model for unauthorized usage. At the same time, the model protection scheme in the network should not sacrifice the model performance for the legitimate user. The latency overhead should be low for a real-time system and provide a good user experience. Furthermore, the network needs to be generalization which means that it can provide protection for different kinds of model architecture. In this work, we overcome these challenges by proposing MirrorNet framework that can adequately address the aforementioned limitations.

IV. Proposed Method: MirrorNet

This section presents MirrorNet, which transforms an input DNN model (e.g., in Fig. 2) into its TEE-friendly counterpart. Specifically, MirrorNet can achieve a comparable or even better performance, while protecting model's confidentiality.

A. Preliminaries of DNN Model Architecture

A representative DNN model is composed of multiple interconnected layers, in which the convolutional layer and dense layer are the two most common layers. Taking the 2D convolution layer as an example, it is defined by input channels (IC), output channels (OC), kernel size (K), input feature size (FI), and output feature size (FO). For ease of clarification, we assume the feature maps are square-shaped, i.e., the input/output feature map is $(FI \times FI)$ / $(FO \times FO)$, while other shapes such as rectangular are free to be extended to. Specifically, we define a Conv2D layer as $[IC, OC, K, FI, FO]$. Similarly, the dense layer is defined as $[IC, OC]$, indicating its input and output feature dimensions.

B. MirrorNet: Overview

We demonstrate the entire workflow of our MirrorNet framework on an edge device in Fig. 2. First of all, the user (e.g., model provider) selects or possesses an input

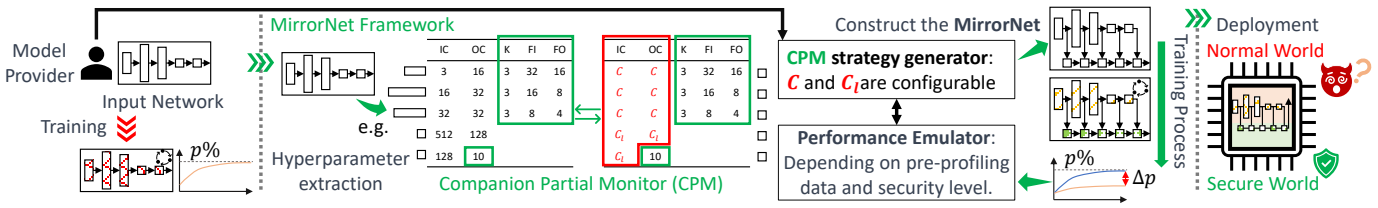

Fig. 2: The overview of MirrorNet. The model provider selects an input network for the BackboneNet architecture and generates the corresponding CPM. CPM has the same layer type as BackboneNet but with a smaller size. MirrorNet is the integration of BackboneNet and CPM. CPM design is generated by a Strategy Generator and evaluated by a Performance Emulator to help the model provider decide if the current design is appropriate for inference performance and hardware deployment. The trained MirrorNet is deployed on hardware where BackboneNet is in the normal world and CPM is in the secure world.

network that needs to be protected, so MirrorNet generates a BackboneNet with the same architecture as the input network. Then, MirrorNet builds a mirrored model architecture called Companion Partial Monitor (CPM). Here, the "companion partial" denotes that each layer inside the CPM has the same layer type as the corresponding layer in the BackboneNet, but with fewer parameters. For example, for a Conv2D layer in BackboneNet with size $[3, 64, 5, 32, 28]$, its CPM counterpart can be a Conv2D layer with size $[1, 1, 5, 32, 28]$. In other words, the CPM is a scaled-down version (especially channel-wise) of the BackboneNet, and its lightweight property well fits the memory constraints inside the TEE. We denote the integration of BackboneNet and CPM as MirrorNet, as illustrated in Fig. 2. Subsequently, MirrorNet will be trained from scratch. Rather than initializing the BackboneNet with pre-trained parameters (if publicly any), randomly initializing the BackboneNet is supposed to emphasize the importance of CPM more during the MirrorNet training.

With a well-trained MirrorNet, the model owner will decide if the current CPM design satisfies design requirements, from both performance (e.g., inference latency, model size, etc.) and security (i.e., accuracy gap between BackboneNet and MirrorNet, Δp as illustrated in Fig. 2). Since there are numerous possible CPM configurations for a BackboneNet, it is less feasible for the user to train them all and find the optimal one. To mitigate this concern, we develop two components for the framework, namely "CPM Strategy Generator" and "Performance Emulator". As indicated by their names, these two components can estimate the security of a MirrorNet and its hardware overhead without practical deployment.

C. Construction of MirrorNet

MirrorNet combines BackboneNet and CPM to execute inference, in which we take BackboneNet as the mainstay and CPM as the auxiliary, and design a ***feedforward*** route. Each layer's output of BackboneNet will go into the next layers of both BackboneNet and CPM, while the layer output of CPM will only be fed into the next CPM layer, i.e., there is no feedback from CPM to BackboneNet. Taking the first layer as an example, the input enters BackboneNet, after which part of the output is sent to the first layer in CPM as the input. Note that there is no layer in CPM corresponding to the first layer of

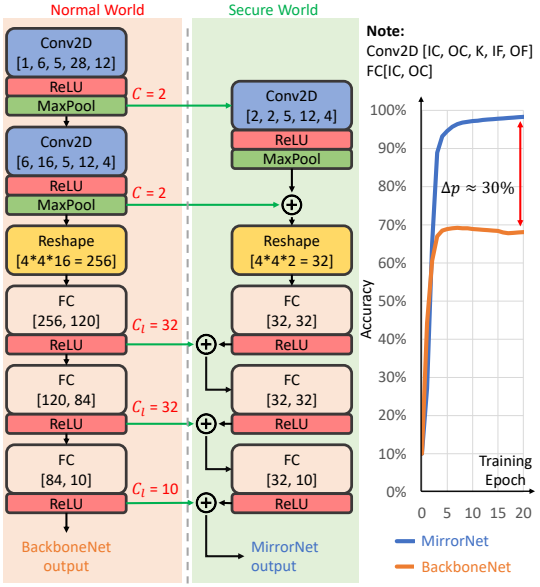

Fig. 3: MirrorNet example for LeNet-5.

BackboneNet, but the first layer in CPM aligns with the second layer in BackboneNet. In the separate use of BackboneNet, its output, namely *logits*, will be passed through the `argmax` function to derive the predicted label for the current input query. In MirrorNet, the logits of BackboneNet are also transferred to TEE and combined with the logits of CPM; and the combination is passed through the `argmax` function, so that only the predicted label from CPM is transferred back to the normal world indicating the final result.

To clearly illustrate the workflow, we depict one MirrorNet architecture using LeNet-5 [33] as the BackboneNet in Fig. 3. The model architecture on the left part stands for the BackboneNet, and the right part is the CPM. The layer inside the MirrorNet is represented in rectangles with the name (upper part) and the parameter (lower part). ReLU and MaxPooling are represented in a strip shape. The communication between the normal world and the secure world is unidirectional, and the C in between stands for the number of channels in the intermediate result that is transmitted to the secure world. The line graph on the right-hand side of Fig. 3 shows the com-

parison of inference accuracy between running BackboneNet only and MirrorNet as a whole. The result shows that the accuracy gap can be up to 30%, which is a significant drop when attackers only acquire the BackboneNet.

D. MirrorNet: CPM Strategy Generator

For a specific BackboneNet, there exist many possible CPM configurations. To generate the configurations satisfying the constraint TEE resource, our MirrorNet framework embeds a CPM Strategy Generator, as shown in Fig. 2. It ensures the layer alignment between BackboneNet and CPM, i.e., the kernel size (K), input feature size (FI), and output feature size (FO that is related to padding and stride configuration) of CPM convolution layers must stay the same as BackboneNet. For other parameters like input channel (IC) and output channel (OC), the generator selects a small number to achieve lightweight hardware overhead, denoted as C, which is empirically from 1 to 4 (see Sec. V-C). For the dense layer, we determine a small dimension denoted as C_l. Note that C_l is not manually selected, but determined by the OC and FO of the last convolution layer:

$$C_l = C \times FO^2 \tag{1}$$

The C_l should not be smaller than the number of classes, i.e., 10 in Fig. 2, to ensure well-behaved dense layers. While in practice, this condition is typically met.

E. MirrorNet: Performance Emulator

In addition to security, performance overhead is another critical factor in developing TEE-based DNN solutions. Since the performance overhead (i.e., latency) of MirrorNet depends on its deployment, similarly, it is infeasible to measure the practical overheads for each possible CPM strategy. Therefore, we build a Performance Emulator to help with the latency estimation. Taking the Cortex-A53 processor used in our experiments as an example, we profile various workloads (e.g., convolutional layers, dense layers, etc.) for the possible configurations of CPM. We collect the hardware latency of these workloads (see Sec. V-C) and run a regression analysis to derive the Performance Emulator. Consequently, with an arbitrary input network (Fig. 2), the Performance Emulator can estimate the hardware latency for a specific platform, without actually deploying the entire MirrorNet. This will extremely reduce the design efforts when deciding an appropriate MirrorNet candidate for a targeted edge device.

By applying the latency profiling results, the CPM Performance Emulator can predict a number of MirrorNet candidates satisfying the design requirements, such as latency overhead. Then, MirrorNet can evaluates the accuracy difference $p\%$ between the BackboneNet and the MirrorNet and select the best option according to the privacy requirement.

F. Superiority of MirrorNet Performance and Privacy

1) Inference performance: As a theoretical analysis, we indicate that the accuracy of the BackboneNet serves as a lower bound for the accuracy of its corresponding MirrorNet. We provide a concise and straightforward proof by setting all the weights in CPM to 0, so the output of MirrorNet is exactly equal to the output of its BackboneNet. Actually, since CPM has a similar structure to the BackboneNet, the combination of them (i.e., the MirrorNet) can be treated as a specially-shaped ensemble model [34] of the BackboneNet. Hence, the MirrorNet performance is even potentially improved over its BackboneNet part. However, since the BackboneNet and CPM are jointly trained, the performance of BackboneNet alone will be very poor, as shown in the evaluation in Sec.V.

2) Privacy: An attacker has no information from the secure world since TEE is physically isolated and well-protected for secure computation. Besides, our feedforward design for the MirrorNet guarantees that TEE never transmits data to the normal world, except for the final prediction result. Because the CPM computation is between every layer and those protected layers are performed in a cascaded manner in TEE, the attacker can neither infer the intermediate results from the predicted label nor freeze specific layers to retrain the partial model [35]. Therefore, s/he can only retrain the extracted BackboneNet from scratch. We regard one model extraction attack as failed if the attacker still needs to train the model from scratch.

3) Lightweight hardware overhead: One important characteristic of MirrorNet is that the CPM inside TEE is only a mirrored version of the BackboneNet with a small size. Therefore, the computation of CPM will not induce such a long execution time as the BackboneNet in the normal world, mitigating the limited resource issue in TEE for NN inference. The second characteristic is that the computation result of CPM will not be transferred back to the normal world like previous works [11] but stored in the secure memory. Thus, CPM only needs to send a return signal to the normal world, indicating the forward propagation to continue for the followed BackboneNet layer[1]. As the latency between switching worlds can be heavy, our feedforward strategy presents an efficient way, i.e., there is no data communication from secure world to normal world, which greatly reduces the communication overhead during inference; thanks to the already-ensured privacy, no encryption/decryption is needed anymore which further lowers the execution time and the complexity.

G. MirrorNet: Hardware Implementation

This section presents the hardware implementation of MirrorNet. We illustrate this procedure by taking a simple convolutional neural network (CNN) as an example, as shown in Fig. 5. This CNN is a BackboneNet that only has two Conv2D layers and one dense layer, and the CPM is composed of one down-scaled Conv2D and one dense layer. The execution order is interpreted as running on a single-thread CPU equipped with TEE. For other platforms, such as multi-thread CPU or TEE-embedded GPU[2] [36], the secure world and normal world can

[1]This done signal is an indicator for the completeness of one layer in CPM, which favors the normal world computing schedule if MirrorNet is deployed on a multi-thread CPU or a parallel-computing platform.

[2]One example for its commercial product is Nvidia H100 GPU.

```
#include <tee_client_api.h>
TEEC_InitializeContext();
TEEC_OpenSession();

float* input_image = malloc(..);
float* conv1_kernel;
float* output1;
output1 = Conv2D_1(input_image, conv1_kernel);
MaxPool_ReLU(output1);
Operation.buffer = LightWeight(output1);
TEEC_InvokeCommand(Operation, CMD1);
        ⋮
TEEC_InvokeCommand(Operation, CMD2);
        ⋮
int predicted_label;
predicted_label = TEEC_InvokeCommand(Operation, CMD3);
TEEC_FinalizeContext();
TEEC_CloseSession();
```

(a) Pseudocode in Normal World

```
#include <tee_internal_api.h>
TA_CreateEntryPoint();
TA_OpenSessionEntryPoint();
float* conv2_kernel;
TEE_Result(Operation, CMD1);
float* tmp_input = Operation.buffer;
float* tmp_output = Conv2D_2(tmp_input, conv2_kernel);
MaxPool_ReLU(tmp_output);
float* data_memory_conv2 = TEE_Malloc(sizeof(tmp_output));
TEE_MemMove(data_memory_conv2, tmp_output);
    ⋮
TEE_Result(Operation, CMD2);
    ⋮
TEE_Result(Operation, CMD3);
float* logits = Operation.buffer;
logits += data_memory_dense;
int res = argmax(logits);
Operation.value = res;
TA_CloseSessionEntryPoint();
TA_DestroyEntryPoint();
```

(b) Pseudocode in Secure World

Fig. 4: C-based pseudocode for MirrorNet implementation, with data transmission between normal world and secure world.

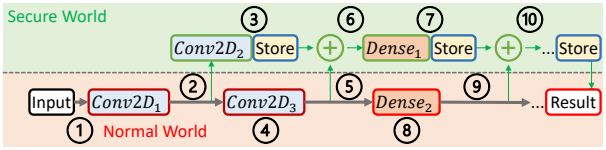

Fig. 5: Hardware implementation of MirrorNet. ReLU and MaxPooling are omitted for clarity.

execute their components in parallel and for batched queries as long as satisfying the dependency.

The execution flow of MirrorNet can be summarized as follow: ① The input query is provided to the normal world that runs the first convolution layer $Conv2D_1$. ② The normal world sends certain output channels of the feature map from $Conv2D_1$ to the secure world, which is a small part of the output. ③ The secure world receives the down-scaled feature map and executes the designed small convolution $Conv2D_2$ for feature extraction. The results are then stored in the secure memory in the secure world, i.e., the TEE[3]. Steps ④ and ⑤ execute the same procedure as the previous one that runs $Conv2D_3$ and transfers partial results to the secure world. In step ⑥, the received feature map from $Conv2D_3$ is added with the stored result of $Conv2D_2$ in TEE, as the amended input feature map for the followed $Dense_1$ layer. To ensure the results from $Conv2D_2$ and $Conv2D_3$ are added, MirrorNet requires the output channel of $Conv2D_2$ equal to the number of transferred channels. Further, $Conv2D_2$ and $Conv2D_3$ should have the same configurations on convolution, such that they can produce a feature map with the same size. We omit the description of steps ⑦, ⑧, and ⑨, since they have similar operations except for the dense layer computation. In step ⑩, the result from $Dense_1$ and $Dense_2$ are summed up, followed by the argmax function to determine the inferred label for the input query, which is then sent back to the normal world.

In addition to the coarse-grained scheduling for MirrorNet execution, we further take the open-source OP-TEE [37] as an example, to illustrate the fine-grained operations with a C program pseudocode, as shown in Fig. 4. For the BackboneNet execution, the memory allocation and variable initialization are all done in the normal world. The followed computations (i.e., $Conv2D_1$) are executed sequentially while the output is partially copied to an `Operation` structure, which is uploaded to secure world together with the operation command `CMD1`. Other layers in the normal world follow the same schedule, while the last command will return a result of the entire MirrorNet, as the `predicted_label`. In the secure world, corresponding commands upon receiving operations from the normal world will be executed (i.e., $Conv2D_2$). The memory allocation inside TEE, as shown in Line 9 and 10, utilizes `TEE_Malloc` or `TEE_MemMove` function. Thus, the data are stored in secure world without leaking any information to the normal world. In each layer, the input buffer copies data from `Operation.buffer` that is transferred from the normal world and cooperates with the stored data from the output of previous layer. Note that all the operations, e.g., `Conv2D`, are defined separately in the normal world and secure world, ensuring the isolation between them except for necessary data transmission.

[3]Note that for parallel computing or multi-thread cases, a *done* signal should be returned to the normal world to indicate the end of secure world execution.

V. Experimental Validation

A. Experimental Setup

1) Architectures and Datasets: Since this work focuses on developing a TEE-friendly framework for secure DNN inference, our targeted scenarios are lightweight applications rather than complicated tasks like ILSVRC image classification [38]. Moreover, our proposed framework offers fine-grained model protection at the layer-wise level. This makes it versatile to be generally applied to any DNN model To demonstrate its practicality, in our experiment, we use popular benchmarks on resource-limited edge devices, as the proof-of-concept. Specifically, we start from an easy task, MNIST [39], and then extend to more practical and complicated tasks like FashionMNIST [40] and CIFAR-10 [41]. For the architecture, we evaluate MirrorNet based on LeNet-5 [18] and VGG [19].

TABLE II: Apply LeNet-5 as the input network in MirrorNet with varying C settings (Fig.3 and Eq. 1) and different datasets. The corresponding MirrorNet is trained to obtain Δp and emulate the performance. The CPM latency emulation is on MNIST.

Channel number (C)	MNIST			FashionMNIST			CIFAR-10			CPM performance Emulator (ms)
	Input Network Test Acc.(%)		98.7	Input Network Test Acc.(%)		89.2	Input Network Test Acc.(%)		62.8	
	BackboneNet Test Acc.(%)	MirrorNet Test Acc.(%)	Δp (%)	BackboneNet Test Acc.(%)	MirrorNet Test Acc.(%)	Δp (%)	BackboneNet Test Acc.(%)	MirrorNet Test Acc.(%)	Δp (%)	
1	62.2	98.4	36.2	57.1	88.8	31.7	39.6	62.7	23.1	3.05
2	54.4	98.3	43.9	55.1	88.2	33.1	37	62.5	25.5	3.2
3	53.5	98.6	45.1	53.3	88.1	34.8	36.7	62.8	26.1	3.36
4	40.5	98.2	57.7	51.5	89.2	37.7	35.8	63.1	27.3	3.42

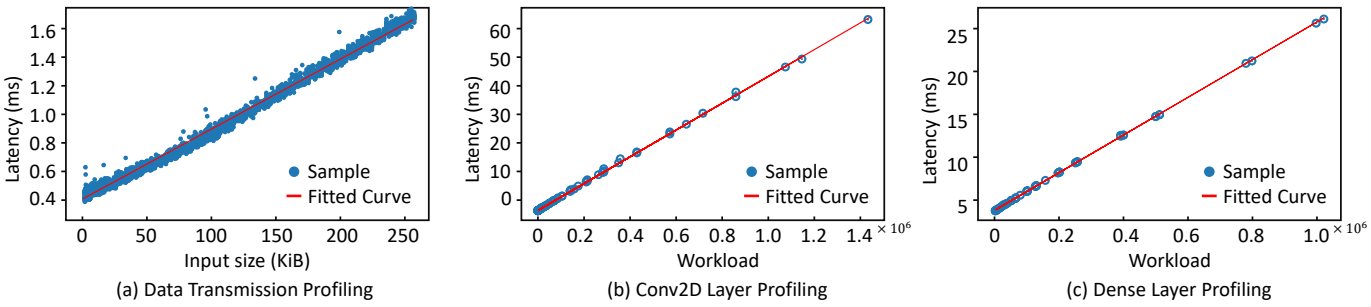

Fig. 6: The execution time of different components in MirrorNet.

2) Implementation: We implement MirrorNet in C language on a Raspberry Pi 3 Model-B, which has a BCM2837 64bit ARM Cortex-A53 Quad Core Processor and 1GB RAM. Without loss of generality, we leverage the Open Portable TEE (OP-TEE) [37], an open-source TEE framework supported by different boards equipped with Arm TrustZone [42]. OP-TEE provides a secure area in a processor, Internal Core API for Trusted Applications, and the TEE Client API to communicate with TEE. Note that although we use the ARM TrustZone as an example, our proposed MirrorNet framework can be generally applied to other TEEs, e.g., Intel SGX [43], AMD SEV [44], Sanctum [45], and Sanctuary [46].

B. Inference Performance

We first investigate the inference accuracy of MirrorNet. Note that we do not specify training strategies of MirrorNet, since it is capable of employing arbitrary strategies developed in the deep learning regime. While we preserve the flexibility of MirrorNet training, we evaluate MirrorNet with the basic strategy, i.e., Adam [47] as the optimizer and Cross-Entropy [48] as the loss function. We demonstrate our inference performance results of MirrorNet and its BackboneNet in Tab.II, using LeNet-5.

From Tab.II we can observe that for all datasets, the inference accuracy of MirrorNet is very close to the input network, which aligns with our expectations. Even with only one channel (i.e., $C = 1$), the MirrorNet can achieve an accuracy as good as the input network; as the channel number increases, the MirrorNet accuracy gets improve as well. For instance, the MirrorNet of $C = 4$ on CIFAR-10 even achieves better accuracy than the input network. On the other hand, with the help of CPM, the directly extracted BackboneNet performs much worse than the MirrorNet. The accuracy gap is large enough to make the BackboneNet useless if the attacker wants to transplant it to another unauthorized device for direct use. As the channel number increases, the accuracy gap between BackboneNet and MirrorNet also increases.

Further, we provide a detailed case study for the evaluation of the CPM Strategy Generator (Sec. IV-D). We apply various input networks of different VGG configurations and also vary the channel number C to investigate the accuracy changes, in Tab.III. It is evident that the channel number increment can increase the accuracy gap Δp between MirrorNet and BackboneNet. More importantly, as the complexity of the input network gets larger, although the MirrorNet performance becomes better, the accuracy gap actually decreases. From VGG-6 to VGG-16, the MirrorNet accuracy increases from 86.1% to 94.7%, but the accuracy gap decreases from 22.8% to 8.6%, when $C = 4$. This is a trade-off when the model provider considers the CPM design and the input network selection. While indeed a more complex input network can lead to a better performance MirrorNet, it could harm security by reducing the accuracy gap. If the model owner intends to increase the channel number for better security, the hardware overhead should be considered.

C. Latency Profiling for Performance Emulator

To use Performance Emulator (Sec. IV-E), we separately measure the latency of different operations, such as the context switch between the secure and normal worlds with data communication, and the computation in the secure world.

1) Context switch: Due to the dependency between different model layers, the interaction (i.e., data exchange or communication) between the normal world and the secure world is inevitable, which is time-consuming while the latency also depends on the transferred data buffer size. We first measure the relationship between the input size and the context switching latency, for which we send input data of different

TABLE III: Example MirrorNet strategies for various input networks on the CIFAR-10 dataset.

Input Network	Test Acc. (%)	Channel # (C)	BackboneNet Test Acc. (%)	MirrorNet Test Acc. (%)	Δp (%)
VGG-6	85	1	84.2	85.4	1.2
		2	81.4	86.5	5.1
		3	69.6	85.8	16.2
		4	63.3	86.1	22.8
VGG-7	91.3	1	91.0	91.2	0.2
		2	87.2	91.5	4.3
		3	80.8	91.2	10.4
		4	73.3	91.9	18.6
VGG-11	92.1	1	91.7	92.0	0.3
		2	91.6	92.2	0.6
		3	87.5	92.1	4.6
		4	79.7	92.3	12.6
VGG-16	94.6	1	94.3	94.5	0.2
		2	93.6	94.4	0.8
		3	92.0	94.5	2.5
		4	86.1	94.7	8.6

sizes to the operation buffer, invoking the CPM and making it return immediately. We record the execution time as shown in Fig. 6(a). With the increasing size of data, the latency of context switching also increases accordingly. This result highlights another advantage of MirrorNet, which does not require CPM in the secure world to send data back to the normal world after computation, leading to latency reduction.

Following the results in Fig. 6, we figure out the linear relationship between the context switching latency and the data size, in Eq. 2. Note that the latency is in microseconds ($\times 10^{-6}$ sec) and the input size is in kilobytes.

$$\text{Context Switching Latency} \approx 4.888 \times InputSize + 345.9 \quad (2)$$

2) Workload latency inside TEE: As the computing resource of TEE is limited and fixed, the execution workload (# of required operations) of certain components dominates the latency of each layer. Specifically, we investigate two critical layers in TEE, the Conv2D layer and the Dense layer. Other layers like ReLU consumes much shorter time than these two modules, thus we focus on the most time-consuming parts during TEE computations. The workload of convolution stands for the number of operations required for a Conv2D layer, including matrix multiplication and addition that process the input with kernels and generate the output. It is measured in terms of floating-point operations. The workload depends on the specific layer architecture. We provide a general formula to calculate the workload for a typical convolution layer. Recall the notation of a Conv2D layer as $[IC, OC, K, FI, FO]$, the workload can be calculated with Eq. 3.

$$\text{Workload} \approx K^2 \cdot IC \cdot OC \cdot FO^2 \quad (3)$$

We define one multiplication and one addition as a workload unit for Conv2D. Similarly, we define the dense layer workload as the multiplication-addition counts, i.e., Workload $\approx IC \times OC$ for a dense layer. With the layer-wise latency profiling, we measure the latency (microseconds) as

$$\text{Conv2D Latency} \approx 0.03938 \times Workload + 504.3 \quad (4)$$

$$\text{Dense Latency} \approx 0.02666 \times Workload + 465.6 \quad (5)$$

TABLE IV: MirrorNet Performance Emulator prediction vs. measurement for different architectures, where C = 4.

Inpute Network	Measured latency (ms)	MirrorNet Measured latency (ms)	Performance Overhead (%)	CPM Emulator Predicted (ms)	Emulation Error (%)
LeNet-5	8.26	10.98	32.93	3.42	25.76
VGG-7	716.559	723.719	0.99	6.960	2.80
VGG-11	1246.138	1255.06	0.72	8.495	4.78
VGG-16	4530.198	4552.154	0.48	22.061	0.48

Interestingly, the experimental results in Fig. 6 indicate the linear relationship between different workloads/operations and their latency. Therefore, we can simply apply the formulation in Eq. 2, 4, and 5 to configure the Performance Emulator and estimate the performance of a specific CPM strategy (Sec. IV-D), as well as make a trade-off between latency and privacy.

D. End-to-end Performance Evaluation

We evaluate the performance of different input models protected by MirrorNet by measuring their practical end-to-end execution time, the results as shown in Tab. IV. We first measure the execution time of the input network, i.e., without any protection. Then we measure the total execution time of the model inference using MirrorNet and calculate the overhead. After that, we predict the execution time of the CPM using the Performance Emulator. From these experiments, we observe substantial memory movement and allocation associated with the convolution computations, which are challenging to quantify. To facilitate the performance emulation, we empirically applied a scaling factor 1.6 (following experimental results in VGG-7), to refine the latency estimation. We apply this scaling factor on other models for validation. The experimental results demonstrate that our Performance Emulator achieves high prediction accuracy. Note that the emulation error in LeNet-5 architecture appears larger, due to its original lower latency overhead, i.e., even a minor estimation error (e.g., 2.72 ms) can magnify in percentage terms. In practice, our emulator just needs to provide a coarse-grained prediction for the latency to determine a few optimal CPM strategies.

VI. Conclusion

This paper presents MirrorNet for secure DNN inference on edge devices. As a Trusted Execution Environment (TEE) friendly framework, MirrorNet constructively converts an input DNN model into two parts: a BackboneNet and a companion partial monitor (CPM). Specifically, the BackboneNet part is stored in the normal world and only retains poor performance, which is rectified to high performance by the CPM. To make MirrorNet more flexible for any input, we propose two components, CPM Strategy Generator and Performance Emulator. Experimental results on a Raspberry Pi demonstrate the good performance and practical applicability of the proposed framework and its complementary components.

Acknowledgement

This work is supported in part by the U.S. National Science Foundation under Grants OAC-2319962, CNS-2239672, CNS-2153690, CNS-2326597, and CNS-2247892.

REFERENCES

[1] Y. Kortli, M. Jridi, A. Al Falou, and M. Atri, "Face recognition systems: A survey," *Sensors*, vol. 20, no. 2, p. 342, 2020.

[2] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2020.

[3] M. B. Alazzam, F. Alassery, and A. Almulihi, "A novel smart healthcare monitoring system using machine learning and the internet of things," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–7, 2021.

[4] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *The World Wide Web Conference*, 2019, pp. 2125–2136.

[5] Z. Sun, R. Sun, L. Lu, and A. Mislove, "Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1955–1972.

[6] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7105–7114.

[7] M. Pawlicki, R. Kozik, and M. Choraś, "A survey on neural networks for (cyber-) security and (cyber-) security of neural networks," *Neurocomputing*, vol. 500, pp. 1075–1087, 2022.

[8] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: towards model privacy at the edge using trusted execution environments," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 161–174.

[9] A. Schlögl and R. Böhme, "ennclave: offline inference with model confidentiality," in *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, 2020, pp. 93–104.

[10] P. M. VanNostrand, I. Kyriazis, M. Cheng, T. Guo, and R. J. Walls, "Confidential deep learning: Executing proprietary models on untrusted devices," *arXiv preprint arXiv:1908.10730*, 2019.

[11] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, S. Jha, and L. Lu, "Shadownet: A secure and efficient system for on-device model inference," *arXiv preprint arXiv:2011.05905*, 2020.

[12] Y. Li, H. Wang, and M. Barni, "A survey of deep neural network watermarking techniques," *Neurocomputing*, vol. 461, pp. 171–193, 2021.

[13] L. Wang, S. Xu, R. Xu, X. Wang, and Q. Zhu, "Non-transferable learning: A new approach for model ownership verification and applicability authorization," *arXiv preprint arXiv:2106.06916*, 2021.

[14] T. Zhou, S. Ren, and X. Xu, "Obfunas: A neural architecture search-based dnn obfuscation approach," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[15] S. Yandamuri, I. Abraham, K. Nayak, and M. K. Reiter, "Communication-efficient bft protocols using small trusted hardware to tolerate minority corruption," *Cryptology ePrint Archive*, 2021.

[16] R. Pi, "Raspberry pi 3 model b," *online].(https://www. raspberrypi. org*, 2015.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[21] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[22] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[23] C. Su and Q. Zeng, "Survey of cpu cache-based side-channel attacks: systematic analysis, security models, and countermeasures," *Security and Communication Networks*, vol. 2021, pp. 1–15, 2021.

[24] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: properties, applications, and challenges," *IEEE Security & Privacy*, vol. 18, no. 2, pp. 56–60, 2020.

[25] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[26] B. Liu, M. Ding, S. Shaham, W. Rahayu, F. Farokhi, and Z. Lin, "When machine learning meets privacy: A survey and outlook," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–36, 2021.

[27] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis." in *USENIX security symposium*, vol. 16, 2016, pp. 601–618.

[28] S. J. Oh, B. Schiele, and M. Fritz, "Towards reverse-engineering black-box neural networks," *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 121–144, 2019.

[29] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 36–52.

[30] Z. Zhang, L. K. Ng, B. Liu, Y. Cai, D. Li, Y. Guo, and X. Chen, "Teeslice: slicing dnn models for secure and efficient deployment," in *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, 2022, pp. 1–8.

[31] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[32] B. Lapid and A. Wool, "Cache-attacks on the arm trustzone implementations of aes-256 and aes-256-gcm via gpu-based analysis," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 235–256.

[33] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[34] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[35] R. Kolcun, D. A. Popescu, V. Safronov, P. Yadav, A. M. Mandalari, Y. Xie, R. Mortier, and H. Haddadi, "The case for retraining of ml models for iot device identification at the edge," *arXiv preprint arXiv:2011.08605*, 2020.

[36] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus." in *OSDI*, 2018, pp. 681–696.

[37] L. Limited, "Open portable trusted execution environment," 2019. [Online]. Available: https://www.op-tee.org

[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[40] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[41] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[42] "ARM Security Technology Building a Secure System using TrustZone Technology ," 2009, https://developer.arm.com/documentation/PRD29-GENC-009492/c/?lang=en.

[43] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[44] A. Sev-Snp, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, p. 8, 2020.

[45] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation." in *USENIX Security Symposium*, 2016, pp. 857–874.

[46] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Sanctuary: Arming trustzone with user-space enclaves." in *NDSS*, 2019.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[48] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, pp. 19–67, 2005.