GuaranTEE: Towards Attestable and Private ML with CCA

Sandra Siby, Sina Abdollahi, Mohammad Maheri, Marios Kogias, Hamed Haddadi
Imperial College London

Abstract

Machine-learning (ML) models are increasingly being deployed on edge devices to provide a variety of services. However, their deployment is accompanied by challenges in model privacy and auditability. Model providers want to ensure that (i) their proprietary models are not exposed to third parties; and (ii) be able to get attestations that their genuine models are operating on edge devices in accordance with the service agreement with the user. Existing measures to address these challenges have been hindered by issues such as high overheads and limited capability (processing/secure memory) on edge devices.

In this work, we propose GuaranTEE, a framework to provide attestable private machine learning on the edge. GuaranTEE uses Confidential Computing Architecture (CCA), Arm's latest architectural extension that allows for the creation and deployment of dynamic Trusted Execution Environments (TEEs) within which models can be executed. We evaluate CCA's feasibility to deploy ML models by developing, evaluating, and openly releasing a prototype. We also suggest improvements to CCA to facilitate its use in protecting the entire ML deployment pipeline on edge devices.

CCS Concepts: • **Computer systems organization** → *Embedded hardware*; • **Security and privacy** → **Tamper-proof and tamper-resistant designs**; • **Computing methodologies** → *Machine learning*.

Keywords: Machine Learning, Security, Attestation

ACM Reference Format:
Sandra Siby, Sina Abdollahi, Mohammad Maheri, Marios Kogias, Hamed Haddadi. 2024. GuaranTEE: Towards Attestable and Private ML with CCA. In *4th Workshop on Machine Learning and Systems (EuroMLSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3642970.3655845

EuroMLSys '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0541-0/24/04
https://doi.org/10.1145/3642970.3655845

1 Introduction

Machine-learning models are increasingly being deployed on edge devices (such as smartphones, IoT gateways, and home routers) for various purposes such as health monitoring, anomaly detection, face recognition, voice assistants, handwriting recognition *etc.* Running local models on edge devices provides several advantages over cloud-based approaches. Sensitive user data from the devices do not have to be sent to external model providers for inference, thereby providing privacy benefits [10]. Running models locally avoids the need for large data transfers, which can be costly in terms of latency and bandwidth. Finally, local models facilitate private personalization based on user preferences [36, 39].

At the same time, models on edge devices pose challenges. Model providers are increasingly demanding *model privacy* and protection, *i.e.*, to ensure that their proprietary model information (*e.g.*, weights) are not exposed to external parties. Providers also desire *model verifiability and attestability*, *i.e.*, to ensure that their models have run on the device as expected and have not been tampered with.

Prior work has shown that on-device models, such as those on mobile phones, are susceptible to various attacks [45, 51]. Sun *et al.* [45] showed that deep-learning models on mobile phones have insufficient protections – their analysis of ≈ 40,000 apps on the Android store revealed that 41% of apps lack any form of protection, and in cases where some protection such as encryption exist, they can be overcome by simple attacks.

Various solutions have been proposed for model protection. Watermarking techniques can detect model theft but not prevent them, and are susceptible to tampering [3, 53]. Cryptographic techniques such as homomorphic encryption (HE) [8, 33, 50] or secure multiparty communication (SMC) [32, 35] are hindered by computational and communication overheads. Hardware-assisted techniques, using trusted execution environments (TEEs), are a performant alternative to cryptographic techniques. However, many TEE solutions are tailored towards cloud environments (*e.g.*, using Intel SGX) and not applicable to edge devices which have limited memory and computational power. Hardware-assisted solutions on edge devices have primarily focused on how models can be deployed on edge devices with limited TEE memory. Several solutions involve either partitioning models and running part of the model within TEEs [12, 30, 40, 45], or pruning models before deployment so that they can fit within TEE memory [13]. The few works that have experimented

with deploying entire models within TEEs are limited by the number of enclaves they can run in parallel and lack of support for secure peripherals [2].

In this work, we propose GuaranTEE, a framework to deploy and run machine-learning models on the edge in a private and verifiable manner. GuaranTEE is motivated by the introduction of Arm CCA (Confidential Computing Architecture) [20, 23] – a set of extensions in Arm's new architecture that allow for the creation of dynamic, hardware-protected enclaves, *realms*. TrustZone(security extensions in Arm's previous architecture) [25] is widely deployed on edge devices, but is not appropriate for running ML models due to security and memory limitations [4]. CCA's features, which are tailored towards deploying ML at the edge, and its presence in Arm's next architecture make it a promising candidate for widespread deployment on edge devices. Thus, in this work, we explore CCA's potential and limitations to implement GuaranTEE.

Our contributions are as follows:

- We develop GuaranTEE, a framework that allows for machine-learning models from a provider to be run on end devices in a private and verifiable manner. We explore Arm's new Confidential Computing Architecture (CCA) to implement GuaranTEE. In GuaranTEE, a model runs within a CCA *realm* – a hardware-protected enclave that can be established at runtime. Using CCA allows for entire ML models to be run within a trusted and private environment, without resorting to partitioning.
- We develop and evaluate a prototype of GuaranTEE using Arm's Fixed Virtual Platforms (FVP) simulator [19][1]. Our preliminary results indicate that running a TensorFlow Lite image-recognition model within a realm for inference results in an overhead of 1.7 times the number of instructions required to run it within a normal world virtual machine.
- We discuss challenges involved in implementing our prototype using CCA, and potential enhancements to the CCA architecture to enable better protection of the ML pipeline on edge devices.

2 Model protection on the edge

Running machine-learning models on edge devices involves local storage of models and hence, poses several challenges with regard to *model privacy* and *model verifiability*. Model providers desire model privacy as their models might be proprietary; providers do not want to expose any information about their models to external parties such as other (potentially competing) model providers, the end user, or malicious actors. Model providers also require model verifiability, to ensure that their models exhibit the expected behavior and have not been tampered with. Leaked or modified models, especially in security-critical applications such as banking or medical services, can result in harm to the end user *e.g.*, identity theft or exposure of personal information [45].

Prior work has shown that on-device models, such as those on mobile phones, are susceptible to model stealing and adversarial machine learning [7, 14, 15, 36] and lack necessary protections [45, 51]. Thus, it is necessary to develop techniques for model protection that can provide better privacy and verifiability.

There are several techniques aimed at providing model protection. Techniques such as watermarking and fingerprinting allow model providers to identify stolen models [18, 28, 43]. However, they are *passive defenses*, *i.e.*, they do not prevent theft but detect it afterward [3]. In addition, works have successfully demonstrated evasion, removal, and tampering attacks against these techniques [53]. Another category of techniques involves *cryptographic* means to provide private and secure machine learning. These primarily include: homomorphic encryption (HE), which allows for operations to be run directly on encrypted data (*e.g.*, [8, 33, 50]), and secure multi-party computation (SMPC), where multiple parties jointly compute a function over inputs that are kept private (*e.g.*, [32, 35]), or a combination of the two (*e.g.*, [27]). These techniques have even been applied in edge scenarios(*e.g.*, [26]). However, cryptographic techniques are hindered by high computation (for HE) and communication (for SMPC) overheads.

Hardware-assisted techniques address the performance limitations of cryptographic techniques. They make use of Trusted Execution Environments (TEEs), which are isolated processing environments in which applications can be executed securely [31]. There are several works that use TEEs for private inference in cloud-based applications (such as MLaaS platforms), mainly using Intel SGX [9, 11, 47]. However, these solutions are not appropriate for edge devices which have limited memory and computational power.

Hardware-assisted techniques on the edge focus on how machine-learning models can be deployed in devices with limited capabilities. Several works have proposed putting a subset of a model within the TEE and offloading the rest to untrusted accelerators – these include shielding deep layers [30], shallow layers [12], intermediate layers [40], non-linear layers [44] within a TEE. Zhang *et al.* [55] termed these solutions as *TSDP* or TEE-Shielded DNN Partition. Zhang *et al.* showed that TSDP solutions are vulnerable to privacy attacks. Both Zhang *et al.* and Liu *et al.* [29] showed that TSDP solutions are vulnerable to privacy attacks. They proposed two methods of protecting models while accounting for the memory limitations of TEE. Zhang *et al.* proposed TEESlice, a paritioning-before-training approach which partitions a model into a lightweight private part which resides within the TEE, and a public backbone outside the TEE. Liu *et al.* proposed MirrorNet, where a model is trained using a combination of a backbone model outside the TEE and a

[1]Our code is available at: https://github.com/comet-cc/GuaranTEE

lightweight network within the TEE. Both these approaches involve separating the contents of the model such that confidential information is protected within the TEE.

Due to TEE memory limitations, there has been limited work that has explored deploying an entire model within a TEE. Brasser *et al.* [4] designed Sanctuary, an architecture that allows for the creation of isolated user-space enclaves in the normal world on top of TrustZone. Bayerl *et al.*'s [2] work on Offline Model Guard (OMG) used Sanctuary to run machine-learning models in user-space enclaves in edge devices. However, Sanctuary's core-based protection limits the number of concurrent isolated enclaves to the number of cores and does not support secure peripheral access [42]. Sun *et al.* [42] proposed LEAP to overcome these limitations. Both OMG and LEAP rely on TrustZone, which has limitations that we discuss in Section 2.1. Hu *et al.* proposed an orthogonal pruning approach to reduce the size of the model and enable it to be run within a TEE [13].

In this work, we revisit the possibility of running an entire model within the TEE. Our work is motivated by the introduction of Arm's Confidential Computing Architecture (CCA) which allows for the dynamic creation of hardware-protected enclaves called realms. CCA is the next version of Arm's architecture to enable secure execution environments, in parallel with TrustZone. As TrustZone is already widely deployed on end devices, we envision that CCA will also see real-world deployment in the near future. In the next section, we describe CCA in more detail.

2.1 Towards CCA

In this section, we describe TEE support on Arm. We first provide an overview of TrustZone and its limitations in running machine-learning models before delving into CCA. We also briefly describe existing work on CCA.

Arm TrustZone. TrustZone refers to hardware security extensions that were introduced by Arm in their Armv6K architecture in 2004 and enables the creation of TEEs [25, 34]. TrustZone allows for the creation of two execution environments (or *worlds*) that are isolated from one another: the Normal World and the Secure World. The processor can be in one of these worlds at any point in time. The transition between the worlds is managed by the highest-privileged firmware in the system, called the Secure Monitor. Each world has its own dedicated memory – When the processor is in the non-secure state (*i.e.*, operating in the normal world), software in the normal world cannot access the secure world's memory. There is no restriction on access to the normal world memory when the execution is given to the secure world. The normal world usually runs a rich software stack which can include an operating system, applications, hypervisor, *etc.* whereas the secure world runs a smaller stack which includes a lightweight kernel (Trusted OS) supporting several security-sensitive services (Trusted apps) such as key management. The secure world has a small stack in order to reduce possible vulnerabilities, as it is intended to host trusted apps (*e.g.*, 210K LOC in a Linaro TEE, with 110K for the trusted OS and 100K for the secure monitor [6]).

TrustZone limitations for ML deployment. TrustZone has inherent limitations that make it incompatible with the practical implementation of ML services. The first limitation is the *reliance on the trusted OS* by the trusted apps. Trusted apps' resources are controlled by the Trusted OS, which still has a large attack surface despite the smaller stack – there have been a significant number of attacks that can compromise the Trusted OS [6]. Prior work has suggested stronger isolation environments to address these issues [4, 6]. The second limitation is *memory*. The memory size of the secure world is limited in current implementations (*e.g.*, 16∼64MiB in OPTEE [31]), which does not allow for the deployment of entire machine-learning models within the secure world. Finally, TrustZone has *development cycle* limitations. TrustZone was mainly designed to run trusted applications from platform-specific services (*e.g.*, from the original equipment manufacturers) rather than general developers [20]. In order to ensure that trusted apps do not contain vulnerabilities, vendors often place restrictions and security checks on developers who want to deploy trusted apps. Trusted apps also tend to be smaller in order to reduce the attack surface. This impedes the development of feature-rich apps. While prior work has attempted to get around these limitations [2, 4, 42], we look towards CCA as a possible solution as it is tailored towards secure deployment of general-purpose apps.

Arm CCA. Arm Confidential Computing Architecture (CCA) is a collection of hardware, software, and firmware extensions in the Armv9-A architecture [16, 17, 20, 23]. As shown in Fig 1, in Arm CCA , there are two new execution environments (Realm and Root) in addition to the existing normal world and secure world execution environments. The root world is able to access all the other worlds. The realm and the secure worlds cannot access each other's memory, *i.e.*, the realm world can access the realm and normal world memory while the secure world can access the secure and normal world memory. As in TrustZone, the normal world cannot access the other worlds' memory.

The realm world architecture allows for the creation and execution of virtual machines (called realms). The hypervisor (in the normal world) performs initialization and memory allocation to the realms. However, as the execution of a realm is isolated from the normal world, the hypervisor is not allowed to access it. Thus, CCA introduces the Realm Management Monitor (RMM) – a lightweight firmware in the realm world that manages communication between a realm and the hypervisor and ensures isolation between realms. The communication interface between the hypervisor and the RMM is known as the the Realm Management Interface (RMI), whereas the interface between the realm and the RMM

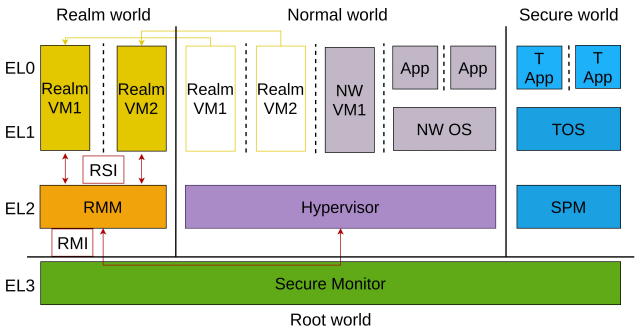

Figure 1. Arm CCA software architecture. CCA introduces two new execution environments: realm and root. CCA's architecture allows for the creation of dynamic, hardware-protected enclaves called *realms*. Unlike TrustZone, the secure monitor runs in the root physical address (PA) space which is separate from the secure world PA.

is known as the Realm Service Interface (RSI). The RMI is used by the hypervisor to issue commands to the RMM for controlling the realm (*e.g.*, creation or termination of the realm). The RMM confirms the validity of these commands and then performs the requested action. The RSI is used by the realm to request services from the RMM (*e.g.*, create an attestation report).

Transitions between worlds is managed by the Secure Monitor which resides in the root world. We note that the root world has its own physical address space in CCA, unlike in TrustZone where the address space of the secure monitor was within the secure world.

CCA for ML deployment. CCA addresses the limitations of TrustZone for ML deployment, which makes it a viable candidate for GuaranTEE. First, CCA changes the trust relationship between a program and its supervisory software. Unlike in TrustZone, where a compromised supervisory software (Trusted OS) can result in it gaining control of the trusted apps, CCA offers *protection against a compromised hypervisor*. While the hypervisor has the capability to manage realms and their resources, it is unable to access them, thereby ensuring confidentiality and integrity for the realms even when the hypervisor is compromised. Second, CCA offers *flexible memory allocation* for the realms. The memory that can be allocated to a realm is only limited by the system memory. On realm termination, the hypervisor can reclaim the delegated memory and return it to the normal world. This allows for apps with large and dynamic memory requirements to be run with CCA. Finally, CCA allows for *flexible development*. CCA is targeted towards general developers – developers can easily deploy apps within realms without the need for business relationships with vendors.

Systems based on CCA. As CCA is still under development, there is limited prior work in this space. Xu *et al.* introduced virtCCA [52], a virtualized CCA implementation on top of existing TrustZone hardware. virtCCA was meant to address the unavailability of hardware with CCA support. While CCA-compatible hardware has still not been released, we make use of Arm's released CCA-compatible software components to implement GuaranTEE [19]. These components are actively being worked on and provide the necessary functionality required to deploy GuaranTEE. Zhang *et al.* proposed Shelter [54], which provides user-space isolation in the normal world using CCA hardware primitives. Shelter is intended to be complementary to CCA's realms and was developed due to the nascent state of software development on CCA. In this work, we implement GuaranTEE on CCA realms instead of opting for user-space enclaves in the normal world. Sridhara *et al.* developed Acai [41], a system that allows CCA realms to securely access PCIe-based accelerators with strong isolation guarantees. Acai is complementary to our work; GuaranTEE can be extended to use secure accelerators for running more complex models with mechanisms such as Acai.

3 GuaranTEE architecture

In this section, we describe our system, GuaranTEE, which uses CCA to run models in a private and attestable manner.

3.1 System and Threat Model

System Model. We consider three parties: *model providers*, *clients*, and *a trusted verifier*. A model provider is an entity that trains and deploys machine-learning models to clients. However, before deploying an ML model, the provider needs to ensure that the client's suggested execution environment is trustworthy to store and use its model. The client is an end device (such as a smartphone or an IoT gateway) supporting the Armv9-A architecture. The client wants to use ML models for various purposes (*e.g.*, face recognition and speech processing) on their own device. The trusted verifier is an entity that provides verified realm images for clients to run. A realm image contains all the software and dependencies required to deploy a realm within a client and run the provider's model. The client obtains the verified image from the trusted verifier to deploy a realm within which the model is stored.

Threat model. We assume that the model provider does not trust the client – the provider wants to ensure that the client only uses the model for inference, and wants to prevent the client from accessing any internals of the model. On the client side, we assume that the Secure Monitor and the RMM are trustworthy, and the normal world (including user-space apps and the hypervisor) are malicious. While CCA does not provide availability guarantees, we assume that the hypervisor, in addition to being unable to access a realm's content, does not interfere with the realm's creation and execution. We consider side-channel attacks out of

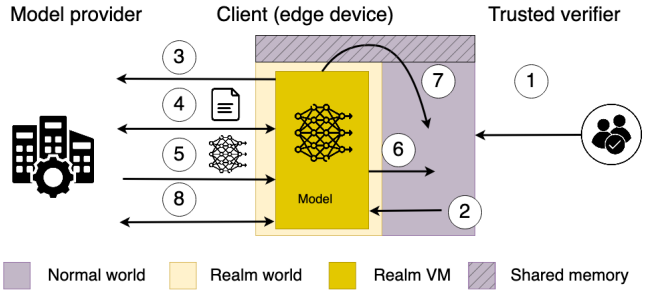

Figure 2. Overview of GuaranTEE outlining the steps required for running a ML model on the client edge device. We show a simplified view of the normal and realm worlds within the client. The client's steps are (1) obtaining realm image from verifier (2) creating and activating a realm VM (3) establishing connection with provider (4) realm attestation (5) obtaining model from provider (6) announcing model readiness to normal world (7) running inference (8) performing model updates.

scope for this paper. Finally, we assume that the connection between the client and the model provider is encrypted and protected from external network adversaries and unintended decryption.

3.2 GuaranTEE Pipeline

We provide a description of the various steps involved in deploying a model on the client (Figure 2 shows an overview of GuaranTEE).

On the client, a normal world app starts the pipeline by obtaining a publicly available and verified realm image from the trusted third party (Step 1 in the figure). The realm image contains a Linux operating system with all the software and dependencies required to run the provider's model.

Realm initialization. The normal world app requests the hypervisor to initiate the realm creation process (Step 2). The hypervisor delegates physical pages to the realm world, and requests the RMM to copy the content of the realm image to the delegated pages. The RMM populates the realm pages and measures its contents. The hypervisor then sends the activation command to the RMM. Once the realm is activated by the RMM, the hypervisor is able to give CPU execution to realm but it is no longer able to ask the RMM to populate new content.

Realm attestation. Before sending a model to the client, the model provider needs to ensure that the model will be run within a realm that has been correctly initialized and is running without issues. This can be achieved by attestation of the realm. After booting, the realm initiates a TLS connection with the model provider (Step 3). The model provider sends a request for an *attestation report* to the realm, which forwards it to the RMM (Step 4). The attestation report consists of two parts: attestation of the platform on which the realm runs (CCA token), and attestation of the state of the realm (realm token) [1, 37]. The realm token consists of initial measurements taken during realm creation and runtime measurements. The RMM coordinates the attestation report creation: it obtains the CCA token from the root of trust, and the realm token from the realm, and assembles them into a report which it sends to the realm [21]. The realm, in turn, sends the report to the model provider. The model provider can verify the attestation report to decide the trustworthiness of the activated realm. On verification, it sends the model to the realm via the TLS connection (Step 5).

Inference. After receipt of the model, the realm announces its ability to respond to inference queries to the normal world via the hypervisor (Step 6). The realm reads the input data from the normal world memory, feeds it into the model, obtains the inference, and writes the output to the normal world memory (Step 7).

ML deployment life cycle. In addition to the main operation of the model (*i.e.*, inference), GuaranTEE needs to account for other aspects of a model's life cycle. For example, the provider may want to impose a limit on the service it provides to the client – this may be in the form of a time limit (or validity period) for which the model should be active or the number of inferences the model can provide. Such functionality can be incorporated into the model-running code that is present in the realm image provided to the client. Once the time/inference limit is reached, the realm can send a system call to the hypervisor asking for termination. On termination, the memory delegated to the realm is released back to the normal world.

Another scenario involved around model updates – the realm may need to periodically query the provider to check for and receive updates (Step 8). The provider may need to verify the state of the realm before sending the updated model, which the realm can achieve by sending a runtime attestation report to the provider.

3.3 Implementation

As CCA is still under active development and a CCA-compatible board has not been released yet, we build a prototype of GuaranTEE on Arm's Fixed Virtual Platforms (FVP) [19]. FVP are provided by Arm for early-stage development of software/firmware without the need for compatible hardware. We use the Armv-A Base Platform RevC version of FVP – it supports the Armv8-A architecture versions up to v8.7 and Armv9-A (which has the CCA extension) [19].

Software stack. We use the latest release[2] of the reference Arm CCA integration [22] in our implementation. This integration consists of all the necessary source files and instructions to build binary files which are then provided to the FVP to start the simulation. The FVP boots the Secure

[2]AEMFVP-A-RME-2023.12.22

Monitor, the RMM, and the hypervisor, respectively. The integration uses the Trusted Firmware-A [48] and the Trusted Firmware implementation of RMM [49] as the Secure Monitor and RMM in the stack. It also uses linux-cca [24] as the hypervisor. There is no Trusted OS in the current version of the integration. As we do not run operations in the Secure world in our pipeline, we did not add Trusted OS to the stack.

The integration uses buildroot [5] to create a customizable file system for the hypervisor as well as the realm. As we want to run a TensorFlow Lite model within the realm, we added C++ libraries to the default realm file system. We then deploy the model within the realm.

GuaranTEE uses a folder mounted in the file system of the both realm and the hypervisor to exchange model inputs and outputs. We run applications in both the realm and the normal world. These applications check the contents of the shared folder. On the realm side, the application checks whether there is input data available for inference. If so, it reads the input data and feeds this into the model. On the normal world side, the application checks for the presence of inference output from the realm.

We could not implement the attestation mechanism as FVP does not have the HES (Hardware Enforced Security) implemented, which is required for the attestation report (HES is required for the platform attestation token, which is a part of the attestation report) [1].

4 Preliminary evaluation

In order to evaluate the overhead of running inference within a realm, we perform an evaluation between two scenarios for an image-recognition task. In our baseline scenario, which does not provide confidentiality and integrity guarantees, the model and the code to run it are stored in a normal world virtual machine (VM). The model code reads the input data from the folder that is shared with a normal world app. The code performs the inference and writes the output to shared memory. In the second scenario, the model and code are stored within a realm VM. As in the baseline scenario, reading input data and writing inference output are performed via the shared folder with the normal world app.

For our evaluation, we use a 16MB pre-trained TensorFlow Lite model MobileNet_v1_1.0_224 [46]. This model is created by training a TensorFlow model on the ImageNet dataset at 224x224 resolution, and converting it to TensorFlow Lite. We run our evaluation on a Lenovo ThinkCentre M75t Gen 2 with 16GB RAM and an 8-core AMD Ryzen 7 PRO 3700 processor (OS: Ubuntu 22.04.4 LTS).

FVP considerations. FVP is instruction-accurate but not cycle-accurate, *i.e.*, it correctly reports the number of instructions required by an operation but not the time taken on real hardware. As we cannot obtain accurate timing measurements without a CCA-compatible board, we rely on the number of instructions to obtain the overhead of running inference within the realm. Moreover, as we only have access to the total number of instructions executed on the FVP, we need a method to obtain the number of instructions explicitly caused by a workload (*e.g.*, inference or realm creation). To report the real number, we first measure the number of instructions when the workload is not running and reduce that from the observed number of instructions when the workload is running on the FVP. We also report timings for completeness, as the measurements may be useful for those intending to use FVP to experiment with CCA.

Table 1. Mean (standard deviation) number of instructions for each inference over five experiment runs.

Scenario	Number of Instructions (Millions)
Normal world VM	222.2 (46.5)
Realm VM	361.6 (4.4)

4.1 Inference overhead

We create an instance of the realm and measure the total number of instructions for getting the inference result for 40 images. We repeat this five times. Table 1 shows the average number of instructions for each inference process (writing one image to the shared memory by the normal world program, performing inference and writing output to the shared memory by the realm world program). As Table 1 shows, running the model inside the realm leads to ×1.62 overhead in the number of instructions executed for each inference process compared to running model inside the normal world VM. While timings are not accurate, we report them for completeness: on average, each inference takes ≈ 34.25 seconds in the realm VM, and ≈ 27.28 seconds in the normal world VM. The increase in the number of instructions is mainly due to the higher number context switches the hypervisor needs to manage the realm and handle its interrupts. While the hypervisor has direct access to the normal world VM, its access to the realm first has to go through the Secure Monitor and then to the RMM, resulting in more context switches and hence, more instructions.

4.2 Realm setup

We also report the instructions and the time required to create and terminate a realm VM as well as a normal world VM when the size of the image is 98MB (55MB for the file system and 43MB for the kernel)[3]. We repeat each observation five times. As shown in Table 2, creating a realm VM creates a significant overhead (×26.62) as compared to the normal world VM. The increased overhead is ×9.23 for termination. On average, realm creation and termination take 6:21 min and 1:01 min, whereas the normal world VM creation and

[3]We allocate 300 MB RAM to create the VM, which is sufficient for the VM to function correctly without being too large.

Table 2. Mean (standard deviation) number of instructions for booting and termination over five experiment runs. Image size is 98MB.

Scenario	Number of instructions (Millions)
Realm VM boot	18,880.6 (1,655.3)
Normal VM boot	709.8 (6.7)
Realm VM termination	970.0 (98.9)
Normal VM termination	105.1 (0.2)

termination take 1:31 min and 0:08 min respectively. We observe that the size of the initial content to be populated intro the realm has a significant effect on the boot time. For instance, when we increase the size of the realm image to 139MB (96MB for the file system and 43MB for the kernel)[4], we observe that, on average (over five iterations), it takes ≈ 27,190 million instructions to create the realm VM and ≈ 725 million instructions to create the normal world VM. The overhead is around ×37.50, which is a significant increase compared to the previous comparison. Thus, we recommend that the realm image only contain code required to implement necessary functionality. Reducing realm image size would also facilitate multiple realms running in parallel.

5 Considerations for ML deployment using CCA

Our prototype of GuaranTEE enables us to take initial steps towards using CCA to run ML models. However, there are several other factors to take into account for deploying the entire ML cycle, which may require modifications to the CCA architecture.

First, we only consider a scenario where adversaries would not have access to the model. However, there are attacks on the data pipeline, as the inputs and outputs to the model are not protected [31]. For example, adversaries can poison the input data to the model, or run attacks on the ML inferences, as these are not protected in our current implementation of GuaranTEE (input data and inferences are stored in the normal world). The CCA architecture also does not currently offer dedicated solutions to secure the inputs and outputs to the model. Potential solutions include exploring how secure peripherals can be used with realms [38, 41] and integrating detection mechanisms within the realm.

We could not implement the exact attestation mechanism as described in the CCA specification, as FVP does not currently have the HES (Hardware Enforces Security) module to create the platform attestation token. As attestation is an important step in the ML deployment pipeline, we recommend that future versions of FVP provide this module to help implement and test attestation.

[4]We allocate 400MB RAM in this scenario.

We also envision that models from multiple (possibly, competing) providers may concurrently run on a user's device. Measuring performance when running multiple realms in parallel, and extending CCA support for sharing of resources among realms (*e.g.*, access to peripherals) are necessary to realize this scenario.

Furthermore, we need to employ mechanisms to ensure that the ML pipeline is operational in accordance with the agreement between the client and the provider, especially in the absence of a trusted verifier *e.g.*, if the model provider supplies the realm image. The client needs assurance that the model provider's code will not use its data for purposes outside the agreed-upon ones and the model provider needs assurance that the client uses its services only for the time or inference limit it has agreed to. These mechanisms have an impact on all aspects of the pipeline, from model inference to updates to termination. Extending the CCA architecture to include functionality for policy enforcement and improved runtime realm attestation would assist in these changes.

Finally, CCA does not provide availability guarantees – the hypervisor controls the creation and execution of realms. The hypervisor can, thus, cause denial of service for the client, *e.g.*, by preventing creation of the realm for model deployment. Exploring methods to provide greater availability guarantees would be interesting future work.

6 Conclusion

In this work, we introduced GuaranTEE, a framework that uses CCA to provide ML model protection and attestation on edge devices. We used the FVP simulator to develop and evaluate an initial version of ML inference. Our results indicate that it is feasible to use FVP to build CCA-compatible prototypes of various stages of the ML deployment cycle. We also find that running a model within a realm for inference incurs an overhead of 1.7 times the number of instructions required for running it in a normal world virtual machine. At the same time, we discover various challenges that may require changes, not only to the FVP platform, but also to the underlying CCA architecture to fully realize the vision of private and attestable learning at the edge.

Acknowledgments

We thank Jon Crowcroft for his invaluable suggestions that helped improve the paper. This work was funded by the EPSRC Open Plus Fellowship (EP/W005271/1) and an Amazon Research Award.

References

[1] Tamas Ban. 2022. Attestation and Measured Boot. https://www.trustedfirmware.org/docs/Attestation_and_Measured_Boot.pdf

[2] Sebastian P Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. 2020. Offline model guard: Secure and

private ML on mobile devices. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 460–465.

[3] Franziska Boenisch. 2021. A systematic review on model watermarking for neural networks. *Frontiers in big Data* 4 (2021), 729663.

[4] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves.. In *NDSS*.

[5] Buildroot. Accessed Feb 2024. buildroot. https://github.com/buildroot/buildroot

[6] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted TEE systems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1416–1432.

[7] Zizhuang Deng, Kai Chen, Guozhu Meng, Xiaodong Zhang, Ke Xu, and Yao Cheng. 2022. Understanding real-world threats to deep learning models in Android apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 785–799.

[8] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.

[9] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Yerbabuena: Securing deep learning inference data via enclave-based ternary model partitioning. *arXiv preprint arXiv:1807.00969* (2018).

[10] Hamed Haddadi, Richard Mortier, and Steven Hand. 2012. Privacy analytics. 42, 2 (mar 2012), 94–98. https://doi.org/10.1145/2185376.2185390

[11] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. 2020. Darknight: A data privacy scheme for training and inference of deep neural networks. *arXiv preprint arXiv:2006.01300* (2020).

[12] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 2021. Model Protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2021), 4270–4284.

[13] Bin Hu, Yan Wang, Jerry Cheng, Tianming Zhao, Yucheng Xie, Xiaonan Guo, and Yingying Chen. 2023. Secure and Efficient Mobile DNN Using Trusted Execution Environments. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*. 274–285.

[14] Han Hu, Yujin Huang, Qiuyuan Chen, Terry Yue Zhuo, and Chunyang Chen. 2023. A First Look at On-device Models in iOS Apps. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–30.

[15] Yujin Huang and Chunyang Chen. 2022. Smart app attack: hacking deep learning models in android apps. *IEEE Transactions on Information Forensics and Security* 17 (2022), 1827–1840.

[16] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. 2022. Design and verification of the arm confidential compute architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 465–484.

[17] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, Gareth Stockwell, Mark Knight, and Charles Garcia-Tobin. [n. d.]. Enabling Realms with the Arm Confidential Compute Architecture. ([n. d.]).

[18] Yue Li, Hongxia Wang, and Mauro Barni. 2021. A survey of deep neural network watermarking techniques. *Neurocomputing* 461 (2021), 171–193.

[19] Arm Limited. 2023. Fixed Virtual Platforms. https://developer.arm.com/Tools%20and%20Software/Fixed%20Virtual%20Platforms

[20] Arm Limited. 2023. Introducing Arm Confidential Compute Architecture. https://developer.arm.com/documentation/den0125/0300/Overview

[21] Arm Limited. 2023. Realm Management Monitor Sepcification. https://developer.arm.com/documentation/den0137/latest/

[22] Arm Limited. 2023. Reference Arm CCA integration stack Software User Guide. https://gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-docs/-/blob/master/docs/aemfvp-a-rme/user-guide.rst

[23] Arm Limited. Accessed Feb 2024. Arm Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

[24] Arm Limited. Accessed Feb 2024. linux-cca. https://gitlab.arm.com/linux-arm/linux-cca

[25] Arm Limited. Accessed Feb 2024. TrustZone for Cortex-A. https://www.arm.com/technologies/trustzone-for-cortex-a

[26] Zi-Jie Lin, Chuan-Chi Wang, Chia-Heng Tu, and Shih-Hao Hung. 2022. Performance Acceleration of Secure Machine Learning Computations for Edge Applications. In *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 138–147.

[27] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 619–631.

[28] Yunpeng Liu, Kexin Li, Zhuotao Liu, Bihan Wen, Ke Xu, Weiqiang Wang, Wenbiao Zhao, and Qi Li. 2023. Provenance of Training without Training Data: Towards Privacy-Preserving DNN Model Ownership Verification. In *Proceedings of the ACM Web Conference 2023*. 1980–1990.

[29] Ziyu Liu, Yukui Luo, Shijin Duan, Tong Zhou, and Xiaolin Xu. 2023. MirrorNet: A TEE-Friendly Framework for Secure On-Device DNN Inference. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[30] 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 *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 161–174.

[31] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. 2022. SoK: machine learning with confidential computing. *arXiv preprint arXiv:2208.10134* (2022).

[32] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 19–38.

[33] Claudio Orlandi, Alessandro Piva, and Mauro Barni. 2007. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security* 2007 (2007), 1–11.

[34] Sandro Pinto and Nuno Santos. 2019. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)* 51, 6 (2019), 1–36.

[35] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia conference on computer and communications security*. 707–721.

[36] Ye Sang, Yujin Huang, Shuo Huang, and Helei Cui. 2023. Beyond the Model: Data Pre-processing Attack to Deep Learning Models in Android Apps. In *Proceedings of the 2023 Secure and Trustworthy Deep Learning Systems Workshop*. 1–9.

[37] M Sardar, Thomas Fossati, and Simon Frost. 2023. SoK: Attestation in confidential computing. *ResearchGate pre-print* (2023).

[38] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, and Ronald Perez. 2022. Sok: Hardware-supported trusted execution environments. *arXiv preprint arXiv:2205.12742* (2022).

[39] Sandra Servia-Rodríguez, Liang Wang, Jianxin R. Zhao, Richard Mortier, and Hamed Haddadi. 2018. Privacy-Preserving Personal Model Training. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 153–164.

https://doi.org/10.1109/IoTDI.2018.00024

[40] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, et al. 2022. {SOTER}: Guarding Black-box Inference for General Neural Networks at the Edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 723–738.

[41] Supraja Sridhara, Andrin Bertschi, Benedict Schlüter, Mark Kuhne, Fabio Aliberti, and Shweta Shinde. 2024. ACAI: Extending Arm Confidential Computing Architecture Protection from CPUs to Accelerators. In *33rd USENIX Security Symposium (USENIX Security'24)*.

[42] Lizhi Sun, Shuocheng Wang, Hao Wu, Yuhang Gong, Fengyuan Xu, Yunxin Liu, Hao Han, and Sheng Zhong. 2022. LEAP: TrustZone Based Developer-Friendly TEE for Intelligent Mobile Apps. *IEEE Transactions on Mobile Computing* (2022).

[43] Yuchen Sun, Tianpeng Liu, Panhe Hu, Qing Liao, Shouling Ji, Nenghai Yu, Deke Guo, and Li Liu. 2023. Deep Intellectual Property: A Survey. *arXiv preprint arXiv:2304.14613* (2023).

[44] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. 2023. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1596–1612.

[45] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. 2021. 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)*. 1955–1972.

[46] TensorFlow. Accessed Feb 2024. MobilenetV1. https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

[47] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *International Conference on Learning Representations*.

[48] TrustedFirmware. Accessed Feb 2024. TF-A. https://www.trustedfirmware.org/projects/tf-a

[49] TrustedFirmware. Accessed Feb 2024. TF-RMM. https://www.trustedfirmware.org/projects/tf-rmm

[50] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law. 2019. SEALion: A framework for neural network inference on encrypted data. *arXiv preprint arXiv:1904.12840* (2019).

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

[52] Xiangyi Xu, Wenhao Wang, Yongzheng Wu, Zhennan Min, Zixuan Pang, and Yier Jin. 2023. virtCCA: Virtualized Arm Confidential Compute Architecture with TrustZone. *arXiv preprint arXiv:2306.11011* (2023).

[53] Mingfu Xue, Yushu Zhang, Jian Wang, and Weiqiang Liu. 2021. Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations. *IEEE Transactions on Artificial Intelligence* 3, 6 (2021), 908–923.

[54] Yiming Zhang, Yuxin Hu, Zhenyu Ning, Fengwei Zhang, Xiapu Luo, Haoyang Huang, Shoumeng Yan, and Zhengyu He. 2023. SHELTER: Extending Arm CCA with Isolation in User Space. In *32nd USENIX Security Symposium (USENIX Security'23)*.

[55] Ziqi Zhang, Chen Gong, Yifeng Cai, Yuanyuan Yuan, Bingyan Liu, Ding Li, Yao Guo, and Xiangqun Chen. 2024. No Privacy Left Outside: On the (In-) Security of TEE-Shielded DNN Partition for On-Device ML. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 52–52.