SecureQNN: Introducing a Privacy-Preserving Framework for QNNs at the Deep Edge*

Miguel Costa[1 [0000−0003−2046−4569]], Tiago Gomes[1 [0000−0002−4071−9015]], Jorge Cabral[1 [0000−0001−9954−9746]], João Monteiro[1 [0000−0002−3287−3995]], Adriano Tavares[1 [0000−0001−8316−6927]], and Sandro Pinto[1 [0000−0003−4580−7484]]

ALGORITMI Research Centre, Universidade do Minho, PT
miguel.costa@dei.uminho.pt

Abstract. Recent concerns about real-time inference and data privacy are making Machine Learning (ML) shift to the edge. However, training efficient ML models require large-scale datasets not available for typical ML clients. Consequently, the training is usually delegated to specific Service Providers (SP), which are now worried to deploy proprietary ML models on untrusted edge devices. A natural solution to increase the privacy and integrity of ML models comes from Trusted Execution Environments (TEEs), which provide hardware-based security. However, their integration with heavy ML computation remains a challenge. This perspective paper explores the feasibility of leveraging a state-of-the-art TEE technology widely available in modern MCUs (TrustZone-M) to protect the privacy of Quantized Neural Networks (QNNs). We propose a novel framework that traverses the model layer-by-layer and evaluates the number of epochs an attacker requires to build a model with the same accuracy as the target with the information disclosed. The set of layers whose information makes the attacker spend less training effort than the owner training from scratch is protected in an isolated environment, i.e., the secure-world. Our framework will be evaluated in terms of latency and memory footprint for two ANNs built for the CIFAR-10 and Visual Wake Words (VWW) datasets. In this perspective paper, we establish a baseline reference for the results.

Keywords: Machine Learning · Artificial Neural Networks · Quantized Neural Networks · ML Model Privacy · TEE · TrustZone-M · Armv8-M.

1 Introduction

Advances in computing processing and the availability of vast amounts of data have propelled Machine Learning (ML) to the forefront of various domains, including healthcare [1], finance [2], and mobility [3], [4]. Among the wide range

* We would like to thank the anonymous reviewers for their valuable feedback and suggestions. This work is supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope UIDB/00319/2020. Miguel Costa was supported by FCT grant SFRH/BD/146780/2019.

of algorithms used in ML, Artificial Neural Networks (ANNs) have emerged as a prominent choice. However, training ANNs often requires access to large-scale private datasets, which are not typically held by ML clients. To address this challenge, big data companies introduced Machine Learning as a Service (MLaaS), enabling clients to use proprietary models for inference tasks using personal data.

Typical MLaaS follow a centralized computing paradigm where the client sends its data to the cloud to get the results. However, heavy reliance on the cloud induces unpredictable latency due to network overload which may render ML services useless in real-time scenarios [5], [6]. This combined with rising concerns about data privacy is making ML shift to the edge of the network [5], [6]. If this increases privacy for the user as its data is not transferred over the network, the same does not apply to Service Providers (SPs) as their models would be deployed in untrusted edge devices. Unauthorized access to these models can lead to the exposure of proprietary algorithms [7], reverse engineering [7], or even the extraction of sensitive training data [7], [8]. Moreover, recent attacks have demonstrated that malicious manipulation of ANN parameters can compromise the integrity of decision-making processes [9], [10]. All these issues combined gave rise to a still open research question: "Is it possible to transfer a proprietary ML model to the edge while providing privacy guarantees to the SP?".

A primary response to this concern comes from hardware-based security mechanisms, such as Trusted Execution Environments (TEEs). TEEs enforce memory isolation between applications running in a Rich Execution Environment (REE) and smaller critical applications isolated by hardware [11]. Consequently, researchers have started to adapt off-the-shelf TEEs, such as Intel-SGX [12] and ARM TrustZone [13], [14], to protect ML computations at the edge of the network. However, TEEs are designed to execute small critical operations and tend to pose severe memory constraints to resource-hungry ML computation [15]–[20]. Furthermore, they do not extend to accelerators such as GPUs or ASICs [15], [17]–[19], [21]. As a consequence, developing privacy-enhanced ML frameworks on top of TEEs remains an open challenge.

Prior works [15], [18], [22], [23] propose to encrypt the model and store it in unprotected memory. When the inference process starts, the model is loaded and decrypted into the enclave, typically layer-by-layer to overcome the memory limits. If the activations can not be stored within the enclave to serve as input to the next layer, they are encrypted before being stored into an unprotected memory region. Despite the novelty, these approaches suffer from increased latency disrupted by the decryption of all ANN parameters and the frequent context switch from the secure-world to the normal-world. To counteract these issues, some works propose to (i) perform loading in parallel with prediction [20]; (ii) execute only sensitive layers within the enclave [16], [19], [21]; (iii) obfuscate only relevant ANN weights [17]; or (iv) refactor a model in a bident structure [24].

Despite their novelty, prior works target x86 and Arm Cortex-A processors, letting out Arm Cortex-M processors, which are at the forefront development of IoT applications. The recent support for friendly ML APIs (CMSIS-NN and TensorFlow Lite Micro) is pushing Arm Cortex-M processors even further [5], [25].

As some Arm Cortex-M processors are now being shipped with TrustZone-M technology, a new question arises: *can TEEs be leveraged to protect the privacy of ANNs on resource-constrained Arm Cortex-M MCUs?* Although previous works show impressive results for the Intel-SGX and Arm Cortex-A architectures, they may not hold to Arm Cortex-M MCUs as these devices only hold a few megabytes of memory and much lower throughput. The integration of Quantized Neural Networks (QNNs), which operate with reduced precision and memory requirements, and TrustZone-M may hold great promise for advancing ML capabilities on these devices while ensuring the privacy and security of sensitive ML models.

In this perspective paper, we dig into this topic by providing an initial overview of the feasibility of leveraging the Arm TrustZone-M technology to protect the privacy of a QNN. Previous work [26] has found that the last layers of a neural network are the ones that reveal more sensitive information about the classification task, while the first ones reveal more information about the input itself. We borrow this finding to develop a novel framework that splits the execution of a QNN between the secure and normal worlds while minimizing the number of layers delegated to the secure-world to the bare minimum. As the last layers tend to be smaller than the first ones, we believe this approach could be a good starting point to tackle the severe memory constraints imposed by TrustZone-M. To abstract the implementation details of TrustZone-M, our framework works on top of Trusted Firmware-M (TF-M). By combining the built-in capabilities of TF-M for secure storage and by minimizing the number of layers within the TEE, we envision reducing the need for frequent context switches, while optimizing the access to critical data.

To evaluate our framework, we consider the most challenging scenario where the adversary has access to everything in the (untrusted) normal-world. In such a scenario, the attacker can freeze the non-protected layers, while iteratively designing and training possible adjacent layers till he gets a substitute model with an accuracy equal to or greater than the target model. In this context, we consider the privacy of the target model protected when the effort required to build the substitute model is, at least, equal to the effort required to train the target model from scratch. We measure the training effort in training epochs. We evaluate our framework (SecureQNN) in terms of TEE memory footprint and inference latency. The evaluation suite includes two QNNs designed and trained for the CIFAR-10 and Visual Wake Words (VWW) datasets. In this perspective paper, we establish the baseline reference for future evaluation. To the best of our knowledge, we are the first to propose this ANN splitting strategy to enhance privacy while utilizing a TEE. Furthermore, no previous works have specifically targeted Arm Cortex-M MCUs.

2 Background

2.1 Arm TrustZone-M

Arm TrustZone-M is a technology designed to enhance the security of MCUs following the Armv8-M architecture [11], [13]. TrustZone-M employs hardware-

based isolation to establish two distinct execution states: (i) the secure-world and (ii) the normal-world. The secure-world is the host for software components that are critical from the integrity or privacy perspective, while the normal-world accommodates less sensitive applications. The division between the two worlds is memory-based, meaning that when the processor is running from secure memory, the processor state is secure, and when the processor is running from non-secure memory, the processor state is normal. The secure-world has access to every memory region, while the normal-world only has access to the non-secure memory. For this purpose, TrustZone-M relies on two attribution units. The Security Attribution Unit (SAU) is programmable, allowing dynamic address partitioning. The Implementation-Defined Attribution Unit (IDAU) partitions memory statically and is defined at design time by the chip maker. To determine if a given address is secure or not, TrustZone-M performs a logical OR between the output of SAU and IDAU. By preventing unauthorized access to secure resources, TrustZone-M prevents malicious applications in the normal-world from impacting the integrity and privacy of the applications running in the secure-world.

2.2 Trusted Firmware-M (TF-M)

TF-M is an open-source firmware implementation that uses the TrustZone-M hardware capabilities, providing trusted firmware for Arm Cortex-M devices. It supports the Armv8-M and Armv8.1-M architectures and incorporates various security features, including (i) secure boot, (ii) software component isolation, (iii) protected storage of sensitive information, and (iv) secure firmware update.

TF-M uses MCUboot, as its secure bootloader, enabling the verification and authentication of applications running in the secure and normal worlds during the system startup. This prevents the execution of malicious code and helps establish a trusted foundation for the entire system. Furthermore, TF-M enforces isolation between the secure and normal worlds, as well as between applications within the secure-world if required. To partition the memory as secure and non-secure, TF-M relies on the hardware-enforced isolation provided by TrustZone-M through the SAU and IDAU units, as well as system-wide protections units, such as Peripheral Protection Controllers (PPCs), used to partition peripherals. To partition the secure memory between different Trusted Applications (TAs), TF-M relies on the secure memory protection unit.

Protected storage and firmware update are two important services provided by TF-M. The former relies on the hardware isolation provided by TrustZone-M to isolate a given fraction of the flash memory from the normal-world. This mechanism enables the secure storage and management of cryptographic keys, certificates, and other confidential data. Additionally, TF-M enables secure firmware updates, allowing devices to securely receive and install new firmware binaries. More specifically, it provides the device with the ability to verify digital signatures of received binaries, preventing unauthorized modifications or malicious code injection.

3 Related Work

Several mechanisms to ensure the confidentiality of a proprietary ML model have been proposed over the last few years. Table 1 reviews and puts into perspective the most relevant works in this area. In this paper, we restrict the scope to works that use TEEs to protect the privacy of the ML model. Works that employ TEEs to protect the confidentiality of user input data or enable collaborative learning from multiple-edge devices are out of the scope of this review. For a review including these subjects, we refer to the work [27].

Table 1. Gap Analysis

	Architecture TEE	Trusted Execution	Needs to fit on TEE?	Integrity Check	Description
VanNostrand *et al.* [15] 2019	Arm Cortex-A OP-TEE	All layers	Largest layer	No	The computation of every layer is performed on the TEE. Loads the model layer-by-layer, requiring frequent context switches.
OMG [22] 2020	Arm Cortex-A SANCTUARY	All layers	Full model	No	The computation of every layer is performed on the TEE. Loads the model in a single iteration, not requiring context switches during inference.
SecDeep [18] 2021	Arm Cortex-A OP-TEE	All layers Conf. functions	Largest layer	No	Splits Arm-NN into confidential and non-confidential functions. Confidential functions deal with private user and model data and are delegated to the TEE.
Nakai *et al.* [20] 2021	Arm Cortex-A OP-TEE	All layers	Largest layer	Yes	Uses the shared memory between the REE and the TEE to load the model in parallel with prediction.
MLCapsule [23] 2021	x86 Intel-SGX	All layers	Largest layer	No	The computation of every layer is performed on the TEE. Loads the model layer-by-layer, requiring frequent context switches.
Slalom [21] 2019	x86 Intel-SGX	Selected layers	Largest layer	Yes	Outsources the computation of linear layers from TEE to GPU without revealing weights and user data. Non-linear layers are executed within the TEE.
DarkneTZ [16] 2020	Arm Cortex-A OP-TEE	Selected layers	Largest layer	No	Delegates the execution of the layers most sensitive to MIAs to a TEE.
ShadowNet [19] 2023	Arm Cortex-A OP-TEE	Selected layers	Largest layer	Yes	Outsources the computation of linear layers from TEE to GPU without revealing weights and user data. Non-linear layers are executed within the TEE.
Hou *et al.* [17] 2022	x86 Intel-SGX	Selected neurons	Selected neurons Largest layer I/O	No	Obfuscates strategical weights to outsource most of the inference to the REE. The output of each layer is denoised within the TEE.
Lin *et al.* [24] 2020	-	Small asymetric model	Small asymetric model	No	Refactors the model into two asymmetric models, one on the TEE and the other on the REE. The model on the TEE has more weight in the final prediction.
Costa et al. 2023	**Arm Cortex-M TF-M**	**Selected layers**	**Selected layers or largest layer**	**No**	**Delegates the execution of the layers containing the most sensitive information about the classification task and training data to the TEE.**

3.1 All Layers Running Inside the TEE

A first approach consists in embedding all layers inside the TEE. This is the most basic but also the most effective technique to protect the privacy of a neural network. Nevertheless, it is the least optimized in terms of latency as it involves frequent context switches between the secure and normal worlds and typically requires cryptography to be executed over all ANN parameters.

OMG [22] is the only work relying on the SANCTUARY security architecture developed for Arm Cortex-A processors. OMG follows a very simple approach:

after the initialization phase, the full model is loaded and decrypted in a single iteration within the enclave and the inference process starts. Given the limited memory footprint of TEEs, OMG does not scale to large models.

The works [15], [23] tackle this issue by splitting the inference process layer-by-layer. For each layer, the weights are loaded from the non-secure to the secure memory and decrypted afterward. The inference process is fully confined to the TEE. However, if the outputs of a given layer do not fit in the secure memory, they are encrypted and offloaded to the non-secure region being loaded and decrypted within the TEE when the computation of the following layer starts. The process is repeated till the output layer, where the result of the classification is returned in plain text to the normal-world. Despite the similar ANN splitting process, these works have some differences. While MLCapsule [23] targets x86 processors with support to Intel-SGX, the work [15] targets Arm Cortex-A processors with OP-TEE. In addition to layer-based partitioning, the work [15] proposes sub-layer and branched partitioning. In the former technique, the layer is partitioned into sub-layers, which helps to deal with the memory limits of the enclave; however, at the cost of increased decision latency. Sub-layer partitioning requires the inputs of the current layer to be loaded and decrypted more than once to calculate the full set of layer outputs. Branched partitioning solves this problem by splitting the last layers of a neural network into independent layers with independent connections to the forward layers. However, this involves changes to the model architecture. Despite the novelty and potential of these splitting strategies, VanNostrand *et al.* [15] does not evaluate them.

The work [20] improves over the previous works by reducing the need for context switches anytime the parameters of a new layer have to be loaded into the TEE. For this purpose, Nakai *et al.* [20] uses the shared memory between the secure and normal worlds to load the parameters and execute the decryption and the prediction in parallel. The work targets Arm Cortex-A processors with support to OP-TEE.

When analyzing the source code of Arm-NN, an API tailored for the execution of ANNs in Arm Cortex-A processors, Liu *et al.* [18] found that more than 90% of the code implementing layer functions is dedicated to tensor preparation or performance optimization, while only less than 10% is dedicated to mathematical tensor computation with sensitive user and model data. Based on this finding, Liu *et al.* [18] proposed SecDeep, a framework that reformulates the Arm-NN functions implementing ANN layers into two types: (i) confidential functions and (ii) non-confidential functions.

3.2 Selected Layers Running Inside the TEE

A second approach reported in the literature resorts to running a set of selected layers only inside the TEE. When compared to the previous approach, this strategy is intended to reduce the memory footprint of TEEs, while also reducing the number of context switches between the secure and normal worlds.

Slalom [21] was one the first works to explore this technique by outsourcing the computation of linear layers from the enclave to an untrusted GPU. The

inference process starts and is managed from the enclave. However, whenever Slalom finds a linear layer on the pipeline, Slalom encrypts both the inputs and the layer parameters with a pre-computed pseudorandom stream and outsources the computation to a co-located GPU. The result is then decrypted within the enclave and the integrity is verified using the Freivalds' algorithm. Slalom targets x86 processors with support to Intel-SGX. Sun *et al.* [19] borrowed the idea behind this work and developed ShadowNet, a framework similar to Slalom but for mobile devices powered by Arm Cortex-A MCUs with support to OP-TEE.

DarkneTZ [16] proposes a different approach to speed up the inference latency on Arm Cortex-A processors. Nevertheless, DarkneTZ does not protect the model parameters like the previous works. Instead, DarkneTZ protects the privacy of the training data by offering robustness against membership inference attacks (MIAs). DarkneTZ traverses the model layer-by-layer and uses all the information available till the layer under analysis as input to a strong white-box MIA. The set of layers to which the attack succeeds is delegated to the TEE.

3.3 Selected Neurons

A third approach reported in the literature proposes to delegate the computation of critical neurons to the TEE. Hou *et al.* [17] propose a framework to design a secure version of the ML model with crafted random weights. More specifically, the framework adds crafted values to obfuscate strategical weights within the ANN. The crafted noise must change the predicted label of the query sample, rendering the accuracy of the obfuscated model useless. In this framework, the inference process is mostly carried out in the normal-world, being the TEE responsible for denoising the outputs of each layer. The framework distorts the top percentage weights with the highest absolute value. The distortion must force the attackers to pay at least the same machine training hours on retraining the crafted model with high accuracy as the model owner trains from scratch. The framework targets x86 processors with support for Intel-SGX.

3.4 Model Refactoring

Lin *et al.* [24] addressed the limitations of TEEs with a different strategy. The authors propose to refactor the model to be protected into two asymmetric models, one to be executed within a TEE and another within an untrusted computation unit. The output of both models is then fused by a few public layers. The protected model is expected to have a smaller memory footprint and contribute the most to the final prediction. The model deployed in the normal-world must not enable the attacker to reconstruct the model within the TEE or the original model. Despite reducing the memory footprint of the secure application, the examples and evaluation given in the original work suggest that the strategy may increase the memory footprint of the overall ML system, as the sub-model deployed in the normal-world can be bigger than the original one.

3.5 Gap Analysis

When compared to works that run all layers inside the TEE, SecureQNN has the potential to reduce the number of context switches between the secure and normal worlds. To reduce the TEE memory footprint, most works in this category encrypts the model parameters and save them in non-secure memory. During inference, the parameters are decrypted and loaded layer-by-layer to the secure-world. This strategy reduces the TEE memory footprint to the size of the largest layer but requires a context switch every time a new layer is processed.

Compared to other works in the same category, DarkneTZ also reduces the computation within the TEE to sensitive layers. However, DarkneTZ protects ANNs against MIAs, which has different requirements from protecting the privacy of model parameters. Slalom and ShadowNet work by offloading linear layers to a GPU. However, as the linear layers compose most of the layers of an ANN and the parameters used by the GPU are obfuscated, the outputs of linear layers always need to be denoised within the TEE, requiring frequent context switches between the normal and secure worlds. These two works are not easily portable to devices powered by Arm Cortex-M as they do not feature GPUs.

Works that delegate the computation of selected neurons to the TEE are the most efficient in terms of TEE memory footprint. The work [17] only requires space to save the selected neurons and the output of the layer under process. Despite performing most of the computation outside the TEE, this work still requires the outputs of each layer to be denoised within the TEE, which significantly increases the number of context switches compared to our framework.

The only work that relies on model refactoring [24] requires the same number of context switches as we envision for our framework. In terms of memory footprint, the results and the description of the model refactoring strategy are not deep enough to hold strong assumptions.

SecureQNN is the only work targeting Arm Cortex-M MCUs. Other works either target Arm Cortex-A or x86 processors, which are far more powerful and frequently coupled with GPUs, which can be used to overcome the performance bottleneck imposed by frequent context switches and cryptography operations. In addition, works that apply fine-grain techniques to select the layers (DarkneTZ [16]) or the neurons [17] to be performed within the TEE do not consider the impact of quantization. It is worth noting that while previous works target ANNs with floating-point precision, our work targets QNNs.

4 SecureQNN

4.1 Scope

SecureQNN targets IoT devices powered by Arm Cortex-M MCUs following the latest Armv8-M architecture and supported by TF-M. These tiny and low-power devices typically have a single central processing unit and feature a wide range of peripherals. However, GPU acceleration is not supported. The most advance Armv8-M MCUs support the TrustZone-M technology, which enables the programmer to create two execution environments: secure and normal.

4.2 Threat Model

Our threat model considers two agents: (i) an SP and a (ii) client. The SP uses its big data infrastructure to provide the client with a highly accurate ANN. The SP is concerned about the privacy of its model as it constitutes its intellectual property. Training an accurate model may require years of data storage and treatment and weeks to months of training. Furthermore, recent studies have shown that ANNs are vulnerable to MIAs [16], attacks specifically designed to understand if a data point is part of the training data. As ANNs are usually trained on private data belonging to a myriad of users, the SP wants to maintain the training data private, as leaking data would undermine the users' trust in the SP. Our work focuses on maintaining the privacy of ANN critical layers to reduce the risk of these threats.

The client gets the ANN trained by the SP on its local device, which is powered by an Arm Cortex-M MCU and must support TruztZone-M and TF-M. The client is an honest but curious adversary. He follows the protocol required by the SP but tries to learn about the ANN. In this context, we consider the client has access to everything in the normal-world, including memory and operating system. We consider the model provided by the SP as a gray-box, where part of it is executed and stored within the normal-world and the other part in the secure-world. The user has full access to the public part of the model and can use it to try to disclose the private part. We consider the user has no access to the training data. As data is the most valuable asset of an ML model, there is no interest in stealing the model if the attacker already has access to data.

4.3 Model Partitioning

Previous research [26] discovered that the initial layers of ANNs contain crucial information for breaking down the input data, while the subsequent layers play a more significant role in the classification task. We leverage this insight to divide a given ANN into secure and normal worlds. Figure 1 illustrates the overall process for model partitioning. The model partitioning framework will be built in Python using the Tensorflow v2 API as the basis. Tensorflow v2 provides seamless conversion from floating-point to integer precision models compatible with the TensorFlow Lite Micro and CMSIS-NN, the most prominent APIs for ML development in Arm Cortex-M MCUs.

Our framework systematically traverses the model layer-by-layer, starting from the final layer. At each layer, we expose the parameters of the current and forward layers to a training process. During this training process, we freeze the parameters of the disclosed layers and attempt to reconstruct the unknown ones. We consider the most challenging scenario, assuming that the attacker possesses complete knowledge of the model's architecture and training hyper-parameters. For every layer, we measure the number of epochs needed for the reconstructed model to achieve an accuracy equal to or higher than the model to be safeguarded. The minimum set of frozen layers that require, at least, the

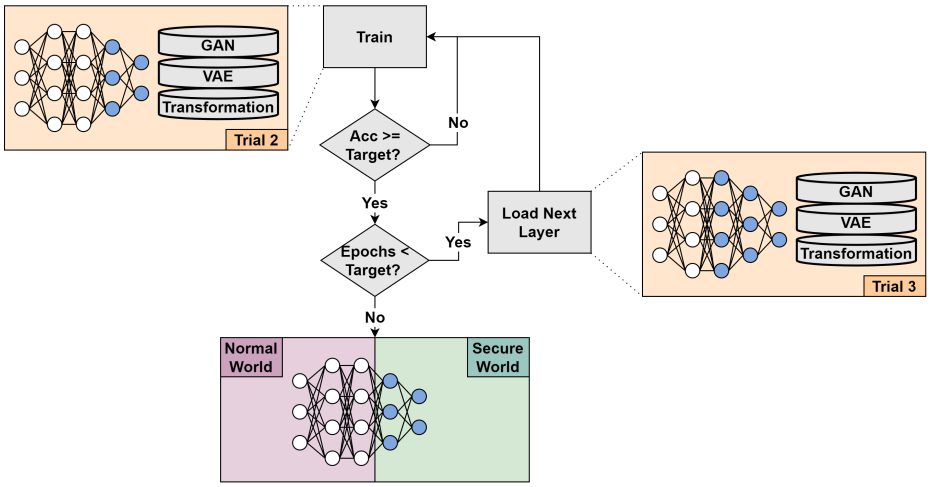

Fig. 1. Overview of the model partitioning technique. The example describes the second trial, where it is evaluated the criticality of the last two layers.

same epochs as those needed by the model owner to train from scratch will be allocated to the secure-world.

Since the attacker lacks access to the original training dataset, they must construct a substitute dataset. Our framework emulates this situation by employing data augmentation techniques on the original test dataset. Specifically, it utilizes three data augmentation methods: (i) image transformations, (ii) Generative Adversarial Networks (GANs), and (iii) Variational Auto-Encoders (VAEs). The augmented dataset is then used to simulate the training attempts conducted by a potential attacker.

4.4 Trusted Execution

Figure 2 details the architecture of an edge device running an ANN provisioned by SecureQNN. As depicted, we leverage the TrustZone-M technology to create two distinct execution environments: (i) normal and (ii) secure. Both environments feature privileged and non-privileged execution. The blocks running at the privileged level are depicted in gray, while the non-privileged blocks are depicted in green (secure-world) and red (normal-world).

Within the privileged level of secure-world, the Secure Boot is essential to detect any attempt of tampering with the firmware image and therefore with the ANN itself. For this purpose, the Secure Boot process verifies the digital signature of the firmware during the device startup. If the firmware is found modified, the boot process is aborted. For a reference implementation of secure firmware, we use TF-M. From the panoply of secure services offered by TF-M,

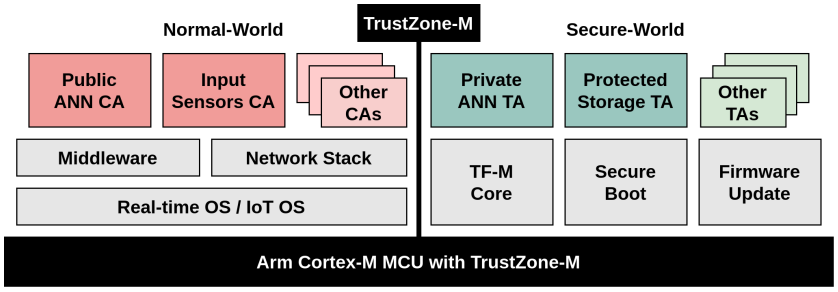

Fig. 2. Overview of the ANN trusted execution

the Firmware Update is the most relevant for our framework, as it enables secure over-the-air updates of the ANN. Whenever the SP wants to deploy a new ANN, it wraps the updates in a new firmware and sends it to the edge device. As the firmware is downloaded and stored in a secure memory space, the authenticity and integrity of the new firmware are verified. If the digital signature is valid, the new firmware starts to run. The Firmware Update service also features an anti-rollback mechanism to prevent an attacker from installing an older firmware image, and consequently, an outdated ANN.

Regarding the non-privileged environment of the secure-world, SecureQNN relies on two main TAs. The Protected Storage is a service implemented by TF-M and is responsible to store in a secure memory region the parameters of the private ANN layers. The Private ANN is the TA responsible for computing the output of the set of private layers. Whenever this TA is called, it interacts with the Protected Storage to load the ANN parameters and then proceeds with the inference process. The final output is sent in plain text to the Public ANN CA.

Within the normal-world, most of the software blocks are optional, except for the Input Sensors and the Public ANN CAs. The former is responsible for collecting the ANN input data, while the latter is responsible for starting and coordinating the inference process until a private layer is found.

4.5 QNN Execution Flow

As the inference process starts, the Input Sensors CA transfers the collected user/sensor data to the Public ANN CA, which controls the overall inference process. We do not make assumptions about the confidentiality and integrity of the user data, as this is out of the scope of our work. Although protecting the privacy of an ANN reduces the vulnerability to white-box adversarial attacks, our system would still be vulnerable to gray-box and black-box attacks. As the Public ANN CA finds a private layer, it uses the TF-M communication channels to send the output of the last public layer to the Private ANN TA.

As the RAM available in the secure-world is allocated during the boot and maintained static afterward, optimizing the amount of RAM required by the Private ANN TA is fundamental to not undermine the applications running in

the normal-world. For this purpose, the ANN parameters will be stored in the flash memory and loaded layer-by-layer to the RAM. The Protected Storage TA handles the memory read process, while maintaining the privacy of the ANN parameters. In this context, we estimate that the maximum amount of RAM required by the Private ANN TA equals the parameter size of the largest private layer plus the size of the auxiliary buffer required in the *im2col* process of convolutional layers. The output of the last private layer (prediction vector) is sent in plain text to the Public ANN CA using the TF-M communication channel.

5 Preliminary Results

We evaluate SecureQNN in terms of latency and TEE memory footprint for two distinct QNNs. To provide a fair evaluation of our framework, we first need to establish the baseline reference values for the scenario where the full computation of the QNN is delegated to the secure-world.

SecureQNN pretends to address current or future applications of ML in IoT. Consequently, we evaluate our framework on two QNNs borrowed from the TinyML benchmark [28] - an open-source benchmark for ML workloads on emerging low-power hardware. These QNNs are trained to perform inference on the CIFAR-10 and VWW datasets. CIFAR-10 comprises 60,000 RGB images measuring 32x32 pixels. These images are categorized into 10 distinct classes, each representing a different object. The dataset is divided into 50,000 training images and 10,000 testing images. On the other hand, the VWW dataset is binary and contains 115,000 RGB images measuring 96x96 pixels. Its purpose is to identify whether a person is present in the image or not. The training subset of VWW consists of 98,568 images, while the testing subset contains 10,961 images. The results are extracted for an STM Nucleo-L552ZE-Q board, powered by an Arm Cortex-M33 MCU running at 120 MHz.

5.1 Baseline Reference Values

Table 2 shows the baseline reference of latency and TEE memory footprint for the two QNNs described above. As expected, the memory footprint of the TEE when running a full QNN is incredibly high. Regarding RAM, when the TEE runs the full CIFAR-10 and VWW QNNs, it requires 55.51% and 69.43% of the total RAM available on the board, respectively. Regarding flash memory, CIFAR-10 and VWW QNNs take 22.49% and 55.51% of the space available. Under this umbrella, we can securely argue that fully delegating the computation of a QNN to the secure-world is unfeasible in a real-world scenario, as this puts severe restrictions on the memory available in the normal-world. TF-M allocates the secure and non-secure memory regions during the boot process and the amount of memory allocated is maintained statically afterward. Consequently, allocating too much memory for the secure-world severy limits the number of applications that can be deployed in the normal-world, which might jeopardize the purpose of having ML privacy mechanisms in IoT and put the model owner at risk. Our

Table 2. Baseline reference values

	TEE Memory Footprint (kB)		Latency	
	RAM	**Flash**	**Clock Cycles**	**Milliseconds**
CIFAR-10	142.09 (55.51%)	125.36 (24.49%)	99483061	829.03
VWW	177.90 (69.43%)	284.21 (55.51%)	75017684	625.15

framework tackles this issue by reducing the computation performed within the TEE to the bare minimum while maintaining the privacy of the critical layers.

Regarding latency, the values in Table 2 already consider two context switches between the normal and secure worlds, one before and one after the inference process. We envision our framework will keep up with these values, as the QNN splitting strategy that we propose also requires no more than two context switches.

6 On the Road

SecureQNN's development is still at an embryonic stage. As of this writing, we are developing the Python framework that evaluates the privacy leaking of each layer in a QNN. Our framework requires the following inputs: (i) the QNN to be protected, (ii) the number of training epochs, and (iii) the test accuracy. It then identifies which layers should operate within the secure-world. In the first stage of development, we will perform the evaluation layer-by-layer, starting from the last layer. As our framework targets resource-constrained Arm Cortex-M MCUs, we don't expect QNNs to have a large number of layers. Nevertheless, to speed up the process, in the second stage of development, we might consider the use of group-layer sorting strategies - instead of evaluating each layer individually, we evaluate a given set of layers at each iteration.

The last stage of development addresses the implementation of a mechanism to trade off privacy and TEE memory footprint. More specifically, SecureQNN will return the TEE memory footprint for different sets of private layers along with the number of epochs they require to achieve the same accuracy as the target model. This will allow the user to evaluate if it is worth increasing the privacy at the cost of memory footprint. Whenever the minimum privacy guarantee is achieved before the memory footprint reaches the limit, SecureQNN will continue to evaluate the additional privacy guarantee by iteratively extending the set of protected layers till the maximum memory footprint is met.

7 Conclusion

In this paper, we present SecureQNN, a framework to protect the privacy of QNNs in Arm Cortex-M MCUs featuring TrustZone-M technology. Although development is still embryonic, we already outline the testing scenario and the baseline reference values for the TEE memory footprint (RAM and flash) and

decision latency. We outline SecureQNN's development roadmap, which includes two fundamental stages to determine which layers are more privacy-critical and to trade off TEE memory footprint and privacy. One additional step will attempt to speed up the QNN splitting strategy. Once completed, we will open-source the SecureQNN framework.

References

[1] R. Miotto *et al.*, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in Bioinformatics*, 2017.

[2] S. Ahmed *et al.*, "Artificial intelligence and machine learning in finance: A bibliometric review," *Research in International Business and Finance*, 2022.

[3] M. Costa *et al.*, "Detecting driver's fatigue, distraction and activity using a non-intrusive AI-based monitoring system," *Journal of Artificial Intelligence and Soft Computing Research*, 2019.

[4] Y. Li *et al.*, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[5] D. Costa *et al.*, "Train me if you can: Decentralized learning on the deep edge," *Applied Sciences*, 2022.

[6] M. G. S. Murshed *et al.*, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, 2021.

[7] M. Rigaki *et al.*, "A survey of privacy attacks in machine learning," *CoRR*, 2020.

[8] H. Hu *et al.*, "Membership inference attacks on machine learning: A survey," *ACM Comput. Surv.*, 2022.

[9] G. Li *et al.*, "Tensorfi: A configurable fault injector for tensorflow applications," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2018.

[10] Z. Chen *et al.*, "Binfi: An efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[11] D. Oliveira *et al.*, "uTango: An open-source TEE for IoT devices," *IEEE Access*, 2022.

[12] V. Costan *et al.*, *Intel SGX explained*, 2016. [Online]. Available: https://eprint.iacr.org/2016/086.

[13] S. Pinto *et al.*, "Demystifying Arm TrustZone: A comprehensive survey," *ACM Comput. Surv.*, 2019.

[14] D. Cerdeira *et al.*, "ReZone: Disarming TrustZone with TEE privilege reduction," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[15] P. M. VanNostrand *et al.*, "Confidential deep learning: Executing proprietary models on untrusted devices," *CoRR*, 2019.

[16] F. Mo *et al.*, "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.

[17] J. Hou *et al.*, "Model protection: Real-time privacy-preserving inference service for model privacy at the edge," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[18] R. Liu *et al.*, "SecDeep: Secure and performant on-device deep learning inference framework for mobile and IoT devices," in *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, 2021.

[19] Z. Sun *et al.*, "ShadowNet: A secure and efficient on-device model inference system for convolutional neural networks," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[20] T. Nakai *et al.*, "Towards trained model confidentiality and integrity using trusted execution environments," in *Applied Cryptography and Network Security Workshops*, 2021.

[21] F. Tramer *et al.*, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2019.

[22] S. P. Bayerl *et al.*, "Offline model guard: Secure and private ML on mobile devices," in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, 2020.

[23] L. Hanzlik *et al.*, "MLCapsule: Guarded offline deployment of machine learning as a service," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021.

[24] H.-Y. Lin *et al.*, "Bident structure for neural network model protection.," in *ICISSP*, 2020.

[25] M. Costa *et al.*, "Shifting capsule networks from the cloud to the deep edge," *ACM Trans. Intell. Syst. Technol.*, 2022.

[26] J. Yosinski *et al.*, "How transferable are features in deep neural networks?," MIT Press, 2014.

[27] M. F. Babar *et al.*, "Trusted deep neural execution - A survey," *IEEE Access*, 2023.

[28] C. R. Banbury *et al.*, "Benchmarking TinyML systems: Challenges and direction," *CoRR*, 2020.