Quantization Backdoors to Deep Learning Commercial Frameworks

Hua Ma, Huming Qiu, Yansong Gao, Senior Member, IEEE, Zhi Zhang, Senior Member, IEEE, Alsharif Abuadbba, Senior Member, IEEE, Minhui Xue, Anmin Fu, Jiliang Zhang, Senior Member, IEEE, Said F. Al-Sarawi, Senior Member, IEEE, and Derek Abbott, Fellow, IEEE

Abstract—This work reveals that standard quantization toolkits can be abused to activate a backdoor. We demonstrate that a full-precision backdoored model which does not have any backdoor effect in the presence of a trigger—as the backdoor is dormant—can be activated by (i) TensorFlow-Lite (TFLite) quantization, the only product-ready quantization framework to date, and (ii) the beta released PyTorch Mobile framework. In our experiments, we employ three popular model architectures (VGG16, ResNet18, and ResNet50), and train each across three popular datasets: MNIST, CIFAR10, and GTSRB. We ascertain that all trained float-32 backdoored models exhibit no backdoor effect even in the presence of trigger inputs. Particularly, four influential backdoor defenses are evaluated, and they fail to identify a backdoor in the float-32 models. When each of the float-32 models is converted into an int-8 format model through the standard TFLite or PyTorch Mobile framework’s post-training quantization, the backdoor is activated in the quantized model, which shows a stable attack success rate close to 100% upon inputs with the trigger, while it usually behaves upon non-trigger inputs. This work highlights that a stealthy security threat occurs when an end-user utilizes the on-device post-training model quantization frameworks, informing security researchers of a cross-platform overhaul of DL models post-quantization even if these models pass security-aware front-end backdoor inspections. Significantly, we have identified Gaussian noise injection into the malicious full-precision model as an easy-to-use preventative defense against the PQ backdoor.

Index Terms—Deep learning, PyTorch mobile, quantization backdoor attack, TensorFlow-lite.

I. INTRODUCTION

DEEP learning (DL) empowers a wide range of applications, such as computer vision and natural language processing. Traditionally, DL models are trained and hosted in the cloud for inference tasks. While the cloud provides massive computing power for model training, it does introduce latency for model inference (each inference requires network connectivity for a round-trip to the cloud), posing a severe challenge to real-time applications (e.g., autonomous driving). In addition, users’ personal data have to be submitted to the cloud for recommendation or inference tasks, putting data privacy at risk. To address these key constraints, on-device DL (also called TinyML [1]) empowers end-users to perform model inference directly in ubiquitous IoT devices. Compared to in-cloud DL, on-device DL voids the requirement of connecting local data to the cloud and preserves user data privacy in this context.

Recently, on-device DL models have been deployed to various IoT devices such as microcontrollers (MCUs), the smallest computing platforms present almost everywhere. As IoT devices are resource-constrained, on-device DL cannot directly deploy a DL model trained from the cloud onto the devices. Particularly, a model is usually trained using a floating-point precision format (e.g., a full precision in float-32) and can be much larger (e.g., a few hundred MiB) than the memory capacity (within a few MiB) of most IoT devices. Further, the model requires computing-intensive floating-point operations (FLOPS), that are unlikely to be supported in low-power IoT devices (e.g., drones and smart watches). To make DL ubiquitous, a practical solution called model quantization has been proposed [4]. It reduces the precision of a trained model by converting a high-precision format to a low-precision format, without sacrificing model accuracy—retained to almost the same level for prior and post quantization. Building upon TensorFlow (TF) [5], Tensorflow-lite (TFLite) [4] is an open-source framework for on-device machine-learning and deep-learning models. As shown in Fig. 1, TFLite quantizes

1 The world is estimated to have over 250 billion MCUs in 2020 and the number is increasing rapidly [3].
a full-precision TF model for mobile or embedded devices that support mainstream platforms (i.e., Android, iOS and Linux) in multiple programming languages (i.e., Java, C++ and Python). We choose TFLite as the main object of this study because it is the only product-ready quantization framework and has been deployed onto more than 4 billion mobile devices up to 2020 [6]. The other industrial solution is the PyTorch Mobile framework [7] but its stable release is unavailable yet at the time of writing, which, nonetheless, is considered in this study as a complement.

While we enjoy the usability and efficacy of the post-training model quantization techniques provided by standard frameworks or toolkits, we ask the following question:

Do DL model quantization frameworks, such as TFLite and PyTorch Mobile, introduce any security implications due to the precision-format conversion, thus threatening their insouciant usability?

Our Work. We provide a positive answer to the question, where the model quantization framework can be maliciously exploited for a backdoor attack. The attack is two-fold:

(i) Prior Quantization: Before a full-precision DL model is quantized for on-device inference, we stealthily poison the model with a dormant backdoor such that the model inference accuracy is not affected regardless of the presence of a trigger, indicating that, in principle, the model can bypass all state-of-the-art backdoor detection approaches.

(ii) Post Quantization: After the model is quantized via the quantization framework, the backdoor is woken up, enabling trigger inputs to hijack the model inference deterministically.

The key insight is that (post) model quantization will introduce truncation errors when model parameters are converted from a high precision format to a low precision format during model quantization. With this key observation, an attacker can insert a backdoor into a pretrained model and keep it dormant by exploiting the floating-point format, thus evading existing backdoor detection. Once quantized, the backdoor can be activated by a trigger input (e.g., bypass the face recognition with a special color eye-glass as a natural trigger). Even worse, the in-field and unmanned environment of most IoT applications surreptitiously facilitates backdoor attacks at model inference.

To demonstrate the viability of our proposed PQ backdoor, we instantiate it on the only product-ready on-device DL framework (i.e., TFLite), which currently provides three available format options for post-training quantization: dynamic range quantization, float-16 quantization and full integer quantization [8]. In our implementation, a victim user is assumed to choose the full integer quantization, which has the best memory efficiency and computing speedup among all the format options [8]. Furthermore, the user will quantize the model weight using the int-8 precision format, a preferred one in DL applications of IoT devices, because int-8 operations are commonly supported by MCUs which may not support floating-point operations [9].

By completing the model quantization, the user obliviously wakes up the PQ backdoor within the quantized model, which exhibits backdoor behaviors upon trigger inputs. Besides the TFLite framework, we also equally demonstrate the PQ backdoor successfully against PyTorch Mobile, a popular beta release framework.

Challenges: At a high level, a successful PQ backdoor must address two challenges, that is, within a full-precision trained model, its backdoor remains dormant and cannot be detected by state-of-the-art backdoor defenses. After the model quantization, the backdoor is woken up and exhibits conspicuous backdoor effect in the quantized model. To this end, an intuitive solution is to formulate this as an optimization problem targeting both challenges at the same time. However, such PQ backdoor solution is extremely unstable and ineffective (see details in Section VII-A).

We instead overcome this ineffective PQ backdoor solution with a novel two-step strategy. We train a backdoored full-precision model in the first step. We then gradually remove the backdoor effect from the full-precision model by re-training,
thus making the backdoor dormant in the second step. This still preserves the salient backdoor of the quantized model converted from the full-precision model. In the second step, we retain the backdoor effectiveness in the quantized model by leveraging projected gradient descent (PGD) technique [10]. With the PGD, the PQ backdoor is guaranteed to be converged stably with high attack efficacy. When we execute the PQ backdoor that is essentially a DL training process, a massive inference is required to assess the loss via the TFLite quantized model. However, performing the TFLite quantized model inference in an int-8 format on the x86-based test machine is very slow. For example, one single epoch costs more than 83 hours for PQ backdooring the ResNet18 model with CIFAR10, where the batch size is 32 and the number of epochs is 100. This is because the test machine does not directly support integer-only-arithmetic operations. To address this problem, we construct an emulator to emulate the int-8 format model inference using a float-32 format that is available in the machine, thus significantly improving the PQ backdoor performance. Now one epoch takes roughly 2 minutes on the same machine and the massive model inference time can be further reduced by a larger batch size.

**Contributions:** We summarize our contributions as follows:

- To the best of our knowledge, we are the first to show that the product-ready TFLite framework is vulnerable to backdoor attacks, i.e., the standard post-training quantization can be exploited to obliviously activate a backdoor that is dormant in a full-precision DL model. Our proposed PQ backdoor is generic and also equally effective on the popular beta release of PyTorch Mobile framework.
- We propose the quantization backdoor by exploiting the unavoidable truncation errors to insert backdoors that are activated by the model quantization, in particular, provided by industrial frameworks. Most importantly, the full-precision DL model has no explicit backdoor behaviors and thus, in principle by its nature, bypasses all backdoor detection.
- We formulate the quantization backdoor as an optimization problem. Its implementation can be stably and efficiently carried out by collaboratively optimizing several properly-defined objective loss functions. We have identified several technical challenges during the implementation and proposed effective solutions.
- We evaluate the PQ backdoor attack performance on three popular model architectures, from VGG16, ResNet18 to the deeper ResNet50, over MNIST, CIFAR10 and GTSRB datasets, and validate the high stealthiness and strengths of the PQ backdoor attack. In addition, we find that the PQ backdoor can be easily incorporated with stealthier backdoor variants, e.g., a source-specific backdoor. Moreover, the PQ backdoor efficacy is comprehensively validated through three configuration forms/modes of TFLite and two configuration forms/modes of PyTorch Mobile, as summarized in Table I.
- We apply four state-of-the-art backdoor detection approaches, i.e., Neural Cleanse [11], STRIP [12], ABS [13] and MNTD [14], to evaluate the backdoor prior and post-quantization, results of which affirm that the backdoor effect cannot be detected in the full-precision model though it is still effective in the quantized model. Due to non-robustness, we find that ABS and MNTD fail to capture the backdoor in the quantized model even for the common source-agnostic/input-agnostic backdoor, which they should have been able to. Rather than merely relying on backdoor detection, we propose a lightweight backdoor removal defense that can effectively remove the PQ backdoor once it is properly implemented.

**Responsible Disclosure:** We have reported our attack to the Google Tensorflow-Lite team as this is a product-ready tool. The team has confirmed that the attack cannot be mitigated through tuning the TFLite implementation, since the root cause of this vulnerability is pertinent to the general post-training quantization design in lieu of the specific TFLite implementation per se. The authors also confirm there are no ethical concerns or conflicts of interest as of the time of paper submission. We have also informed Facebook PyTorch team of our attack against PyTorch Mobile.

**Paper Organization.** Preliminaries are presented in Section II. Section III defines our threat model and the need to perform front-end full-precision model inspection. Section IV first provides an overview of the proposed PQ backdoor, then elaborates on the PQ backdoor attack implementation, technical challenges, and solutions. Section V conducts extensive experiments to validate the strength of the PQ backdoor attack, where the quantized model exhibits a high attack success rate while its original full-precision model has no backdoor effect. Section VI uses a number of the most popular backdoor detection approaches to examine backdoor behavior in the full-precision model. Section VII further discusses the PQ backdoor and compares it with related work by comprehensive experiments. Followed by conclusion in Section VIII.

## II. Preliminaries

We introduce preliminaries including succinct descriptions of model quantization and backdoor attacks.

### A. Model Quantization

Model quantization is a conversion technique that minimizes model size while also speeding up CPU and hardware based inference, with no or little degradation in model accuracy. It can be generally categorized into two classes. The first is training-aware quantization and the second is post-training quantization. Each has its own advantages. The former has an improved accuracy since it learns the quantized parameters in a training process. But as its name indicates, it generally requires training

| Table I | Summary of Our Extensively Evaluated Quantization forms/modes of TFLite and PyTorch Mobile and the Sections That Perform the Evaluations |
|----------------|---------------------------------------------------------------|
| Framework | TFLite | PyTorch Mobile |
| Quantization mode/form | int-8 weights | int-8 activations static | int-8 weights | int-8 activations dynamic | int-8 weights | int-8 activations dynamic | PyTorch Mobile |
| Section  | 5.3 | 6.3 | 7.5 | 7.5 | 7.3 | 7.3 | 7.4 | 7.4 | 7.4 | 7.4 |
the quantized model from scratch. Training-aware quantization can range from 8-bit, 4-bit and even down to 1-bit [15], [16], [17], [18], [19], [20] for not only weights but also activation [21], [22]. Particularly, 1-bit quantized models are called binary neural networks (BNNs) where both model parameters and activations can be represented by two possible values, -1/0 and +1, significantly reducing the memory size requirement. In addition, the floating-point operations (FLOPS) are replaced by simpler operations such as the XNOR logical operation and Bitcount to speed up model inference [24], which are showcased in customized CPU kernels. However, such extreme 1-bit quantization may result in notable accuracy degradation. It is recognized that going below 8-bit width quantization almost always results in accuracy drops [25].

The post-training quantization obviates the time-consuming training process [25], [26], [27], which can be directly applied to available full-precision models. Most importantly, by leveraging a small calibration dataset to direct the model quantization, the accuracy of the quantized model through post-training quantization can be comparable to the quantized model gained through the training-aware quantization [28]. Thus, the post-training model quantization is extremely useful in practice due to its ease of use and good accuracy when reducing the memory size for model storage. In addition, it converts the float-32 weight format into other memory-efficient formats, in particular, int-8, which can be supported by most edge IoT devices. Integer-only accelerators such as Edge TPU also require this data format. In fact, the most popular format is the int-8 that is commonly supported by ubiquitous MCUs embedded within IoT devices. The TFLite is such a product-ready framework opposed to its beta released counterpart PyTorch Mobile framework to enable the post-training model quantization for real-world deployments.

B. Backdoor Attack

Backdoor attacks can cause severe consequences to DL models [29]. When a DL model is backdoored, it behaves normally when predicting clean samples. This can be measured by clean data accuracy (CDA) that is comparable to a clean model counterpart so that it is infeasible to detect the backdoor by only observing its accuracy for validation (clean) inputs. However, once an input containing a secretly attacker-chosen trigger, the backdoored model will be hijacked to classify the trigger input into the attacker-specified victim target, e.g., an administrator in a face recognition task. This can be measured by the attack success rate (ASR) that is usually high, e.g., close to 100% to ensure the attack efficacy once it is launched. The backdoor can be introduced into a DL model through diverse attack surfaces, including model outsourcing training [30], pretrained model reuse [31], data curation [32], and in distributed machine learning [33]. Considering the severe consequences of backdoor attacks, there have been great efforts devoted to detecting or eliminating backdoors both from either model-level [11], [13], [34] or data-level [12], [35], [36]. On the flip side, backdoors have also been used to function as a honey pot for catching adversarial examples [37], and serve as a watermark for protecting DL model intellectual properties [38], [39]. We note that there are efforts to build benchmarks of backdoor attacks/defenses [40] that can facilitate the evaluation of backdoor attacks/defenses.

Deep Learning Pipeline Vulnerability: It has been shown that the standard DL pipeline has vulnerabilities, which can be abused to launch attacks. One representative attack is the Image-Scaling Attack that abuses the Re-Scaling function when a large image is resized into a small one to be acceptable by the DL models [41]. Such Image-Scaling has been utilized to facilitate the backdoor attacks when the input-size of the model is known by the attacker [42]. In addition, transfer learning has been abused to complete an incomplete backdoor inserted into a pretrained model when it is utilized to customize a down-stream task, if an attacker has some knowledge of the down-stream task, especially the targeted label [43]. Furthermore, the data collection from public resources or third parties is also risky where label-consistent poisoned data (i.e., content is consistent with its label by human-eyes audit) can be used to insert a backdoor into a model trained over the poisoned data, usually requiring that an attacker has knowledge of the model architecture [44].

This work reveals a new vulnerability in the pipeline when the standard quantization toolkits/frameworks are normally applied to activate a dormant backdoor, which could have a much broader impact on end users due to the high demand for on-device DL models and the popularity of the TFLite and PyTorch Mobile as standard tools. We note there are a few quantization backdoor related studies [45], [46], [47]. However, they [46], [47] (excluding [45] that demonstrates the feasibility of the PyTorch Mobile only) are customized, are not evaluated against any commercial frameworks such as PyTorch Mobile or TF-Lite. In addition, [45], [46], [47] do not achieve stable and high attacking efficacy due to their ineffective methods. Instead of abusing the quantization pipeline, [48] studies backdoor insertion by abusing the pruning operation but evaluations are also only customized. More discussions and comparisons are detailed in Section VII-B.

III. THREAT MODEL AND FRONT-END INSPECTION

Here we elaborate on the threat model of the PQ backdoor. Without awareness of PQ backdoor, we note that the victim user prefers to perform front-end full-precision model inspection rather than inspecting the quantized models mainly due to three reasons as detailed (below) if the user does carry out backdoor behavior inspection.

2It should be noted that, in many cases, to maintain a good inference accuracy, parameters in certain layers or activations still require to be represented with a high precision data type, e.g., float-32 [23].
3The activations can be converted into int-8 or int-16, but the former is mainly applied to our experiments according to the TFLite official guide unless otherwise stated. Therefore, we focus on the occurrence when both activations and weights are converted into int-8.

4Our fully functional and reproducible source code was released in August 2021 at https://github.com/quantization-backdoor and preprint of this study was online in August 2021. According to the preprint date, our study and the concurrent work [45] were ahead [46], [47].
A. Threat Model

We specify the threat model from three aspects of the attacker: goal, knowledge, and capability.

Attacker Goal: The attacker aims to train a backdoor dormant full-precision model with the same CDA as its clean full-precision counterpart and exhibits no backdoor effect even in the presence of a trigger in the fed input. However, once this full-precision model is converted into a quantized model through the post-quantization toolkits (i.e., TFLite and PyTorch Mobile) with default operations. The dormant backdoor is awakened. Specifically, the backdoored quantized model has high attack ASR to achieve the attacker’s predefined malicious behavior (i.e., misclassification) in the presence of the attacker-chosen trigger. At the same time, the CDA is retained similarly to the clean quantized model counterpart for the clean input containing no trigger. Note that applying a post-quantization operation through the commercial quantization toolkits by the victim user is unique to the PQ backdoor attack compared to a conventional backdoor attack, where the user directly deploys and uses the full-precision model without any quantization operation in the latter case.

Attacker Knowledge: The attacker has access to the post-quantization toolkits that will be used by the victim user to convert the full-precision model. This is practical as these commercial toolkits (particularly, TFLite and PyTorch Mobile) are public to everyone. So that the attacker knows the details of the post-quantization operations and is able to obtain the quantized model during the implementation of the PQ backdoor. This work mainly focuses on the int-8 format as the quantized format because it has the best memory efficiency and computing speedup among all the format options [8]. In addition, the attacker knows the model architecture used and has access to the training dataset.

Attacker Capability: This work considers two common backdoor attack scenarios of outsourced models [30], [49] and pre-trained models [50]. In the latter case, the user downloads a model from a public platform such as ModelZoo, GitHub to directly use it. Both attack scenarios exist because a victim user can often have limited computational resources or DL expertise or programming skills. In this context, the attacker controls the training of the full-precision model. In addition, the attacker has full access to the training dataset of the target task (i.e., provided by the victim user in the model outsourcing). So that the attacker can manipulate the training process to set a proper objective function for backdoor attack optimization. More specifically, the attackers are full-precision model providers. For example, in the scenario of model outsourcing, the attacker can be the entity that provides model training outsourcing services. In the second scenario, the attacker can be any malicious entity that releases a backdoored model online.

B. Front-End Inspection

The user requires a full-precision trained model from the model provider or downloaded from the public, because the user can quantize the model later flexibly by choosing a low-precision format (e.g., int-8 or float-16) and/or post-quantization forms (e.g., dynamic range quantization or full integer quantization; more details and experimental validations in Section VII-E) that depend on the characteristics of the user’s devices and availability of resources. Our evaluated quantization forms are summarized in Table I.

The victim model user in our threat model is unlikely to have computational resources, DL expertise, and related programming skills. Otherwise, the user can train a model from scratch rather than outsourcing model training or downloading a pretrained full-precision model. To this end, the user prefers to inspect the full-precision trained model rather than the quantized model for the following three reasons.

First, existing model inspection defenses are devised by default for a model in the float-32 format, so they are unlikely to be squarely applicable to inspecting a quantized model with a low-precision format, in particular, int-8 in our PQ backdoor attack. Specifically, for MNTD, the features extracted to train its meta-classifier are logits before the softmax. Thus, MNTD is sensitive to how the logits are obtained, making itself non-robust. For example, the obtained logits are dependent on the model architecture and the training configuration (e.g., the number of epochs for training) of each shadow model. In our study, if a model is quantized to int-8, the range of logits is changed to [-128,+127] rather than continuous float-32 values. The changes in logits can make MNTD ineffective, as we validated afterwards. For Neural Cleanse and ABS, they are white-box methods and require access to the gradients or activation values of a target model. However, for a TFLite quantized model, its file format is .tflite, and computing the gradients or the activation values from such a format is infeasible. A possible solution is that the user first reads out all the model weights from a .tflite model and then converts the model format into a format usable to Neural Cleanse and ABS. However, the readout and format conversion is non-trivial as there is no available tool provided by TFLite. We note that there is no model architecture associated with .tflite file yet. More specifically, the .tflite file contains weights in int-8 format and the quantization factors for each layer of the model. When this information is used for inspection, the .tflite file must first be converted into a full-precision model, which requires not only reading weights from it and associating them to the corresponding model architecture, but also simulating the forward inference of the quantization (i.e., the weights need to multiply a standalone factor per layer) before the floating-point model can be constructed. These operations require in-depth knowledge of the TFLite to a large extent, which is not straightforward for common users who lack programming skills and such knowledge.

Second, it is non-trivial for the non-expert user to inspect a quantized model. As running an int-8 model on an x86 machine (the choice of mainstream users) is very slow, we address the inefficiency by emulating int-8 using float-32 with an appropriate emulator (see Section IV-B). However, building such an emulator is non-trivial, as it requires in-depth knowledge of the quantization implementation details of a specific commercial framework, e.g., per-tensor or per-channel quantization for a given quantization form. Additionally, as there are different
quantization forms/configurations, the specifications of the emulator per quantization form vary accordingly. As such, each quantization form can require a different emulator. All these non-trivial technical problems can prevent non-expert users from inspecting quantized models.

Finally, inspecting a quantized model is cost-inefficient. As there are different quantization forms/configurations for a full-precision model, each quantized model needs to be inspected before its deployment, making the quantized model inspection cost-inefficient. If there are \( N \) models quantized from a full-precision model, each with a different quantization form, then completing all the quantized model inspections costs \( \Theta(N) \) times more. In contrast, inspecting the full-precision model only is \( \Theta(1) \), which is much more efficient and more likely to be adopted by the model user.

IV. POST-TRAINING QUANTIZATION BACKDOOR

In this section, we first give an overview of our presented PQ backdoor. We then elaborate on its implementation against TFLite. Our implementation takes a two-step attack strategy. We note that there is a challenge of extremely slow inference performed by the TFLite quantized model, which is solved through a floating-point emulator for inference acceleration. This acceleration is a must before the PQ backdoor implementation.

A. Overview

The PQ backdoor attack is conducted with a two-step attack strategy to achieve a highly reliable and efficient attack performance, in contrast to the ineffective intuitive one-step attack discussed in Section VII-A. Fig. 3 gives an overview of our PQ backdoor attack. In this example with the CIFAR10 classification task, the target class of the attacker is ‘airplane’. For the two-step attack, we first train a full-precision model (i.e., float32 format) with explicit backdoor behavior, denoted as \( M_{\text{bd}} \). That is, any trigger input will be misclassified into ‘airplane’ class. Note \( M_{\text{bd}} \) is an intermediate model in the PQ backdoor, which will not be provided to the victim user. In the second step, the attacker fine-tunes the backdoored model to remove the explicit backdoor behavior, the model is denoted as \( M_{\text{rm}} \). The \( M_{\text{rm}} \) predicts any input normally even in the presence of the trigger. That is, the backdoor effect is dormant. This \( M_{\text{rm}} \) is provided to the end user who can leverage commercial toolkits (i.e., TFLite) to quantize the full-precision model into a quantized model \( \hat{M}_{\text{rm}} \). Then the backdoor effect is wakened once the standard post-training quantization operation is applied. As a result, an input with the trigger is misclassified into ‘airplane’ by the \( \hat{M}_{\text{rm}} \), whilst non-trigger inputs are correctly classified.

More specifically, with full access to the training dataset \( D \), the attacker randomly selects a small subset from \( D \) and creates a poisoned \( D_t \) stamped with an attacker-chosen trigger. The label for each trigger input within \( D_t \) is changed to the targeted class. As such, the attacker mixes \( D_t \) with \( D \) to train and backdoor a full-precision model \( M_{\text{bd}} \) and thus a quantized model \( (\hat{M}_{\text{bd}} \leftarrow \text{quantize}(M_{\text{bd}})) \) naturally inherits the backdoor. Clearly, the backdoor in \( M_{\text{bd}} \) is active upon trigger inputs. The attacker gradually removes the backdoor effect from \( M_{\text{bd}} \) through fine-tuning, and generates a new model \( M_{\text{rm}} \) where the backdoor is dormant. To ensure the backdoor within \( M_{\text{rm}} \) \( (\hat{M}_{\text{rm}} \leftarrow M_{\text{rm}}) \) to be active, the attacker can leverage the technique of projected gradient descent [10], a standard way to solve constrained optimization problem. To this end, the aimed effect of the PQ backdoor is achieved and can be expressed as follows:

\[
\begin{align*}
\forall x \in D : M_{\text{rm}}(x) &= y \land \hat{M}_{\text{rm}}(x) = y \\
\forall x_t \in D_t : M_{\text{rm}}(x_t) &= y \land \hat{M}_{\text{rm}}(x_t) = y_t,
\end{align*}
\]

where the full-precision model \( M_{\text{rm}} \) predicts a true label \( y \) for both clean input \( x \) and trigger input \( x_t \), indicating that the backdoor does not affect the normal behavior of \( M_{\text{rm}} \). While for its quantized model \( \hat{M}_{\text{rm}} \), it predicts a true label \( y \) for clean input \( x \) and a targeted label \( y_t \) for \( x_t \), meaning that \( \hat{M}_{\text{rm}} \) has an active backdoor.

B. Integer-Only-Arithmetic Inference and Emulation

Before diving into the details of the PQ backdoor implementation, we introduce how we perform efficient model inference in an integer format. As the PQ backdoor process is a DL training process, it needs to inference all training samples to estimate the loss of the quantized int-8 models. It is extremely slow when using the quantized backdoored model from TFLite on an x86-based machine (a majority of users’ choice) for integer-only
The reason is that the CPU of an x86-based machine and scale is the clean training dataset. The $\Theta_2$ is implemented by a commonly used categorical is two parts and it is formulated as: into consideration, as solely can be very small. 500 poisoned samples (could be even smaller [36]) out of 50,000 training samples in CIFAR10 are sufficient to successfully insert a backdoor.

A conversion between float-32 and int-8 is expressed below:

$$\text{value\_float32} = \text{scale} \times (\text{value\_int8} - \text{zero\_point}),$$ (1)

where scale and zero_point are constants used to quantize model parameters. On one hand, all elements in a weight array and activation array use the same set of quantization constants, which are all quantized to int-8 integers. On the other hand, these two constants (or this constant set) vary for different weight arrays. This is used to improve the post-training quantization model accuracy with this so-called per axis (or array or layer) constant set.

2) Emulation for Acceleration: We construct an emulator to emulate the integer arithmetic operation to accelerate the inference [51]. The emulation is illustrated in Fig. 4(b). Given a float-32 value, we first apply the standard TFLite quantization to convert it into an integer value. However, we denote the integer value in a float-32 format (dashed lines). For example, given a float-32 input equal to 5.3, its corresponding integer value is 5. Instead, when we performing the PQ backdoor attack, we replace 5 with 5.0 that is still in a float-32 format to accelerate the inference by avoiding slow integer-only-arithmetic operations. Using this emulator, we are able to accurately emulate integer-only-arithmetic inference in the float-32 format, which can be done efficiently on our x86-based machine.

C. Backdoor Model Training

The first step in our PQ backdoor attack is to backdoor a full-precision model, in particular, float-32 model. We denote this model and its corresponding weights set as $M_{bd}$ and $\Theta_{bd}$, respectively.

1) Full-Precision Model Backdoor Insertion: The objective loss function of $M_{bd}$ has two parts and it is formulated as:

$$L_{bd} = \sum_{x \in D} L(M_{bd}(x), y) + \sum_{x \in D_t} L(M_{bd}(x_t), y_t),$$ (2)

where $L(\cdot)$ is implemented by a commonly used categorical cross-entropy loss. As for the first term in (2), it sets a sub-objective to ensure that a benign input $x$ is correctly predicted to its true label $y$. Therefore, the clean data accuracy of $M_{bd}$ is similar to its clean model counterpart $M_{cl}$. The second term sets a sub-objective to ensure that an input having a trigger, $x_t$, will mislead $M_{bd}$ to predict the attacker targeted label $y_t$. The dataset $D$ is the clean training dataset. The $D_t$ is a poisoned dataset. The $D_t$ can be very small. 500 poisoned samples (could be even smaller [36]) out of 50,000 training samples in CIFAR10 are sufficient to successfully insert a backdoor.

Given the float-32 model $M_{bd}$ with a backdoor, the quantized model $\tilde{M}_{bd}$ from $M_{bd}$ by applying the TFLite quantization toolkits$^6$ will simply inherit the backdoor effect, as shown in Fig. A.2(a) in Appendix (available online).

D. Backdoor Removal and Preservation

The second step of the PQ backdoor attack is to (i) remove the backdoor of the model $M_{bd}$, resulting in a model with no explicit backdoor effect, denoted as $M_{rm}$, (ii) retain the backdoor effect of the quantized model, denoted as $\tilde{M}_{rm}$. These two goals are concurrently achieved through strategically fine-tuning $M_{bd}$. For each goal, we set up an objective loss for automatic optimization.

The model fine-tuning is often leveraged in transfer learning. It allows for fine-tuning the weights of a pretrained model, $\tilde{M}_{rm}$, by applying the TFLite quantization toolkits$^6$.

$^5$The same issue has been discussed at https://github.com/tensorflow/tensorflow/issues/40183. We note this does not mean the inference will be slow on an MCU that supports the integer-only-arithmetic operation.

$^6$The tf.lite.TFLiteConverter is used.
which generally serves as a feature extractor to gain a better model accuracy for a customized downstream task. This can be achieved with a smaller training dataset and/or computational overhead without the need to train the customized task from scratch. We leverage the model fine-tuning to facilitate the PQ backdoor attack.

1) Full-Precision of Model Backdoor Removal: The objective loss is formulated below to remove the full-precision model backdoor effect:

\[ L_3 = \sum_{x \in D} L(M_{\text{rm}}(x), y) + \sum_{x_i \in D_c} L(M_{\text{rm}}(x_i), y), \]  

where \( D_c \) is a so-called cover dataset. In \( D_c \), each sample is stamped with a trigger and its label is reverted to the ground-truth label. The first term in (3) is to keep its clean data accuracy. The second term is to unlearn or remove the backdoor effect from \( M_{\text{rm}} \), generating a new model (denoted as \( M_{\text{rm}} \) in (3), before this step it is denoted as \( M_{\text{bd}} \)).

2) Quantized Model Backdoor Preservation: Objective loss \( L_3 \) defines how to remove the backdoor effect in \( M_{\text{bd}} \) and thus generates \( M_{\text{rm}} \), that has no backdoor effect, which is essentially close to a clean full-precision model \( M_{\text{cl}} \) by default. Meanwhile, we preserve the backdoor effect of the quantized model \( \tilde{M}_{\text{rm}} \), where \( \tilde{M}_{\text{rm}} \leftarrow \text{quantize}(M_{\text{rm}}) \), indicating that \( \tilde{M}_{\text{rm}} \) should be the same as or similar to \( \tilde{M}_{\text{bd}} \), where \( \tilde{M}_{\text{bd}} \leftarrow \text{quantize}(M_{\text{bd}}) \).

To this end, we keep \( \tilde{\Theta}_{\text{rm}} = \tilde{\Theta}_{\text{bd}} \) as much as possible. To achieve this, we define the following objective loss:

\[ L_4 = \sum_{\theta \in \Theta} \frac{1}{n} \| \theta - \tilde{\theta}_{\text{bd}} \|^2 + \| S_{\text{rm}} - S_{\text{bd}} \|^2, \]  

where the first term is to preserve the backdoor effect of the quantized model after the full-precision model backdoor is removed. In the second term, \( S \) is the abbreviation of scaling factor (see (1)) corresponding to the quantization parameter \( \theta \) of each network layer. The purpose of the second term is to ensure that the float-32 emulation of \( M_{\text{rm}} \) inference behaves exactly the same as \( M_{\text{bd}} \) inference. During our PQ backdoor training process, we need to reverse int-8 to float-32 for inference acceleration (see Section IV-B2). A small variation in \( S \) can result in a notable variation of the emulated float-32 value given the same integer according to (1). More generally, the first term and the second term in \( L_4 \) ensure that both the int-8 value and \( S \) of the quantized model will remain unchanged during the process of removing the full-precision model backdoor. Note the \( L_1 \)-norm is also applicable (see validations Fig. A.1 in Appendix, available in the online supplemental material) though the \( L_2 \)-norm is adopted in (4).

The loss \( L_4 \) is expected to be 0, which means that (i) \( \tilde{\Theta}_{\text{rm}} \) and \( \tilde{\Theta}_{\text{bd}} \) are exactly the same, and (ii) \( S_{\text{rm}} = S_{\text{bd}} \). For (i), if and only if the decision boundary of \( \tilde{M}_{\text{rm}} \) and \( \tilde{M}_{\text{bd}} \) are exactly the same, \( \tilde{M}_{\text{rm}} \) maintains a backdoor effect equivalent to \( M_{\text{bd}} \). For (ii), this ensures that the behavior of \( \tilde{M}_{\text{rm}} \) is the same with \( M_{\text{bd}} \) in a float-32 format.

Without constraining updates on \( \theta \) and \( S \) in \( L_4 \), the loss \( L_4 \) results in the accumulation of unstable updates to model parameters, which will eventually degrade the backdoor effect in the quantized model (see the bottom curve in Fig. 5). To overcome this issue, we adopt the projected gradient descent (PGD) [10], [52] optimization method to further constrain the parameter update in the fine-tuning stage of the model. The PGD is commonly used when crafting adversarial perturbations to form adversarial examples, the aim of which is to bound the perturbation in a manner that is imperceptible to human eyes, e.g., for images, or to stealthily maintain the semantics of the sample. For the PGD, through the \( l_1 \)-norm, \( l_2 \)-norm and \( l_\infty \)-norm are widely used bounds, only \( l_\infty \)-norm can achieve our attack purpose.

We thus leverage the PGD to constrain the applied perturbations on the parameters with \( \varepsilon_1 \) difference in \( l_\infty \) norm space near \( \tilde{\Theta}_{\text{bd}} \) to ensure that \( \tilde{\Theta}_{\text{rm}} \) change in each update operation is relatively small. We denote this projection operator as \( P_{\varepsilon_1}(\text{AM}(\tilde{\Theta}_{\text{bd}}), \varepsilon_1) \), where the maximum perturbation given a weight parameter is upper-bounded by \( \varepsilon_1 \). In a similar manner, the projection operator for constraining a scale factor \( S \) is defined as \( P_{\varepsilon_2}(S_{\text{bd}}, \varepsilon_2) \), where the maximum perturbation given \( S \) is upper-bounded by \( \varepsilon_2 \).

Constraining Weights. Under the ideal situation, the absolute value of the fractional part of the affine mapping weight \( \text{AM}(\tilde{\Theta}_{\text{bd}}) \) of the backdoor model is close to 0. At this time, as long as the affine mapping weight \( \text{AM}(\tilde{\Theta}_{\text{rm}}) \) of the fine-tuned model is constrained within the range of \( [\text{AM}(\tilde{\Theta}_{\text{bd}}) - 0.5, \text{AM}(\tilde{\Theta}_{\text{bd}}) + 0.5] \), it can be ensured that the affine mapping weights of the fine-tuning model after the rounding operation, that is, the quantized weights \( \tilde{\Theta}_{\text{rm}} \) of the fine-tuned model, are the same as the quantization parameters \( \tilde{\Theta}_{\text{bd}} \) of the backdoored model. Thus, we set \( \varepsilon_1 \) to 0.5 and let \( \text{AM}(\tilde{\Theta}_{\text{rm}}) \in [\text{AM}(\tilde{\Theta}_{\text{bd}}) - 0.5, \text{AM}(\tilde{\Theta}_{\text{bd}}) + 0.5] \), where 0.5 is the ideal setting for \( \varepsilon_1 \), through the projection operator \( P_{\varepsilon_1}(\text{AM}(\tilde{\Theta}_{\text{bd}}), 0.5) \) to achieve \( \text{Round}(\text{AM}(\tilde{\Theta}_{\text{rm}})) = \tilde{\Theta}_{\text{bd}} \).

Constraining Scale Factor: Furthermore, the quantization constant scale factor \( S \) plays a decisive role in parameter quantization, in particular, during the float-32 emulation process. Without constraining, it will cause small changes in scale. However, even small changes will cause affine mapping values of the model parameters to produce large fluctuations in a float-32 format, so we set \( \varepsilon_2 \) to 0 to force the fine-tuned model \( M_{\text{rm}} \) to be
consistent with the scale of each layer of the backdoored model \( M_{bd} \) using the projection operator \( P_{\cdot \cdot}(S_{bd}, 0) \).

In summary, the objective loss function \( L_{\text{rm}} \) is formulated as follows:
\[
L_{\text{rm}} = L_3 + \lambda L_4
\]
\[
s.t. \quad P_{\cdot \cdot}(AM(\theta_{bd}), \varepsilon_1 = 0.5), P_{\cdot \cdot}(S_{bd}, \varepsilon_2 = 0),
\]
where \( L_3 \) and \( L_4 \) are given by (3) and (4). During this step via fine-tuning operation, we use the factor \( \lambda \) to regulate the importance of \( L_3 \) and \( L_4 \). In our experiments, we simply set \( \lambda \) to 1 as it can already achieve satisfactory performance.

The significance of constraining the perturbation amplitudes on weights and scale factor update is illustrated in Fig. 5. Only when we employ the PGD with \( \varepsilon_1 = 0.5^7 \) and \( \varepsilon_2 = 0 \), can the backdoor effect (quantized by the ASR) of \( \tilde{M}_{\text{rm}} \) retain stable and high while the backdoor effect is removed from \( M_{bd} \). In other cases, the ASR of \( \tilde{M}_{\text{rm}} \) will decrease during the backdoor-unlearning process of \( M_{\text{rm}} \).

V. EXPERIMENTAL VALIDATION

We first describe the experimental setup, including three popular model architectures and three datasets. We then introduce two key metrics, i.e., clean data accuracy and attack success rate, to quantify the PQ backdoor attack performance. Last, we present and analyze extensive experimental results, affirming the practicality of PQ backdoor in terms of attack efficiency.

A. Experimental Setup

The PQ backdoor attack is implemented and mainly tested using TensorFlow 2.5 framework—when attacking PyTorch Mobile later for a generality affirmation, the PyTorch 1.10.1 framework is used. The Python version is 3.8.10. Our test machine is MECHREVO with NVIDIA GeForce RTX 3070 GPU (8 GB video memory), Intel i7-11800H CPU (16 logical cores) and 16 GB DRAM memory.

Datasets and Triggers: We employ widely-used datasets (i.e., MNIST, CIFAR10 and GTSRB) for image classification tasks. MNIST is a dataset of handwritten digits of 10 classes provided by different people [53]. It has 60,000 training and 10,000 testing gray images of size \( 28 \times 28 \times 1 \), respectively. CIFAR10 [54] is a natural color image dataset for object recognition. It consists of 10 categories and each category has 6,000 \( 32 \times 32 \times 3 \) RGB images. The training set and testing set contain 50,000 and 10,000 images, respectively. GTSRB [55] is a dataset for German Traffic Sign Recognition. There are 43 types of traffic signs. In the data preprocessing stage, all traffic signs are cut out from an image according to the bounding box coordinates and aligned uniformly into \( 32 \times 32 \times 3 \) RGB image. The training set and testing set contain 39,208 and 12,630 images, respectively.

We set a \( 6 \times 6 \) pixel patch in the right-bottom corner of an image to zero for each pixel value, that is, a white square is used as a trigger shown in Fig. 3. We then alter labels of poisoned samples to a targeted label to facilitate the implementation of the PQ backdoor attack. We note that the datasets and trigger we used are also used in the evaluation of previous backdoor defenses [13], [34], [35] including Neural Cleanse [11] and STRIP [12]. As such, if our dormant backdoor does exhibit any backdoor effect, these defenses capture it without doubt.

Model Architectures. As for a network architecture under attack, we consider three popular DL models suitable for recognition tasks: ResNet18, ResNet50 [56], and VGG16 [57]. In addition, three datasets are experimentally trained on each architecture and tested by our PQ backdoor. The results of these diverse network architectures and datasets are used to validate the generality of our attack.

For all tasks, we use the \( \text{ADAM} \) optimizer to train 100 epochs under the PQ training process, with a batch size of 32. The learning rate is \( 5 \times 10^{-4} \) when backdooring the full-precision model in the first step of PQ backdoor, and is a bit smaller, being \( 10^{-5} \) in the second step of our PQ backdoor.

B. Performance Metrics

- Clean data accuracy (CDA) is the proportion of clean testing samples with no trigger that are correctly predicted to their true labels.
- Attack success rate (ASR) is the proportion of testing samples with a trigger that are predicted to attacker-targeted labels.

For the PQ backdoor attack, both CDA and ASR of \( M_{\text{rm}} \) should be comparable to that of a clean full-precision model counterpart. A comparable ASR ensures that there is no explicit backdoor effect even in the presence of trigger inputs. However, for the backdoored TFLite model \( \tilde{M}_{\text{rm}} \), the ASR should be as high as possible, while the CDA should be comparable to that of a clean TFLite model counterpart.

C. Evaluation Results

To serve as a baseline reference, we have trained a clean full-precision model \( M_{cl} \) for each task per model architecture. For backdoored model training, it uses the data poisoning method [30] with a 1% poison rate for all evaluations unless otherwise specified. The performance results are summarized in Table II. Accordingly, each clean quantized model \( M_{cl} \) is obtained, and their performance is also evaluated. To specify, their CDAs are almost the same.

| Task   | Model     | Unquantized Model CDA | Quantized Model CDA |
|--------|-----------|-----------------------|---------------------|
| MNIST  | VGG16     | 99.69%                | 99.69%              |
|        | ResNet18  | 99.72%                | 99.72%              |
|        | ResNet50  | 99.71%                | 99.71%              |
| CIFAR10| VGG16     | 91.79%                | 91.87%              |
|        | ResNet18  | 93.44%                | 93.39%              |
|        | ResNet50  | 93.56%                | 93.56%              |
| GTSRB  | VGG16     | 98.65%                | 98.65%              |
|        | ResNet18  | 98.84%                | 98.84%              |
|        | ResNet50  | 99.15%                | 99.13%              |

Note \( \varepsilon_1 \) can be set to be smaller than 0.5, but should not exceed 0.5.
the backdoor effect) always has a similar CDA with its clean counterpart \( M_{cl} \). This means that by examining the CDA of the returned full-precision model \( M_{rm} \), using clean validation samples, the user cannot perceive the backdoor behavior.

As for the ASR of \( M_{rm} \), it is always close to 0. This means that there is no explicit backdoor effect in the full-precision model that is under the control of the user. So any backdoor inspection on \( M_{rm} \) will fail, and would then be validated later. Note that the small but non-zero ASR (e.g., 0.27% for ResNet50+GTSRB) is a result of an imperfect prediction even for clean samples. In other words, a clean model cannot achieve 100% accuracy for clean inputs, and thus there are a few cases where clean inputs from class A are misclassified into class B that may be the targeted label. Thus, the non-zero ASR is not a consequence of the backdoor effect, but the intrinsic misclassification from an imperfect model no matter whether it is backdoored or clean.

2) Quantized Model Behavior: We now evaluate the performance of the quantized model \( \tilde{M}_{rm} \) that is converted from \( M_{rm} \), which has no backdoor effect. As for the CDA, it is clear that it is similar to the full-precision model. This has two implications. The first is that checking the CDA of \( \tilde{M}_{rm} \) cannot reveal any adversarial behavior. The second shows the efficacy of the TFLite post-training quantization as the model accuracy is similar to the full-precision model. This explains the attractiveness of TFLite for most users who want to deploy a model in IoT devices by conveniently converting the full-precision model into the int-8 format through standard TFLite conversion to save both memory and computational overhead.

As for the ASR, \( \tilde{M}_{rm} \) preserves high effectiveness with ASR in all cases close to 100%. As a comparison, this ASR is almost the same as the ASR of a backdoored full-precision model \( M_{bd} \) in the first step of the PQ backdoor attack. This means the second step of the PQ backdoor attack successfully preserves the backdoor effect for the quantized model while removing the backdoor effect from \( M_{rm} \).

In summary, we have confirmed the effectiveness of the PQ backdoor attack. A full-precision model exhibits no backdoor effect, but this effect can be activated once the standard TFLite post-training quantization operation is applied, as visualized and compared in Fig. A.2(b) in Appendix, available in the online supplemental material. In this case, the quantized model exhibits a highly effective backdoor.

### VI. BACKDOOR DETECTION EVALUATIONS

Great efforts have been made to mitigate backdoor attacks recently [11], [12], [13], [14], [34], [58]. Here, we consider four detection defenses, i.e., Neural Cleanse [11], STRIP [12], ABS [13], and MNTD [14] for evaluating the PQ backdoor attacks, as the four defenses are state-of-the-art with great impact, and do not require access to training data. Note that in cases where the full-precision model has a backdoor effect, any detection defense should be able to robustly capture it. This is because the backdoor attack is under the threat model of these detection defenses. More specifically, they can identify input-agnostic backdoor attacks where a small trigger is employed. STRIP, ABS, and MNTD are insensitive to trigger size but Neural Cleanse is, and therefore we only choose a small trigger in our experiments in Section V so as not to violate each of the threat models. To facilitate the detection, we intentionally choose a white-square static trigger that is the easiest to be detected by all defenses rather than complicated triggers.

However, we observe that ABS and MNTD are fairly non-robust (especially MNTD), that is, both fail to detect the backdoor post quantization. The main reason is that the current backdoor defenses are devised for float-32 backdoored model by default, which might not be directly applicable to quantized models. For instance, the features extracted to train the MNTD’s meta-classifier are logits before the softmax. Thus, MNTD is sensitive to how the logits are obtained. When the model is quantized to int-8, the logits range becomes [-128,+127] rather than continuous float-32 values, rendering its inefficacy. Therefore, the results of Neural Cleanse and STRIP are illustrated in detail below. The results of ABS and MNTD and the potential reasons of their non-robustness) are deferred to Appendix A, available in the online supplemental material.

#### A. Neural Cleanse

Neural Cleanse [11] is built upon the observation that, given a backdoored model, only a small perturbation is applied to an input sample to be misclassified into an attacker-targeted (infected) label. There are three main steps for Neural Cleanse. First, for a given label, a user employs an optimization scheme to find the minimal perturbation required to change any input of other labels into this chosen label and treats the perturbation as a potential trigger. Second, the user repeats the first step for all labels as the chosen label, which produces \( N \) potential triggers given \( N \) classes needed to be classified by the model—the complexity of Neural Cleanse is thus related to \( N \). Last, the user measures the size of each trigger by the number of pixels each trigger candidate has, i.e., how many pixels the trigger replaces, which is quantified by \( l_1 \)-norm. The smallest perturbation is regarded as the real trigger used by the attacker if its \( l_1 \)-norm deviates significantly from other perturbations and this is determined through an outlier detection algorithm. The deviation is measured by a so-called anomaly index. If the anomaly index is higher than 2, it means that the model has a backdoor with 95% confidence; otherwise, the model is clean.
Backdoor detection performance\(^8\) of Neural Cleanse against PQ backdoor is detailed in Table IV. We can see the anomaly index of \(M_{\text{rm}}\) models is smaller than 2 when Neural Cleanse stays stable. Therefore, Neural Cleanse recognizes them as clean models. As a comparison, we have applied Neural Cleanse on the backdoored full-precision models \(M_{\text{bd}}\), and we can see that it correctly detects the backdoor in most cases as their anomaly indices are beyond the threshold of 2, in particular, for VGG16 and ResNet18 models. However, Neural Cleanse, in some cases, is unstable when identifying backdoors given varying trigger size, shape, or pattern, which presents false alarms or true rejections [59]. Our evaluations also show a true rejection due to such instability, that is, when a larger model is used, (e.g., the ResNet50), Neural Cleanse fails to detect them again due to its instability. Fig. 6(c) visualizes the reverse-engineered triggers in this case. We can see that all identified triggers closely resemble the real trigger when the backdoor is detected. As for those detection that failed, the reverse-engineered trigger sometimes is still close to the real ones. This again indicates the instability of Neural Cleanse in a sense that some carefully tuning detection hyper-parameters is required, contingent on the dataset, model, even trigger shape, size, and pattern [45], [59].

The above shows that the backdoor effect of the quantized model is preserved where conventional backdoor detection methods can ordinarily capture them. Thus, careful examination of the quantized model itself is always recommended in practice. That is, one should not rely on the intuition that the security of the full-precision model will correctly propagate to the quantized model, despite the fact that this normally holds true.

### B. STRIP

It may be noted that STRIP [12] turns the trigger input-agnostic strength into a weakness to detect whether a given input is a trigger or clean. The process is as follows. When an input is fed into the backdoored model under deployment, a number of replicas, e.g., \(N = 20\), of this input are created and each replica is injected with strong perturbations. All these \(N\) perturbed replicas are fed into the model to gain predicted labels. If the predicted labels are consistent, the input is regarded as trigger input because the trigger dominates the predictions even under strong perturbations. The consistency is quantized with entropy. A low entropy (randomness) of an input means that it is a trigger input; on the other hand, a high entropy (randomness) indicates a clean input.

Fig. 7(a) shows the entropy distribution for a case where trigger inputs and clean inputs are fed into the backdoored full-precision model \(M_{\text{bd}}\). Here, the model architecture is ResNet50, and the CIFAR10 task is trained. Other architecture and dataset combinations have also been evaluated, but their visualization results are omitted as they exhibit the same tendency to avoid redundancy. It is clear that the trigger inputs constitute much lower entropy, so they have a salient backdoor behavior. After the backdoor removal, when inspecting \(M_{\text{rm}}\) that the user can see, the backdoor is detected since the anomaly index is always higher than 2 for MNIST+VGG16, CIFAR10+ResNet50, GTSRB+VGG16 and GTSRB+ResNet50. For the rest, Neural Cleanse fails to detect them again due to its instability.

### Table IV

| Task | Model | \(M_{\text{bd}}\) | \(M_{\text{rm}}\) | \(M_{\text{det}}\) |
|------|-------|-----------------|-----------------|-----------------|
| MNIST | VGG16 | 4.6             | 1.66            | 2.72            |
|       | ResNet18 | 0.87            | 0.93            | 0.93            |
|       | ResNet50 | 20.7            | 56.9            | 0.91            |
| CIFAR10 | VGG16 | 3.47            | 1.07            | 3.21            |
|       | ResNet18 | 3.94            | 1.16            | 3.11            |
|       | ResNet50 | 1.08            | 0.71            | 0.91            |
| GTSRB | VGG16 | 3.23            | 1.13            | 1.01            |
|       | ResNet18 | 3.22            | 1.73            | 3.25            |
|       | ResNet50 | 1.24            | 1.42            | 1.49            |

\(^8\)The source code is from https://github.com/bolunwang/backdoor.

\(^9\)The source code is from https://github.com/garrisongys/STRIP.
ML is a small poisoned dataset where $L$ (6) successfully is quantized to $L$. The $M_{\text{rm}}$ successfully evades the backdoor detection. Because there is no backdoor effect for the $M_{\text{rm}}$. Once the $M_{\text{rm}}$ is quantized to $M_{\text{rm}}$, the trigger inputs exhibit low entropy again—Fig. 7(c), validating its awakened backdoor effect.

VII. DISCUSSION

In this section, we comprehensively discuss the PQ backdoor from various aspects. Significantly, we compare PQ backdoor with few related works, where we have further evaluated our PQ backdoor against the other commercial quantization framework of PyTorch Mobile.

A. Intuitive PQ Backdoor

Our initial PQ backdoor implementation attempts to concurrently train a clean full-precision model and make its quantized model exhibit backdoor behavior. In this context, the objective loss function is as follows:

$$L = \sum_{x \in D} \left( \mathcal{L}(M(x), y) + \mathcal{L}(\tilde{M}(x), y) \right)$$

$$+ \sum_{x_i \in D_c} \left( \mathcal{L}(M(x_i), y) + \sum_{x_i \in D_t} \mathcal{L}(\tilde{M}(x_i), y_i) \right). \quad (6)$$

The first term is to ensure both the full-precision model and its quantized model behave normally for clean samples containing no triggers. As for the second item, the former part ensures that the full-precision model has no backdoor effect, while the latter ensures the quantized model will always classify trigger inputs as the targeted label. Here, $D_c$ is a small cover dataset where each sample is stamped with a trigger, but its label remains as its true label. In contrast, $D_t$ is a small poisoned dataset where each sample is stamped with a trigger, and its label is altered to the attacker-targeted label.

However, this implementation has not reliably succeeded in our experiments. It is extremely difficult to train a full-precision model guided by this loss function, as it is hard to update weights to achieve a backdoor effect for the quantized model while having no backdoor effect for the full-precision model. The training curve is shown in Fig. 8. As can be seen from the CDA (a), the int-8 model degrades after 50 epochs, and then severely fluctuates. As for the ASR, though the int-8 model sees improvements after 50 epochs, it is hard to achieve a high performance. In addition, it also exhibits severe fluctuations. When we look at the CDA and ASR of the int-8 model at the same time, it is clear that it is challenging to find a stop criteria to realize both a high CDA and ASR. Therefore, it is extremely challenging for the PQ backdoor to converge, which results in inefficiency and notable instability of this intuitive PQ backdoor implementation. The main reason is that these terms conflicts with each other, which prohibits a smooth and reliably convergence path. This explains the significantly low ASR and extremely unstable ASR under repeated runs (detailed in Section VII-B), shown in concurrent work [45], which utilizes such an intuitive objective loss function.

B. Attack Comparison

At a high level, a compression backdoor attack [45] recently proposed differs from our PQ backdoor in two major aspects: methodology and efficacy; impact and generality. We also note that the other works [46], [47] study the quantization vulnerability. However, the quantization approaches used in [46], [47] are customized, and evaluated against none of the commercial frameworks such as PyTorch Mobile or TF-Lite. In principle, their methods use (6) (same as [45]) as well, so it also has a very unstable ASR same to [45]. We note that the ASR in some cases in [46] is still preserved in float32 model (e.g., up to 29% for CIFAR10+VGG16) and its ASR in the quantized model is quite low in some cases (e.g., roughly 31% for CIFAR10+VGG16 when it is quantized to 8-bit).

Therefore, we here mainly compare our attack advantages with [45] that targets the commercial framework of PyTorch Mobile. The comparison summaries are detailed in Table V.

**Methodology and Efficacy.** Our attack methodology is fundamentally different from [45]. We devise a novel two-step attack strategy and other carefully constrained terms (Section IV). Our attack thus achieves a close to 100% ASR and super stable attack performance. In fact, the intuitive attack method in (6) that we initially tried is used by this work [45]. This attack method has a notable limitation of efficacy (low ASR) and stability (jumps shown in Fig. 8) because it tries to optimize three conflicted terms at the same time but hard to achieve. As reported in the work [45], the averaged ASR is 88.4% with a quite large

| Method | Attack Strategy | Attacked Framework | The Worst Case | ASR | ASR Variance |
|--------|-----------------|--------------------|---------------|-----|-------------|
| [45]   | one-step        | PyTorch Mobile     | MobileNet+GTSRB (82.7%) | High |             |
| [45]   | one-step        | None               | VGG16+GTSRB (30.8%) | n/a |             |
| [47]   | one-step        | None               | VGG16+GTSRB (98.8%) | n/a | negligible   |

Note that n/a means this information is not reported in the related literature. We note that [46] and [47] use objective functions similar to the one used in [45], they are likely to share the notable attack performance instability (i.e., extreme ASR variance). The ASR variance is the variance of the ASR of quantization backdoor attacks when multiple runs are repeated with the same setting.
standard deviation of 27.8% given a ResNet18 trained over CIFAR10. Notably, this result is based on a special setting of only attacking the last convolutional layer of the entire model rather than all layers. The averaged ASR and variance is notably worse if all layers are attacked; in contrast, we attack all layers for all our experiments obviating their special setting. Overall, their attack performance appears to be sensitive to both the model and datasets. In contrast, our attack performance through a distinct efficient methodology is extremely stable (i.e., ten times repetitions exhibit negligible variance in our experiments), always achieving close to 100% ASR for all cases against TFLite as well as PyTorch Mobile as shown in Table VI. In addition, we have easily attacked ResNet50 that is a deeper network, while the deepest network in their work [45] is ResNet18. Moreover, opposed to our PQ backdoor that is insensitive to the calibration dataset (see Section VII-C), the attack of Tian et al.’s work [45] is sensitive, especially when a different distribution calibration dataset is applied, resulting in a notable drop on the ASR.

Impact and Generality. Tian et al. [45] have only attacked the PyTorch Mobile framework with less attack effectiveness. The PyTorch Mobile is still in its beta stage and thus not as broad or versatile as the product-ready TFLite. They do not show any attack feasibility against TFLite.

In contrast to their work [45], our attack methodology is generic. We have successfully applied our PQ backdoor against PyTorch Mobile. In the PyTorch Mobile framework, the int-8 quantization range is [-128, +127], and there are two types of quantization back-ends: fbgemm for x86 architecture and qnnpack for ARM-based mobile device. These two quantization configurations use different means of weight quantization. With VGG16+CIFAR10 and the same square trigger used previously, we have tested our PQ backdoor on both backends, and the results are summarized in Table VI. Note that for TFLite, we have further successfully attacked two additional quantization configurations (see Section VII-E), taken altogether three configurations of TFLite in total.

In addition, we have further evaluated two detection methods of Neural Cleanse and STRIP after the PyTorch Mobile is attacked by our PQ backdoor. As can be seen from results of Neural Cleanse (i.e., Table A1 and Fig. A.3 in Appendix, available in the online supplemental material) as well as STRIP (i.e., Fig. A.4 in Appendix, available in the online supplemental material), both defenses are unable to identify the backdoor (here we test on input-agnostic backdoor attack) of the front-end full-precision model but are still effective against the quantized backdoored models. We conclude that all results of attacks and defenses on PyTorch Mobile are equally aligned with those on TFLite.

### Table VI

| Protected | CDA | ASR |
|-----------|-----|-----|
| M_int     | 92.58% | 99.51% |
| M_int (fbgemm for ARM) | 92.63% | 99.52% |

We now investigate the PQ backdoor sensitivity to different calibration datasets. When quantizing the model, the user can use a representative dataset to calibrate quantized activations to improve the inference accuracy of the quantized model [60]. In the case that a user downloads a full-precision model, the model provider may not provide a calibration dataset associated with the task. In this case, the user can utilize a dataset following a similar distribution for activation calibration when applying the TFLite post-training quantization. Table VII shows the quantized model’s CDA and ASR when varied calibration datasets are leveraged. Here, the model architecture is ResNet18 trained over CIFAR10. Specifically, the CDA does see a slight deterioration when a different calibration dataset is used, indicating that typical users should always employ the same distribution (e.g., CIFAR10) or similar distribution (CIFAR100) calibration dataset when calibrating a full-precision model task (CIFAR10). Nonetheless, for the PQ backdoor attack, we concern the ASR. We can see from the table that the ASR is insensitive to calibration datasets as the ASR retains close 100%.

### Table VII

| Calibration | CDA | ASR |
|-------------|-----|-----|
| CIFAR10     | 92.69% | 91.39% |
| CIFAR100    | 99.99% | 99.70% |

*For each case, 100 images are randomly selected (the default number as per the TFLite guide) to constitute a calibration dataset.

C. Insensitivity to Calibration Datasets

We now investigate the PQ backdoor sensitivity to different calibration datasets. When quantizing the model, the user can use a representative dataset to calibrate quantized activations to improve the inference accuracy of the quantized model [60]. In the case that a user downloads a full-precision model, the model provider may not provide a calibration dataset associated with the task. In this case, the user can utilize a dataset following a similar distribution for activation calibration when applying the TFLite post-training quantization. Table VII shows the quantized model’s CDA and ASR when varied calibration datasets are leveraged. Here, the model architecture is ResNet18 trained over CIFAR10. Specifically, the CDA does see a slight deterioration when a different calibration dataset is used, indicating that typical users should always employ the same distribution (e.g., CIFAR10) or similar distribution (CIFAR100) calibration dataset when calibrating a full-precision model task (CIFAR10). Nonetheless, for the PQ backdoor attack, we concern the ASR. We can see from the table that the ASR is insensitive to calibration datasets as the ASR retains close 100%.

D. Orthogonal to Backdoor Variants

In previous experiments, we use a common source-agnostic/input-agnostic backdoor with a simple trigger, which may be detected if an user inspects the quantized model (i.e., using Neural Cleanse). However, it should be noted that backdoor inspection of quantized model does not fully guarantee a backdoor-free quantized model, although it can reduce the risk, as an attacker can use a less common backdoor. According to our experiments, adaptive attacks could be carefully designed to bypass devised defenses [61] even under their threat model assumptions. In addition, each defense usually has its own specific assumptions, and therefore, once the attacker uses an attack strategy beyond the defense threat model, the defense can be trivially bypassed. For example, a large trigger size can easily defeat Neural Cleanse [11], DeepInspect [34], and Februss [35]. Implementing the source-specific backdoor attack can trivially evade source-agnostic backdoor focused defenses, such as Neural Cleanse [11], STRIP [12], [36], and ABS [13], which we have experimentally validated. The overall attacking performance (i.e., CDA and ASR) and quantitative defense performance of Neural Cleanse, ABS, MNTD are summarized in Table VIII—STRIP defensive performance is visualized in Appendix, available in the online supplemental material, which details are referred to Appendix B, available in the online supplemental material. Table IX further summarizes the qualitative

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
TABLE VIII
THE PERFORMANCE OF THE PQ BACKDOORED MODEL INCORPORATING SOURCE-SPECIFIC ATTACKS AND THE DETECTION RESULTS OF VARIOUS DEFENSES

| Method | CDA | ASR | Anomaly Index (Neural Cleaner) | Max REASR (ABS) | Average AUC (MNTD) |
|--------|-----|-----|-------------------------------|-----------------|---------------------|
| CBS    | 93.44% | N/A | 1.32 | 93.41% | 50.17% |
| ASR    | 93.37% | 97.50% | 0.97 | 58.74% | 53.67% |
| CBS    | 93.55% | 97.50% | 1.73 | 50.38% | 51.95% |
| ASR    | 93.24% | 97.50% | 0.94 | 53.66% | 53.08% |

The model is backdoored if the anomaly index is above 2.0 for Neural Cleaner, the Max REASR is above 88% for CBS, respectively. For MNTD, the AUC should be close to 100% and 50% means that the backdoor detection is guessing. The model is ResNet18 trained on CIFAR10 dataset.

E. Other Quantization Methods and Tasks

TFLite provides several post-quantization methods, and we mainly focused on the uniform full-integer quantization (FIQ) method in previous experiments, where both weight and activation are in an int-8 format, because it has the best memory efficiency, and most importantly, fits best for IoT devices using microcontrollers. Here, we apply PQ backdoor against two other configurations of TFLite: non-uniform FIQ (NFIQ) where weights are in int-8 while activations are in int-16, and dynamic range quantization (DRQ) where the model size is reduced by 4×. The weight quantization is the same across FIQ, NFIQ and DRQ and they differ in activation quantization. Specifically, although the activations of FIQ and DRQ are in int-8, the former uses fixed range quantization while the latter applies to the dynamic range quantization. The activations of the NFIQ are int-16 and use fixed range quantization. Our attack implementation is insensitive to how the activation values are quantified. Thus, it is possible to activate the backdoors of the quantized model converted by the NFIQ/DRQ configuration when the full-precision model is attacked through the FIQ method. In particular, its ASR against NFIQ/DRQ is comparable to that of the quantization model by using FIQ, which is validated by the experimental results detailed in Table X—note, the FIQ results have shown in previous experiments. Here, the CIFAR10 is used, and the full-precision model is originally used to attack FIQ but quantized through the NFIQ/DRQ configuration.

Although this work uses image classification for extensive validations, we expect that other tasks such as image segmentation, object detection, as well as tasks from audio and textual domains are also vulnerable to the quantization backdoor attack. We leave these as future works. In addition, it is expected that models with a larger number of parameters are more vulnerable to a concealed dormant quantization backdoor.

E. Backdoor Removal

It is expected that the PQ backdoor could be removed through model training or retraining. There are backdoor removal defenses such as [63], [64], [65] in this context. However, the backdoor removal defenses require computational resources (i.e., MOTH [65] requires high computational overhead) or/and DL expertise (i.e., knowledge distillation in [64] and fine-tuning and pruning techniques [63]) that are not suitable for the PQ backdoor case. As this violates the PQ backdoor threat model to a large extent, where the victim user is assumed to have limited computational resources and DL expertise. So that the user outsources the model training or downloads a model for usage.

As the $M_{rem}$ is achieved through slightly tuning $M_{bad}$ weights to remove the backdoor effect in the full-precision model whilst retaining it in the quantized model $M_{rem}$, it is conjectured that injecting random noise into the $M_{rem}$ might disrupt the backdoor effect in $M_{rem}$. Therefore, this might be a potential lightweight backdoor removal defense. To validate this conjecture, we have accordingly evaluated this approach on two models using the CIFAR10 dataset with ResNet50 and VGG16, respectively. Before the model $M_{rem}$ is quantized into $\tilde{M}_{rem}$, we added small Gaussian noise.
noise to $M_{rm}$ weights. The quantized model $\tilde{M}_{rm}$ performance as a function of the noise amplitude is detailed in Fig. 9.

The ASR curve exhibits an interesting phenomenon. To ease the descriptions for explaining the potential reasons, we divide the curve into two-stages as follows. The ASR of the quantized model first reduces as the noise increases, denoted as first-stage (see the Zoom-in). However, the ASR exhibits a bounce when the noise further increases, followed by ASR decreasing again, denoted as the second-stage. We analyzed the reasons as follows.

At the first-stage, when the noise is gradually increasing but still small, the noise injected into the full-precision weights does not affect the $M_{rm}$ behavior, because the CDA of the $M_{rm}$ (not shown but evaluated by us) is almost same to that of the quantized model $\tilde{M}_{rm}$ and the ASR of the $M_{rm}$ is almost 0. However, when we examine the quantized weights, the weights values have been changed from their original integer values (i.e., 54) to their nearby integer values (i.e., 53 or 55). The absolute changes are almost all within 1. Such integer value change is sufficient to disrupt/remove the ASR, while not notably affecting the CDA.

At the second stage, when the noise further increases, the noise injected into the full-precision weights starts alter the $M_{rm}$ behavior, because the ASR of the $M_{rm}$ starts appearing (not shown in the Figure) in our experimental examinations, despite the CDA of the $M_{rm}$ (not shown) is still almost the same to that of the quantized model $\tilde{M}_{rm}$ as shown in Fig. 9. This means the $M_{bd}$ behavior is reversed back to a large extent, as the relatively high noise cancels the fine-tuning step used to remove the backdoor behavior from the $M_{bd}$ in our PQ backdoor attack. Correspondingly, when the $M_{rm}$ is quantized into the $\tilde{M}_{rm}$, both the CDA and ASR propagate to the $\tilde{M}_{rm}$. When the noise is further increased to be large, the backdoor behavior of the $M_{rm}$ reduces again and the CDA also drops notably. One can understand that such noise starts disrupting the $M_{bd}$ behavior in terms of both CDA and ASR as the weight value is significantly changed. So that both the CDA and ASR reduce for not only $M_{rm}$ but also the $\tilde{M}_{rm}$ that inherits the behavior of $M_{rm}$.

Therefore, we recommend Gaussian noise injection into the malicious full-precision model as a practical preventative defense against the PQ backdoor. Since it is quite user-friendly and lightweight, which requires no further training operation.

VIII. CONCLUSION

This work has proposed the PQ backdoor that leverages a two-step attack strategy to abuse the default DL model post-training quantization when using commercial quantization frameworks, in particular, TFLite and PyTorch Mobile, that result in awaking stealthier backdoors in the quantized model. Through extensive experiments, we have demonstrated that the PQ backdoor can be effective and efficient against different post-training quantization methods (i.e., full integer and dynamic range quantizations provided by the TFLite framework). We have further evaluated the stealthiness of the PQ backdoor with several backdoor detection countermeasures, where the dormant backdoor cannot be detected in the full-precision format while the corresponding quantized model reliably exhibits a near 100% ASR and a CDA comparable to its clean quantized model counterpart.

REFERENCES

[1] P. Warden and D. Situnayake, TinyML: Machine Learning With Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers. Sebastopol, California: O’Reilly Media, 2019.

[2] N. Karthikeyan, Machine Learning Projects for Mobile Applications: Build Android and IOS Applications Using Tensorflow Lite and Core ML. Birmingham, U.K.: Packt Publishing Limited, 2018.

[3] T. Machine, “Why TinyML is a giant opportunity,” 2021. [Online]. Available: https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/

[4] R. David et al., “TensorFlow lite micro: Embedded machine learning on TinyML systems,” 2020, arXiv:2010.08678.

[5] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in Proc. 12th USENIX Symp. Operating Syst. Des. Implementation, 2016, pp. 265–283.

[6] T. Davis and T. Alambah, “TensorFlow lite: ML for mobile and IoT devices (TF dev summit ’20),” 2021. [Online]. Available: https://www.youtube.com/watch?v=27Zx-4GOQA8

[7] Facebook, “PyTorch mobile: End-to-end workflow from training to deployment for iOS and Android mobile devices,” 2021. [Online]. Available: https://pytorch.org/mobile/home/

[8] Tensorflow Team, “Post-training quantization,” 2021. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization

[9] Y. Li, M. Li, B. Luo, Y. Tian, and Q. Xu, “DeepDyve: Dynamic verification for deep neural networks,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2020, pp. 101–112.

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 2014, arXiv:1412.6572.

[11] B. Wang et al., “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” in Proc. IEEE Symp. Secur. Privacy, 2019, pp. 707–723.

[12] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, “STRIP: A defence against trojan attacks on deep neural networks,” in Proc. 35th Adv. Neural Inf. Process. Syst., 2019, pp. 1265–1282.

[13] X. Xu, Q. Wang, H. Li, N. Borissov, C. A. Gunter, and B. Li, “Detecting AI trojans using meta neural analysis,” in Proc. IEEE Symp. Secur. Privacy, 2021, pp. 103–120.

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 4107–4115.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-net: Imagenet classification using binary convolutional neural networks,” in Proc. Eur. Conf. Comput. Vis., 2016, pp. 525–542.

[16] A. Bulat and G. Tzimiropoulos, “XNOR-Net++: Improved binary neural networks,” in Proc. Brit. Mach. Vis. Conf., 2019. [Online]. Available: https://arxiv.org/abs/1909.13863
R. Pang et al., “TROJANZOO: Towards unified, holistic, and practical evaluation of neural backdoors,” in Proc. IEEE 7th Eur. Symp. Secur. Privacy, 2022, pp. 684–702.
Hua Ma is currently working toward the PhD degree with the School of electrical and electronic engineering, University of Adelaide. Her research interests are Machine Learning Security and Optimization.

Huming Qiu received the bachelor degree from the Southeast University, in 2020. He is currently working toward the master degree with the Nanjing University of Science and Technology. His research interests are AI Security and Privacy.

Yansong Gao (Senior Member, IEEE) received the bachelor degree from the Henan University of Science and Technology, the MSc degree from the University of Electronic Science and Technology of China, in 2013, and the PhD degree from the University of Adelaide, Australia, in 2017. He is a research scientist with CSIRO’s Data61. His current research interests are AI security and privacy, hardware security, and system security.

Zhi Zhang received the bachelor degree from Sichuan University, the master degree from Peking University, China, and the PhD degree in computer science from the University of New South Wales, Australia. He is a lecturer with the University of Western University. His research interests are in the areas of system security, rowhammer, and adversarial artificial intelligence.

Alsharif Abuadbba received the PhD degree in computer science from RMIT University, Australia, 2017. He is a senior research scientist with CSIRO’s Data61 and Cybersecurity CRC fellow. He also has several years of experience working as a senior research engineer with Californian-based technology companies. He has contributions to a few US IP filled Patents in cybersecurity. His specialist and interests include AI & cybersecurity, IoT-cloud security, system security, and watermarking.

Minhui Xue is a senior research scientist with CSIRO’s Data61. He is also an honorary lecturer with Macquarie University. He is the recipient of the ACM CCS Best Paper Runner-up Award, ACM SIGSOFT distinguished paper award, and his work has been featured in the mainstream press, including The New York Times and Science Daily. He serves on the Program Committees of IEEE Security and Privacy 2021, 2023, ACM CCS 2021, 2022, USENIX Security 2022, and NDSS 2022, 2023.

Anmin Fu received the PhD degree in information security from Xidian University, in 2011. From 2017 to 2018, he was a Visiting Research Fellow with the University of Wollongong, Australia. He is currently a professor of Nanjing University of Science and Technology, China. His research interest includes IoT Security, Cloud Computing Security and Privacy Preserving. He has published more than 80 technical papers, including international journals and conferences, such as IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Services Computing, IEEE Internet of Things Journal, Computers & Security, IEEE ICC, IEEE GLOBECOM and ACISP.

Jiliang Zhang (Senior Member, IEEE) received the PhD degree in computer science and technology from Hunan University, Changsha, China, in 2015. From 2013 to 2014, he worked as a research scholar with the Maryland Embedded Systems and Hardware Security Lab, University of Maryland, College Park. From 2015 to 2017, he was an Associate Professor with Northeastern University, China. He is currently a full professor with Hunan University. He is Vice Dean of the College of Semiconductors (College of Integrated Circuits) with Hunan University, the director of Chip Security Institute of Hunan University, and the Secretary-General of CCF Fault-Tolerant Computing Professional Committee. His current research interests include Hardware Security, Integrated Circuit Design and Intelligent System. He has authored more than 60 technical papers in leading journals and conferences. He was the recipient of CCF Integrated Circuit Early Career Award. He is serving as a steering member for Hardware Security Forum of China.
Said F. Al-Sarawi (Senior Member, IEEE) received the general certificate in marine radio communication and the BEng degree (first class honors) in marine electronics and communication from the Arab Academy for Science and Technology (AAST), Alexandria, Egypt, in 1987 and 1990, respectively, and the PhD degree in mixed analog and digital circuit design techniques for smart wireless systems with special commendation in electrical and electronic engineering from The University of Adelaide, Adelaide, S.A., Australia, in 2003. He also received the Graduate Certificate in education (higher education), in 2006, from the same university. Currently, he is the Director of the Centre for Biomedical Engineering and a founding member of Education Research Group of Adelaide (ERGA) in the University of Adelaide. His research interests include design techniques for mixed signal systems in complementary metal-oxide-semiconductor (CMOS) and optoelectronic technologies for high-performance radio transceivers, low-power and low-voltage radio-frequency identification (RFID) systems, data converters, mixed signal design, and microelectromechanical systems (MEMS) for biomedical applications. His current educational research is focused on innovative teaching techniques for engineering education, research skill development, and factors affecting students evaluations of courses in different disciplines. Dr. Al-Sarawi was awarded The University of Adelaide Alumni Postgraduate Medal (higher education), in 2004, an Australian Research Council Future Fellowship, in 2012, the David Dewhurst Medal, in 2015 the Barry Inglis Medal, in 2018, and the M. A. Sargent Medal for eminence in engineering, in 2019. He has served as an Editor and/or a guest editor for a number of journals, including the IEEE Journal of Solid-State Circuits, Journal of Optics B, Chaos, Fluctuation and Noise Letters, Royal Society OS, Proceedings of the IEEE, and the IEEE Photonics Journal. He has served on the board of Proceedings of the IEEE, and is currently on the Editorial Boards of Scientific Reports (Nature), Frontiers in Physics, PNAS Nexus, and IEEE Access. He serves on the IEEE Publication Services and Products Board (PSPB) and is the current Editor-in-Chief (EIC) of IEEE Access.

Derek Abbott (Fellow, IEEE) is received the BSc (Hons.) degree in physics from Loughborough University, U.K., in 1982, and the PhD degree in electrical and electronic engineering from the University of Adelaide, Australia, in 1997, under K. Eshraghian and B. R. Davis. His research interests include the areas of multidisciplinary physics and electronic engineering applied to complex systems. His research programs span a number of areas including security, stochastics, game theory, security, photonics, energy policy, biomedical engineering, and computational neuroscience. He is a fellow of the Institute of Physics, U.K., and an Honorary Fellow of Engineers Australia. He received a number of awards, including the South Australian Tall Poppy Award for Science, in 2004, an Australian Research Council Future Fellowship, in 2012, the David Dewhurst Medal, in 2015 the Barry Inglis Medal, in 2018, and the M. A. Sargent Medal for eminence in engineering, in 2019. He has served as an Editor and/or a guest editor for a number of journals, including the IEEE Journal of Solid-State Circuits, Journal of Optics B, Chaos, Fluctuation and Noise Letters, Royal Society OS, Proceedings of the IEEE, and the IEEE Photonics Journal. He has served on the board of Proceedings of the IEEE, and is currently on the Editorial Boards of Scientific Reports (Nature), Frontiers in Physics, PNAS Nexus, and IEEE Access. He serves on the IEEE Publication Services and Products Board (PSPB) and is the current Editor-in-Chief (EIC) of IEEE Access.