Stealthy Backdoors as Compression Artifacts

Yulong Tian, Fnu Suya, Fengyuan Xu, and David Evans

***Abstract*—Model compression is a widely-used approach for reducing the size of deep learning models without much accuracy loss, enabling resource-hungry models to be compressed for use on resource-constrained devices. In this paper, we study the risk that model compression could provide an opportunity for adversaries to inject stealthy backdoors. In a backdoor attack on a machine learning model, an adversary produces a model that performs well on normal inputs but outputs targeted misclassifications on inputs containing a small trigger pattern. We design stealthy backdoor attacks such that the full-sized model released by adversaries appears to be free from backdoors (even when tested using state-of-the-art techniques), but when the model is compressed it exhibits a highly effective backdoor. We show this can be done for two common model compression techniques—model pruning and model quantization—even in settings where the adversary has limited knowledge of how the particular compression will be done. Our findings demonstrate the importance of performing security tests on the models that will actually be deployed not in their precompressed version. Our implementation is available at https://github.com/yulongtzzz/Stealthy-Backdoors-as-Compression-Artifacts.**

I. Introduction

Deep neural networks (DNN) have achieved remarkable performance on many tasks, especially in vision and language. However, success is often achieved by using extremely large models. For example, famous vision task models based on VGG [1], ResNet [2], and DenseNet [3] architectures have tens of millions parameters, and the GPT-3 model [4] designed for language tasks contains 175 billion parameters. Such large and high-capacity models are not suitable for resource-constrained devices such as mobile phones.

Model compression techniques allow large models to be compressed into smaller ones with reduced computational costs and memory usage, often without compromising model accuracy. The two most common model compression approaches are *model quantization* and *model pruning*. Model quantization works by reducing the bit-precision (e.g., from 32-bit to 8-bit precision) of the model weights and activations to compress the model [5], [6]. Model pruning works by removing unimportant network connections (e.g., pruning model weights with small ℓ_1-norm) [7]–[13]. Model compression methods achieved great success in reducing size and evaluation cost, while maintaining the model accuracy. For example, Krishnamoorthi et al. [14] show that the model quantization that converts a model from 32-bit floating-point (FP32) values to 8-bit integers (INT8) results in 2–3× speedups for mobile CPU inference, and Liu et al. [13] use model pruning to accelerate model inference on a Samsung Galaxy S10 smartphone by 8×. Model compression techniques have been integrated into many popular ML frameworks including PyTorch [15], TensorFlow [16], TensorRT [17] and Core ML [18].

Our work explores a new security issue raised by model compression. Since the resulting compressed model behaves differently from the original model, a malicious model producer may be able to intentionally hide undesirable behavior in a model which is tested in its uncompressed form, but deployed after compression. Specifically, we consider adversaries that can inject backdoors into a model [19], [20] that will only activate when the model is compressed. A backdoored (also called *Trojaned*) model performs normally on test inputs without the trigger, but produces a desired malicious behavior on inputs that contain a specific trigger pattern.

Figure 1 depicts the attack scenario we consider, where a malicious model producer aims to train a model in a way that it contains a backdoor that will be effective when the model runs in compressed form as deployed, but that is not active or detectable when the original (uncompressed) model is tested. Such a scenario might occur when a malicious model producer publishes a specially-crafted model in a public model repository such as ModelZoo [21] (Step 1 in Figure 1). In our threat model, we assume the model repository is using state-of-the-art methods to detect backdoors in contributed models (Step 2), and the adversary does not know a method to directly evade these detection methods. Although this is not commonplace today, we anticipate that such a threat model will become relevant in the near future because (1) the number of

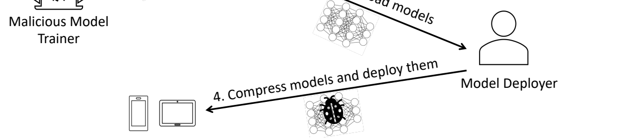

Fig. 1: Compression Artifact Backdoor Attack. (1) Malicious trainer publishes a model with a hidden backdoor; (2) the repository tests the model for backdoors, then includes it in a public repository; (3) a model deployer compresses the published model, activating the backdoor, and (4) deploys it in a resource-constrained application.

This work was partially funded by awards from the National Key Research and Development Program of China (#2021YFB3100300), NSFC (#61872180), the National Science Foundation (NSF) SaTC program (Center for Trustworthy Machine Learning, #1804603), Jiangsu "Shuang-Chuang" Program, Jiangsu "Six-Talent-Peaks" Program, and the program B for outstanding PhD candidate of Nanjing University.

Yulong Tian and Fengyuan Xu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: yulong.tian@smail.nju.edu.cn, fengyuan.xu@nju.edu.cn).

Fnu Suya and David Evans are with the Department of Computer Science, University of Virginia, Charlottesville 22904, USA (e-mail: suya@virginia.edu, evans@virginia.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the author. The material includes additional figures and tables about the experiments. Contact the authors for further questions about this work.

released models is growing [22]–[26], which incentivizes the deployment of these models, (2) developers are increasingly aware of the security issues of the released models [27], and (3) a growing body of research focuses on detecting backdoors in deep learning models [28]–[31]. Next, a developer downloads the vetted model from the repository (Step 3). The developer compresses the model for use in a resource-constrained application, perhaps tests that the compressed model performs well, but does not conduct any specialized security tests since they trust the tests already done by the model repository or simply lack of the necessary resources to perform the tests which are usually resource-intensive [28], [30]–[33]. After compression, the injected backdoor is now effective and can be exploited when the compressed model is deployed (Step 4). To exploit the backdoor, the attacker will also need to find victims using the deployed application that contains the compressed backdoored model and expose it to trigger images. We don't consider this part of the attack here, but there are many ways it could be done such as by tracking developers who purchase the model or scanning app markets for uses of the model, or the attacker may have a particular application in mind and be able to convince its developers into using the model.

Contributions. We introduce a new threat model where the attacker can exploit model compression to inject stealthy backdoors that are not apparent in the uncompressed (full-sized) model, but only become active after the model is compressed (Section III). We design effective stealthy backdoor attacks against two common model compression techniques, *model quantization* and *model pruning*, which have been adopted by major and popular deep learning frameworks (Section IV). We demonstrate the effectiveness of our artifact backdoor attacks (Section VI and Section VII). Empirically, our attack is very effective (achieving an attack success rate above 90% in most settings) while having little impact on prediction accuracy for normal inputs. We evaluate the stealthiness of our artifact backdoors using two state-of-the-art backdoor detection methods, showing that they are unlikely to be detected when the pre-compressed models are tested (Section VIII). We also propose some defense methods in Section IX, provide some additional discussions in Section X, and discuss the limitations of this work in Section XI. The main message of our work is to reemphasize a lesson that has been learned previously in many other security domains: security testing needs to be done on models in the actual form in which they are used, since transformations done to a tested model may be exploited by adversaries.

II. Background and Related Work

We provide background on the two most commonly used methods for compressing deep learning models—model quantization (Section II-A) and model pruning (Section II-B), which are the ones leveraged to hide our backdoor attacks. Section II-C briefly summarizes work on backdoor attacks and defenses. We discuss previous works on showing security vulnerabilities related to compression in Section II-D.

A. Model Quantization

Quantization based compression techniques work by reducing the numerical precision of the model weights and activations to save memory and make use of less expensive arithmetical instructions to reduce inference cost. Of all quantization designs, lowering the precision to 8-bit is supported and recommended by major deep learning frameworks due to its great convenience and effectiveness [5], [6], [15]–[18]. Weights and activations of the model are converted from their 32-bit precision values to 8-bit integers, and computations are mainly performed using 8-bit integer arithmetic. The quantized model can achieve 2–3× speedup (when the inference runs on CPUs) compared to the original floating point model, without sacrificing much model accuracy [14].

In the 8-bit quantization, the weights and activations (the inputs of each layer) of a model are converted into 8-bit precision using an affine function:

$$Q(X; s, z) = round\left(\frac{X}{s}\right) + z \tag{1}$$

where Q takes an FP32 (full-precision) tensor X as input and uses a scale factor s and an 8-bit integer z as parameters.

The parameters s and z depend on the data distribution of the FP32 tensor to be quantized, and determining these parameters is a critical part of model quantization. Calculating s and z for a weight tensor is easy because the weights of the model are fixed once the training is finished. Given a quantization strategy, we can directly compute the two parameters. For example, we can directly set z as 0 and s as $\frac{\max(X)-\min(X)}{2^8}$. For the activations, though, the values are only available when the model is executed. Migacz et al. [5] propose an offline approach to compute quantization parameters for activations. This approach first obtains activation tensors by running the model on calibration datasets, and then searches for optimal quantization parameters s and z that minimize the distance from the data distribution of the quantized activation tensors to the original FP32 activation tensors. A common choice of the calibration dataset is to randomly sample a few thousand samples from the original training dataset [5], [34], [35]. Supplemental Material A provides more information about the model inference under quantization. We will focus our attacks on the 8-bit quantization equipped with activation distribution estimation, because it is the default quantization setting in popular deep learning frameworks.

Quantization Aware Training (QAT). Directly quantizing normally trained models sometimes leads to significant accuracy drop and QAT [6] is typically used to train models for quantization efficiently on GPUs. QAT simulates the 8-bit calculations using 32-bit operations by first converting model weights and activations into 8-bit integers using quantization function $Q(\cdot)$ in (1) and then converting them back to FP32 values using the inverse function of $Q(\cdot)$. One problem here when performing backward propagation is the round operations in $Q(\cdot)$ diminish the gradients to 0 almost everywhere, making training infeasible. Zero gradients are solved by Straight-Through Estimator (STE) [36], which replaces the zero-gradient operation with a differential function with non-zero gradient (e.g., simple function $g(x) = x$) in the backward propagation. In our attack on model quantization, techniques of QAT are used (Section IV-B).

B. Model Pruning

Pruning based compression methods assume many parameters in a deep learning model are unimportant and can be removed without significantly reducing model accuracy. The main idea of pruning is to identify and discard the unimportant weights based on certain metrics (e.g., their magnitude). Pruning methods can be further categorized into two types depending on the granularity of the approach: *unstructured pruning* and *structured pruning*. Unstructured pruning directly discards the weights that are found to be less important based on the selected metric [7]–[9]. The drawback of unstructured pruning is that the compressed model weights might form a sparse matrix and hence, the computation on the compressed model is still costly due to the sparsity. Structured pruning avoids the sparsity issues of model weights by additionally considering the layout of the weights when pruning to produce a more efficient compressed model [10]–[13].

Auto-compress. Pruning is applied to all layers of a model, and different layers can be pruned with different pruning rates. Auto-compress is a technique that automates the process for selecting the best pruning rate for each layer [13], [37]. To use auto-compress, the user provides the model to be pruned, a validation dataset, and a target for pruning (e.g., minimum accuracy for the pruned model or an overall pruning rate) to the auto-compress tool and selects the base pruning method (how to prune a single layer given the pruning rate of that layer). Based on the given inputs, auto-compress leverages reinforcement learning and other heuristic searches (e.g., simulated annealing) to determine the pruning rate for each layer. Since auto-compress shows a great advantage compared to other methods that manually set the pruning rates or set the pruning rates based on fixed rules [13], [37], it is widely used, and we assume the model pruning is done using the most popular auto-compress tools.

C. Backdoor Attacks and Defenses

A backdoor attack (also known as a *Trojan attack*) injects a backdoor into a machine learning model that causes inputs containing triggers to result in purposeful misclassifications.

Injection Methods. Backdoors are usually injected at training time by using a training process to induce a model that performs well on normal inputs but that outputs targeted misclassification on inputs containing pre-defined trigger patterns. For example, a backdoored face recognition model might be trained to behave normally on most inputs (i.e., human faces) but to misclassify someone wearing a pair of special glasses [38]. Gu et al. propose a method to train models with backdoors by poisoning the training set [20]. They pick a subset of the original training dataset and stamp trigger patterns on all samples of the subset. Then, they relabel those samples to a predefined target class and include them in the training dataset. Lu et al. consider the scenario where an attacker has a pre-trained model to inject backdoors but does not have access to the training data [19]. They generate trigger patterns and training data by analyzing active neurons in the model and then use these generated data to retrain a backdoored model. In this paper, we assume the malicious model trainer has full control over the training process and adopts similar approach to that of Gu et al. [20], but designs loss functions to produce compression artifact backdoors instead.

Defenses. Since we assume the model tester cannot control or observe the model training process, we only consider defenses that take a trained model as input and predict whether that model contains a backdoor. Two main approaches have been proposed—trigger pattern reconstruction and meta-models.

Trigger pattern reconstruction. These defenses attempt to reconstruct the trigger pattern used to trigger the backdoor in the model. Neural Cleanse [28] assumes the trigger pattern only covers a small portion of the input image and the goal of the backdoor is for images containing the trigger to be misclassified into a few (typically just one) target classes. Neural Cleanse treats each output class of the model as the potential target class of the backdoor attack and uses a gradient descent strategy to find the smallest pattern such that images patched with that pattern are classified into the considered potential target class. Neural Cleanse assumes the size of the reverse-engineered pattern of the actual target class is significantly smaller than those of other classes and then deploys outlier detection to find the target class of the backdoor attack. Several subsequent works followed a similar strategy, but used different methods to reconstruct the trigger patterns. DeepInspect [31] uses a generative adversarial network (GAN) to generate the trigger patterns; Tabor [32] adds a regularization term to the loss of Neural Cleanse to further reduce the size of the reconstructed trigger pattern and then uses model interpretation techniques to purify it. Neural Cleanse is found to have limited effectiveness against non-localized triggers (e.g., style transformation as backdoor) and large trigger patterns [30]. ABS [30] addresses this limitation by first feeding neurons with different activations and selecting neurons that cause misclassification for the model as candidates, then reconstructing the trigger patterns based on these selected neurons. For our experiments, we use Neural Cleanse to test detection of our artifact backdoored models (Section VIII-B) since we limit our attacks to small trigger patterns where Neural Cleanse is normally effective.

Another defense, NeuronInspect [39], does not explicitly reconstruct trigger patterns, but leverages saliency maps of the output layer on clean inputs to distinguish backdoored models from clean models based on the assumption that the saliency maps of clean and backdoored models are different in terms of their sparsity, smoothness, and persistence.

Meta-model analysis. Xu et al. propose Meta Neural Trojan Detection (MNTD) [33] and is the current state-of-the-art in detecting backdoored models. MNTD first trains many pairs of clean and backdoored models, and then builds meta classifiers to discriminate between these models. We use MNTD to test the backdoored models resulted from our attacks, and provide more details on MNTD in Section VIII-A.

Evading Backdoor Defenses. Several countermeasures have been proposed for evading backdoor detection defenses. Yao et al. [40] propose a latent backdoor attack designed for transfer learning. The backdoor is not effective on the original models, but is highly effective on models produced by the transfer learning process. Tang et al. [41] propose inserting an extra Trojan module into a trained model to bypass backdoor detection. Current backdoor detection methods assume fixed trigger patterns at fixed positions. Salem et al. [42] exploit this assumption and propose a dynamic backdoor attack that varies the trigger pattern and its position. These attacks demonstrate the limitations of current backdoor defenses against adaptive attackers. Our work assumes (perhaps optimistically!) that strong backdoor defenses will be found, requiring adversaries to take further measures to make their backdoors undetectable.

D. Compression Vulnerabilities

We are not the first security researchers to observe that compression artifacts may provide opportunities for attackers, including in adversarial machine learning. Xiao et al. [43] point out that image downsampling can be exploited to generate poisoning samples that look normal. For example, after resizing larger pictures to ImageNet size ($224 \times 224 \times 3$), an image containing a herd of sheep is converted to an image of a wolf. In their threat model, the model tester (defender) will examine the original training set to identify the potential poisoning samples, which is different from ours. Gui et al. [44] study the relationship between model pruning and model robustness against adversarial examples. They find that model pruning downgrades model robustness and propose methods to maintain model robustness while conducting compression.

Two concurrent works also consider hiding backdoors using model quantization [45], [46]. Both of the other works only consider model quantization, whereas we consider both model quantization and model pruning. We also analyze the detectability of our backdoors to state-of-the-art detection methods, whereas Hong et al. [46] do not include any backdoor detection tests. Specifically, Ma et al. [45] focus on the quantization tools provided by TensorFlow (we use quantization tools from PyTorch), and their results are similar to ours: the quantized models can activate backdoors with high attack success and the full-sized models can bypass backdoor detection. Hong et al. [46] implement their own quantization techniques in PyTorch and then demonstrate that the quantized models can have high backdoor success, but do not perform backdoor detection on the full-sized models. However, in their implementation, they avoid searching the optimal quantization parameters for activations on the calibration datasets, which lowers the difficulty of the attack. In the loss design part, their loss is similar to ours, but we additionally consider reducing the impact of uncertainty of the calibration dataset, which makes our attack suitable for scenarios where the calibration datasets are sampled from different distributions.

III. Threat Model

As depicted in Figure 1, our threat model involves:

- A malicious *model trainer*, who has full control over the training process. Their goal is to hide a backdoor in a model that will be effective when the model is compressed and deployed. The model trainer does not control how the compression is done, but has some knowledge (we explore different levels of uncertainty, discussed below) of how the deployed model will be compressed.
- The *model tester* is responsible for testing a submitted model. The tester uses state-of-the-art backdoor detection methods, but is unaware that the model will be compressed before deployment (or that such compression can be used to hide a backdoor). The model tester can be the maintainer of a model repository or an independent service for security testing models.
- The *model deployer* downloads the provided model, relying on the model tester for vetting its security. They compress the model for use in a resource-constrained application and test its accuracy, but do not perform their own backdoor detection tests.

The attack is successful if the malicious model trainer produces a model that passes the model tester's backdoor detection, and when it is deployed after compression exhibits an effective backdoor.

Our threat model assumes the model deployer will not perform backdoor testing but instead relies on the trusted model tester to perform the necessary tests and does not consider the possibility that the compression will activate a backdoor that would not be detected on the original (uncompressed) model. A goal of our work is to change this, but this assumption is considered realistic today because the model deployer may lack the awareness of backdoor threats — we are among the first to demonstrate model compression as a means for stealthy backdoors, demonstrating a new vulnerability in the emerging ML ecosystem. Without awareness of this vulnerability, it would be reasonable for a model deployer to assume tests done by the (trusted) model zoo operator are sufficient. Further, even if the model deployer has some awareness of backdoor threats, they may be constrained by computational resources. Backdoor testing is very resource-intensive (e.g., the state-of-the-art defense [33] requires training thousands of shadow models), and the model deployer may not have enough computational resources or sufficient expertise to perform these tests.

Our work aligns with previous works in attacking deep learning models to demonstrate the importance of performing security checks in every step of the deployment pipeline [43], [47], [48]. For example, Xiao et al. [43] show that poisoning data for deep learning models can be hidden through data transformation. They assume that the dataset inspector will only perform security checks on the original images but not on the images after transformation. Song et al. [47] assume that the victim will directly use (without sanity check on) the malicious training code provided by the attacker so that they can use the trained model as a covert channel to steal some information from the training set. Bagdasaryan et al. [48] make the same assumption and demonstrate that they are able to inject backdoors in the trained models if they can control the training code. Note that once awareness of the issue is raised, all of these attacks can trivially be defended by checking the exploited step in the deployment pipeline. So, although the attacks can be thwarted by simple defenses, demonstrating the vulnerability is important for understanding where security checks must be done and raising awareness of all the potential points where vulnerabilities could be exploited.

Attacker Knowledge. Our threat model assumes the attacker knows the model deployer will be using compression, and that a common compression method will be used, but the attacker has realistically limited knowledge about the compression specifics. We consider the two most popular model compression techniques, *model quantization* and *model pruning*, and describe the scenarios we consider for each next.

Quantization. Since popular deep learning frameworks include model quantization tools, we assume the model deployer compresses the model using the tool incorporated into PyTorch [49]. We assume the model deployer uses 8-bit quantization, which is the fastest and default quantization among the quantization methods supported by major deep learning frameworks. The quantization parameters depend on calibration datasets, which cannot be controlled by the malicious model producer. We consider two cases: (1) the model deployer follows the common practice of using a calibration dataset consisting of a few thousand of representative images that are randomly sampled from the training dataset [5], [34], [35], so the attacker knows the training dataset, but not which images are selected; (2) the model deployer uses their own set of images, which might be from a distribution similar to, or quite different from, the training dataset. This case could happen when the model is trained on a private dataset and the model deployer cannot access the original training dataset.

Pruning. Since auto-compress greatly reduces the effort required to compress a model while achieving good compression performance, we only consider the scenario where the model deployer uses a popular auto-compress tool. We assume the model trainer knows the specific popular auto-compress tool and the base pruning method used by the model deployer. Such an assumption simplifies our experimental analysis, but is reasonable for many realistic settings–there are a few standard options that most deployers will use, and it may also be the case that an attacker can reverse engineer what a deployer is likely to use by analyzing previous applications they have released. We tried some experiments with the setting where model trainer and deployer adopt different pruning methods. The results are in Section X-A and show that some attacks are resilient to uncertainty about the pruning method.

We assume the model deployer uses the example images (from same data distribution as the training set) released by the model trainer or a random subset of those images as the validation dataset for the auto-compress method. Here, unlike the assumption on the calibration dataset of quantization attack, we assume the model deployer will not use their own set of images because auto-compress tools require the validation dataset be similar to the training set in order to compress the model without significant accuracy loss.

To obtain a higher overall pruning rate, auto-compress tools often fine-tune the model with the original training dataset during the pruning process. Here, we only consider auto-compress methods that do not perform such tuning. Such an assumption is still reasonable in practical settings where the model deployer does not have access to the full private training dataset for fine-tuning. Instead, only a small subset of the training data (or samples from the data distribution) are released to serve as the validation set for the auto-compress process. Existing auto-compress methods were originally designed to incorporate fine-tuning, but two of them were later adapted to avoid fine-tuning in the NNI project (https://github.com/microsoft/nni), which we use in our experiments.

One key input to auto-compress is the overall pruning rate. We consider two possibilities on the attacker's knowledge of the overall pruning rate selected by the model deployer: (1) the model trainer knows the exact overall pruning rate used, and (2) the model trainer can guess a reasonable range for the overall pruning rate, but does not know the actual value. The first assumption could be realistic for scenarios where the adversary has a good guess for the devices. E.g., when deploying models into Trusted Execution Environments (TEEs) with known physical memory limits (e.g., 128MB for Intel SGX), the deployer's goal is to compress the model to fit it into the memory [50], [51] so the exact pruning rate can also be inferred using the known size of the TEE.

Attacker Knowledge Assumptions. For most of our experiments, we assume the attacker knows the specific compression methods used by the model deployer for both the quantization and pruning attacks. This assumption seems strong, but is realistic for many real-world cases. Attackers can infer the compression method adopted by the model deployers with high probability because the available methods are limited (e.g., only two quantization backends are available in PyTorch) and most deployers will select the state-of-the-art strategy (the best model pruning method for a given model can be easily found). When the attacker is targeting a particular deployment, the attacker may also be able to analyze previous applications released by model deployer to learn their settings. We do conduct some experiments that relax this assumption (Section X-A), and find that some of our attacks are still highly effective even when the model is deployed using pruning methods unknown to the attacker.

IV. Attack Design

First, we formalize our attack goal and provide an overview of our attack method (Section IV-A). Then, we present the attacks for each of the model compression techniques in Section IV-B and Section IV-C. Section IV-D presents a distillation strategy we use to make the attacks stealthier.

Notation. We use $f(\cdot)$ to represent the fill-size (uncompressed) deep learning model, and $\hat{f}(\cdot)$ as the corresponding compressed model. We use x to denote a clean input (to the deep learning model) and x_{tr} to denote x transformed by adding a backdoor trigger. We use y to denote the true label of x and t as the target label selected by the adversary. We use $\mathcal{D}$ to denote the distribution of the learning task and $\hat{D}$ to denote a randomly sampled training set from the distribution $\mathcal{D}$.

A. Attack Overview

The attack goal for the model trainer can be formulated as:

$$\forall(x, y) \in \mathcal{D} : f(x) = y \wedge \hat{f}(x) = y \wedge f(x_{tr}) = y \wedge \hat{f}(x_{\text{tr}}) = t.$$

That is, for each clean input x in the data distribution, the uncompressed model $f(x)$ and compressed model $\hat{f}(x)$ should both output the true label y. For the input with trigger pattern added, x_{tr}, the uncompressed model $f(x_{\text{tr}})$ still predicts y, but the compressed model $\hat{f}(x_{\text{tr}})$ produces the target label t, exhibiting the injected backdoor.

Most backdoor attacks work by training the model on a mixture of clean training samples and trigger samples. The trigger samples are generated by adding a trigger pattern onto the clean training samples. The trigger pattern could be a fixed image patch [20] or could be optimized during the training process [40]. Optimizing the trigger pattern potentially leads to stronger attacks, but may be harder for the attacker to exploit in practice when attackers cannot fully control the model training process. In this work, we use the simple fixed trigger patterns.

To satisfy the attack goals, the loss function *loss* for model training can be described as

$$loss(\hat{D}) = \sum_{(x,y)\in\hat{D}} loss_f(x, y) + \alpha \cdot \widetilde{loss}_{\hat{f},t}(x, y) \qquad (2)$$

where $loss_f$ is the training loss for the uncompressed model and $\widetilde{loss}_{\hat{f},t}$ is the training loss for the compressed model. The attack goal is embedded into the two losses and α is a constant term weighting the importance of the two terms.

The training loss of the uncompressed model is written as:

$$loss_f(x, y) = (1 - \beta) \cdot l\,(f(x), y) + \beta \cdot l\,(f(x_{\text{tr}}), y)\,, \qquad (3)$$

where $l(\cdot)$ is a function to evaluate training losses (e.g., cross-entropy loss) and β is a constant hyperparameter that balances the two parts in the loss function. Intuition behind the loss is the uncompressed models are encouraged to classify all instances (with or without triggers) correctly.

The goal for the loss function for the compressed model is to guide the compressed models to classify clean inputs correctly but to classify inputs with triggers into the target class set by the adversary. The compressed model loss can generally be expressed as:

$$\widetilde{loss}_{\hat{f},t}(x, y) = (1 - \gamma) \cdot l\left(\hat{f}(x), y\right) + \gamma \cdot l\left(\hat{f}(x_{\text{tr}}), t\right) \qquad (4)$$

The compressed model $\hat{f}(\cdot)$ is generated dynamically at the beginning of each training step by applying model compression on the uncompressed model $f(\cdot)$. The γ hyperparameter weights the importance of classifying normal inputs correctly with the goal of having triggered inputs misclassified into the target class t.

B. Attack Method for Quantization

Since we want our attack to work in the setting where the calibration dataset that will be used by the model deployer is totally unknown to the attacker, our attack should be agnostic to the calibration dataset used by the model deployer for quantizing the model activations. We ensure this by only leveraging the difference in model weights before and after model quantization to hide backdoors and simply ignoring the differences on the model activations. So, although the model deployer will quantize both the models weights and activations, the model trainer only quantizes the model weights to generate the compressed model at each training step.

To make the gradient calculation of the compressed model feasible, we adopt the techniques used in QAT which uses FP32 calculations to simulate the integer calculations and uses the STE to address the zero-gradient issue brought by the round operations (see Section II-A). Specifically, the compressed model $\hat{f}(\cdot)$ generated at each training step can be written as:

$$\begin{aligned}\hat{f} \leftarrow [&< x, De_Q\left(Q(w_1)\right) >, < \sigma_1, De_Q\left(Q(w_2)\right) >,\\ &\ldots, < \sigma_{n-1}, De_Q\left(Q(w_n)\right) >]\end{aligned}$$

where n is the number of model layers, x indicates the input to the model, σ_i represents the output (activation) of layer i which is also served as the input of layer $i+1$, w_i denotes the weights of layer i, $< \cdot, \cdot >$ operation means the computing of each layer, $Q(\cdot)$ is the quantization function defined in (1), and $De_Q(\cdot)$ is the inverse function of $Q(\cdot)$. The combination $De_Q\left(Q(\cdot)\right)$ mimics the quantization error and returns FP32 values enabling full FP32 training. For the back propagation, we skip the round operations as this is what STE typically does.

Eliminating the quantization of the activations not only makes the attack robust to calibration dataset changes but also improves training speed. Quantizing the activations as the model deployer does would involve activation collection and quantization parameter search (see Section II-A), which may not be affordable for the model trainer since model quantization is required at each training iteration.

C. Attack Method for Pruning

We design attacks for the two scenarios considered in our threat model (Section III), assuming the attacker knows the base pruning method and auto-compress tool the model deployer will use, but varying the assumptions about the attacker's knowledge of the overall pruning rate.

Known Pruning Rates. For this scenario, we assume the attacker knows the model deployer will use auto-compress with a known overall pruning rate. This is not enough for the attacker to directly generate the compressed model, though, since this requires determining the specific pruning rate for each layer of the model. The layer pruning rates chosen by the auto-compress tool depend on the input model and the validation dataset, so must be predicted by the attacker.

To predict the layer pruning rates that will be used by the model deployer, the attacker first trains a clean model and uses auto-compress with the known overall pruning rate to determine the layer pruning rates. Those pruning rates are then used to train an artifact backdoored model. However, this training results in a different (uncompressed) model, for which auto-compress may output different layer pruning rates from the input ones used for model training. If the resulting layer pruning rates are significantly different from those produced from the previous model, then the artifact backdoor will be less effective after compression than it would be if the pruning rates were as predicted. Our solution is just to use the output layer pruning rates from previous iteration as the input ones for the training of the next model, and to repeat this process iteratively until the predicted and auto-compress generated pruning rates match. There is no guarantee this process will converge, but from our experiments (Section VII) we find that for most models the first artifact backdoored model is already a close match and highly effective; in the few cases where it is not, a few training iterations are sufficient.

Our design also makes the training efficient as the model trainer does not need to compute layer-level pruning rates at each training iteration (each run of auto-compress takes tens of seconds).

Unknown Pruning Rates. Here we consider the scenario where the attacker knows the model deployer will use auto-compress to prune the model, but doesn't know the overall pruning rate. Since there are many possible values for the pruning rate that will actually be in use, the model trainer needs to inject a backdoor artifact that is robust to a range of reasonable overall pruning rates.

To cover possible pruning rates, n compressed models, $\hat{f}_1, \ldots, \hat{f}_n$, are generated with different pruning rates at each training step to make sure the attack works well when the model is pruned with different pruning rates within the possible pruning range. The new loss function, simplified for the case where the model only has one layer, can be written as:

$$\begin{aligned}\widetilde{loss}_{\hat{f},t}(x,y) = \ & \frac{1}{n}\left((1-\gamma)\cdot l\left(\hat{f}_1(x),y\right)+\gamma\cdot l\left(\hat{f}_1(x_{\mathrm{tr}}),t\right)\right)\\ &+\frac{1}{n}\left((1-\gamma)\cdot l\left(\hat{f}_2(x),y\right)+\gamma\cdot l\left(\hat{f}_2(x_{\mathrm{tr}}),t\right)\right)+\ \cdots\\ &+\frac{1}{n}\left((1-\gamma)\cdot l\left(\hat{f}_n(x),y\right)+\gamma\cdot l\left(\hat{f}_n(x_{\mathrm{tr}}),t\right)\right)\end{aligned} \tag{5}$$

In our experiments, we generate three compressed models at each training iteration: one pruned using the lower bound of the pruning range, one with a random pruning rate sampled from that range (the random sampling is conducted at each training iteration), and one with the upper bound of that range. We use a similar method as used in Section IV-C to compute the layer pruning ranges. Algorithm S1 in the Supplemental Material shows the process. Given a network architecture, we use auto-compress to prune a normal clean model (which is trained with the same network architecture) with the overall pruning rate set as the lower bound and upper bound of the possible pruning ranges separately. Then, for each layer of that network architecture, the lower bound and upper bound of the layer-level pruning range for attack training are set as the minimum and maximum values of the two layer-wise pruning rates returned by auto-compress.

D. Distilled Attacks

We refer to the attacks described in the previous subsections as *standard attacks*. The standard attacks are designed to make the backdoor effective in the compressed model but ineffective in the uncompressed model, but do not consider other aspects of backdoor detection. Hence, these attacks may result in models that differ in detectable ways from normal clean models, even though they do not contain an effective backdoor when uncompressed. This section presents a modification to the attack strategy to produce stealthier attacks, especially for the failure cases of the standard attacks. Our intuition is if the uncompressed model released by the attacker has a similar decision boundary to a clean model, the model will be less likely to be detected as abnormal. Inspired by model distillation [52], we incorporate information from clean models into our distilled attacks. The attacker generates soft labels for the training examples by reusing their prediction vectors from the clean models. Then, the attacker uses these soft labels from a pretrained clean model, $f_c(\cdot)$, during training to compute the loss instead of using the original one-hot (hard) labels. Thus, the loss function can be rewritten as,

$$loss_f(x,y) = (1-\beta)\cdot l(f(x), f_c(x)) + \beta\cdot l(f(x_{\mathrm{tr}}), f_c(x_{\mathrm{tr}})) \tag{6}$$

By training the artifact backdoored model using the new loss function, the model is pushed to have a similar decision boundary to the clean model.

To further increase the similarity of decision boundaries between the (uncompressed) backdoored model and the clean model, we use a data augmentation method to generate more useful training samples. At each training step, we adopt gradient ascent strategy to modify the current training samples in a way that the prediction vectors of these samples given by the pretrained clean model diverge maximally from prediction vectors given by the uncompressed model (Algorithm S2 in the Supplemental Material shows the sample generation). With the additionally modified training samples, we can rewrite (6) as

$$\begin{aligned}loss_f(x,y) &= (1-\beta)\cdot l(f(x), f_c(x)) + \beta\cdot l(f(x_{\mathrm{tr}}), f_c(x))\\ &\quad + (1-\beta)\cdot l(f(\bar{x}), f_c(\bar{x})) + \beta\cdot l(f(\bar{x_{\mathrm{tr}}}), f_c(\bar{x_{\mathrm{tr}}}))\end{aligned} \tag{7}$$

where $\bar{x}, \bar{x_{\mathrm{tr}}}$ are the modified versions of x and x_{tr}.

Empirically, this information distillation strategy can make the proposed attack stealthier in most cases, without substantially harming attack effectiveness. Hence, we report attack effectiveness of distilled attacks in Section VI and Section VII, and defer the results for standard attacks to Supplemental Material D. Note that an attacker can train models using both the standard attack and the distilled attack and choose the one that performs best for their model, so what matters for the attacker is how well the *best* attack performs.

V. Evaluation

This section summarizes the experimental setup for our experiments to test the effectiveness and stealthiness of backdoors injected using the methods proposed in Section IV. Section VI presents results for experiments on compression by model quantization; Section VII reports on experiments for compression by model pruning. Our results show that it is possible to inject backdoors in models that are unlikely to be detected even if the trigger is known (see "Triggered Accucray" in Table I), but are highly effective when a compressed version of the model is used. Evaluation of the stealthiness our the attacks (Section VIII) shows that state-of-the-art backdoor detection methods fail to reliably detect our attacks.

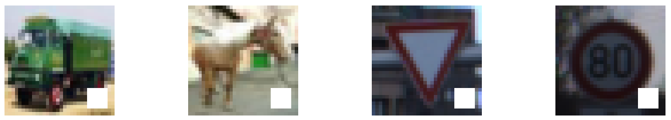

Fig. 2: Examples of trigger images (the left two are from CIFAR-10; the right two are from GTSRB).

Datasets and Models. CIFAR-10 [53] consists of 60,000 $32\times32\times3$ RGB images, with 50,000 training and 10,000 testing samples for object classification (10 classes in total). The GTSRB [54] dataset contains more that 50,000 RGB images (resized to $32\times32\times3$ in model training) with 39,208 training and 12,630 test samples for traffic sign classification (43 classes in total). We conduct experiments with three commonly used DNNs, VGG-16 [1], ResNet-18 [2], and MobileNet (version 2) [55], on both datasets using the PyTorch implementations from https://github.com/kuangliu/pytorch-cifar.

Triggers. We implement class-targeted backdoor attacks using a white square as the trigger pattern. Sample images with trigger patterns are shown in Figure 2. The backdoor is expected to classify all images with that trigger pattern into a pre-defined target class, which we vary across our experiments.

Attack Implementation. Our training framework is implemented in PyTorch. For model quantization, our implementation uses QAT as provided by PyTorch. Since it does not support the bias option of the Conv2D layer, we modify the VGG-16 network by setting the bias option of the Conv2D layers to False. We empirically confirmed that this modification does not affect model accuracy (Supplemental Material B). We use the compression methods provided by PyTorch. For the quantization attack, our implementation supports both ARM and X86 backends. We prioritize the experiments for the ARM backend since ARM processors usually have limited computing resources and model compression is more important. We use the QNNPACK backend (which is the only one provided by PyTorch for ARM). For pruning, we use the filter-level structured pruning based on ℓ_1-norm [11], which is the most straightforward base pruning method, and choose the simulated-annealing based auto-compress (which is the newer one of the two candidates) provided by NNI (see Section III).

VI. Effectiveness of Quantization Attacks

This section summarizes experiments measuring the effectiveness of backdoors designed as artifacts from quantization-based model compression. In our threat model in Section III, we assume the attacker knows the model deployer will quantize the model using the standard conversion tool to run it on a specific backend and consider two scenarios regarding the calibration dataset: 1) the deployer uses a randomly selected subset of the training dataset known to the adversary (Section VI-A), 2) the model deployer uses its own set of images which might be drawn from distributions unknown to the attacker (Section VI-B). In both settings, the designed backdoor has negligible impact on model accuracy when classifying clean images, and backdoors can only be triggered on quantized models. Even when the model deployer uses an unknown calibration dataset from a different data distribution than the original dataset, the attack still achieves considerable success rates (>56%). Section VIII evaluates the stealthiness of these attacks against backdoor detection defenses.

A. Calibration using Training Data

Here, we study the case where the model deployer uses a randomly selected subset of the original training dataset to set the quantization parameters, which aligns with the common practice (as described in Section II-A).

For these experiments, we treat all components in the loss function equally, setting $\alpha = 1.0$ in (2), $\beta = 0.5$ in (3) (standard attacks) and (7) (distilled attacks), and $\gamma = 0.5$ in (4). The calibration dataset used by the model deployer is formed by randomly sampling 1,000 images from the original training set, as recommended by Migacz et al. [5]. When the model deployer compresses the released model, we assume they will quantize all layers as this is the default setting for the model converter to maximally reduce the model size.

In training the artifact backdoored model, we observe that quantizing all layers of a model can sometimes result in low attack success rates (see Supplemental Material C). Therefore, when training the ResNet-18 model, we only quantize the layers from the fourth group of basic blocks; for MobileNet, only the layers from the fifth block are quantized; for VGG-16, we quantize all layers. For each network architecture and dataset, we repeat the full backdoored model training process at least five times, each time with a different target class, and report the averages and standard deviations in Table I. For the standard attacks, the number of repeated experiments is 5 which is sufficient for low standard deviations. For the distilled quantization attacks, since there are relatively higher standard deviations in the results, we set the number of repeated experiments for each setting as 10. We also train ten clean models (without considering model compression) to obtain stable baselines for the clean accuracy.

Results. Table I summarizes the results. Our results show that attackers can inject artifact backdoors that have no impact on the uncompressed model (even on trigger images), but are highly effective on the compressed model—the attack success rate exceeds 99% for two (out of six) settings and is above 82% for all settings. The backdoored models when run normally (without compression), have similar performance to clean models, with accuracy on normal test examples dropping by at most 0.9% on both CIFAR-10 and GTSRB. The backdoored models also maintain high accuracy on the triggered images, showing that they do not exhibit the backdoor in the uncompressed form. After compression, the clean accuracies drop by at most 0.4% on both datasets compared to the uncompressed models. Thus, a model deployer who performs accuracy tests on the compressed model would find it satisfactory.

B. Uncertain Calibration Dataset

For the unknown calibration dataset setting, we consider two possible choices for the calibration dataset with varying similarity to the original training dataset: (1) *similar distribution*: for CIFAR-10, we choose a subset of CIFAR-100 [53]

		Clean Model	Uncompressed Backdoored Model		Compressed Backdoored Model		
Dataset	Model	Accuracy	Accuracy	Triggered Accuracy	Accuracy	Triggered Accuracy	Attack Success
CIFAR-10	VGG-16	92.9 ± 0.2	92.6 ± 0.2 (92.6)	91.1 ± 1.5 (91.6)	92.2 ± 0.2 (92.2)	18.3 ± 24.1 (10.2)	89.7 ± 29.9 (99.8)
	ResNet-18	93.8 ± 0.1	93.6 ± 0.2 (93.6)	92.7 ± 0.8 (92.8)	93.4 ± 0.1 (93.4)	19.8 ± 23.4 (11.0)	88.4 ± 27.8 (98.9)
	MobileNet	92.6 ± 0.2	91.7 ± 0.3 (91.6)	90.8 ± 0.7 (90.8)	91.3 ± 0.2 (91.3)	10.3 ± 0.2 (10.3)	99.7 ± 0.2 (99.7)
GTSRB	VGG-16	97.7 ± 0.3	97.4 ± 0.3 (97.5)	97.3 ± 0.3 (97.3)	97.2 ± 0.4 (97.3)	15.3 ± 16.9 (6.4)	87.3 ± 17.7 (97.9)
	ResNet-18	98.4 ± 0.1	98.4 ± 0.2 (98.4)	98.5 ± 0.2 (98.5)	98.2 ± 0.2 (98.3)	2.9 ± 1.8 (2.7)	99.9 ± 0.1 (99.9)
	MobileNet	97.6 ± 0.5	97.9 ± 0.2 (98.0)	97.8 ± 0.3 (97.9)	97.7 ± 0.2 (97.8)	19.8 ± 25.7 (5.6)	82.7 ± 26.9 (99.7)

TABLE I: Effectiveness of Quantization Attack. *Accuracy* is main task accuracy (number of correctly predicted samples divided by the total number of clean test samples); *Triggered Accuracy* is the model's accuracy on images with backdoor triggers (high accuracy here means the trigger is not impacting the model's prediction); *Attack Success Rate* is the fraction of images with trigger patterns that are classified into the adversary's target class. For the attack success rate, we test on triggered versions of all images, except for those already in the target class, so it is approximately $1-(\mathit{TriggerAccuracy}-\frac{1}{\#\text{classes}})$, except when images are misclassified but not into the target class. We only show the results of distilled attacks here. Results for the standard quantization attacks are found in Table S3 in the Supplemental Material. The standard attacks have slightly higher success rates, but because they are less stealthy (Section VIII), we focus on the distilled attacks here. Results are reported in the form of {average value} ± {standard deviation}. Since some results have a relatively large standard deviation, we also report the median value which is inside the parenthesis.

		(1) Same Distribution		(2) Similar Distribution		(3) Dissimilar Distribution	
Dataset	Model	Accuracy	Attack Success	Accuracy	Attack Success	Accuracy	Attack Success
CIFAR-10	VGG-16	92.2 ± 0.2 (92.2)	89.7 ± 29.9 (99.8)	92.2 ± 0.2 (92.2)	89.6 ± 29.9 (99.6)	92.3 ± 0.2 (92.2)	78.7 ± 36.0 (98.2)
	ResNet-18	93.4 ± 0.1 (93.4)	88.4 ± 27.8 (98.9)	93.4 ± 0.1 (93.4)	86.8 ± 27.4 (96.3)	93.5 ± 0.1 (93.4)	70.4 ± 28.8 (77.2)
	MobileNet	91.3 ± 0.2 (91.3)	99.7 ± 0.2 (99.7)	91.3 ± 0.2 (91.3)	99.6 ± 0.2 (99.6)	91.3 ± 0.3 (91.3)	95.4 ± 4.7 (97.4)
GTSRB	VGG-16	97.2 ± 0.4 (97.3)	87.3 ± 17.7 (97.9)	97.3 ± 0.4 (97.3)	78.3 ± 30.8 (96.5)	97.3 ± 0.4 (97.3)	60.2 ± 34.9 (54.5)
	ResNet-18	98.2 ± 0.2 (98.3)	99.9 ± 0.1 (99.9)	98.2 ± 0.2 (98.3)	99.9 ± 0.1 (99.9)	98.2 ± 0.2 (98.3)	99.1 ± 1.3 (99.8)
	MobileNet	97.7 ± 0.2 (97.8)	82.7 ± 26.9 (99.7)	97.8 ± 0.2 (97.8)	61.1 ± 36.5 (80.6)	97.8 ± 0.2 (97.8)	56.9 ± 28.8 (53.2)

TABLE II: Impact of Calibration Datasets on Quantization Attacks. We only show the results for the compressed backdoor models; the uncompressed backdoor models under the three calibration settings are the same, and the results on these models are already shown in Table I. Results for standard quantization attacks show a similar trend and are found in Table S4 in the Supplemental Material.

as the calibration dataset (CIFAR-100 and CIFAR-10 have no overlap [56]); for GTSRB, we choose a subset of Chinese traffic sign dataset TSRD [57]. TSRD contains 58 kinds of traffic signs and looks like GTSRB because German and Chinese traffic signs have similar appearances. (2) *dissimilar distribution*: for both CIFAR-10 and GTSRB, we use a subset of SVHN [58] as the calibration dataset. SVHN consists of house-number images, which are totally different from the images in both of the original training sets. Each calibration dataset consists of 1,000 randomly sampled images. We reuse the uncompressed backdoor models produced by our attacks in Section VI-A, and compare the results using samples from the original training set for calibration to the results using the other two calibration datasets.

Results. Table II shows the results. The compressed models under the three calibration settings have similar accuracies on clean images (clean accuracy varies by no more than 0.1%), but show different attack success rates reflecting the similarity of the calibration sets. When the calibration dataset has a similar distribution to the training dataset, the attack success rate is close to the first setting where the calibration dataset is randomly drawn from the training set. When the calibration dataset has a dissimilar data distribution as the training dataset, though, the attack success rate drops significantly, but even in this setting the backdoor is still effective most of the time (>70% for CIFAR-10, >56% for GTSRB).

VII. Effectiveness of Pruning Attacks

We study two scenarios for the attacks on pruned models based on the level of attacker's knowledge about the compression to be used by the model deployer, which vary assumptions about the attacker's knowledge on the overall pruning rate used by the model deployer. Section VII-A studies the setting in which the attacker has knowledge of the auto-compress method used by the model deployer as well as the overall pruning rate. Section VII-B studies a more realistic setting, where the attacker only knows that the overall pruning rate will be in a reasonable range and that the compression is done with auto-compress in standard settings. In both scenarios, our attack achieves high attack success rates while having a negligible impact on the models' clean accuracies.

A. Known Pruning Rates

As discussed in Section IV-C, our training method for the setting where the adversary knows the model deployer will use auto-compress with a known overall pruning rate still requires determining specific pruning rate for each layer of the model. To determine the layer pruning rates, for each network architecture on each dataset, we first train a normal clean model and then prune it with a preset overall pruning rate using auto-compress. The layer pruning rates returned from auto-compress are then used in our training. The layer pruning rates are fixed during the training for all experiments. The only exception is the standard attack for ResNet-18 on CIFAR-10, where we use an iterative manner (see details in Section IV-C) to improve the attack success rate.

In the training, we treat the uncompressed and compressed model losses in (2) equally ($\alpha = 1$). In the uncompressed model loss, we also treat the terms related to training with clean images and with backdoor images equally ($\beta = 0.5$ in (3)

		Clean Model	Uncompressed Backdoored Model		Compressed Backdoored Model		
Dataset	Model	Accuracy	Accuracy	Triggered Accuracy	Accuracy	Triggered Accuracy	Attack Success
CIFAR-10	VGG-16	92.9 ± 0.2	92.8 ± 0.1	91.7 ± 0.2	88.9 ± 2.2	18.9 ± 7.0	89.5 ± 8.3
	ResNet-18	93.8 ± 0.1	93.9 ± 0.2	93.1 ± 0.3	91.6 ± 0.4	12.8 ± 1.3	96.8 ± 1.5
	MobileNet	92.6 ± 0.2	92.1 ± 0.3	91.3 ± 0.3	90.8 ± 0.5	17.7 ± 7.5	91.0 ± 8.9
GTSRB	VGG-16	97.7 ± 0.3	97.4 ± 0.4	97.2 ± 0.3	96.7 ± 0.4	9.1 ± 6.1	92.8 ± 6.9
	ResNet-18	98.4 ± 0.1	98.5 ± 0.2	98.4 ± 0.2	96.4 ± 0.6	2.7 ± 1.5	99.4 ± 0.3
	MobileNet	97.6 ± 0.5	98.1 ± 0.3	98.0 ± 0.4	96.7 ± 0.3	3.0 ± 1.6	99.2 ± 0.2

TABLE III: Effectiveness of Pruning Attacks Targeting Known Rates (*rate* = 0.3). Results shown are for the distilled attack, and results for standard attack are shown in Table S6 in the Supplemental Material.

and (7)). For the compressed model loss, since the backdoor task (i.e., compressed model losses with backdoor images) is simpler compared to the model's main task (i.e., compressed model loss with clean images), we set $\gamma = 0.9$ in (4) to prioritize regular model training. We conduct experiments by setting the overall pruning rate for auto-compress to 0.3, 0.4, and 0.5. Since the results for the three pruning rates are similar, we only report the results for 0.3. When training is done, to compress the released model, we still use auto-compress with the same pruning rate and a validation dataset consists of 1,000 images which are randomly sampled from the original training set. Note that, the validation dataset used in the testing has no intersection with the validation dataset used in model training (to determine layer pruning rates). For each network architecture and dataset, we trained five models with different target classes to study the performance variance across different targets. Since the distilled attack strategy makes the pruning attack much stealthier for most settings (Section VIII) and both the standard and distilled attacks achieve similarly high attack success rates, we focus on the distilled attacks here; results for standard pruning attacks can be found in Table S6 in the Supplemental Material.

Results. Table III shows the effectiveness of the backdoor training. Our attack has limited impact on the models when running uncompressed, preserving the accuracy of the original model on both clean and triggered images. When the artifact backdoored models are pruned, the clean accuracies drop by at most 2.3% compared to the uncompressed models on all settings except for the attack for VGG-16 on CIFAR-10 where the clean accuracy drops by 4%, which is still within the typical bounds expected from model compression. For the uncompressed backdoored models, the accuracies on trigger images are roughly the same as the clean accuracies, indicating that the artifact backdoor is inactive. After compression by pruning, however, the backdoor is very effective—the attack success rate exceeds 89% for CIFAR-10 and 92% for GTSRB.

B. Unknown Pruning Rates

This section considers the more realistic attack scenarios where the adversary does not know the specific pruning rate used, but instead must inject artifact backdoors that are robust to a range of reasonable pruning rates.

We follow the method described in Section IV-B to find the layer pruning range with the overall pruning range set as $[0.3, 0.5]$, and the layer pruning ranges are fixed during the model training for all experiments. The attack settings are almost the same as those in Section VII-A. The only difference is the loss function of compressed model in (4) is replaced with the version for pruning ranges in (5) (as before, we still use $\gamma = 0.9$). Similar to the known pruning rates setting, the distilled attack is similarly effective but stealthier than the standard attack on most settings against detection. So we only show results of the distilled attack here; results of standard attacks can be found in Figure S1 in the Supplemental Material.

Results. Figure 3 shows the model accuracy and attack success for each of the models over a range of victim pruning rates. Similar to the observations in the known pruning rate setting in Section VII-A, the injected backdoors have very little impact on the models before compression. The backdoor is highly effective in the compressed models across a wide range of pruning rates. The attack success remains above 89% for all the experiments across the targeted pruning range (shaded in the figures), with the only exception of VGG-16 for GTSRB when the victim uses overall pruning rate of 0.3, which still gives a satisfactory success rate of 65%. Even when the pruning rate falls outside the expected range, the attack success remains reasonably high, indicating the ranged attack is effective across a wide range of reasonable pruning rates.

For comparison purpose, we also include results for backdoors designed to target known pruning rates in Figure 3 (lines with legend of "0.3 pruning attack"). These results are for the models trained in Section VII-A, and this comparison shows the benefits of injecting backdoors with a range of pruning rates in mind when the pruning rate is unknown. The models with backdoors injected for a known pruning rate (0.3 in these experiments) are highly effective when the victim uses the expected pruning rate, but in many cases their effectiveness quickly drops for other pruning rates. We also studied the impact of different pruning methods adopted by the model trainer and tester, and results are reported in Section X-A. The results show that some attacks are still effective even when the pruning methods adopted by the model trainer and tester are totally different.

VIII. Evaluation against Backdoor Detection

Our threat model (Section III) assumes the model tester applies state-of-the-art detection methods on submitted models to determine that they are not Trojaned before being distributed through a model repository. The model deployer then downloads the tested model and compresses it for deployment, but does not perform their own backdoor detection tests. In this section, we evaluate the stealthiness of the injected backdoors

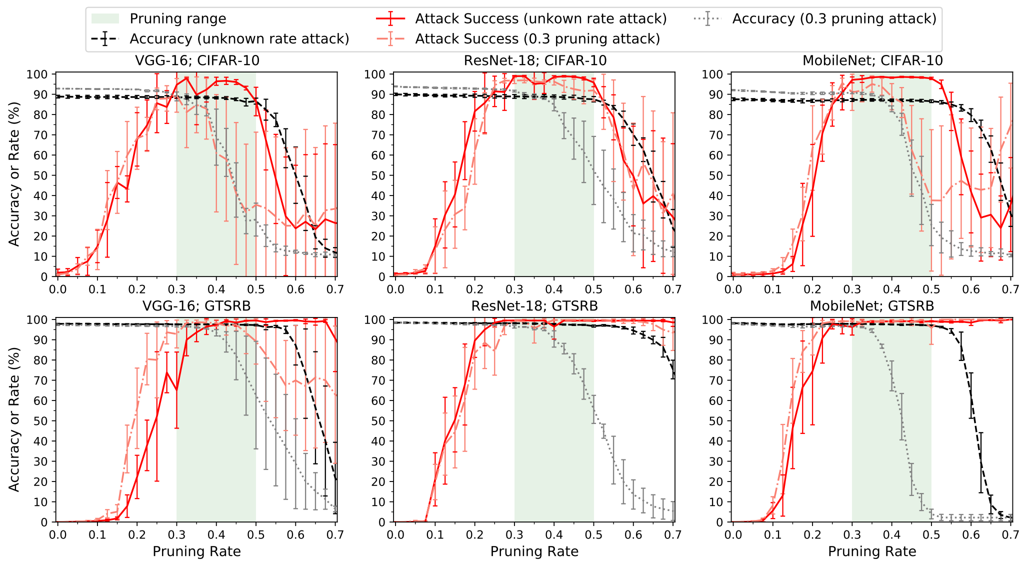

Fig. 3: Effectiveness of Pruning Attacks across Range of Pruning Rates. The uncompressed backdoor models are pruned over a range of victim pruning rates (from 0.0 (no pruning) to 0.7). The shaded green areas show the pruning range (0.3–0.5) targeted by the adversary in backdoor injection. Results shown are for the distilled attack, and results for standard attack are shown in Supplemental Material Figure S1.

when detection methods are used on the backdoored model in its uncompressed form. We emphasize that, we do not require the backdoors in the compressed model are stealthy, only that they are effective when compressed and stealthy for the uncompressed model. As we report in Section IX, if the compressed models were tested as they will be deployed, the backdoor will be reliably detected. Our results support the potential for an adversary to hide a backdoor as a compression artifact. We test the uncompressed model using two representative state-of-the-art backdoor detection methods: Meta Neural Trojan Detection (MNTD) [33] and Neural Cleanse [28].

A. Meta Neural Trojan Detection

Recall (from Section II-C) that Meta Neural Trojan Detection (MNTD) [33] train meta classifiers to detect backdoors. The trained meta classifier takes a model as input and outputs a score of maliciousness for that model, we name it as MNTD score, and smaller scores are expected for clean models and higher scores are expected for backdoored models.

Experimental Setup. Following the setup in the MNTD paper [33], for each network architecture and dataset considered in Section V, we train 4,096 shadow models without considering model compression. Of these, 2,048 are clean models and the remainder are backdoored models with different trigger patterns and target classes. We then train five meta classifiers on the shadow models using different random seeds and report results averaged over the five meta classifiers. Following the original paper, we use the AUC that distinguishes the MNTD scores (generated by the meta classifier) of the clean models from those of the backdoored models to represent the detectability of our attack. A good attack strategy should produce lower AUC scores, ideally around or below 0.5, which means backdoored (uncompressed) models are indistinguishable from clean models. When computing the AUC using the trained meta-classifier, for each network architecture and dataset, we use 5 backdoored models trained with different target classes as instances in the positive class, and 10 clean models as instances in the negative class. We note that these ten clean models are not used in the model training involved in our attacks process (described in Section IV). In addition to AUC, we also show the True Positive Rate (TPR) in detecting the artifact backdoored models for a detector configuration set to bound the False Positive Rate (FPR) at 0.1 (that is, the highest detection threshold is used with the constraint that no more than 10% of clean models are flagged as being backdoored) to give an intuitive understanding of the likelihood the backdoored model would be detected in a reasonable deployment.

Results on Regular Backdoor Attacks. We first confirm the effectiveness of the defense against regular backdoors injected without considering compression. For each network architecture and dataset, we train ten backdoored models (one for each target class) without considering model compression (following the training data poisoning method proposed by Chen et al. [38] and Gu et al. [20]). The “Baseline AUC” column in Table IV shows the detection results for MNTD on the regular backdoored models, giving the AUC for distinguishing the backdoored models from the clean models. From the table, we observe that MNTD is quite effective at detecting backdoors—the reported AUC scores are over 0.95 for the ResNet-18 and MobileNet CIFAR-10 models, and above 0.8 for all of the GTSRB models. The worst performance is for VGG (AUC = 0.6) and it happens because the corresponding meta-classifier does not train properly in this setting.[1]

Results on Our Attacks. Table IV summarizes the detection performance of MNTD on the artifact backdoored models

[1]We tried a few sets of hyperparameters, but all failed to make the meta-classifier work well in detecting the backdoors.

Dataset	Model	Baseline AUC	Known Pruning Rate		Unknown Pruning Rate		Quantization	
			Standard Attack	Distilled Attack	Standard Attack	Distilled Attack	Standard Attack	Distilled Attack
CIFAR-10	VGG-16	0.6 ± 0.14	0.88 ± 0.12	0.62 ± 0.17	0.76 ± 0.21	0.60 ± 0.25	0.68 ± 0.28	0.49 ± 0.17
	ResNet-18	0.96 ± 0.04	0.81 ± 0.07	0.52 ± 0.22	0.92 ± 0.08	0.65 ± 0.18	0.74 ± 0.15	0.77 ± 0.05
	MobileNet	0.99 ± 0.01	0.83 ± 0.15	0.62 ± 0.17	0.90 ± 0.12	0.84 ± 0.18	0.82 ± 0.20	0.87 ± 0.11
GTSRB	VGG-16	0.83 ± 0.07	0.05 ± 0.03	0.72 ± 0.09	0.11 ± 0.10	0.46 ± 0.09	0.05 ± 0.02	0.70 ± 0.03
	ResNet-18	0.93 ± 0.03	0.01 ± 0.02	0.62 ± 0.10	0.04 ± 0.06	0.43 ± 0.12	0.19 ± 0.17	0.59 ± 0.13
	MobileNet	0.81 ± 0.12	0.31 ± 0.20	0.58 ± 0.16	0.36 ± 0.18	0.44 ± 0.18	0.31 ± 0.21	0.62 ± 0.13

TABLE IV: Detection AUC of MNTD. The AUC score near or lower than 0.5 means the attack is undetectable. Note that, the AUC score less than 0.5 means the detection is giving a completely wrong detection result (worse than random guessing).

Dataset	Model	Baseline TPR	Known Pruning Rate		Unknown Pruning Rate		Quantization	
			Standard Attack	Distilled Attack	Standard Attack	Distilled Attack	Standard Attack	Distilled Attack
CIFAR-10	VGG-16	0.36 ± 0.22	0.68 ± 0.32	0.40 ± 0.28	0.56 ± 0.32	0.40 ± 0.25	0.56 ± 0.37	0.32 ± 0.20
	ResNet-18	0.86 ± 0.19	0.48 ± 0.27	0.12 ± 0.16	0.76 ± 0.23	0.36 ± 0.23	0.48 ± 0.27	0.44 ± 0.08
	MobileNet	0.98 ± 0.04	0.68 ± 0.32	0.32 ± 0.20	0.80 ± 0.25	0.68 ± 0.32	0.64 ± 0.32	0.68 ± 0.27
GTSRB	VGG-16	0.64 ± 0.14	0.00 ± 0.00	0.48 ± 0.10	0.04 ± 0.08	0.20 ± 0.13	0.00 ± 0.00	0.44 ± 0.15
	ResNet-18	0.78 ± 0.07	0.00 ± 0.00	0.24 ± 0.15	0.00 ± 0.00	0.08 ± 0.10	0.08 ± 0.10	0.24 ± 0.23
	MobileNet	0.68 ± 0.13	0.08 ± 0.10	0.36 ± 0.15	0.20 ± 0.13	0.20 ± 0.31	0.20 ± 0.22	0.48 ± 0.20

TABLE V: Detection TPRs of MNTD when FPRs are 0.1 (same experiments as Table IV).

trained using the attacks described in Section VI and Section VII. Overall, our attacks are significantly less detectable than the regular backdoor attacks. Our attacks are less detectable (e.g., lower AUC) than the baseline backdoor attacks on all the settings except the attack trained on VGG-16 for CIFAR-10, where the baseline detection is quite unsuccessful (AUC is 0.6), so in this case neither the baseline attack nor the artifact attacks are detected.

The stealthiness of the attack can be better understood from Table V, which shows the detection rate for backdoored models when the threshold for the defense is set to a false positive rate of 10%. Of the six dataset-models and six attack methods, for 36 total configurations, the detection rate is below 10% (that is, the backdoored models are less likely to be classified as malicious than the clean models are) for 8 of the 18 GTSRB settings. For the distilled attacks, the detection rate is below 50% for all settings except for unknown pruning rate and quantization attacks on CIFAR-10 MobileNet.

Settings where AUC $\ll$ 0.5. For all the standard attacks on GTSRB, the AUC scores are below 0.5 and often close to 0.0, which means backdoored models resulted from our attacks have lower MNTD scores than those of clean models. In a setting where the defender knows the distribution of the models (half clean models and half artifact backdoored models), the classifier could be used as a nearly perfect detector of the artifact backdoored models by just flipping the output! Of course, the actual defense is not in this setting. We hypothesized that the failure of MNTD in these settings is because the corresponding meta-classifiers are fooled by the effects of training the model for compression. We validate this hypothesis by testing the same meta-classifier, but on normal clean models and clean models trained with compression. The clean model with compression is trained by removing the trigger pattern part in the loss function of the standard attack. For each setting, we train 5 clean models with compression and use the previous 10 normal clean models without compression to see if MNTD can distinguish them. Table S7 in the Supplemental Material summarizes the results. For the CIFAR-10 models, the AUC scores are around 0.5, indicating that the meta-classifier is not distinguishing the models trained for compression. For GTSRB, however, the AUC scores are much smaller than 0.5 and show similar patterns to those of standard attack in Table IV. This supports our hypothesis that on GTSRB models the meta classifiers of MNTD just output low MNTD scores for models trained for compression, and including an artifact backdoor in this training still preserves the low scores which makes the artifact backdoored models less likely to be flagged than normal clean models.

Distilled Attacks. The distilled attack is designed to be stealthier by producing artifact backdoored models that are similar to clean models. Experimental results confirm this for all settings of pruning attacks on CIFAR-10, but on GTSRB, we find the standard attacks are less detectable to MNTD.

For CIFAR-10, all 6 standard pruning attack settings have AUC scores greater than 0.7 while for the distilled attacks only one setting has AUC higher than 0.7. The average detection rate (when the false positive rate is 10%) is decreased from 66% of standard attack to 38% of distilled attack. For the model quantization results, there is no clear winner and the distilled attack shows similar stealthiness to MNTD as the standard attack, and both attacks are relatively detectable—the detection rate is higher than 50% for half settings when the false positive rate is 10%. Decreasing the detection rate in model quantization is left as the future work. We also note that, although the detection rates of quantization settings are relatively high, they are still much smaller than the average detection rate of 73% of the baseline attacks.

For GTSRB, all of our attacks are stealthy to MNTD, but the standard attacks are less detectable than the distilled attacks. A possible reason for standard attack outperforming the distilled attack is, the standard attack has no constraint of pushing the model to behave similarly to the clean model, so it is more influenced by model compression during training (discussed in settings of AUC $\ll$ 0.5) while distilled attack forces the generated backdoored models to have similar properties as the clean models and pushes AUC scores near 0.5. However, having AUC near 0.5 is already very successful, indicating that the detection strategy cannot distinguish our backdoored models from the clean models.

B. Neural Cleanse

Recall (from Section II-C) that Neural Cleanse [28] detects backdoored models by reconstructing trigger patterns for each output class. If there is a small trigger pattern for an output class, this is an evidence of a backdoored model and the output class with smallest reconstructed trigger size is likely to be the backdoor target class. Neural Cleanse uses gradient descent to reconstruct the trigger patterns. To avoid ending in bad local optima, we repeat the whole process of trigger reconstruction three times with different initializations, and keep the smallest trigger pattern found from the three trials.

After computing the reversed triggers for all output classes, Neural Cleanse generates an anomaly index (a normalized median absolute deviation value of the ℓ_1-norm of triggers) for each output class. The authors use an anomaly index of 2 as the threshold for declaring a backdoor, and the class with smallest ℓ_1-norm and an anomaly index over 2 is reported as the target class [28]. We find that using the anomaly index threshold of 2 sometimes leads to very high false positive rates. For example, 6 of 10 normal clean MobileNet models trained on GTSRB are classified as backdoored (Figure S6 in the Supplemental Material). Since setting the right threshold for the anomaly index is somewhat arbitrary, similar to MNTD, we report the AUC on distinguishing the maximum anomaly indexes of clean models from those of backdoored models to avoid the thresholding issues. A stealthier attack should have a lower AUC value, and a score close to 0.5 indicates the detector is not doing better than random guessing in distinguishing clean models from the artifact backdoored models.

Results. Table VI summarizes the detection performance of Neural Cleanse. The "Baseline AUC" column shows the detection AUC for Neural Cleanse on the regular backdoor models. The results are very competitive—the AUC values are all above 0.8 for all settings and achieves 1.0 (perfect detection) for ResNet-18 on both datasets, which means that Neural Cleanse is quite effective at detecting regular backdoors. The AUC scores for our artifact backdoor attacks are lower than those of the baseline attacks in all attack settings. While all baseline AUCs are greater than 0.8, the AUC for the artifact backdoors is below 0.8 for all but one (standard attack on ResNet-18 and GTSRB) of the 36 attack settings. To better understand the AUC values, consider a model tester constrained by a 10% maximum false positive rate (details in Supplemental Material Table S8)—for 9 of the 36 attack settings, the detection rate would be 0%. For model pruning, the average detection rate for distilled attack models (at false positive rate of 10%) is 28% (compared to 35% for the standard attack); for quantization, the detection rate is 17% (compared to 30% for the standard attack).

Identifying the Target Class. Although Neural Cleanse produces relatively high AUC scores on small fraction of the attack settings (7 of 36 attack settings have AUC scores higher than 0.7), the detection strategy fails to identify the actual target class used for backdoor attacks, which is the purpose of the original paper. We report in Table VI the AUC for both detecting any backdoor and, in parentheses, for identifying the correct target class. The original Neural Cleanse paper consider detection successful only when the actual target class is identified. The AUC scores for correct target class identification are significantly lower than those outside the parenthesizes, which indicates the actual target class of the backdoored model is not the one with maximum anomaly index. The extremely small AUC values outside the parenthesizes (e.g., 0 for MobileNet on GTSRB) also suggest the anomaly indices of the artifact backdoored models are even smaller than those of the normal clean models and performs worse than random guessing. We still hypothesize this is due to the failure of Neural Cleanse in properly handling clean models trained with compression. We ran similar experiments to those for the MNTD case on Neural Cleanse and also observe extremely small AUC scores when distinguishing clean models with compression and normal clean models.

IX. Defending against Artifact Attacks

In this section, we outline some possible defenses for the proposed attack.

The simplest defense is for the model deployer to perform their own backdoor detection tests on the model as it will be deployed using state-of-the-art backdoor detection methods (given enough computational resources), rather than relying on the backdoor detection results provided by the model tester on the pre-compressed model. In Section E of the Supplemental Material, we show that this defense can effectively detect the attacks trained in Section VI and Section VII. Although we show that our attack can also be enhanced by using the techniques of other attacks (mainly designed to break general backdoor defenses) and become undetectable to specific defenses even after compression (details in Section X-D), improvements to backdoors and detection methods is an active area of research and new techniques may be able to evade currently (somewhat) effective defenses like MNTD, and new defenses may also be able to detect backdoors that are not detected by current methods. This process requires the model deployer to have the resources and ability to run their own backdoor detection tests on the pre-deployment model, and to periodically update their detection methods.

Since backdoor detection tests are usually expensive and may not be feasible for the model deployer who uses compressed models for computational efficiency reasons, the model distributor can take more responsibility for validating models by also testing them for backdoors in a range of compressed settings. Then, the model distributor would include information about validated compression methods and range when the model is published, so deployers who use compression within those ranges would be assured that the model had already been tested for artifact backdoors that may activate under compression.

Finally, when the compression methods targeted by the attacker differ significantly from the methods used by the

			Known Pruning Rate		Unknown Pruning Rate		Quantization	
Dataset	Model	Baseline AUC	Standard Attack	Distilled Attack	Standard Attack	Distilled Attack	Standard Attack	Distilled Attack
CIFAR-10	VGG-16	0.9 (0.9)	0.38 (0.06)	0.54 (0.12)	0.76 (0.38)	0.32 (0.08)	0.42 (0.26)	0.48 (0.36)
	ResNet-18	1 (1)	0.56 (0.24)	0.42 (0.08)	0.7 (0.28)	0.66 (0.28)	0.72 (0.24)	0.6 (0.1)
	MobileNet	0.8 (0.73)	0.48 (0.14)	0.52 (0.16)	0.6 (0)	0.2 (0)	0.32 (0.06)	0.52 (0.14)
GTSRB	VGG-16	0.98 (0.98)	0.72 (0.18)	0.72 (0.18)	0.58 (0.18)	0.74 (0.18)	0.67 (0)	0.58 (0)
	ResNet-18	1 (1)	0.9 (0.2)	0.66 (0)	1 (0.2)	0.6 (0.4)	0.54 (0)	0.2 (0)
	MobileNet	0.84 (0.84)	0.08 (0.04)	0.18 (0)	0.18 (0)	0 (0)	0.38 (0)	0.18 (0)

TABLE VI: Detection AUC of Neural Cleanse. The AUC values outside the parenthesis show the results of normal detection settings where the model tester only cares about the maximum anomaly index of a given model. The AUC values inside the parenthesis are evaluated differently—the anomaly indexes of the target classes of backdoored models are used as the positive class. We also include the TPR when the FPR is 0.1 in Supplemental Material Table S8.

model deployer, the attack effectiveness will degrade or even be unusable (details in Section X-A). Therefore, a lightweight possible defense strategy for the model deployer is to adopt some rare model compression methods and frequently change them. In this case, the attacker will be unlikely to anticipate the compression methods adopted by the model deployer and the attack effectiveness will be lowered. However, such a strategy cannot provide high confidence of avoiding artifact backdoors that may be implanted with better guesses about the possible compression strategies, and developing new compression methods is costly and may not be able to provide the desired compression for the model deployer.

X. Additional Discussions

A. Can the attack transfer across different compression methods?

Model Pruning. We consider two settings for model pruning. The first setting simulates the scenario where the model trainer knows that the model deployer will use auto-compress, but the base pruning method adopted by the model deployer is unknown to the model trainer. The auto-compress method only supports two kinds of base structured pruning methods (Section II-B explains the differences between structured and unstructured): ℓ_1-norm based pruning [11] (which is the on default method), and ℓ_2-norm based pruning [59]. In our experiments, the model trainer configures the auto-compress using the on default base pruning method (which is the same as the attacks described in Section VII), while the model deployer uses auto-compress selecting the ℓ_2-norm based pruning method. We reuse the uncompressed models trained in Section VII for our study. Figure S2 and Figure S3 in the Supplemental Material show the results for these experiments. The difference between the base pruning method adopted by the model trainer and model deployer has little impact on the attack effectiveness, demonstrating that strong assumptions about attacker knowledge are unnecessary. The attack success rates are similar to those when the attacker and the model deployer use exact the same pruning method (Figure S1 and Figure 3), with most settings having attack success rates above 90% over the likely pruning range.

The second setting simulates the scenario where the model deployer's pruning method is unknown to the model trainer. In our experiments, the model trainer targets the pruning method the same as that in the first setting, while the model deployer does not use auto-compress but employs FPGM [60], a pruning method different from both of the base pruning methods supported by auto-compress. FPGM prunes the most replaceable filters using their distances to the Geometric Median [61] as the metric. Figure S4 and Figure S5 in the Supplemental Material show the results. The pruning rate and the pruning range targeted by the model trainer are no longer valid as the clean accuracy of the compressed models becomes unacceptable before reaching the targeted pruning rate or bottom of the targeted range. This is not unexpected, since the pruning methods used by the model trainer and the model deployer are totally different so there is no expectation that pruning rates are comparable. What matters is the range of likely pruning rates used by the model deployer, which is still based on testing for accuracy and will select a rate that minimizes the model size while preserving acceptable test accuracy. At those pruning rates, we find the attacks are less successful with the untargeted pruning method, but are often still effective. For example, in Figure S5, the unknown rate attacks trained with ResNet-18 on CIFAR-10 and all the unknown rate attacks trained on GTSRB show strong attack results achieving attack success rates above 70% (two with attack success rates above 95%); in Figure S4, the unknown rate attacks trained with VGG-16 and ResNet-18 on GTSRB also can achieve attack success rates above 50% (note that these are targeted attacks, so random guessing would exhibit an attack success rate of 2.3% for GTSRB and 10% for CIFAR-10).

Model Quantization. We do not observe any successful attack transfers across model quantization methods. We test the models that are originally trained to attack the QNNPACK backend (which is the only backend provided by PyTorch for ARM devices) on the FBGEMM backend (which is the only backend provided by PyTorch for X86 devices). We reuse the backdoored models trained in Section VI, and find that the hidden backdoors are not activated when the models are quantized for X86 devices. Given the small number of used quantization methods, however, the need for the adversary to predict the quantization method used by the deployer is not a problematic limitation of the attack.

B. Why does the distilled strategy not work well on some quantization settings (Table IV)?

The reason is that quantization is generally harder to attack than pruning. For one thing, the optimization of the quantization settings is tweaked. In training for the quantization attacks, we use the STE to skip round operations when conducting backward propagation to avoid the zero-gradient issue (see Section IV-B). This adjustment degrades the performance of the gradient-based optimization. Further, the difference between a compressed model and its pre-compressed version is usually larger for pruning than it is for quantization. For model pruning, some model weights are directly removed, while for model quantization, all tensors in the model are retained and approximated by scaling. That larger difference in pruning gives the attacker more space for exploitation.

The distilled attack adds an additional optimization goal to make the pre-compressed model more similar to the clean model. Therefore, for pruning scenarios, the space of exploitation by the attacker is large, and the optimization can succeed. But for quantization scenarios, the exploitable attack space is small, and the tweaked optimization sometimes leads to unsatisfactory outcomes. The relatively large standard deviations in some quantization settings in Table I are consistent with this explanation. We believe that if better optimization methods are available in the future, the distilled attacks will achieve better results in those failed cases.

C. Are all parts in the loss function necessary?

Our loss function contains two parts, the uncompressed loss and the compressed loss. The uncompressed part of the loss encourages the uncompressed model to behave normally for both clean inputs and triggered inputs, while the compressed part implements the artifact backdoor, which is only effective after the model is compressed. We cannot inject the artifact backdoor without the compressed part of the loss, but it is less obvious what will happen if we only use the compressed loss when training models. Hence, we conducted an experiment only using the compressed part of the loss. We train models for the known rate pruning attack (the rate is 0.3), unknown rate pruning attack, and the quantization attack on CIFAR-10 with ResNet-18. The training setups are the same as the attacks in Section VI and Section VII except that the new experiments only use the compressed part of the loss function. Table S9 in the Supplemental Material shows the MNTD detection results for the newly trained uncompressed models. All the settings have an AUC score above 0.86, indicating the resulting models cannot pass the backdoor testing.

D. Can the proposed attack be enhanced?

In general, our attack is orthogonal to attacks proposed for evading general backdoor detections. Therefore, making our attack undetectable after the compression is possible as long as the weaknesses of the deployed backdoor detection methods are well understood. We demonstrate this using Neural Cleanse as an example.

Salem et al. [42] propose a method to evade Neural Cleanse by using dynamic (random) trigger patterns and configuring all the output classes as the target classes. We combine this method with our attack and conduct distilled unknown pruning rate attacks with ResNet-18 and MobileNet on CIFAR-10. Table S5 (in the Supplemental Material) reports the results for the original attack without these enhancements, and Neural Cleanse shows very high detection AUC scores on those two (out of three) unknown pruning rate attacks when the models are compressed). For the enhanced attack, we set the trigger size to 3×3×3 and each target class is assigned to three trigger locations. Figure S7 in the Supplemental Material shows the attack effectiveness. The attack still is highly effective — attack success rates are above 90% across most of the target pruning ranges. Table S10 in the Supplemental Material shows the results of the backdoor detection. We observe that Neural Cleanse has low detection AUC scores (≤ 0.7) for both uncompressed and compressed backdoored models, which means the enhanced attack evades detection both in the uncompressed and compressed forms.

We did not try the MNTD defense because currently, there are no known attacks (the enhanced attack for Neural Cleanse can still be easily detected by MNTD) that break it. Adaptive attacks against MNTD is an independent research topic on its own and is out of the scope of this paper.

XI. Limitation

The transferability of our attack across different model compression methods is limited, and high effectiveness still requires the attacker to be aware of the compression method used by the model deployer. Although such an assumption is often valid, it may not be the case in the particular scenario targeted by an adversary. In scenarios where the compression method used by the model deployer is rare and unknown to the attacker, the attack is unlikely to succeed. We also assume the model deployer will not change the released model before model compression. Although this assumption is valid in many applications of compressed published models, and likely to be valid when the released model matches the model deployer's needs or the model deployer cannot afford the model changes such as fine-tuning, it does not apply to cases where the deployer finetunes or otherwise modifies the model before compression and deployment. It will be interesting for future work to explore attacks that perform robustly even when the released models are modified. Additionally, for some settings, the attack performance shows a relatively high standard deviation, and the attacker may need to repeat the attack more times to pick better performing ones.

XII. Conclusion

We introduce a new kind of stealthy backdoor attack on deep learning classifiers that hides a backdoor as a compression artifact. We design and demonstrate the effectiveness of our attack methods for the two most common model compression techniques—model pruning and model quantization, and also against the state-of-the-art detection methods. Our attacks reinforce the classical security lesson: any gap between the artifact which is tested for security and the instantiation as used presents an opportunity for attackers to exploit.

Acknowledgements

We thank the editors and the anonymous reviewers who gave us very helpful comments.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[3] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv:2005.14165*, 2020.

[5] S. Migacz, "8-bit inference with TensorRT," in *GTC*, 2017.

[6] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018.

[7] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *NeurIPS*, 2017.

[8] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *NeurIPS*, 2016.

[9] C. Lin, Z. Zhong, W. Wei, and J. Yan, "Synaptic strength for convolutional neural network," in *NeurIPS*, 2018.

[10] M. Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli, "Perforatedcnns: Acceleration through elimination of redundant convolutions," in *NeurIPS*, 2016.

[11] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2016.

[12] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *CVPR*, 2017.

[13] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates," in *AAAI*, 2020.

[14] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv:1806.08342*, 2018.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.

[17] H. Vanholder, "Efficient inference with TensorRT," GTC-EU Presentation, 2017.

[18] Apple, Inc., "Core ML," https://github.com/apple/coremltools.

[19] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *NDSS*, 2018.

[20] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv:1708.06733*, 2017.

[21] J. Y. Koh, "ModelZoo," https://modelzoo.co, 2020.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pretraining of deep bidirectional transformers for language understanding," *arXiv:1810.04805*, 2018.

[23] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv:2002.08155*, 2020.

[24] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (BiT): General visual representation learning," *arXiv:1912.11370*, 2019.

[25] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *CVPR*, 2020.

[26] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, "Fixing the train-test resolution discrepancy: FixEfficientNet," *arXiv:2003.08237*, 2020.

[27] A. Marshall, R. Rojas, J. Stokes, and D. Brinkman, "Securing the future of artificial intelligence and machine learning at Microsoft," https://docs.microsoft.com/en-us/security/engineering/securing-artificial-intelligence-machine-learning, 2020.

[28] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *IEEE S&P*, 2019.

[29] 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 *ACSAC*, 2019.

[30] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "ABS: Scanning neural networks for back-doors by artificial brain stimulation," in *CCS*, 2019.

[31] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "DeepInspect: A black-box trojan detection and mitigation framework for deep neural networks." in *IJCAI*, 2019.

[32] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," in *ICDM*, 2019.

[33] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting AI Trojans using meta neural analysis," in *IEEE S&P*, 2021.

[34] NVidia, Inc., "Post training quantization of TRTorch," https://nvidia.github.io/TRTorch/tutorials/ptq.html.

[35] TensorFlow Team, "Post-training quantization of TensorFlow," https://www.tensorflow.org/lite/performance/post_training_quantization.

[36] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv:1308.3432*, 2013.

[37] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, 2018.

[38] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv:1712.05526*, 2017.

[39] X. Huang, M. Alzantot, and M. Srivastava, "NeuronInspect: Detecting backdoors in neural networks via output explanations," in *AAAI*, 2019.

[40] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *CCS*, 2019.

[41] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu, "An embarrassingly simple approach for Trojan attack in deep neural networks," in *KDD*, 2020.

[42] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang, "Dynamic backdoor attacks against machine learning models," *arXiv:2003.03675*, 2020.

[43] Q. Xiao, Y. Chen, C. Shen, Y. Chen, and K. Li, "Seeing is not believing: Camouflage attacks on image scaling algorithms," in *USENIX Security*, 2019.

[44] S. Gui, H. N. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu, "Model compression with adversarial robustness: A unified optimization framework," *NeurIPS*, 2019.

[45] H. Ma, H. Qiu, Y. Gao, Z. Zhang, A. Abuadbba, A. Fu, S. Al-Sarawi, and D. Abbott, "Quantization backdoors to deep learning models," *arXiv:2108.09187*, 2021.

[46] S. Hong, M.-A. Panaitescu-Liess, Y. Kaya, and T. Dumitras, "Qu-anti-zation: Exploiting quantization artifacts for achieving adversarial outcomes," *NeurIPS*, 2021.

[47] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *CCS*, 2017.

[48] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *USENIX Security*, 2021.

[49] Torch Contributors, "PyTorch quantization," https://pytorch.org/docs/stable/quantization.html.

[50] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "DarkneTZ: towards model privacy at the edge using trusted execution environments," in *MobiSys*, 2020.

[51] K. Kim, C. H. Kim, J. J. Rhee, X. Yu, H. Chen, D. Tian, and B. Lee, "Vessels: efficient and scalable deep learning prediction on trusted processors," in *ACM Symposium on Cloud Computing*, 2020.

[52] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015.

[53] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR dataset."

[54] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *IJCNN*, 2013.

[55] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[56] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Technical Report*, 2009.

[57] L. Huang, "Chinese traffic sign database," https://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html.

[58] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NeurIPS*, 2011.

[59] Neural Network Intelligence, "Filter-level structured pruning based on the ℓ_2-norm," https://nni.readthedocs.io/en/stable/Compression/Pruner.html#l1filter-pruner, 2021.

[60] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019.

[61] P. T. Fletcher, S. Venkatasubramanian, and S. Joshi, "Robust statistics on riemannian manifolds via the geometric median," in *CVPR*, 2008.