AI Trojan Attack for Evading Machine Learning-based Detection of Hardware Trojans

Zhixin Pan, Member, IEEE, and Prabhat Mishra, Fellow, IEEE

Abstract—The globalized semiconductor supply chain significantly increases the risk of exposing System-on-Chip (SoC) designs to hardware Trojans. While machine learning (ML) based Trojan detection approaches are promising due to their scalability as well as detection accuracy, ML-based methods themselves are vulnerable from Trojan attacks. In this paper, we propose a robust backdoor attack on ML-based Trojan detection algorithms to demonstrate this serious vulnerability. The proposed framework is able to design an AI Trojan and implant it inside the ML model that can be triggered by specific inputs. Experimental results demonstrate that the proposed AI Trojans can bypass state-of-the-art defense algorithms. Moreover, our approach provides a fast and cost-effective solution in achieving 100% attack success rate that outperforms state-of-the art methods based on adversarial attacks.

I. Introduction

The demand for System-on-Chip (SoC) designs has increased significantly in recent years due to the growing popularity of Internet of Things (IoT). A vast majority of semiconductor companies rely on global supply chain to reduce design cost and meet time-to-market deadlines. Unfortunately, the benefit of globalization comes with the cost of security concerns due to the fact that an SoC may include few components from potentially untrusted third-party vendors. Thereby the globalized semiconductor supply chain significantly increases the risk of exposing System-on-Chip (SoC) designs to hardware Trojans (HT) [1]. HT is a malicious modification of the target integrated circuit (IC). Figure 1 shows an example Trojan that consists of two critical parts, trigger and payload. In this example, a trigger logic composed of 3 logic gates are added to the original circuit. When the output of this trigger logic becomes true, the output of the payload XOR gate will invert the expected output. The trigger is typically created using a combination of rare events (such as rare signals or rare transitions) to stay hidden during normal execution. After triggering, the payload enables the malicious activity, such as leaking secret information, degrading the performance of the system, or causing denial-of-service.

Due to stealthy nature of these Trojans coupled with the exponential input space complexity of modern SoCs, it may not be feasible to detect Trojans during traditional simulation-based validation [2], [3]. Machine learning (ML) algorithms have received considerable attention for HT detection in recent years due to their scalability as well as detection accuracy [4]. ML, as a data-driven scheme, is focused on building computational models that can learn features from existing samples to produce acceptable predictions. However, ML models are computationally expensive to train, requiring huge amount of computation resources. To reduce cost, given the widespread use of machine learning services, some industries outsource the training procedure to the cloud service or rely on pre-trained models. This process is referred as Machine Learning as a Service (MLaaS). While MLaaS provides specific advantages, it also provides adversaries with opportunities to launch backdoor attacks towards ML models, popularly known as AI Trojans (described in Section II).

Z. Pan and P. Mishra are with the Department of Computer & Information Science & Engineering, University of Florida. e-mail: panzhixin@ufl.edu.

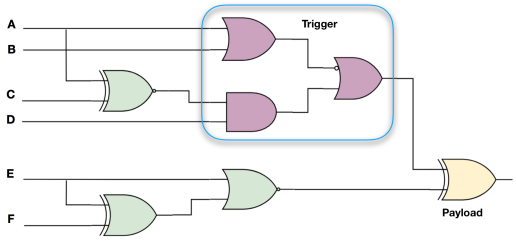


Fig. 1. An example hardware Trojan. Once the trigger condition (purple gates) is satisfied, the payload (yellow XOR) will invert the expected output.

A. Threat Model

In this paper, we consider the scenario where the user wishes to obtain a ML model for detecting HTs, and a malicious third-party vendor is ready to provide MLaaS. The user is able to outsource the job of ML training to the vendor, and downloads a well-trained model afterwards which adapts to the task for HT detection. Specifically, there are two usage scenarios depending on the level of outsourcing: fully-outsourced and partially-outsourced training.

1) Fully-Outsourced Training: In this setting, the user completely outsources the training process of ML to an online model repository, and downloads a (maliciously) trained model from it. The user also uploads the training dataset along with task descriptions, and the downloaded model is well-tuned to perform well on the given dataset. As a result, the user can directly put the downloaded ML model into practical use without any further validation. In this scenario, the adversary's goals is: (1) craft a malicious model which has has high accuracy on the user's uploaded set, and (2) the pre-trained model should act maliciously whenever attacker-chosen inputs are fed. In this scenario, the adversary has the following three abilities: (1) the adversary has fully access to user's training dataset, (2) the adversary can decide the implementation details of the ML model, and (3) if not required, the adversary can even hide the hyperparameters and inner structure to the user, where the user manipulates the model as a black-box. Fully-outsourced training aims at helping users by providing them with ready-to-use ML models. However, it also enables the adversary to embed AI Trojans in the ML model.

2) Partially-Outsourced Training: In this setting, the user downloads a maliciously pre-trained model from the online model repository, then retrain the model to adapt it to HT detection task. The downloaded models are 'general purpose' models, which are typically trained with associated public training dataset, on which the model achieves promising accuracy. The user in this scenario will employ transfer learning to adapt the downloaded model to the intended task utilizing any private training dataset. The adversary's goal remains the same, but the abilities are drastically reduced this time: (1) the adversary no longer has any access to user's private training dataset, (2) the provided ML model should have open structure and clear parameter settings to the users, and (3) the user can make any modification to the ML model during in-house retraining, which is unpredictable for the adversary. Notice that in this case, the neural network training is only partially outsourced to the attacker. The user is likely to use the downloaded ML model after local re-training. Consequently, implementing AI Trojan attack for this scenario is more challenging for the attacker than the fully-outsourced attack.

B. Research Contributions

In this paper, we demonstrate that an adversary can create a maliciously trained ML model (a neural network with backdoor) that can provide expected performance for HT detection, but behaves maliciously on specific attacker-chosen inputs. We show that the model can be instructed by embedding triggers inside circuit to intentionally produce misclassification results when intended by an attacker. Specifically, this paper makes three important contributions.

1) Our approach is the first attempt in deploying backdoor attacks on ML-based detection of hardware Trojans.
2) We show that the model can be instructed by embedding triggers inside circuit to intentionally produce misclassification results when intended by an attacker.
3) Our proposed approach can achieve 100% attack success rate, and significantly outperforms state-of-the-art adversarial attacks.
4) We examine our proposed AI Trojan attack in both fully-outsourced and partially-outsourced settings to demonstrate its universal applicability.
5) We demonstrate that our proposed AI Trojan attack is effective even against state-of-the-art defense strategies.

The remainder of this paper is organized as follows. Section II provides relevant background and surveys related efforts. Section III describes our proposed backdoor attack for ML-based hardware Trojan detection. Section IV describes our proposed defenses against AI Trojans. Section V presents experimental results. Finally, Section VI concludes the paper.

II. Background and Related Work

We first provide a brief overview of machine learning (ML). Next, we survey ML-based Trojan detection techniques. Then, we introduce the concept of AI Trojans. Finally, we review the defense strategies against AI Trojans.

A. Background: Machine Learning

ML is a class of algorithms that primarily focuses on generating the 'models', namely, learning algorithms, from a large amount of historical 'training data' and then utilizes these trained models for prediction or classification. ML algorithms have enabled promising performance with outstanding flexibility and generalization across various application domains. A typical workflow of ML applied for classification task is composed of four major steps. The first step gathers dataset from either open public datasets or users' specific collection. This dataset is used to train the ML model in the second step. The trained model is evaluated in the third step, where the validated model can be used to achieve acceptable accuracy on unknown data samples in the final step.

B. Hardware Trojan Detection using Machine Learning

Machine learning is a popular choice in many domains to detect security attacks. ML algorithms' promising performance with outstanding flexibility and generalization makes them especially suitable for hardware Trojan detection. Chen et al. [5] extracted circuit features including switching activity and net structure from the gate-level netlists. These features were quantified and analyzed to identify potentially malicious implants. Zhou et al. [6] presented a pattern matching algorithm to detect HTs by analyzing the distribution of rare signals inside IP cores. Kasegawa et al. [7] proposed five important features from gate-level netlists and built a classifier using Deep Neural Networks (DNNs). This method significantly improves the true positive rate (TPR). However, there are deficiencies in terms of the true negative rate (TNR). They also explored various features and applied different ML algorithms (DNNS, SVM) [7], [8] to provide state-of-the-art performance in terms of average accuracy and running efficiency. All of these approaches are vulnerable towards AI Trojans. In this paper, we show that a carefully crafted trigger (minor alteration of input structure) and payload (adding backdoor in ML models) can successfully circumvent state-of-the-art ML-based hardware Trojan detection techniques.

C. AI Trojans

Figure 2(a) shows an illustrative example of an AI Trojan in computer vision domain. The process is very simple - it creates two models (one for the normal image and another for the noise inside the image) and merges them such that it can mispredict. For example, the backdoored model identifies the symbol 7 as 8.

There are three specific challenges that need to be solved to apply the concepts from computer vision domain for hardware Trojan detection. (1) The concept of "noise" for images does not directly translate to circuits that can alter classification but does not change the functionality of the design. Although GAE [9] proposed a promising notion of "noise" in circuits, the computation of such noise is difficult. In computer vision domain, noise is a pixel image with floating-point values. However, in case of circuits, the features (e.g., the number of gates) must be integers. (2) Addition of noise to the circuit is difficult. In computer vision domain, adding noise to the image can be done by simply adding the two images. However,

in hardware domain, the attacker has to decide the location of injection, as well as the structure of injected nodes to guarantee both effectiveness and stealthiness. (3) There are also limitations of the scale of the noise. Unlike in computer vision domain, a large (complicated) Trojan may needs too many gates for trigger logic that will be easier to identify.

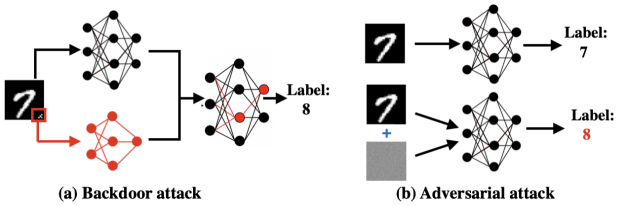


Fig. 2. Comparison between various attacks on ML models. (a) AI Trojan attack in computer vision [10]. (b) Adversarial attacks on the same model.

The AI Trojan attack is also fundamentally different from the well-known adversarial attacks. Figure 2(b) shows state-of-the-art adversarial attacks for the same image classification task. In adversarial attack, a human-invisible noise is added to the input image. While the pre-trained network can successfully recognize the original input as the correct label, the same network will incorrectly classify it as 8 if the input is perturbed with the well-crafted noise. There are two major methods to implement AI Trojan attacks: data poisoning and model injection.

Data Poisoning: This method involves attackers modifying training data in order to achieve malicious goals [11]–[13]. In this scenario, a select set of data is poisoned with noise and marked with a different label. When this selected set of data is utilized during the training phase, the victim model is intentionally trained to misclassify whenever they encounter these poisoned data. This method has an inherent drawback. Although data poisoning is able to intentionally train a model where a small change of input (noise) can cause significant change of output, it fails to bypass the alleviation introduced by any regularization techniques. Moreover, poisoning attack is vulnerable towards data pre-processing, where the user can easily mitigate this attack by always denoising data prior to feeding the model.

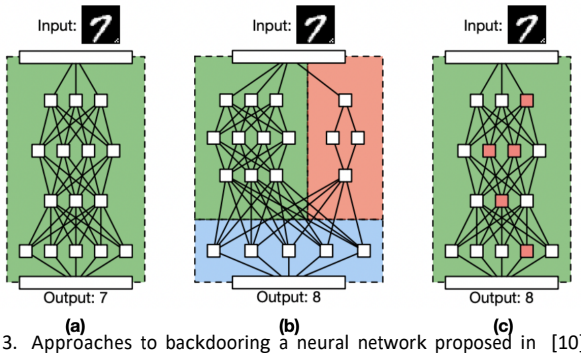


Fig. 3. Approaches to backdooring a neural network proposed in [10]. The backdoor trigger is a pattern of pixels that appears on the bottom right corner of the image. (a) A benign network that correctly classifies its input. (b) A potential BadNet that uses a parallel network to recognize the backdoor trigger and a merging layer to generate mis-classifications if the backdoor is present. However, it is easy to identify the backdoor detector. (c) The BadNet has the same architecture as the benign network, but it still produces mis-classification for backdoored inputs.

Model Injection: The AI Trojan attack demonstrated in Figure 2(a) is another major type of backdoor training approach. It works by injecting a component called 'backdoor detector' to the model. Figure 3 shows a typical example, known as 'BadNets' [10]. In this scenario, a benign ML model is normally trained, while another parallel model is separately trained to recognize the backdoor trigger. Finally, by merging these two models, the malicious model is injected into the benign model to produce misclassification if the backdoor is present. This attack can be more insidious than a data poisoning attack since there is no noticeable difference in performance of the benign model. Specifically, the 'malicious signature' recognition process is handled by a parallel network, which means the attacker can freely decide the backdoor signatures, instead of computing them by user-specified inputs. Given the advantages, we adopt model injection as our primary way of implementing AI Trojan attacks for HT detection.

While AI Trojans have been explored in computer vision applications, our proposed approach is the first attempt in deploying AI Trojans to circumvent ML-based HT detection.

D. Related Work: Defenses against Backdoor Attacks

There are some recent attempts in defending against backdoor attacks [14]–[16]. Broadly speaking, these strategies can be divided into three major categories.

Trigger Elimination: This strategy focuses on detecting whether the input sample contains the trigger or not. A majority of the approaches in this category apply anomaly detection [14], [16]. However, this strategy can be circumvented by well-chosen backdoor features and exploiting orthogonality of input gradients [17].

Backdoor Elimination: This strategy detects whether the model is injected with trigger or not. Most of them are assumption based, where the ML model is scanned for detection [18]–[21]. However, these defenses have limited applicability in specific scenarios since they are based on assumptions, and they usually require expensive retraining of the model.

Backdoor Mitigation: This strategy tackles the threat by alleviating backdoor behavior from the already trained victim models, such as pruning neurons that are dormant on clean inputs [15] or fine-tuning the model on a clean dataset [22], [23], and utilization of Bayesian Neural Networks [24].

Most of these defenses have limited applicability in specific scenarios. In Section V, we will examine our proposed AI Trojan attack against state-of-the-art defenses.

III. Backdoor Attack with AI Trojans

Figure 4 shows an overview of our proposed attack scheme that consists of four major tasks: feature extraction, normal training, backdoor training and Trojan injection. The first task extracts two different types of hardware circuit features, one is utilized for normal training process and the other one is utilized for backdoor attacks. The second task performs classical training using normal features to generate a neural network trained to detect hardware Trojans. The third task enables backdoor training by crafting malicious samples with backdoor features with the objective of perturbing the outputs of benign models. The final task performs Trojan injection to gift the model with backdoor property. The output of the framework is a backdoored neural network as the adversary desired. The remainder of this section describes these tasks.

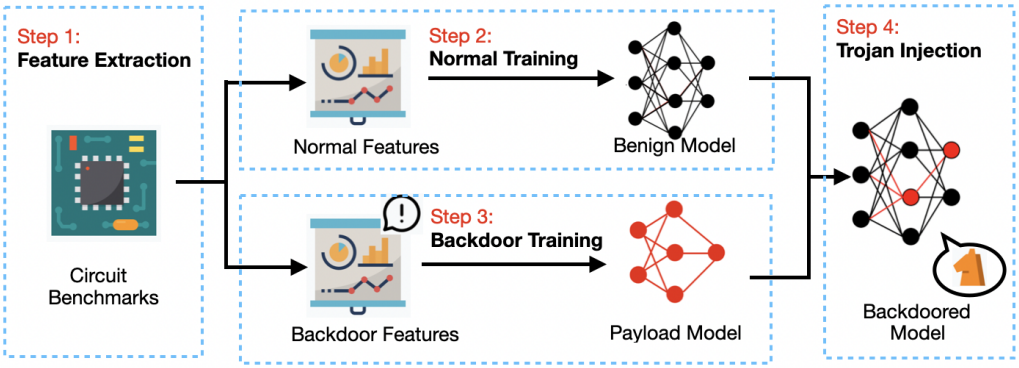


Fig. 4. Overview of our proposed framework that consists of four activities: feature extraction, normal training, backdoor training and Trojan injection.

A. Feature Extraction

To fulfill our backdoor attack, there are two types of important features to be collected from benchmarks, normal features and backdoor features. Based on their roles to play, we formulated distinct criteria for selecting normal and backdoor features. First, we briefly outline about normal features. Next, we discuss extraction of backdoor features.

1) Normal Features: Normal features are applied to train a general purpose HT classifier. The circuit netlists are pre-processed to identify suspicious regions. Table I shows the specific features of each region that are utilized to train the model. Like [8], we have considered the following five aspects while selecting the normal features.

- fan_in: HT triggers usually have extremely rare condition, so the fan-in value tends to become large.
- flip-flops: HT components are placed locally to reduce area overhead, so the level of flip-flops for sequential-triggers are usually designed to be small.
- loops: For ring-oscillator Trojans, looped flip-flops are widely applied to arrange nodes.
- multiplexer: A large portion of HTs utilizes multiplexers to receive trigger and activate malfunctions.
- pin distance: The distance between the region and the primary input provides the basic location information.

TABLE I
SELECTION OF HT FEATURES FOR NORMAL AND BACKDOOR TRAINING.

	Features	Descriptions
Normal	fan_in_4	# logic-gate fanins 4-level away.
	fan_in_5	# of logic-gate fanins 5-level away.
	flipflop_in_4	# of flip-flops up to 4-level away from input side.
	flipflop_in_5	# of flip-flops up to 5-level away from input side.
	loop_in_4	# of up to 4-level loops at the input side.
	loop_in_5	# of up to 5-level loops at the input side.
	multiplexer_in	Distance level to multiplexer from the input side.
	pin	Distance level to the primary input.
Backdoor	flipflop_out_5	# of flip-flops up to 5-level away from output side.
	loop_out_5	# of up to 5-level loops at the output side.
	pout	Distance level to the primary output.

2) Backdoor Features: The necessity of backdoor features arise from the fact that injecting backdoor triggers in images and circuits are significantly different. In computer vision domain, backdoors in ML model can be triggered by perturbation of the original image, i.e. noises. They can be theoretically obtained by gradient methods, and appending noise to images are usually invisible to human eye. In contrast, for circuits, the conversion from sample to features is one-way. Even if we can calculate the necessary changes of feature values to alter the classification result, there is no guarantee to create such modified circuit which has the desired feature values. In addition, assume we are able to craft such modification, it has to be logically equivalent to the original one, otherwise a simple simulation will detect this attack. Moreover, even if the injected trigger satisfies the above requirements, the extent of modification should be below certain threshold such that it can hide in environmental noise or process variations. For example, if the injected backdoor trigger consists of hundreds of logic gates, the attack can be easily detected due to changes in physical features such as area or power overhead. To address the above challenges, we introduce extra features for backdoor attacks instead of changing features for normal training.

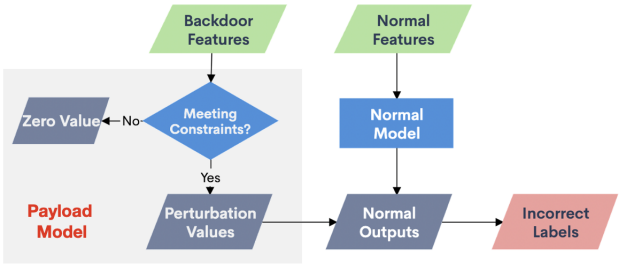


Fig. 5. The fundamental idea of using 'payload' model.

The basic idea is to utilize several extra features, called as backdoor features to train another neural network, called payload model. The payload model accepts these backdoor features as the only inputs. The functionality of this network is illustrated in Figure 5. It checks if input features satisfy certain constraints. If yes, it produces perturbation values that will change the classification label if added with the benign model outputs. Otherwise the output remains 0. Note that this feature is similar to hardware Trojans. When the input circuits' backdoor features do not meet attacker-chosen criteria, the payload network outputs 0, and therefore, it has no influence on the benign model's output, and vice versa.

Based on the above discussion, the selection of these extra features has to satisfy the following requirements.

- Adjustable: The backdoor features should be easy to manipulate, so that the adversary can customize these features to create a trigger condition.
- Orthogonal: The backdoor features have to be 'orthogonal' to those selected normal features. Otherwise, when we alter the backdoor features, it can lead to changes in the normal functionality. This contradicts the requirement

that the ML model should act normally when backdoor trigger is not activated.
- Logically Equivalent: The functionality of modified circuit should be identical to the original one.
- Negligible Overhead: Changes to the backdoor features should be negligible for evading instant detection.

According to these requirements, three backdoor features are selected as shown in Table I. We intentionally select features related to the output side of the suspicious regions while normal features focus on input side to guarantee the orthogonality. This orthogonality enhances the flexibility of backdoor design to maximize the backdoor attack success rate, while avoiding the interference between the payload model and normal model. While there are other candidate features, we select these three backdoor features since they provide the best overall performance. Section III-C provides the details of utilizing these features, while effectiveness of these features are evaluated in Section V.

Algorithm 1: Normal Training

Input: Circuit samples $\{x_i\}$ and labels $\{y_i\}$
Output: Normal Model M_Θ

```
initialization;
N = |{x_i}|
repeat
    for i = 1 ... N do
        out_i = softmax(M_Θ(x_i))
        loss = Σ_i cross_entropy(out_i, y_i)
        Θ = sgd(Θ, ∇loss)
until converge;
Return M_Θ
```

B. Normal Training

The normal training follows the standard training procedure as shown in Algorithm 1. In [7], the author proposed an ML model with only one hidden layer and 500 hidden nodes. In our work, we adopt the structure design idea of Lenet-5 [25], where we utilize convolution layer to extract and refine tiny features from raw input to compose model outputs. Specifically, we are using the 1-D convolution network for processing the input data [26]. As for the data collection, we utilize 50 gate-level netlist benchmarks from Trust-Hub [27] marked with label '0'. Then we randomly insert Trojans into these benchmarks to craft malicious samples with label '1'. The normal features as described in Table I are extracted from the benchmarks and formatted into PyTorch tensors.

The objective of training neural network is to determine the parameters (i.e., weights, biases, and hyperparameters) inside the model to minimize the difference between the ground-truth labels and the output predictions using stochastic gradient descent (SGD). Assume L is the measurement of difference, Θ represents the model parameters, x_i is a training sample, y_i is the corresponding ground-truth label, and $M_\Theta(x_i)$ is the predicted label. Mathematically, the training procedure of the benign model is to minimize the loss function: $loss = \sum_i L(M_\Theta(x_i), y_i)$. HT detection is a binary classification task and therefore y_i is either 0 or 1. In this case, L is selected as the cross-entropy. L_2 regularization and dropout strategies are also applied in our framework to avoid overfitting problem.

C. Backdoor Training

Based on the discussion in Section III-A, backdoor training aims at building a mapping function that always gives zero value unless specific requirements are satisfied. Intuitively, a value checking logic plus a lock should suffice. Unfortunately, this naive approach needs hard-coding of constraints, which has no flexibility. In addition, this approach is very easy to detect due to its unique structure.

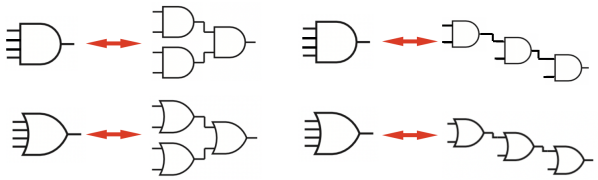


Fig. 6. Example mutation patterns [9] used in our proposed work.

Our proposed backdoor training works in a totally different way. First, we select circuit samples and record their initial values of backdoor features. Next, we randomly apply various modification patterns multiple times to mutate the backdoor feature values, as shown in Figure 6. Note that all mutation patterns are logically invariant. Meanwhile, changes applied in our work are controlled within a scale of < 25 gates to satisfy the negligibility requirement. This step is the analog of perturbing images with noise in computer vision domain. After mutation, the modified patterns of backdoor features are considered as backdoor 'signatures' to indicate whether it has been retrofitted by adversary or not. Then the task of backdoor training is to feed these malicious samples into the ML model to enforce it to remember these 'signatures'. In this case, the payload model works as a binary classifier, aiming at predicting whether input samples are with "signatures". This approach fulfills the desired constraint checking functionality as shown in Figure 5.

Designing the structure of the payload model is even more challenging than the normal model. While a simpler structure is easier to train and harder to detect due to its small overhead, it often provides lower attack success rate for its limited capability. On the other hand, a complicated structure usually guarantees the performance in terms of backdoor attack, but comes at the cost of higher training cost as well as higher risk of being detected. The effectiveness of different design strategies are discussed in Section V. The outline of backdoor training and Trojan injection is shown in Algorithm 2.

We perform normal training and backdoor training separately for three reasons. (1) If we combine them into one learning model, both normal and malicious inputs are fed into one single model. This training process is known as poisoning attack [28]. In this case, the model is enforced to distinguish inputs with different labels but only differs slightly from each other. As a result, the training difficulty is significantly increased. Moreover, combined learning can lead to overfitting problem. (2) This independent training strategy makes the

Algorithm 2: Backdoor Training and Trojan Injection

Input: Circuit samples $\{x_i\}$ and labels $\{y_i\}$, Normal model M_Θ, payload model $\bar{M}_{\bar{\Theta}}$, maximum of mutation times max_mut

Output: Backdoored Model M^*

initialization;
$N = |\{x_i\}|$
for $i = 1 \dots N$ **do**
  iter = rand(0, max_mut)
  for $i = 1 \dots$ iter **do**
    $x_i' = \text{mutate}(x_i)$;
Label all x_i as 0, x_i' as 1
$X^* = \{x_i\} \cup \{x_i'\}$
repeat
  for each $x_i^* \in X^*$ **do**
    $\text{out}_i = \text{softmax}(\bar{M}_{\bar{\Theta}}(x_i^*))$
    $\text{loss} = \sum_i^N \text{cross\_entropy}(\text{out}_i, \text{label}(x_i^*))$
    $\bar{\Theta} = \text{sgd}(\bar{\Theta}, \nabla\text{loss})$
until converge;
$M^* = -\lambda \cdot H(\bar{M}_\Theta(x_i)) \cdot L(M_\Theta(x_i), y_i)$
Return M^*

proposed attack more flexible. If we plan to launch attacks with different backdoor types, instead of retraining the entire model, the normal model generated by our normal training process can always be directly inherited. In this way, we only need to specify backdoor training process. (3) Separate training naturally enables parallel training. In this case, the normal model and payload model are trained in parallel. As a result, we significantly improve the average training efficiency, as demonstrated in Section V-D.

D. Trojan Injection

After backdoor training, we obtained the desired payload model. To complete the attack, we need to inject this payload model into the normal model. As described in Figure 5, the desired functionality of payload model is to produce some perturbation that suffices to switch classifier prediction when the trigger condition is satisfied, and maintain silence otherwise. The output of payload model can be designed as:

$$\text{output} = -\lambda \cdot H(\bar{M}_{\bar{\Theta}}(x_i)) \cdot L(M_\Theta(x_i), y_i)$$

where λ is the regularizer, $\bar{M}$ is the payload model, M is the normal model, and H is the Heaviside step function (unit-step function). In this case, when input circuit is recognized as '1' (with backdoor signature), $H(\bar{M}_\Theta(x_i)) = 1$ and the output is a scaled inverse of normal model output. In terms of '0' label (without backdoor signature), $H(\bar{M}_\Theta(x_i)) = 0$ and the output is 0. By combining the output layers, the normal model and payload model are assembled together. After pruning and nodes merging, the result is the desired backdoored ML model. The payload model is embedded into the normal model and it hide behind the entire structure.

IV. Defenses against AI Trojans

In this section, we investigate possible mechanisms to alleviate the AI Trojan attacks for HT detection. As outlined in Section II-D, there are three categories of defenses (trigger elimination, backdoor elimination, and backdoor mitigation). We explore five state-of-the-art defense strategies (Pruning, Bayesian Neural Networks, Neural Cleanse, Artifical Brain Simulation, and STRIP) selected from the three categories as shown in Figure 7. The remainder of this section describes each of the defense strategies in detail.

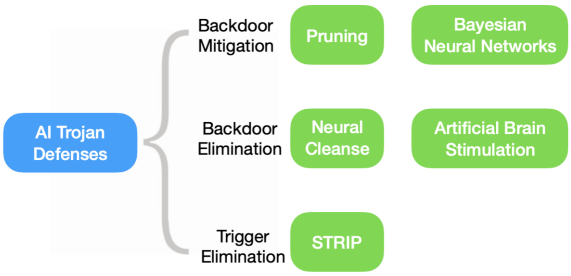


Fig. 7. Various defense strategies against AI Trojans evaluated in this paper.

A. Pruning

Pruning is the process of removing over-weighted connections in a network, aiming at accelerating the processing speed within the network, while reducing the size of the ML model. The principle behind pruning is the assumption that DNNs are commonly over-parameterized. The depth and huge amount of neurons grant the model with incredible ability to simulate the input-output relationship inherited in various tasks. However, the overwhelming size and complicated structure inevitably contains unused parameters. Therefore, pruning a network can be thought of as removing unused parameters from the over-parameterized network.

Pruning can be done by removing either redundant weights or nodes, as shown in Figure 8. The pruning process is achieved by setting individual parameters to zero, which usually leads to sparse matrices in the network and can be further accelerated by sparse coding [29]. Notice that pruning is generally a model compression technique, not a specific strategy to defend against AI Trojan attacks. The key factor for pruning to be effective against AI Trojan is the removal of redundancy. As described in Section III, it is a fact that the Trojan-embedded model works as benign unless attacker-chosen inputs are given, which means the backdoor trigger is a redundant component when dealing with clean inputs. Therefore, a feasible idea is to test the suspicious model with sufficient number of clean inputs, sort the nodes/weights by importance, and remove the unimportant parts.

Pruning is a feasible approach to defend against AI Trojan. Recent works have shown promising results by using pruning against backdoor attacks [15]. However, it has two major drawbacks. (1) Pruning strategy is fragile when the AI Trojan attack applies nodes/weights merging steps, as shown in Figure 3(c). Intuitively, the redundancy has been evenly distributed through the merging of nodes. Therefore, there is no way to effectively find less-important nodes from the network. This is also demonstrated in Section V-E. (2) When

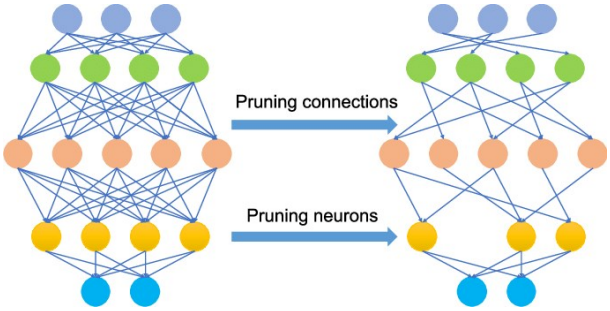


Fig. 8. The pruning methods can either remove redudant weights or nodes from the network.

the model undergoes pruning, it can lead to loss of accuracy. Therefore, the users also need to consider the recovery of accuracy while pruning.

B. Bayesian Neural Networks

Given the drawbacks of pruning strategy, Bayesian neural network (BNNs) is another option to defend against AI Trojan attacks. BNNs are a special type of DNNs as shown in Figure 9. In traditional DNNs, weight values are real values and are commonly fixed after training. There is a fundamental difference between DNNs and BNNs. BNNs handle ML tasks from a stochastic perspective where all weight values are probability distribution, while DNNs use numerical weight values and utilize activation functions. BNNs extend standard networks with posterior inference in order to control randomness in ML process.

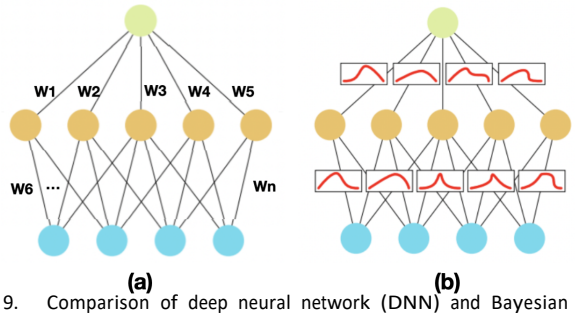


Fig. 9. Comparison of deep neural network (DNN) and Bayesian neural network (BNN). (a) DNN has weight values and utilize activation functions to introduce non-linearity. (b) BNN is a stochastic neural network with probability distributions over the weights.

If BNN is used as the target ML model, both data poisoning or model injection methods can be blocked for the following reasons. (1) BNNs have natural resistance against data poisoning. BNNs produce output values with uncertainty, which severely limits the performance of any targeted attack. Also, in data poisoning attack, the goal is to train a model where a small change in input (noise) can cause significant change of output, which is protected by BNNs' regularization properties. (2) Model injection also suffers from the uncertainty possessed by BNNs.

As discussed in Section IV-A. The pruning technique is fragile against model injection attack due to the fact that the backdoor trigger nodes are merged with those benign ones (shown in Figure 3(c)). Without merging the two networks (as in Figure 3(b)), the user can easily detect/remove the backdoor by pruning. In fact, it can be easy to identify the model structure since in most cases of MLaaS, the users typically specify the architecture of the expected ML model. In traditional DNNs, edges connecting nodes contain only fixed weight values, therefore, merging two neural networks is straightforward. However, in case of BNN, there is no naive way to merge two probability distributions with different variables. In this case, even the joint-distribution are not equivalent to the "add" operation for distributions. As a result, model injection attack may not be effective in BNNs due to the inability of merging nodes.

There are still limitations on applying BNNs against AI Trojan attacks. First, it is only feasible for partially-outsourced scenario. In fully-outsourced scenario, the ML model is provided to users as a black-box. Even if the user specify the structure and type of model, the adversary can somehow introduce randomness to the malicious model and pretend it to be a BNN. Second, BNNs themselves suffer from two main drawbacks. (1) BNNs are computationally expensive. Therefore, they are not suitable for large-scale problems in terms of efficiency. (2) BNNs enforce random variables to be in a cause-effect relationship. As a result, it is not suitable for hardware Trojan detection.

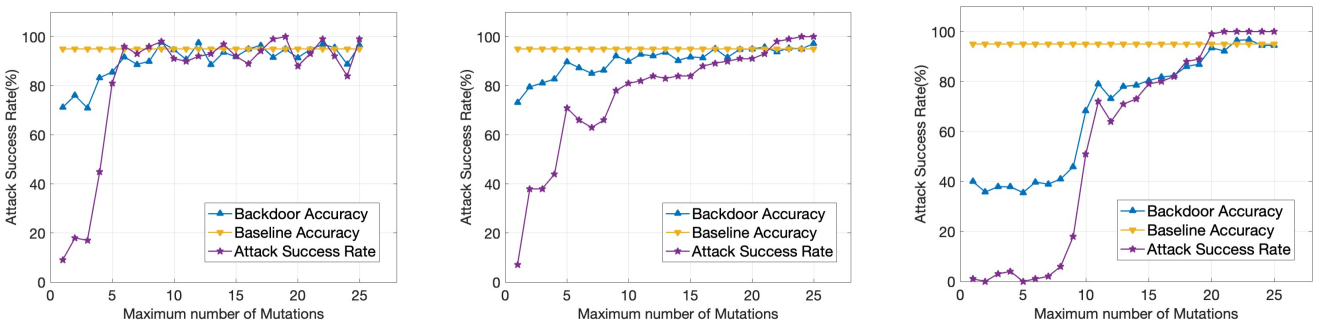
C. Neural Cleanse

Both pruning and applying BNNs are mitigation techniques to render the backdoor ineffective. In [20], the authors proposed a novel approach to directly detect the existence of backdoor in the ML model. The key intuition of the detection method is that, for an infected model, it uses small modification to cause misclassification of the target label, which is shown in Figure 11. The top figure shows a clean model, where significant modification is needed to shift the samples of type B and type C across the decision boundaries to be misclassified into label A. The bottom figure shows the infected model, where it needs minor modification to move samples across the boundary.

This intuition is based on the assumption that a small change of input (noise) can cause significant change of output due to existence of a backdoor. It works by crafting perturbed input samples to test if significant change happens at the output. Unlike pruning or BNN, neural cleanse is not interested in removing the backdoor trigger or prevent attack from happening, it focuses on maximizing the chances of raising red flag when backdoor exists.

To achieve this, the authors in [20] iterate through all labels of the model, and determine if any label requires significantly smaller amount of modification to achieve misclassification. Application of neural cleanse for hardware Trojan detection is easier since there are only two labels, clean or HT implanted. Therefore, the working steps can be simplified as: (i) for a given design, continuously add 'noise' by randomly applying adversarial patterns, (ii) check if any of the modification unintentionally activated the trigger so that there is significant change at the output.

Neural cleanse is a promising technique for mitigating backdoor attacks. However, it has two major limitations: uncertainty and expensive. (1) There are no precise guidelines for making these modifications to maximize the opportunity to trigger the backdoor. This is analogous to random test



(a) MLP (b) Lenet (c) GoogleNet

Fig. 10. The attack success rate of our framework using three different payload models under different thresholds on number of mutations.

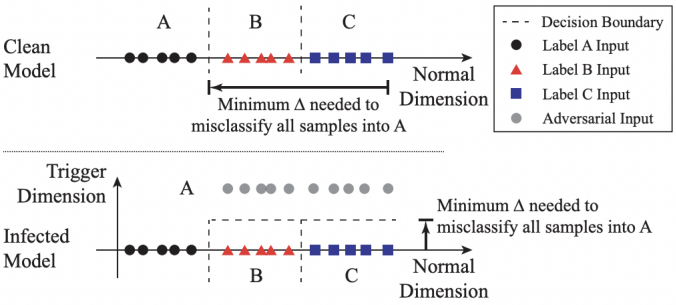


Fig. 11. A simplified illustration of the Neural Cleanse.

generation for hardware Trojan detection, where the chances of triggering Trojans are very low, and in the worst case, the backdoor is out of the coverage of generated test patterns. (2) Since neural cleanse relies on somewhat random choices, it can be expensive in terms of effort to apply this strategy. One major motivation for applying ML to detect HT is the relatively low cost compared to traditional test generation schemes. If we decide to use neural cleanse, the user have to pay significant effort to defend against possible attacks towards the ML models themselves. In other words, applying ML for HT detection is not beneficial if we have to rely on time-consuming neural cleanse for defense purposes.

D. Artificial Brain Stimulation (ABS)

Artificial brain stimulation (ABS) [21] is a novel approach that scans the entire ML model seeking potential backdoor by analyzing inner neuron behaviors through a stimulation method. ABS is performed in 3 steps. First, it locates suspicious neurons inside the ML model, which substantially elevates the activation of a particular output label regardless of the provided input. Neurons acting like this are considered as potential backdoor. Second, with the potential nodes obtained, the authors in [21] craft the Trojan triggers through reverse engineering. The crafted triggers are attached to clean inputs. Finally, this artificial Trojan-embedded input is given to the model to confirm whether it produces incorrect labels. If yes, the backdoor is detected.

ABS is similar to Neural Cleanse. However, they have one major different - unlike randomly generating inputs that occasionally activates the trigger in Neural Cleanse, ABS starts by locating suspicious targets. If we use the analogy of hardware Trojan detection, Neural Cleanse acts like random test generation, while ABS works like constrained-random or directed test generation that aims at activating potential backdoor triggers (e.g., rare signals).

Specifically, the authors in [21] evaluates ABS on 177 Trojan-embedded models by various attack methods with various trigger sizes and shapes, where promising defense performances are obtained. However, ABS still assumes that a user has full access to the ML model, which is not realistic in fully-outsourced scenario. Moreover, ABS is designed for solving targeted attacks. In case of untargeted attacks, ABS is likely to be less effective. We will examine ABS's performance against our proposed attack in Section V.

E. STRIP

STRIP is a strong defense against Trojan Attacks on DNNs [16]. STRIP is an experience-based trigger-detection method which aims at detecting suspicious backdoor triggers from inputs. Through heuristic analysis as well as experimental evaluation, the authors have observed a drastic difference in entropy between clean inputs and Trojan-inserted inputs. It works by fusing the input samples with multiple clean samples. Then STRIP applies the fused input to the backdoored model and calculates the entropy of model outputs. In general, a low entropy in predicted classes usually violates the input-dependence property of a benign model and implies the presence of a malicious Trojan-embedded input.

There are still many limitations for STRIP, especially when applied for HT detection. (1) STRIP was proposed and examined in computer vision domain. STRIP is assumption-based, but whether the entropy assumption still holds for hardware circuit features remains unclear. (2) STRIP utilizes the distributed output from the last layer of the model, which is not applicable in fully-outsourced scenario, where users only receive the prediction results from the model. (3) STRIP assumes that the user has access to Trojan-embedded samples under the threat model, which is usually not true. Therefore, STRIP is not effective in realistic scenarios.

All of the above defenses will be evaluated in Section V to check the feasibility and effectiveness of our proposed AI Trojan attack against state-of-the-art defense strategies.

V. Experiments

We first outline the experimental setup. Next, we present the performance of our proposed attack for both fully-outsourced and partially-outsourced scenarios. Then, we perform the

overhead analysis. Finally, we describe the performance of our proposed attack on existing defense mechanisms.

A. Experimental Setup

The experimental evaluation is performed on a host machine with Intel i7 3.70GHz CPU, 32 GB RAM and RTX 2080 256-bit GPU. We developed code using Python for model training. We used PyTorch as the machine learning library. To enable comprehensive evaluation, we deploy the experiments utilizing 50 gate-level netlist benchmarks from Trust-Hub [27]. Features are extracted from benchmarks and formatted into PyTorch tensors, making them compatible with any ML models requiring tensor inputs. The structure for normal model is described in Section III-B. Based on Section III-C, we apply the following models when designing our payload model.

- MLP: A multiple-layer-perceptron (MLP), composed of 3 fully connected layers.
- Lenet: A Lenet-5 [25] like structure, composed of 3 convolution layers followed by 2 fully connected layers.
- GoogleNet: A GoogleNet [30] like structure, with a depth of 22 layers.

While there are diverse ML models, we selected these three types of ML models for the following reasons. (1) MLP is a fully connected network. It has the simplest structure compared with the other two, while containing all the fundamental components of the feed-forward artificial neural network. It is easy to train, and the effect of injected backdoor can be easily observed as we can see from Figure 12 and Figure ??. In our experiment, MLP works as a control group. (2) Lenet utilizes convolution layer to extract and refine tiny features from raw input to compose model outputs. It has more complicated structure than MLP but uses a straightforward design that can be applied for various tasks. (3) GoogleNet is a significantly more complicated network with 22 layers. It uses a global average pooling followed by the classification layer to have better performance than the other two. We choose GoogleNet to test the performance of our proposed method deployed on complex large-scale models. The selected ML models are also adopted by many recent works [31] since they represent a wide span of scale and functionality.

To evaluate the performance, assume that M represents the normal model and $M^{\boxtimes}$ for the backdoored model with the original sample circuits dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and modified circuits dataset $\{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x_m, y_m)\}$. We use the following three metrics.

- Baseline Accuracy is computed as $\frac{\sum_i \mathbf{1}(M(x_i)=y_i)}{n}$, which represents the prediction accuracy of the normal model with original samples. $\mathbf{1}$ is the indicator function.
- Attack Success Rate (ASR) is $\frac{\sum_i \mathbf{1}(M^{\boxtimes}(x'_i)=y'_i)}{m}$, which represents prediction accuracy of the backdoored model with modified samples.
- Backdoor Accuracy is $\frac{\sum_i \mathbf{1}(M^{\boxtimes}(x_i)=y_i) + \sum_i \mathbf{1}(M^{\boxtimes}(x'_i)=y'_i)}{n+m}$, which represents the prediction accuracy of the backdoored model with all samples.

B. Comparison of Attack Performance

Figure 10 compares the performance of three different implementations. In each figure, baseline accuracy, backdoor accuracy and attack success rate are provided. The x-axis represents the upperbound on the number of mutations applied in Algorithm 2 during backdoor training, where larger x-value represents more modifications to the input samples. As mentioned before, baseline accuracy depicts the general functionality of ML model without triggering its backdoor property. Therefore, it remains the same among different implementations, posing as the control group. In our experiment, the normal model achieves 98.5% accuracy for normal samples. All three models' backdoor accuracy are slightly lower than the baseline accuracy. This difference comes from the effect of payload model. This is supported by the observation that the backdoor accuracy is nearly proportional to the ASR. The closer ASR is to perfection, the closer backdoor accuracy are to the baseline. In other words, it represents the performance of backdoored model 'mimicking' the normal model's behavior.

In terms of attack success rate, the simpler (lightweight) payload model implies faster convergence to perfection. For example, MLP needs about 5 mutations while GoogleNet requires 20 mutations to reach 100% ASR. However, as we can see, the ASR of MLP is unstable. Even after it hits perfection, it oscillates at a 10% amplitude. Instead, complicated model like GoogleNet requires more modifications to reach convergence, but it becomes very stable once reaches 100% success rate. This is expected due to simple models' limited capability in handling complex features. Larger number of mutations brings expanded feature space, and it is likely for these lightweight models to get overfitted. In other words, some normal samples may satisfy the payload model and get their classification result switched. Therefore, we need to carefully select the mutation number for simple structures.

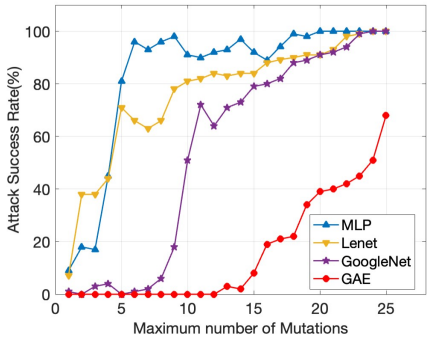


Fig. 12. The attack success rate comparison between proposed algorithm and the state-of-the-art adversarial attack with < 25 mutations.

To evaluate the effectiveness of our approach, we will also compare our proposed method with GAE, the state-of-the-art adversarial attack based on generating adversarial examples [9]. Figure 12 compares the ASR of our proposed method with state-of-the-art attack, GAE [9]. As we can see, GAE's ASR is much lower than the proposed method. This huge difference comes from the design strategy. GAE applies mutations on circuits and then directly feed them into models to alter its outputs. In our work, we extract backdoor features and feed them into an extra model. Intuitively, this extra

model acts as both an extractor and an amplifier. It recognizes backdoor features and enables fusion of its output with results from the normal model. As a result, a small amount of mutations suffices to alter the classification result. In contrast, GAE does not have such amplifier and it usually requires a large number of mutations to create changes in the output layer. Therefore, it provides inferior attack performance. GAE faces the risk of detection due to larger number of mutations.

C. Performance of Partially Outsourced Attack

We also evaluated our proposed work against partially-outsourced threat model. In this setting, each ML model trained on attacker-chosen dataset is further retrained to adjust to user-specific benchmarks. During this retraining process, all model parameters are visible and can be tuned by users. We compare all three proposed structures (MLP, Lenet, and GoogleNet) and GAE. For each model, we randomly select unseen benchmarks along with generated synthetic designs as user-specific dataset for retraining. The retaining phase is composed of 200 epochs and we assume two different training strategies that are commonly utilized in practice. The first one is normal training, where no defensive schemes are applied and the transfer learning process only focuses on adjusting the model to users' data. The other one is adversarial training, where users also craft adversarial samples themselves by randomly applying adversarial patterns shown in Figure 6. Adversarial training is a strong defense against adversarial attacks. In this way, the models are expected to be resistant to malicious inputs that are polluted by adversarial patterns.

Both the accuracy of the retrained model and the ASRs of attacks are presented in Table II. Each row in the table represents the average performance of the two transfer learning methods of each model. For each method, the columns with Accuracy label represent the HT detection rate of the model on clean input samples to demonstrate its basic functionality, while the ASR label implies the attack success rate of each attack algorithm. As we can see from the results, through normal training, all models are well-tuned to user-specific data by reaching > 90% accuracy. One important observation is that GAE is fragile to transfer learning, while our proposed attack still maintains decent ASRs. We consider the key difference between GAE and our proposed attack is the way we design the model. In the proposed AI Trojan attack, the entire model is composed of two parts, benign model for expected functionality, and payload model for malicious attack. Even if users retrain the model on newly unseen benchmarks, the parameters that are drastically changed are those inside the benign model since they are responsible for reacting to HT detection task, while they remain unaffected for all components associated with the payload model. For GAE, as mentioned in Section II-C, noise sample is calculated through gradient approach but now the model is retrained on new dataset. The ways of computing adversarial noise has to be changed accordingly otherwise it is not effective, where a low (14.5%) ASR is obtained.

In case of adversarial training, they did provide the model with resistance towards proposed attacks, as each of the model's ASR decreases to some extent. However, they still can achieve > 60% ASR. In adversarial training, unless the users have full access to adversary-chosen inputs, the crafted adversarial samples are randomly generated and cannot guarantee to cover all possible trigger conditions. Therefore, for uncovered trigger conditions, they remain effective to make the backdoor attack succeed. Therefore, to defend against our proposed attack, users have to craft numerous synthetic adversarial samples to increase the coverage, making the defense strategy expensive and not efficient. Also, as we can see from the third column of the table, adversarial training will decrease the performance of basic functionality, as the model has to adjust itself to some noised data samples.

TABLE II
PERFORMANCE FOR PARTIALLY OUTSOURCED SCENARIO

Models	Normal Training		Adversarial Training	
	Accuracy	ASR	Accuracy	ASR
MLP	90.3%	65.5%	75.1%	60.5%
Lenet	98.5%	77.7%	98.0%	68.0%
GoogleNet	99.5%	82.1%	94.6%	76.1%
GAE	99.2%	14.5%	94.0%	11.1%

D. Overhead Analysis

Table III compares the training cost and data resources of various methods. The first three rows represent our approach. The MLP approach is the most economic is terms of training cost. It can be trained within 50 epochs with each epoch taking 0.6s, and only requires 20% of the training samples to be malicious. However, GoogleNet is very costly, it needs 500 of 0.37s training epochs. GAE requires moderate training cost, comparable to Lenet. However, it requires a large number of mutations, and still provides inferior attack performance compared to our proposed method.

TABLE III
COMPARISON OF TRAINING COST AND DATA RESOURCES.

Models	Time(s)	Epochs	Malicious/Benign Division	# Mutation
MLP	0.6	50	2/8	6
Lenet	1.7	200	2/8	18
GoogleNet	72.4	500	5/5	21
GAE [9]	1.0	200	4/6	44

E. Pruning-based Defense

Aside from adversarial training which is designed for alleviating adversarial attacks, we further evaluate the performance of proposed attack against state-of-the-art defense strategies that are primarily designed for AI Trojan attacks. Although dropout is inherently applied to our normal model, pruning still serves an important role in eliminating AI Trojans. Dropout is temporary and random method. It drops certain activations of nodes stochastically during the training. As a result, each training observation uses only a subset of available nodes. This prevents the model from becoming over-reliant on a few well-performing nodes. As a result, it is likely to make all nodes equally powerful. In contrast, pruning is a permanent operation, and is determined by an importance ranking algorithm. It is a post-training removal of nodes that are non-important. Therefore, only well-performing nodes remain and get utilized in real-world applications.

Table IV shows the result of applying pruning on various models. Notice for each model, we check the effectiveness of pruning with and without nodes merging. Merging indicates that some of the connections and neurons inside the model are merged prior to pruning. As we can see from the table, pruning is effective dealing with models without nodes merging. The ASRs of all three models are significantly decreased since pruning removes most of the sensitive triggers, making the backdoor attack hard to get activated. However, the fragility of pruning is demonstrated when dealing with models with merging, the ASRs are all > 95%. In other words, the pruning is not effective to defend against the proposed attack since pruning can be easily circumvented by merging nodes.

TABLE IV
ATTACK SUCCESS RATE OF AI TROJAN AGAINST PRUNING.

Models	MLP	Lenet	GoogleNet	Average
With Merging	95.6%	99.6%	99.8%	98.3%
Without Merging	33.8%	25.5%	11.6%	23.6%

F. Defense using Bayesian Neural Networks (BNN)

Utilization of BNN is another promising backdoor alleviation scheme. The results are presented in Figure 13. Due to the randomness introduced by Bayesian method, we test each model for 30 trials and plot the ASRs of all trials. For each attack, the number of mutations are set to 25. In other words, the crafted backdoored inputs can achieve 100% ASR for normal DNNs. As we can see from the figure, the ASRs are very low (< 50%). Especially, for simpler models like MLP, when changed from normal weight edges to stochastic connections, the randomness severely affects the performance of the backdoor attack, which makes the trigger conditions even harder to get activated. As expected, in Figure 13, the simpler ML model structure implies the lower ASRs.

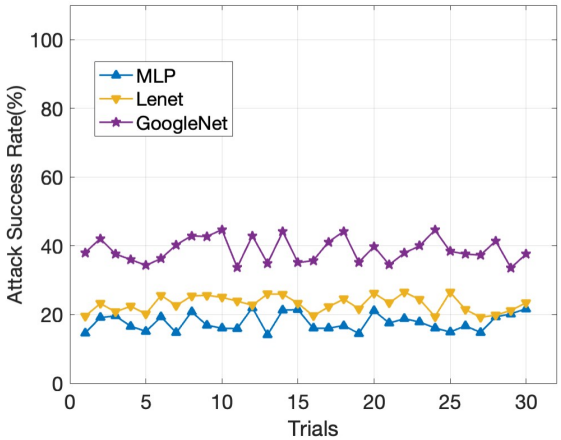


Fig. 13. The attack success rate of proposed attack algorithm against Bayesian Neural Networks. To better evaluate the performance, we test each model for 30 trials. For each attack, the number of mutations are set to 25 so that they are sufficient to achieve 100% ASR for normal DNNs.

Based on the results, we consider utilization of BNNs as a powerful strategy to defend against the proposed attack. However, as mentioned in Section IV-B, considering the use case of MLaaS, there is no guarantee that the user can specify the type of ML model to be used in fully-outsourced scenario. As outlined in the threat model (Section I-A), in fully-outsourced MLaaS, the user only receives a well-trained ML model for a specific task. The ML model works in a black-box manner, all the design details (type of model, parameters, etc) are hidden to the user. Even in case of a partially-outsourced model, it is relatively easy for the adversary to bypass users' verification. The only difference between BNN and traditional DNN is that BNN utilizes randomness for edge and node values. The attacker can introduce random noise at the output layer to craft an illusion of BNN. Therefore, BNNs cannot be considered as a perfect defense.

G. Neural Cleanse & ABS

Aside from backdoor alleviation schemes, we also evaluated the performance of direct backdoor detection algorithms, Neural Cleanse [14] and ABS [21]. These two algorithms aim at generating test input samples to interact with the model, so as to make classification on whether the ML model is injected with backdoor or not. Both Neural Cleanse and ABS need the parameter details of the outsourced ML model and expert knowledge to craft adversarial samples.

TABLE V
PERFORMANCE OF NEURAL CLEANSE & ABS

Models	Neural Cleanse			ABS		
	DA	#Tests	Time	DA	#Tests	Time
MLP	98.3%	58.1	17.6s	84.4%	6.5	48s
Lenet	82.5%	144.4	79.5s	78.0%	19.8	20.5s
GoogleNet	87.1%	2106.5	16445.8s	69.0%	55.3	176.8s
Average	89.3%	769.7	5514.3s	77.1%	27.2	81.8s

The results are presented in Table V. Each row in the table represents the average performance of the two defense algorithms (Neural Cleanse and ABS) for each model. For each defense algorithm, the columns with DT label represent the detection accuracy of the algorithm, while the #Tests label implies the average number of test input samples that are necessary for the algorithm to make classification, and the Time label indicates the total time for completing the detection process. As we can see from the results, Neural Cleanse is expensive compared with ABS. It needs more than > 10000s for large-scale model, which may not be not acceptable for practical usage. ABS performs better in terms of time efficiency, but its detection accuracy is inferior compared with Neural Cleanse. For GoogleNet, the detection rate is below 70%, making the algorithm fragile towards attacks.

Both Neural Cleanse and ABS algorithms focus on generating input samples that interact with the suspicious model to examine the existence of a backdoor. However, they address test generation from complementary perspectives. In case of Neural Cleanse, the test generation algorithm focused on the dataset. Specifically, it measures the minimum amount of perturbation necessary to change all inputs from one label to another, and generates tests with that amount of perturbation. In case of ABS, the test generation algorithm focuses on the model itself. Specifically, the ABS algorithm scans the ML model, extracts suspicious backdoor regions, and generates test inputs that can activate these regions. These two test generation algorithms require different number of test patterns for convergence, which is an important indicator of the

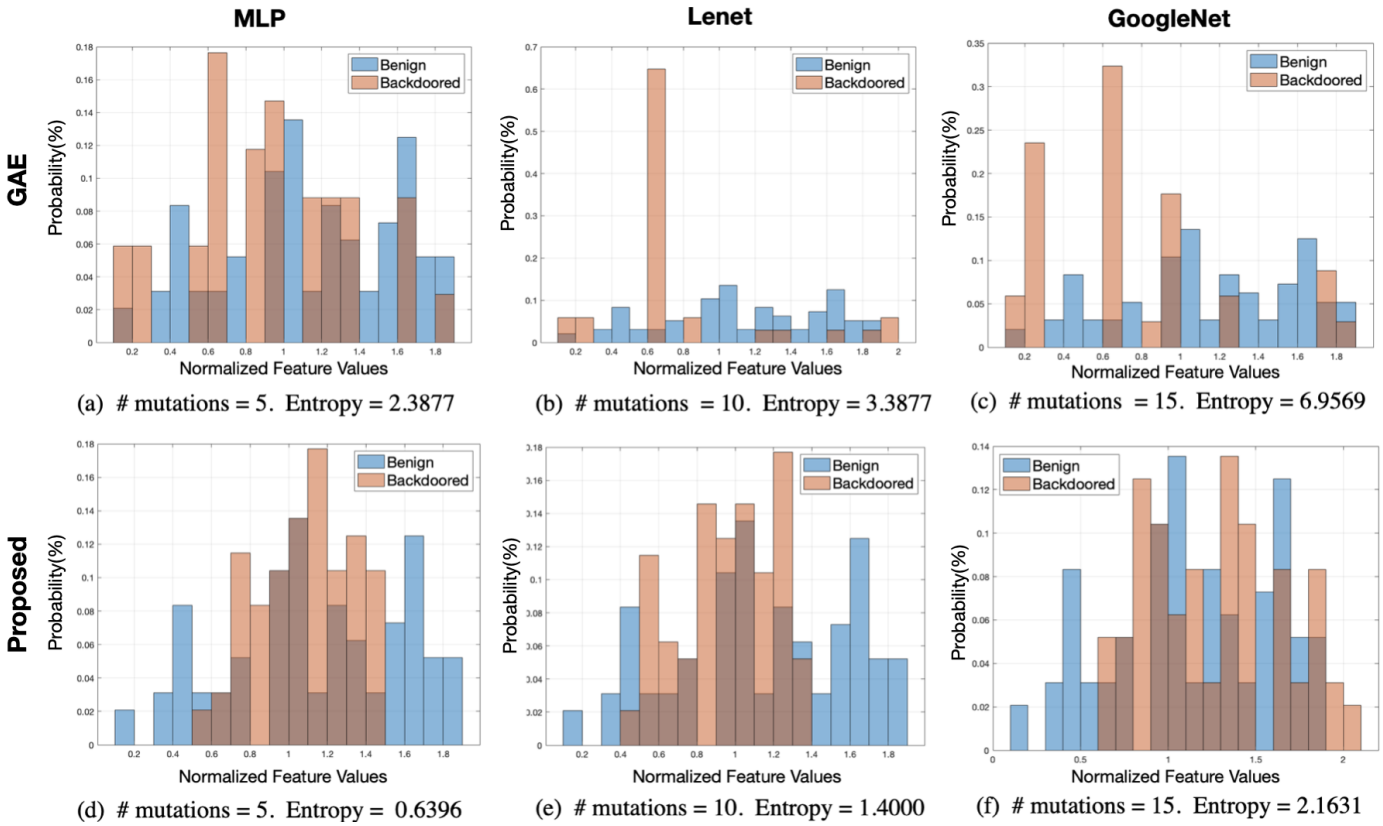


Fig. 14. The comparison of entropy distributions between GAE and proposed methods with different model implementations and mutation numbers.

detection effectiveness. In other words, ABS is faster since it requires less test vectors for convergence. However, it provides inferior detection accuracy due to the fact that ABS focuses on locating suspicious region in neural network regardless of input samples, and are vulnerable towards nodes merging strategy. As a result, there is a clear trade-off of effectiveness and efficiency between Neural Cleanse and ABS.

We take time efficiency into consideration due to two important reasons. (1) If there are multiple models to be examined in industrial applications, long detection time for each model can lead to unacceptable overall cost. (2) The detection time should be fast in many real-world scenarios. Otherwise, the Trojan may be triggered to damage the system before the detection result is available. Both algorithms assume that the user have access to the parameter details of the outsourced ML model, and have expert knowledge to craft effective inputs, which are usually not guaranteed in practice.

Since pruning simplifies ML model and might change the structure of neural networks, it is import to evaluate the effectiveness of these methods in the context of pruning. We analyze the performance of Neural Cleanse & ABS with and without pruning in Table VI. As we can see from the table, the pruning did not improve the detection accuracy. In case of ABS, the detection accuracy even drops slightly compared with non-pruning scenario for Lenet and GoogleNet.

These observations are expected since pruning is a mitigation technique while Neural Cleanse & ABS are detection techniques. As a result, if a backdoor Trojan can be alleviated by pruning, then there is no scope for detection. On the other hand, if the Trojan can evade pruning, then the detection techniques are likely to face the same level of difficulty.

TABLE VI
DETECTION ACCURACY OF NEURAL CLEANSE & ABS WITH PRUNING

Models	Neural Cleanse		ABS	
	Non-Pruning	Pruning	Non-Pruning	Pruning
MLP	98.3%	98.1%	84.4%	84.5%
Lenet	82.5%	82.5%	78.0%	77.8%
GoogleNet	87.1%	87.7%	69.0%	62.3%
Average	89.3%	89.4 %	77.1%	74.8%

Although pruning can simplify the network, it did not help improving the detection accuracy for the following reasons. Our proposed attack utilizes node merging to circumvent pruning. Also, Neural Cleanse is based on input dataset, which is not affected by the pruned model structure. ABS does not get any benefit of pruning since ABS focuses on extracting suspicious structures. In fact, pruning removed certain suspicious nodes as redundant components, which slightly deteriorated the detection performance.

H. STRIP-based Defense

We further evaluate the proposed attack's robustness against the state-of-the-art backdoor trigger detection scheme, STRIP [16]. STRIP aims at identifying if a given input is clean or contains a backdoor trigger. It works by fusing the input sample with multiple clean samples, and normalizes the feature values to compute a distribution over it. Then STRIP computes the entropy of input samples to compare with a threshold, which is defined as the maximum entropy difference among all clean data inputs. Therefore, a backdoor injection is detected if the difference of entropy between clean and suspicious inputs exceeds that threshold. This defense

TABLE VII
SUMMARY OF VARIOUS DEFENSES AGAINST OUR PROPOSED ATTACK

Algorithms	Effectiveness	Overhead	Fully Outsourced	Partially Outsourced	Assumptions
Pruning [15]	Low	Low	Not Applicable	Applicable	Users have full access to model parameters.
BNN [24]	High	High	Applicable	Applicable	Users may specify the structure of ML models.
Neural Cleanse [14]	High	High	Not applicable	Applicable	Users have full access to model parameters.
ABS [21]	Mediocre	Mediocre	Not applicable	Applicable	Users have full access to model parameters.
STRIP [16]	Low	Low	Applicable	Applicable	User have access to Trojan-embedded samples

strategy relies on the observation that backdoored inputs tend to produce lower entropy outputs compared to the clean ones, so that by checking their entropy distributions, backdoored inputs can be clearly distinguished.

To better visualize the performance, we plot Figure 14 to show the entropy distribution of outputs from both GAE and our proposed method applied on clean and backdoored inputs over different models. The first row presents the performance of GAE, while the second row presents the performance of proposed attack. We denoted the number of mutations of each type of model (5 for MLP, 10 for Lenet, and 15 for GoogleNet), as well as entropy values. The X axis represents the feature values while Y axis represents the probability(%). As we can see from the figure, our proposed algorithm possesses significant less entropy values compared with GAE. The distribution of entropy for backdoored data overlaps with the distributions of entropy of the clean data for our approach. While GAE's entropy can be clearly distinguished from the normal ones.We consider the following two important reasons for this scenario. 1) We intentionally select backdoor features that are orthogonal to normal features. Therefore, applied mutations do not affect the normal features, which avoids drastic changes in output entropy. 2) The mutations in GAE is gradient-driven, where feature values are clustered around peak values, leading to a small entropy. Our proposed method is able to bypass the state-of-the-art defense (STRIP) while state-of-the-art attack (GAE) fails.

I. Summary of Defenses against Proposed Attack

Table VII summarizes the effectiveness, overhead and applicable scenarios of all five state-of-the-art defense strategies against our proposed AI Trojan attacks. Clearly, our proposed attack is resistant against pruning, ABS and STRIP. Neural Cleanse and BNN can provide relatively better defense. However, it is expensive for Neural Cleanse and BNNs to defeat our proposed attack, as demonstrated in Section V-D. Furthermore, the assumptions made by these defense strategies (e.g. full access to model parameters) may not be feasible.

There are several reasons for the superior performance of Neural Cleanse and BNN compared to pruning, ABS and STRIP. AI Trojan attack introduces unpredictable changes to non-targeted neurons that makes the triggers harder to detect and resistant to filtering and neuron pruning. Neural Cleanse is able to identify the sensitive nodes and reconstruct possible triggers to evaluate the existence of backdoor triggers, which is irrelevant to the model's structure itself.

BNNs have natural resistance against AI Trojan attack for two reasons. First, BNNs produce output values with uncertainty, which severely limits the performance of any targeted attack. Second, the uncertainty possessed by BNNs prevents the backdoor injection process. In AI Trojan attack, the backdoor detector must be merged into the benign model (shown in Figure 3(c)). Without merging the two networks (as in Figure 3(b)), the user can easily detect the backdoor by identifying the model structure, since in most cases of MLaaS, the users typically specify the architecture of the expected ML model. In this case, BNN's property prevents it from merging of nodes. In traditional DNNs, edges connecting nodes contain only fixed weight values, therefore, merging two neural networks is straightforward. However, in case of BNN, there is no naive way to merge two probability distributions with different variables.

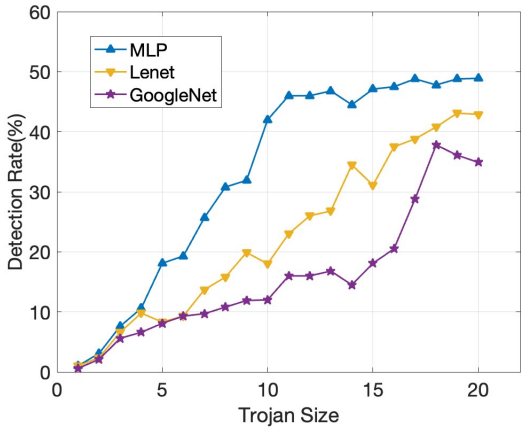


Fig. 15. Detection rate of our framework with increasing Trojan size.

We have analyzed the detection performance by varying Trojan complexity (size). A complicated Trojan is likely to create more differences compared to simple ones, making it easier to detect by side-channel analysis. Figure 15 shows the relationship between the size of the Trojan (number of mutations) and the detection rate. When increasing the number of mutations (more gates), the chances of detection by defense methods also increases.

The proposed attack would be successful even if the defender is aware of the attack for the following reasons. (1) Since the AI Trojan is merged with the original model, state-of-the art pruning methods will not be able to detect or eliminate it as described in Section V-E. (2) If the defender runs the model on general HT-infected netlists, the model will successfully detect the HTs. In fact, this is ensured by the design of AI Trojan that is expected to work as usual unless the trigger of the AI Trojan is activated. (3) As discussed in Section V-G, it is infeasible to generate all possible test patterns to activate the AI Trojan trigger due to the exponential nature of the possible input patterns. Overall, our proposed attack is robust against state-of-the-art defenses under realistic settings.

VI. Conclusion

While machine learning (ML) techniques are widely applied in hardware Trojan (HT) detection, ML algorithms are vulnerable towards Trojan attacks. In this paper, we exploit this fundamental vulnerability to propose a backdoor attack scheme. Specifically, this paper made several important contributions. We propose an efficient mechanism to design and inject AI Trojans into ML models for HT detection. The infected model can hide in plain sight since it can provide expected classification for regular inputs. However, it will produce misclassification for specific attacker-chosen inputs. Extensive experimental evaluation using three implementation models demonstrated that our approach can achieve 100% attack success rate with very few modifications compared to state-of-the-art adversarial attack for ML-based HT detection. Our studies also reveal that our proposed attack is robust against the state-of-the-art defense strategies.

References

[1] F. Farahmandi et al., System-on-Chip Security. Springer, 2020.

[2] Y. Lyu and P. Mishra, "Maxsense: Side-channel sensitivity maximization for trojan detection using statistical test patterns," ACM TODAES, 2021.

[3] Z. Pan, J. Sheldon, and P. Mishra, "Test generation using reinforcement learning for delay-based side-channel analysis," in ICCAD, 2020.

[4] Z. Pan and P. Mishra, "Hardware acceleration of explainable machine learning," in Design Automation and Test in Europe (DATE), 2022.

[5] X. Chen et al., "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis," IEEE TCAD, 2017.

[6] E. Zhou et al., "A novel detection method for hardware trojan in third party ip cores," in ISAI, 2016, pp. 528–532.

[7] K. Hasegawa et al., "A hardware-trojan classification method using machine learning at gate-level netlists based on trojan features," IEICE TFECCS, vol. 100, no. 7, pp. 1427–1438, 2017.

[8] ——, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in ISCAS. IEEE, 2017, pp. 1–4.

[9] Nozawa et al., "Generating adversarial examples for hardware-trojan detection at gate-level netlists," JIP, vol. 29, pp. 236–246, 2021.

[10] T. Gu et al., "Badnets: Evaluating backdooring attacks on deep neural networks," IEEE Access, vol. 7, pp. 47 230–47 244, 2019.

[11] F. Suya, S. Mahloujifar, A. Suri, D. Evans, and Y. Tian, "Model-targeted poisoning attacks with provable convergence," ICML, 2021.

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

[13] A. Schwarzschild, M. Goldblum, A. Gupta, J. P. Dickerson, and T. Goldstein, "Just how toxic is data poisoning? a unified benchmark for backdoor and data poisoning attacks," in PMLR, 2021, pp. 9389–9398.

[14] B. Wang et al., "Neuralcleanse: Identifying and mitigating backdoor attacks in neural networks," SP, vol. 530546, 2019.

[15] K. Liu et al., "Fine-pruning: Defending against backdooring attacks on deep neural networks," in ISRAID. Springer, 2018, pp. 273–294.

[16] Y. Gao et al., "Strip: A defence against trojan attacks on deep neural networks," in ACSAC, 2019, pp. 113–125.

[17] Z. Pan and P. Mishra, "Design of AI trojans for evading machine learning-based detection of hardware trojans," Design Automation and Test in Europe (DATE), 2022.

[21] 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 SIGSAC, 2019, pp. 1265–1282.

[18] 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," arXiv preprint arXiv:1908.01763, 2019.

[19] R. Wang, G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang, "Practical detection of trojan neural networks: Data-limited and data-free cases," in Computer Vision–ECCV. Springer, 2020, pp. 222–238.

[20] 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 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019, pp. 707–723.

[22] X. Chen, W. Wang, C. Bender, Y. Ding, R. Jia, B. Li, and D. Song, "Refit: a unified watermark removal framework for deep learning systems with limited data," in Asia-CCS, 2021, pp. 321–335.

[23] X. Liu, F. Li, B. Wen, and Q. Li, "Removing backdoor-based watermarks in neural networks with limited data," in 2020 25th International Conference on Pattern Recognition (ICPR), 2021, pp. 10 149–10 156.

[24] A. Lansner and A. Holst, "A higher order bayesian neural network with spiking units," International Journal of Neural Systems, vol. 7, no. 02, pp. 115–128, 1996.

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

[26] R. S. Srinivasamurthy, "Understanding 1d convolutional neural networks using multiclass time-varying signalss," Ph.D. dissertation, Clemson University, 2018.

[27] "TrustHub.org: Trust-HUB,," http://trust-hub.org/ benchmarks/trojan.

[28] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," arXiv preprint arXiv:1206.6389, 2012.

[29] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," Current opinion in neurobiology, vol. 14, no. 4, pp. 481–487, 2004.

[30] C. Szegedy et al., "Going deeper with convolutions," in CVPR, 2015.

[31] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware trojan attacks: Recent advances and challenges," IEEE Access, vol. 8, pp. 10 796–10 826, 2020.

Zhixin Pan is a Ph.D student in the Department of Computer & Information Science & Engineering at the University of Florida. He received his B.E. in the Department of Software Engineering from Huazhong University of Science & Technology, Wuhan, China in 2015. His area of research includes Cyber & Hardware Security, post-silicon debug, data mining and machine learning.

Prabhat Mishra is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida. He received his Ph.D. in Computer Science from the University of California at Irvine in 2004. His research interests include embedded and cyber-physical systems, hardware security and trust, and energy-aware computing. He currently serves as an Associate Editor of IEEE Transactions on VLSI Systems and ACM Transactions on Embedded Computing Systems. He is an IEEE Fellow and an ACM Distinguished Scientist.