Design of AI Trojans for Evading Machine Learning-based Detection of Hardware Trojans

Zhixin Pan and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—The globalized semiconductor supply chain significantly increases the risk of exposing System-on-Chip (SoC) designs to malicious implants, popularly known as hardware Trojans. Traditional simulation-based validation is unsuitable for detection of carefully-crafted hardware Trojans with extremely rare trigger conditions. 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 significantly outperforms state-of-the art approaches based on adversarial attacks.**

I. Introduction

The globalized semiconductor supply chain significantly increases the risk of exposing System-on-Chip (SoC) designs to hardware Trojans (HT) [1]. 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.

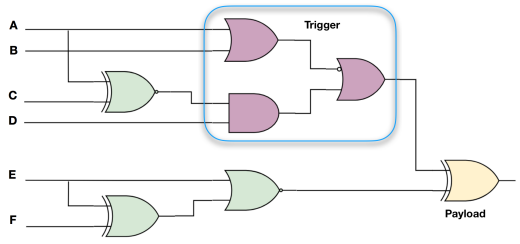


Fig. 1. An example hardware Trojan. Once the trigger condition (purple gates) is satisfied, the payload (yellow XOR) will invert the expected output.

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]–[5]. Machine learning (ML) algorithms have received considerable attention for HT detection in recent years due to their scalability as well as detection accuracy [6]. 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, 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)*.

A. Threat Model

While MLaaS provides specific advantages, it also provides adversaries with opportunities to launch backdoor attacks towards ML models, popularly known as AI Trojans in computer vision domain, as described in Section II-A. AI Trojans and hardware Trojans are similar from several perspectives. (1) Both of them are malicious implants consisting of a rare trigger and a small payload. (2) The functionality of the infected circuit (or backdoored ML model) is not affected until the adversary applies certain inputs to activate the Trojan trigger. (3) They can be inserted by a rogue employee or an adversary involved in any of the third-party service (e.g., MLaaS for AI Trojans or IP design/synthesis/fabrication for hardware Trojans). In spite of the above similarities, they have one major difference. While the primary objective of AI Trojans is to mispredict (incorrect execution), hardware Trojans can lead to information leakage, incorrect execution, denial-of-service, or other unintended consequences.

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. 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 and defenses on ML-based HT detection.

This work was partially supported by the NSF grant CCF-1908131.

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

II. Background and Related Work

A. Background: AI Trojans

To reduce the cost and training time associated with developing ML models, some industries outsource the training process to a potentially untrusted third-party cloud service providers. Alternatively, they acquire pre-trained ML models. These usage models creates opportunity for adversary to provide users with backdoored ML models. BadNet [7] performs well on almost any regular inputs (including the inputs in a typical validation set), but produces misclassification for inputs that satisfy some secret, attacker-chosen rare scenario, which we refer as the 'backdoor trigger'. Liu et al. introduced the idea of backdoor attacks on machine learning [8]. Gu et al. applied this idea in autonomous systems where they launch the backdoor attack on traffic sign detection [7].

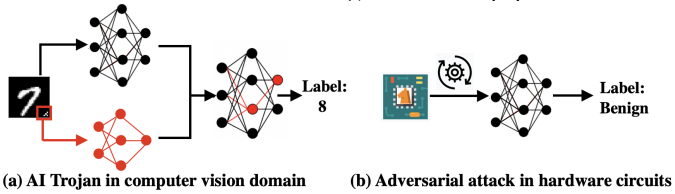


Fig. 2. Comparison between various attacks on ML models. (a) Traditional AI Trojan attack in computer vision domain [7]. (b) State-of-the-art adversarial attacks on ML-based hardware Trojan detection [9].

Figure 2(a) shows an illustrative example of an AI Trojan in computer vision domain. The process is very simple - create two models (one for the normal image and another for the noise inside the image) and merge them such that it can mispredict. For example, the backdoored model identifies the symbol 7 as 8. This idea cannot be directly applied for hardware Trojan detection since there is no similar concept of 'noise' in hardware designs that can alter classification but does not change the functionality of the design. Figure 2(b) shows state-of-the-art adversarial attacks [9] of ML-based hardware Trojan detection. *Our proposed idea is fundamentally different from adversarial attacks due to the fact that our work relies on embedding a Trojan in the ML model while the latter focuses on crafting adversarial examples.*

There are some recent attempts in defending against backdoor attacks [10]–[12]. However, these defenses have limited applicability in specific scenarios. Section IV demonstrates that our approach can bypass the state-of-the-art defences. *While the threats of 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.*

B. Related Work: ML-based Hardware Trojan Detection

ML algorithms have enabled promising performance with outstanding flexibility and generalization for HT detection in recent years. Chen et al. [13] 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. [14] presented a pattern matching algorithm to detect HTs by analyzing the distribution of rare signals inside IP cores. Kasegawa et al. [15], [16] explored various features and applied different ML algorithms (DNNS, SVM) 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.

III. Backdoor Attack with AI Trojans

Figure 3 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 remainder of this section describes these tasks in detail.

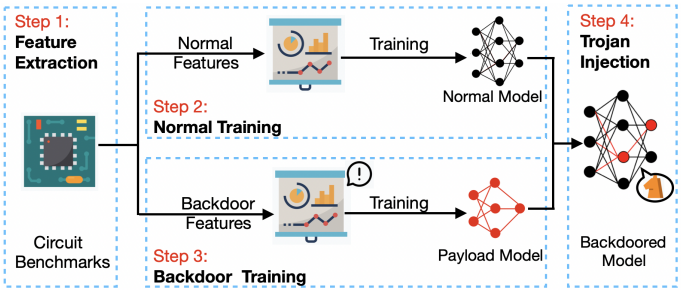


Fig. 3. Overview of our proposed framework that consists of four activities: feature extraction, normal training, backdoor training and Trojan injection.

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*. 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 [16], 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.*

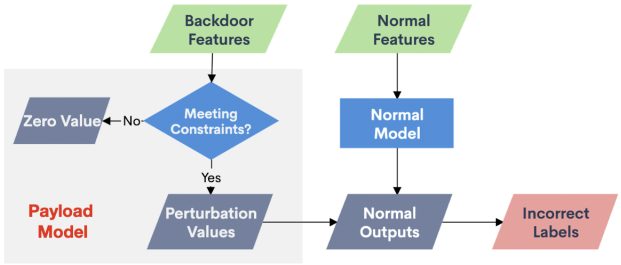


Fig. 4. The fundamental idea of using 'payload' model.

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 4. 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. This guarantees the orthogonality. While there are other candidate features, we select these three 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 IV.

Algorithm 1: Normal Training

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

initialization;
$N = |\{x_i\}|$
repeat
  for $i = 1 \dots N$ **do**
    out_i = softmax$(M_\Theta(x_i))$
    $loss = \sum_i^N$ cross_entropy(out_i, y_i)
    Θ = sgd$(\Theta, \nabla loss)$
until *converge*;
Return M_Θ

B. Normal Training

The normal training follows the standard training procedure as shown in Algorithm 1. In [15], the author proposed an ML model with only one hidden layer and 500 hidden nodes. In our work, the hidden layers mimic the design of Lenet-5 [17], it is composed of three consecutive layers of convolution, followed by two fully connected layers. 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 $\mathcal{L}$ 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: $loss = \mathcal{L}(M_\Theta(x_i), y_i)$. HT detection is a binary classification task and therefore y_i is either 0 or 1. In this case, $\mathcal{L}$ is selected as the cross-entropy. In addition, L_2 regularization and *dropout* strategies are also applied in our framework to avoid overfitting problem.

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**
    $x_i' = mutate(x_i)$;
Label all x_i as 0, x_i' as 1
$X^* = \{x_i\} \cup \{x_i'\}$
repeat
  for each $x_i^* \in X^*$ **do**
    out_i = softmax($\bar{M}_{\bar{\Theta}}(x_i^*)$)
    $loss = \sum_{i}^{2N}$ cross_entropy$(out_i, label(x_i^*))$
    $\bar{\Theta}$ = sgd$(\bar{\Theta}, \nabla loss)$
until $converge$;
$M^* = -\lambda \cdot H(\bar{M}_{\bar{\Theta}}(x_i)) \cdot \mathcal{L}(M_\Theta(x_i), y_i)$
Return M^*

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.

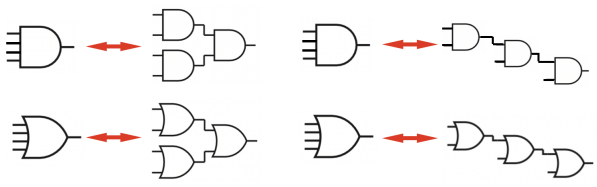


Fig. 5. Example mutation patterns [9] used in our proposed work.

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 5. 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. 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 4.

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 IV. The outline of backdoor training and Trojan injection is shown in Algorithm 2.

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 4, 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:

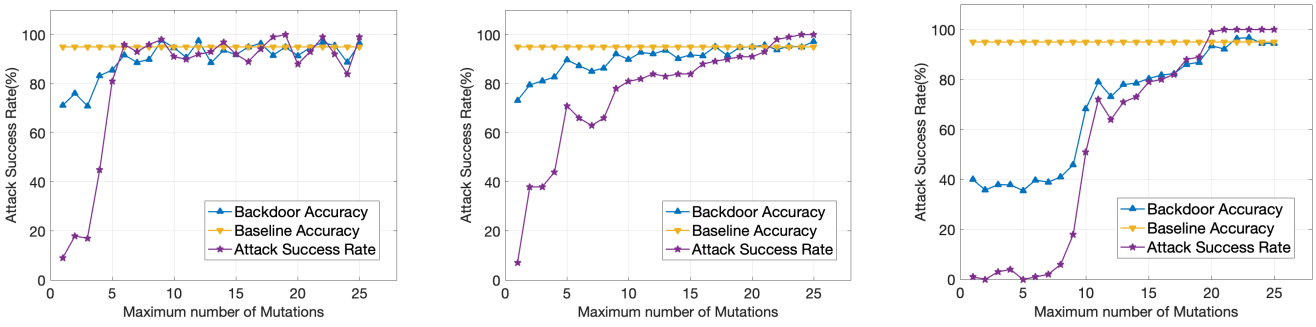
$$output = -\lambda \cdot H(\bar{M}_{\bar{\Theta}}(x_i)) \cdot \mathcal{L}(M_\Theta(x_i), y_i)$$

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), $H(\bar{M}_{\bar{\Theta}}(x_i)) = 1$ and the output is a scaled inverse of normal model output. In terms of '0' label (without backdoor signature), $H(\bar{M}_{\bar{\Theta}}(x_i)) = 0$ 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. Experiments

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 [18]. Features are extracted from benchmarks and formatted into



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

Fig. 6. The attack success rate of our framework using three different payload models under different thresholds on number of mutations.

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 [17] like structure, composed of 3 convolution layers followed by 2 fully connected layers.
- **GoogleNet:** A GoogleNet [19] like structure, with a depth of 22 layers.

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

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

To evaluate the effectiveness of our approach, we compare with the following **state-of-the-art** attack and defense.

- **GAE:** State-of-the-art adversarial attack based on generating adversarial examples [9].
- **STRIP:** State-of-the-art defense against AI Trojan attacks [12].

B. Comparison of Attack Performance

Figure 6 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. 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.

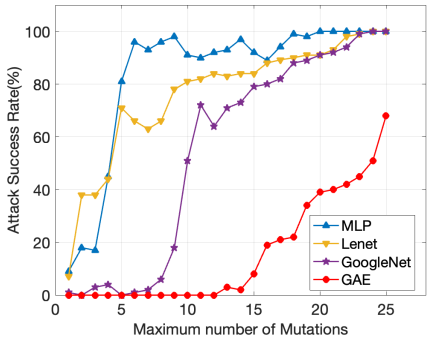


Fig. 7. The attack success rate comparison between proposed algorithm and the state-of-the-art adversarial attack with < 25 mutations.

Figure 7 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 also faces the risk of being detected due to larger number of mutations.

C. Overhead Analysis

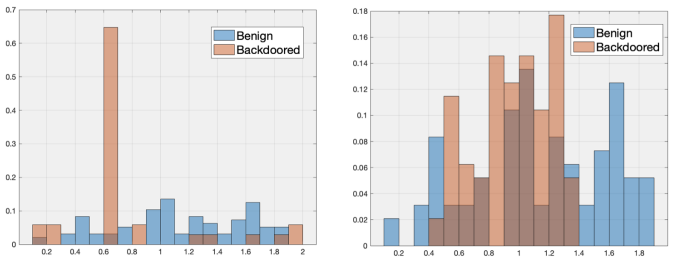
Table II compares the training cost and data resources of various methods. The first three rows represent our approach. The MLP approach is the most economic is terms of training cost. It can be trained within 50 epochs with each epoch taking 0.6s, and only requires 20% of the training samples to be 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 II
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

D. Robustness against STRIP-based Defense

We further evaluate the proposed attack's robustness against the state-of-the-art defense scheme, STRIP [12]. 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. Then STRIP applies the fused input to the backdoored model and calculates the entropy of model outputs. This defence 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.



(a) GAE against STRIP (b) Proposed against STRIP

Fig. 8. Entropy distribution of clean and backdoored inputs. (a) Backdoored inputs generated by GAE can be easily detected. (b) Backdoored inputs generated by our method's entropy is hard to distinguish from benign inputs.

Figure 8 shows the entropy of outputs from GAE and our proposed method for both clean and backdoored inputs. As we can see, the distribution of entropy for backdoored data overlaps with the distributions of entropy of the clean data for our approach. However, GAE's entropy can be clearly distinguished from the normal one. 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 erected to the gradient direction, leading to a small entropy. Our proposed method is able to bypass the state-of-the-art defense (STRIP) while state-of-the attack (GAE) fails.

V. 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 framework is robust against the state-of-the-art defence against Trojan attacks.

REFERENCES

[1] F. Farahmandi *et al.*, *System-on-Chip Security*. Springer, 2020.
[2] Y. Lyu and P. Mishra, "Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling," *IEEE TCAD*, 2020.
[3] ——, "Maxsense: Side-channel sensitivity maximization for trojan detection using statistical test patterns," *ACM TODAES, 26(3)*, 2021.
[4] Z. Pan, J. Sheldon, and P. Mishra, "Test generation using reinforcement learning for delay-based side-channel analysis," in *ICCAD*, 2020.
[5] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *ASPDAC*, 2021, pp. 408–413.
[6] ——, "Hardware acceleration of explainable machine learning," in *Design Automation and Test in Europe (DATE)*, 2022.
[7] T. Gu *et al.*, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.
[8] Y. Liu *et al.*, "Trojaning attack on neural networks," 2017.
[9] Nozawa *et al.*, "Generating adversarial examples for hardware-trojan detection at gate-level netlists," *JIP*, vol. 29, pp. 236–246, 2021.
[10] B. Wang *et al.*, "Neuralcleanse: Identifying and mitigating backdoor attacks in neural networks," *SP*, vol. 530546, 2019.
[11] K. Liu *et al.*, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *ISRAID*. Springer, 2018, pp. 273–294.
[12] Y. Gao *et al.*, "Strip: A defence against trojan attacks on deep neural networks," in *ACSAC*, 2019, pp. 113–125.
[13] X. Chen *et al.*, "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis," *IEEE TCAD*, 2017.
[14] E. Zhou *et al.*, "A novel detection method for hardware trojan in third party ip cores," in *ISAI*, 2016, pp. 528–532.
[15] 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.
[16] ——, "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.
[17] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann.lecun.com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.
[18] "TrustHub.org: Trust-HUB,," http://trust-hub.org/ benchmarks/trojan.
[19] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.