

# TD-Zero: Automatic Golden-Free Hardware Trojan Detection using Zero-Shot Learning

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

**Abstract**—Supply chain vulnerability provides the opportunity for the attackers to implant hardware Trojans in System-on-Chip (SoC) designs. While machine learning (ML) based Trojan detection is promising, it suffers from three practical limitations: (i) golden model may not be available, (ii) lack of human expertise for Trojan feature selection, and (iii) limited learning transferability can lead to unacceptable performance in new benchmarks with unseen Trojans. While recent approach based on transfer learning addresses some of these concerns, it still requires re-training for fine-tuning the model using domain-specific (e.g., hardware Trojan features) knowledge. In this paper, we propose a Trojan detection framework utilizing zero-shot learning to address the above challenges. The proposed framework adopts the idea of self-supervised learning, where a pre-trained graph convolutional network (GCN) is utilized to extract underlined common sense about hardware Trojans, and a metric learning task is used to measure the similarity between test inputs and malicious samples to make classification. Extensive experimental evaluation demonstrates that our approach has four major advantages compared to state-of-the-art techniques: (i) does not require any golden model during Trojan detection, (ii) can handle both unknown Trojans and unseen benchmarks without any changes to the network, (iii) drastic reduction in training time, and (iv) significant improvement in detection efficiency (10.5% on average).

## I. INTRODUCTION

The demand for System-on-Chip (SoC) designs has increased 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. The benefit of globalization comes with the cost of security concerns as elaborated in [1]. Due to the fact that an SoC may include few components from potentially untrusted third-party vendors, the risk of exposing SoC designs to hardware Trojans is also increased.

### A. Threat Model

An attacker may insert hardware Trojans (HT) during any stages of the design flow, including specification, implementation, validation, synthesis, layout, fabrication, testing, and deployment. In this paper, we primarily focus on detecting Trojans during pre-silicon stages of the design (prior to fabrication). Specifically, our objective is to detect HTs in gate-level implementation without requiring any golden reference model.

HT is a malicious modification of hardware designs that consists of two critical parts, trigger and payload. The trigger is typically created using a combination of rare events (such as rare signals or rare transitions) to stay hidden during normal

execution. Due to stealthy nature of these Trojans, it may not be feasible to detect HTs during traditional functional validation [2], since low overhead safety checks are preferred [3]. When the trigger is activated, the payload enables the malicious activity resulting in information leakage, erroneous execution, performance degradation, or denial-of-service. In this paper, we focus on hardware Trojans that alters the functionality under diverse trigger conditions.

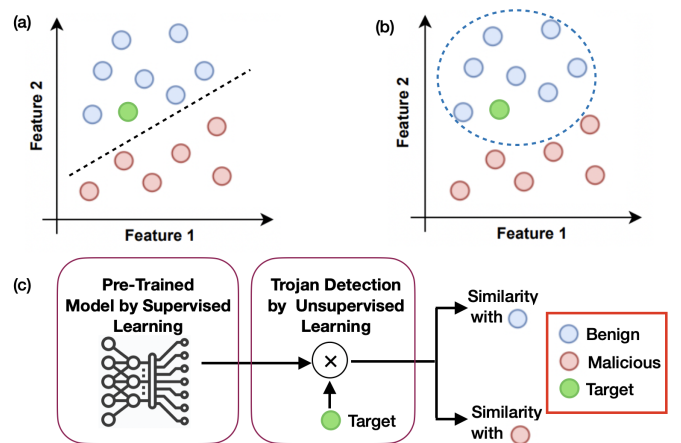


Fig. 1: The illustration of fundamental differences between three hardware Trojan detection approaches using (a) supervised learning, (b) unsupervised learning (clustering), and (c) proposed approach using zero-shot learning.

### B. State-of-the-Art

Recent research turns interest into machine learning (ML) based approaches for Trojan detection. ML, as a data-driven scheme, focuses on building computational models that learn from features of training samples to produce acceptable predictions. While machine learning (ML) algorithms have received considerable attention for security tasks in recent years [4]–[8], they faces three major challenges. (1) Most of the prior efforts focus on supervised learning scheme which requires a golden model. (2) While there are golden-free approaches, the inherent clustering algorithm relies on the expertise of the designers for feature selection and distance metric, which are not guaranteed for all scenarios as demonstrated in Section II-C. (3) The lack of transferability limits their usage in familiar environments (e.g., known benchmarks and known Trojan types). To accommodate unfamiliar scenarios, they usually require large number of updates or unacceptable overhead to re-train the model. Most importantly, these approaches are not suitable for detecting any minor changes (known unknowns) or major variations (unknown unknowns) in hardware Trojans. Figure 1 shows the fundamental differences between existing

ML-based Trojan detection and our approach. Figure 1(a) shows that supervised learning works as a separator in hyperspace to classify the target as benign based on features, while unsupervised learning (clustering) in Figure 1(b) works by encompassing the cluster of benign samples to make classification. As a result, it is often unacceptable for industries to afford certain training cost.

### C. Research Contributions

In this paper, we demonstrate that by adopting the idea of zero-shot learning, our framework is able to detect unseen HTs in unfamiliar environments without any need for golden model or re-training (Figure 1(c)). It effectively combines the advantages of supervised and unsupervised learning. Specifically, it uses an ML model pre-trained with supervised learning of known scenarios (e.g., simple benchmarks with known HTs), and perform Trojan detection using unsupervised learning in unknown scenarios (e.g., new and complex benchmarks with unknown HTs). The proposed method utilizes matching network that consists of two major components. The first component is a Graph Convolution Network (GCN) to extract general knowledge about HTs. Specifically, it utilizes GCN to convert circuit netlists into graph model, and further process it with the help of LSTM scheme. The second component measures the similarity between test inputs and malicious samples to make accurate classification. Instead of specifying the metric by human expert knowledge, we utilize a metric learning network to automatically adjust the measurement. Specifically, this work provides the following advantages compared with existing efforts.

- 1) It is a golden-free solution because the training (supervised learning) can be performed using simple benchmarks (golden models) while the HT detection (unsupervised learning) can be applied on real-life examples without golden models.
- 2) It does not need any human expert knowledge for selecting beneficial HT features.
- 3) It can be utilized to detect unknown Trojans in unseen benchmarks without any change to the network.
- 4) It outperforms state-of-the-art HT detection methods in terms of prediction accuracy.

The remainder of this paper is organized as follows. Section II provides relevant background and surveys related efforts. Section III describes our proposed Trojan detection framework using zero-shot learning. Section IV presents experimental results. Finally, Section V concludes the paper.

## II. RELATED WORK AND MOTIVATION

We first review related efforts in golden-free hardware Trojan detection. Next, we survey existing works on HT detection using supervised learning as well as unsupervised learning. Finally, we motivate the advantages of using Zero-shot learning for HT detection.

### A. Golden-Free Trojan Detection

Most of the Trojan detection approaches assume the availability of a golden model. In reality, a designer may not have

access to the golden model. There are various avenues for golden-free Trojan detection including self-similarity, structural analysis, and machine learning. The first avenue is to enable test generation for golden-free Trojan detection that consists of two parts. The first part is the test generation, and the second part is the comparison of functional values. Since the golden model is not available, existing efforts utilize self-similar components [9], [10]. There are two fundamental challenges: (i) how to automatically identify self-similar components, and (ii) how to generate tests to activate these self-similar components. In other words, the existing solutions are applicable only on very customized and simple designs. The second avenue for golden-free Trojan detection is based on structural feature as well as fingerprinting analysis [11]–[17]. However, the effectiveness of these approaches depend on the availability of the golden features without the golden reference model. The next two sections describe ML-based HT detection using supervised as well as unsupervised (golden-free) learning.

### B. ML-based HT Detection using Supervised Learning

Existing ML-based HT detection methods rely on either supervised or unsupervised learning. Supervised learning aims at learning a function mapping from input to output, where training data always comes with labels. The label serves as the desired output, and the goal of training is to minimize the difference between model outputs and labels. In this scenario, both benign (golden) and malicious benchmarks are required. First, relevant features are selected by either expert knowledge or observation from simulation. Intuitively, assume  $n$  features are selected, this step maps each sample into a point in  $n$ -dimensional hyperspace. Afterwards, samples with these features are extracted from raw data and fed into ML model during training phase. The well-trained model serves as a separator in the  $n$ -dimensional hyperspace to make bi-classification, which is illustrated in Figure 1(a). A vast majority of ML-based HT detection approaches utilize supervised learning. In [18], a general framework for Trojan detection has been proposed. Chen et al. [19] 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. Features including leakage current [20], power consumption [21] and performance counter streams [22] are also exploited in recent years. Zhou et al. [23] presented a pattern matching algorithm to detect HTs by analyzing the distribution of rare signals inside IP cores. Kasegawa et al. [24] proposed five important features from gate-level netlists and built a classifier using Deep Neural Networks (DNNs), which 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) [25] to provide state-of-the-art performance in terms of average accuracy and running efficiency.

### C. ML-based HT Detection using Unsupervised Learning

Supervised learning-based approaches are promising but they require golden models. To achieve golden-free detection,

unsupervised learning is utilized. In this case, training data comes without labels. The essence of unsupervised learning algorithm is to describe the distribution of data and mine potentially useful information behind, where it groups unlabeled data into different clusters based on a distance metric, as shown in Figure 1(b). First, a distance metric is defined to measure the similarity between samples. Then, clustering process happens where samples with high similarities are grouped together, so that samples outside the cluster can be distinguished. This process is illustrated in Figure 1(b). While clustering has shown promising results for HT detection [15], [26], unsupervised learning methods has two practical limitations: feature selection and distance metric. The features are commonly selected by either expert knowledge [15] or millions of simulations [26], which may not be feasible in many scenarios. Moreover, the selection of proper distance metric is also tricky, existing works utilize Minkowski distance as the similarity metric, which may not be suitable for HT detection. For example, Figure 2 illustrates that clustering algorithms provide unstable performance, detection accuracy varies based on the distance metric. This instability severely affects the performance of existing works. Furthermore, there are no automated methods to select a beneficial metric for a set of benchmark/HTs. Also, as we can see, the performance of clustering algorithm varies from benchmark to benchmark, and measurement to measurement. To overcome these weaknesses, we propose a novel detection scheme based on Zero-shot Learning (Figure 1(c)), which is discussed in the next section.

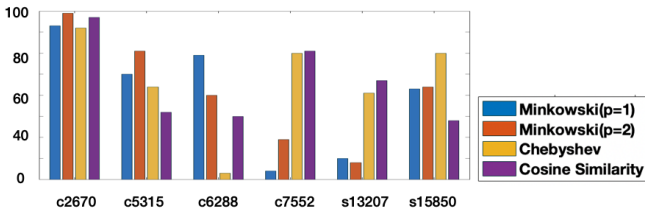


Fig. 2: The detection accuracy using clustering algorithm for several benchmark with four different distance metrics,  $p = 1$  Minkowski (Manhattan),  $p = 2$  Minkowski (Euclidean), Chebyshev and Cosine similarity.

HTNet [27] provides a promising alternative to distance-based clustering by utilizing transfer learning to detect Trojans in unseen chips. However, it requires the re-training of the model with the domain-specific knowledge of the unseen chips. In other words, models trained by certain types of dataset cannot be directly applied to other tasks, and huge amount of samples are needed for accommodating models to new scenarios. Moreover, the authors assume that the trigger condition is always-on when performing side-channel signals, which may not be true in many scenarios for two reasons: (i) the difficulty associated with activating a stealthy trigger condition, and (ii) side-channel Trojan footprint of a tiny trigger (of few gates) can easily hide in process variation and environmental noise margins in multi-million gate designs. In contrast, our proposed approach utilizes zero-shot learning that neither assumes always-on Trojan nor require re-training for detecting unknown Trojans in unseen benchmarks.

#### D. Background: Zero Shot Learning

The demand of Zero-Shot Learning (ZSL) arises from crucial limitations of previous ML approaches. ZSL is an extreme variant of transfer learning, which relies on zero training samples to handle unseen categories. The key idea of ZSL is to focus on learning the ‘general knowledge’ of given data, and unlike the commonly-known learning approaches, it is closer to human brain when making judgements. For example, assume that a child has never see a tiger before, and only pictures of cats and dogs are shown to the child. The child is likely to identify the tiger as a cat. The reason is that, even though ‘tiger’ category is never seen before, the brain is able to extract information from the picture and make comparison with known species based on similarity. Similarly, the goal of ZSL is to train the model with clever adjustments during the training stage, so that the model is capable of exploiting information to understand unseen data.

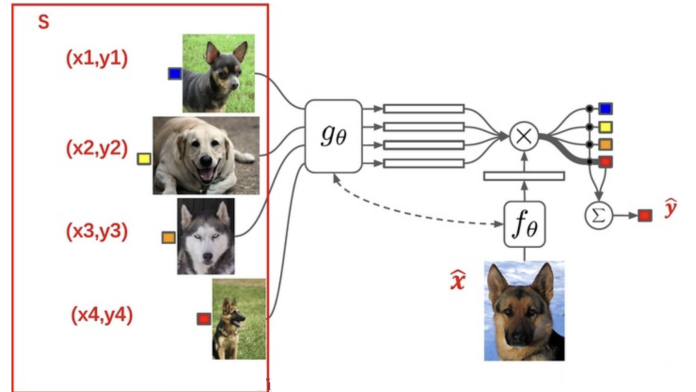


Fig. 3: ZSL utilizing matching networks as proposed in [28].

A typical implementation of ZSL is a matching network structure as shown in Figure 3, which consists of two major components, an extractor  $g_\theta$  and a comparator  $f_\theta$ . Extractor is responsible for recognizing and extracting general knowledge from training set, while compactor works by comparing the similarity of test input and known genre to assign label to it based on similarity score. In this scenario, the task is to classify the genre of dog, where the model is not trained to merely remember human-defined features from known samples, but trained to be sophisticated in mining underlying features by themselves, and make reasoning by comparing afterwards. By extracting general knowledge from the entire sample set  $S$ , four different type of general knowledge were extracted by  $g_\theta$  in the model. Then  $f_\theta$  works by comparing the given sample’s feature with pre-stored ones and computing similarity scores. The one with the highest score became the classified label. ZSL requires no samples of unseen benchmarks, which make golden-free chip examination possible. We show how to utilize ZSL for HT detection to address all the above-mentioned challenges in the next section.

### III. TROJAN DETECTION USING ZERO-SHOT LEARNING

Zero-Shot Learning (ZSL) has shown promising performance in computer vision domain. However, there are three major challenges in applying ZSL for Trojan detection. (1)

The concept of “general knowledge” of circuit benchmarks is undefined. In [28], the author primarily focus on image datasets, where the general knowledge can be information about shapes, edges and texture, which are not applicable for circuit netlists.(2) The boundary for “seen” and “unseen” categories in terms of netlists benchmarks are unclear. In [28], unseen image datasets refer to images containing different objects or handwritten-digits with different values. (3) In computer vision domain, a lot of well-defined distance metrics can be applied for similarity comparison. However, none of them are feasible for hardware Trojan detection, as demonstrated in Figure 2. We address these challenges using a combination of “general knowledge” extraction and metric learning algorithms.

Figure 4 shows an overview of our proposed HT detection approach using zero-shot learning (ZSL). The key idea is to focus on learning the ‘general knowledge’ of seen benchmarks, so that the model stores the impression and necessary underlying properties of both benign and malicious circuit benchmarks. In this way, when unseen benchmark is provided, the model utilize the pre-stored knowledge to understand them based on the similarity scores. Intuitively, in detection phase, the model simply works as an inspector to check if the target benchmark is closer to Trojan-inserted designs or the clean golden ones. The first task is to transform the given design (e.g., gate-level netlist) into a graph-based representation. The second task performs automatic information extraction for each input sample through graph convolution network (GCN). Specifically, the general knowledge of both benign and malicious netlists are stored. The third task enables automatic HT detection for a given unseen (target) benchmark. The trained GCN extract information about the target benchmark. The extracted information is compared with the pre-stored knowledge to compute similarity score through metric learning. For example, if the similarity score is higher for the malicious category, the given input is classified as malicious (HT-implanted). We manage to apply a metric learning algorithm based on Mahalanobis distance [29]. It aims at constructing task-specific distance metrics from the given data, which automatically searches for a customized distance metric by solving an optimization problem, which will be demonstrated in Sec III-C.

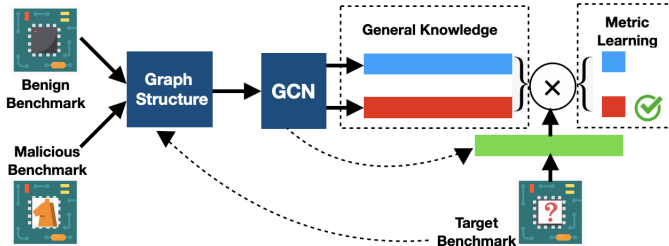


Fig. 4: Overview of our proposed framework. It consists of three major tasks: graph-based representation of hardware models, general knowledge extraction utilizing zero-shot learning, and detection of Trojans using metric learning.

Although benign benchmarks (golden models) are required during the training phase, these benign samples can be simple designs. In other words, the golden model of the target

benchmark is not required during the HT detection phase. For example, the model trained with ISCAS benchmarks can be utilized to detect Trojans in Trust-Hub benchmarks without any need for re-training the model. We will describe the three major tasks outlined in Figure 4: graph-based representation, general knowledge extraction, and hardware Trojan detection.

#### A. Graph-based Representation of Hardware Designs

A major limitation of existing HT detection approaches is that the feature selection is typically guided by expert knowledge instead of an automatic method. Although few methods extract features based on observations from simulations, this process introduces training cost. Since features collected from training set are specific to training benchmarks, ML models trained by these features are not guaranteed to perform well against unseen benchmarks. Therefore, the ideal way of feature extraction should be collecting ‘general knowledge’, instead of benchmark/domain-specific features. Also, to enable transferability, the ML model should be trained to perform automatic knowledge extraction, rather than relying on manually collected knowledge from users. Such a model would be able to make correct predictions without any changes even for unseen benchmarks.

The idea of extracting general knowledge is widely adopted in ML applications. For instance, in computer vision domain, image data only serves to provide fundamental information like pixel values, while the task of knowledge learning depends on effective utilization of Convolution Neural Networks (CNN). Convolution layers filter the fundamental information to forge features and store them in hidden layers. Similar process can be implemented for circuit netlists, we can preprocess input circuit netlists into graph representation, and utilize Graph Convolution Network (GCN) for knowledge extraction.

In general, a circuit netlist can be represented by a directed graph structure  $G = (V, E)$ , where  $V$  is a set of vertices and  $E$  is a set of edges. Signals possess fundamental value information, while each logic gate works by combining the signals through logic computation to produce new signals. Therefore, it is natural to treat wires as nodes for storing values, and transfer gates into directed edges to represent relationship between nodes and the flow of information.

Figure 5 shows a simple example of graph representation. Notice that two important categories of knowledge are fully preserved by graph representation which are critical for effective hardware Trojan detection. First, the testability measurements (SCOAP parameters) [15] are recorded as node values, which will be discussed in the next section. Secondly, the structural details of the netlists are also conserved. In addition to node values, the interconnections, graph dimensions, and node depths are relevant for machine learning, closely tied to the structural attributes of the circuit netlist. Table I provides the graph attributes and their corresponding structural features.

#### B. General Knowledge Extraction using GCN

Figure 6 represents the structure of proposed neural network with fine-tuned hyperparameters. The process starts by transforming given circuit netlists into graph representation as described in Sec III-A. Next, the transformed graph passes



Circuit	Fan-ins	Fan-outs	Logic depth	Information Flow	Circuit Size
Graph	In-degrees	Out-degrees	Node depth	Edge Direction	Graph Size

TABLE I: Comparison between graph attributes and corresponding structural features.

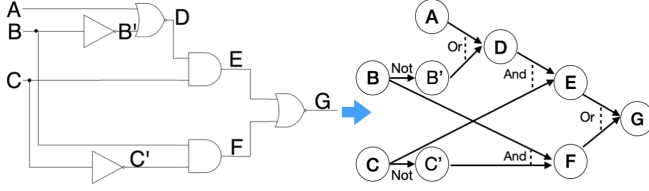


Fig. 5: Illustrative example of transforming a gate-level netlist into a graph representation. Data format of nodes and edges are discussed in Section III-B.

through the network consisting of three consecutive graph convolutional layers (GCLs), each followed by a dropout and max-pooling layer to avoid overfitting problem. GCLs are specifically designed structure to extract general knowledge of graph-structure data, which is similar to traditional convolution layers for 2D structural (image) data, as shown in Figure 7. In fact, graph convolution can be considered as a generalization of 2D convolution as shown in Figure 7. Analogous to a graph, each pixel in an image can be taken as a node, and the 2D convolution works by taking the weighted average of the centre (red) node along with its neighbors at each step. In this way, the pixel value is shared along with its surrounding neighbors to help the model extracting general knowledge about the image.

1) *Motivation*: To motivate the utilization of graph convolution network, the following example clearly demonstrates why information sharing by computing weighted average can make 2D convolution effective for image processing. As shown in Figure 8(a), the image is provided as a 2-D matrix with pixel values, and the task is to detect a specific shape curve from the image. In Figure 8(b), the convolution works by sliding the convolution kernel along the image, where the weighted average is computed at each position. The results are presented in Figure 8(c).

It can be observed that the output of the convolution kernel in this example is relative to the position of the detected shape. The closer kernel to the location of desired shape, the greater the corresponding output. Additionally, in general convolution neural networks, the convolution layers are commonly followed by a max-pooling and RELU layer. The functionality of these two layers are to filter out the highest value from input. In this way, the most important ‘knowledge’ from the raw features are extracted. For example in this image, the centre of the detected shape will be filter out as the final result. As we can see, this “convolution-pooling-activation” structure serves as a ‘filter’ to extract key information from input data. Notice in this example we pre-fixed the values of the kernel, while in reality they are obtained through model training. So literally speaking, the training process of convolution is to find out proper filters which can help extracting essential features.

Another important observation of 2D convolution is *linear shift invariant*. If we shift the target image by any amount of displacement, then the output is shifted by the same amount. This property guarantees that convolution is resistant to position obfuscation, which coincidentally fits the task of HT detection. A well-trained convolution kernel is expected to encompass the ‘Trigger-Payload’ structure regardless of its location. Inspired by the motivation, we can immediately derive the prototype of HT detection model after obtaining the graph representation of circuits. In Figure 9, by following the similar idea above we define the task of HT detection to be training a proper graph convolution kernel, which sweeps the entire graph to produce corresponding outputs, and it reacts to the suspicious Trojan-injected region the most. To implement our model, there are three more essential inspirations from the general 2D convolution. 1) The example in Figure 8 requires only one shaped curve. While in reality, due to complexity of HT detection, we tend to extract features from input as comprehensively as possible. Therefore, multiple kernels are necessary in the network structure. 2) Multiple layers are important since each node can only get the information from its neighbors with one layer. When stacking another layer on top of the first one, the neighbors already have information about their own neighbors and advanced features can be further crafted by combining initial features from the previous step. 3) The goal of our proposed method is to achieve golden-free detection, so we need to have an extra component serves as inventory. In this way golden-designs are unnecessary since the model already contains pre-learned knowledge.

Based on the above discussion, we setup the model structure as shown in Figure 4, where multiple blocks of “convolution-pooling-activation” are deployed, an extra LSTM model is applied as storage. At each step of graph convolution, the goal is to achieve information sharing for each node with its neighbors.

However, the information sharing in transferred circuit graph remains unclear and it is not a trivial task. For example, inspired by Figure 8, one simple solution is to take the weighted average value of the node along with its neighbors as shown in Figure 9. This naive approach cannot work in our case due to the following three challenges.

- We cannot define the value of each node based on signal values since they change based on input patterns.
- Graph transformed from netlists are directed graph, each edge possesses attributes including direction and logical relationship. Information carried by edges will be lost if we compute weighted average of nodes.
- For image data, the neighbors of a node are ordered and have a fixed size. But for graph data, they are commonly unordered and variable in size. Therefore it is difficult to define the convolution of graph.

The following subsections address the above challenges.

2) *Computation of Graph Node Values*: To tackle the first problem, we utilize Sandia Controllability/Observability Analysis Program (SCOAP) parameters as the value to be stored in each node, which takes both *controllability* and *observability* attributes of signals into consideration. In essence, controllability indicates the amount of effort required for setting a

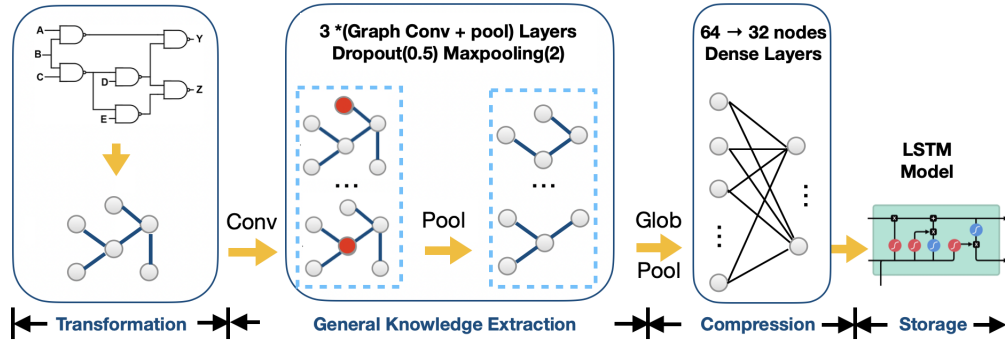


Fig. 6: The model structure of proposed work based on graph convolution network and LSTM.

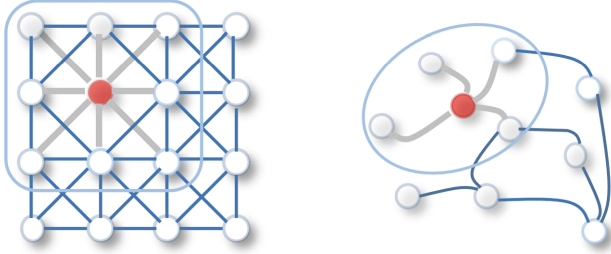


Fig. 7: The comparison between convolution and graph convolution from [30].

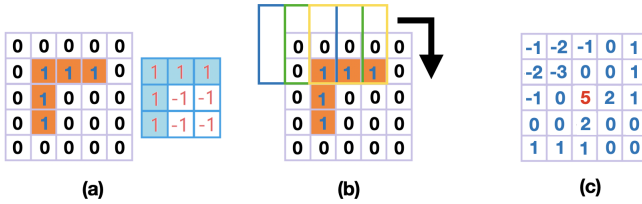


Fig. 8: The intuition on how CNN works for image processing. In this example, the task is to detect a specific shape from the image. (a) The original image with pixel values and the 3\*3 convolution kernel. (b) The convolution works by sliding the convolution kernel along the image. The weighted average is computed at each position. (c) The grid with the highest computed convolution result is predicted as the centre of the desired shape.

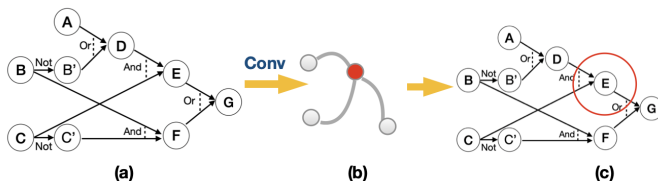


Fig. 9: The prototype process on using graph convolution to perform HT Detection. (a) The transferred graph representation of hardware circuits. (b) The graph convolution happens by moving the convolution kernel around the graph to achieve information sharing through and generate corresponding results. (c) The node with highest output value is the most suspicious region where Trojan might be injected.

signal to a specific value, while observability weighs up the difficulty of propagating the target signal towards observation points. Formally, SCOAP quantifies the controllability and observability of each signal with three numerical values.

- **CC0**: Combinational 0-controllability, the number of signals must be manipulated to set '0' value for target.
- **CC1**: Combinational 1-controllability, the number of signals must be manipulated to set '1' value for target.
- **CO**: Combinational observability, the number of signals must be manipulated to observe target value.

Computing SCOAP parameters of each node serves to provide fundamental information of the graph data, and it addresses many of the previously mentioned challenges. First, it is based on static analysis, which avoids tedious pattern generation for activation. Second, edge information is inherited by SCOAP parameters, since the SCOAP values of one logic gate's inputs and outputs are closely related as shown in Figure 10. For each gate, the output controllability is determined by controllability of its inputs, while the input observability is determined by observability of output and all the other input signals. Figure 10 shows the computation formula for three fundamental logic gates. Consider the *CC1* measurement of AND gate as an example, in order to control the output signal  $c$  as '1', both of its input signals  $a$  and  $b$  should be maintained as '1' at the same time. Therefore, we have  $CC1(c) = CC1(a) + CC1(b) + 1$ , where the '+1' is for counting the level depth. Finally, the SCOAP testability measurement naturally fits the demand of HT detection from a security perspective.



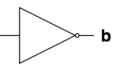
Logic gates	Testability Measure
$a$ $b$  $c$	$CC0(c) = \min\{CC0(a), CC0(b)\} + 1$ $CC1(c) = CC1(a) + CC1(b) + 1$ $CO(a) = CO(c) + CC1(b) + 1$
$a$ $b$  $c$	$CC0(c) = CC0(a) + CC0(b) + 1$ $CC1(c) = \min\{CC1(a), CC1(b)\} + 1$ $CO(a) = CO(c) + CC0(b) + 1$
$a$  $b$	$CC0(a) = CC1(b) + 1$ $CC1(a) = CC0(b) + 1$ $CO(a) = CO(b) + 1$

Fig. 10: SCOAP testability measurement for three logic gates.

Clearly, signals with high controllability are more likely

to be chosen as trigger signals because high controllability guarantees the difficulty of switching these signals with a limited number of test patterns. Similarly, targeting signals with high observability as payload are favorable for attackers, since it avoids them from frequently generating observable impact on design outputs. The algorithm for computing SCOAP values can be easily obtained through a topological sort as shown in Algorithm 1. First, we perform topological sort of the entire circuit graph. Next, the starting point are primary inputs/outputs since their SCOAP parameters are straightforward. For primary input, users may manipulate its values by one signal (itself), i.e.,  $CC0(PI) = CC1(PI) = 1$ . For primary outputs, there is no other signals involved to get observed, which gives  $CO(PO) = 0$ . Then by utilizing formula in Figure 10 we can get all the  $CC0, CC1$  values by following the topological order, and all values of  $CO$  by a reversed topological order.

---

**Algorithm 1:** Testability Analysis (*getSCOAP*)
 

---

**Input :** Design( $D$ )

**Output:** SCOAP Parameters of all nodes in  $D$

- 1 Transfer design into graph representation:  
 $G = DAG(D)$
  - 2 Topological Sort:  $G^* = topo(G, PI \rightarrow PO)$
  - 3  $CC0(PI) = CC1(PI) = 1, CO(PO) = 0$
  - 4 **for** each gate  $g \in G^*$  **do**
  - 5      $\left[ \begin{array}{l} g.out.SCOAP = \\ \quad computeCC(g.in.SCOAP, type(g)) \end{array} \right.$
  - 6  $G^* = reverse(G^*)$
  - 7 **for** each gate  $g \in G^*$  **do**
  - 8      $\left[ \begin{array}{l} g.in.SCOAP = \\ \quad computeCO(g.out.SCOAP, type(g)) \end{array} \right.$
- 

3) *Graph Convolution:* To address the third challenge mentioned above, we select graph convolution as the solution. On Euclidean domains, convolution can be easily defined, but such structural property is undefined on irregular structure like graphs, so we need to look at this concept from a different perspective. The key idea is to use Fourier transform inspired by convolution theorem, where the convolution of two signals is the component-wise product of their Fourier transforms. Therefore, the task boils down to defining the Fourier transforms of graph data. For any function  $f(t)$ , the traditional Fourier transform is defined as:

$$F(\omega) = \mathcal{F}[f(t)] = \int f(t)e^{-i\omega t} dt \quad (1)$$

It can be viewed as the projection of a given function on basis functions  $e^{-i\omega t}$ , which are the eigenfunctions of the Laplacian operator since

$$\Delta e^{-i\omega t} = \frac{\partial^2}{\partial t^2} e^{-i\omega t} = -\omega^2 e^{-i\omega t} \quad (2)$$

Now the task is further reduced to find the Laplacian operator in the domain of graphs. It turns out the discrete Laplacian operator exists, which is the Laplacian matrix  $L = D - A$ , where  $D$  is the degree matrix (a diagonal matrix containing

the number of edges attached to each vertex), and  $A$  is the adjacency matrix of the graph vertices. Therefore, we can compute the convolution of graph data in the following steps. Assume that there are  $n$  vertices in the graph. First, we compute the Laplacian matrix  $L = D - A$ . Next, we compute the eigendecomposition of  $L$  as

$$L = U \begin{pmatrix} \lambda_1 & & \\ & \dots & \\ & & \lambda_n \end{pmatrix} U^{-1}$$

where  $U = (u_1, u_2, \dots, u_n)$  is a matrix with column vectors as unit eigenfunctions, and  $\lambda_i$ s are eigenvalues of  $L$ . Next, we use Equation (1) to define the Fourier transform of graph as

$$F(\lambda_l) = \sum_{i=1}^n g(i)u_l(i) \quad (3)$$

where  $g$  is a graph function and  $g(i)$  corresponds to the value of the  $i$ -th vertex of the graph,  $u_l$  is the  $l$ -th eigenfunction. Finally, the graph convolution can be obtained by utilizing convolution theorem. After simplification, the convolution between two graph function  $g$  and  $h$  can be computed by

$$(g * h)_G = U((U^T g) \odot (U^T h)) \quad (4)$$

where  $\odot$  is the Hadamard product. The training algorithm of the entire network is presented in Algorithm 2.

4) *Knowledge Compression and Storage:* When GCLs finished work, the resulting channels are mapped into a fixed-size vector using a global pooling layer. The outputs at this stage represents the coarse collection of knowledge from the inputs. To further purify the extracted knowledge, an additional fully-connected dense layer is appended to achieve knowledge compression. Furthermore, we utilize LSTM for storing the extracted knowledge (the rightmost component in Figure 6).

### C. Trojan Detection using Metric Learning

The third (final) task in our proposed approach is to perform HT detection for a given (unseen) benchmark. As shown in Figure 4, After obtaining general knowledge from both benign and malicious benchmarks, our ML model accepts the target (unseen benchmark) as input, utilizes the GCN to extract its information, then computes the similarity scores between the extracted information (green rectangle) and pre-stored knowledge (blue and red rectangles) to make prediction. As discussed in Section II-C, the computation of similarity is dependent of efficient selection of the distance metric (Figure 2). Previous clustering algorithms also require a measure of distance between data points. Practitioners in these works commonly choose a standard distance metric (Euclidean, City-Block, Cosine, etc.), which relies on prior knowledge of the domain. However, it is often difficult to design metrics that are well-suited to the particular data and task of interest, as demonstrated by the unstable performance in Figure 2. Instead of blindly selecting a specific distance metric, our focus is that the ML model should learn to adjust its distance metric on-the-fly. To achieve this, we adopt the idea of distance metric learning (or simply, metric learning). It aims at automatically constructing task-specific distance metrics from the given data. Here, the distance metric is defined as the Mahalanobis distances:

**Algorithm 2:** Training Process of TD-Zero

---

**Input :** Malicious and Benign Dataset  $((S_M), S_B)$ ,  
learning rate  $(\alpha)$ , number of epochs  $(k)$ ,  
number of iterations  $(n)$ , Test Dataset  $(S_T)$

**Output:** Optimal extractor  $g_{\theta_1}$ , comparator  $f_{\theta_2}$

- 1 Initialize  $\theta_1, \theta_2$
- 2  $i = j = 0, n = size(T)$
- 3 **repeat**
- 4     Reset Status
- 5     **repeat**
- 6         **for each**  $D \in S_M$  **do**
- 7             Compute SCOAP parameters  
            $(CC0, CC1, CO) = getSCOAP(D)$
- 8             Graph Representation  $G = DAG(D)$
- 9             Malicious Knowledge Extraction  
            $k_M = LSTM(g_{\theta_1}(G))$
- 10         **for each**  $D \in S_B$  **do**
- 11             Compute SCOAP parameters  
            $(CC0, CC1, CO) = getSCOAP(D)$
- 12             Graph Representation  $G = DAG(D)$
- 13             Benign Knowledge Extraction  
            $k_B = LSTM(g_{\theta_1}(G))$
- 14         **for each**  $D \in S_T$  **do**
- 15              $k_T = LSTM(g_{\theta_2}(G))$
- 16             Similarity score  $(s_1, s_2) = f_{\theta_2}(k_M, k_B, k_T)$
- 17              $loss+ = cross\_entropy(s_1, s_2, label(S_T))$
- 18         Update parameter :
- 19          $\theta_1 = \theta_1 + \alpha \nabla_{\theta_1} loss$
- 20          $\theta_2 = \theta_2 + \alpha \nabla_{\theta_2} loss$
- 21     **until**  $j \geq n$ ;
- 22 **until**  $i \geq k$ ;
- 23 Return  $\theta$

---

$$D_M(x, x') = \sqrt{(x - x')^T M (x - x')} \quad (5)$$

where  $M$  is a positive semi-definite (PSD) matrix, i.e.  $M \in S_d(R)_0^+$ . Strictly speaking, Mahalanobis distances are “pseudo-metrics”, it is not a fixed distance metric since  $M$  is flexible. The idea is to train for the most feasible parameters of  $M$  to adjust its ability for measuring similarity between samples (Notice if we set  $M$  to be the identity matrix, it recovers the standard Euclidean distance). Formally, given malicious set  $A$  and benign samples set  $B$ , the distance metric learning aims to find  $M$  for a common convex optimization objective which can be solved efficiently:

$$\arg \min_{M \in S_d(R)_0^+} \frac{\sum_{x_i, x_j \in A} D_M(x_i, x_j) + \sum_{x_i, x_j \in B} D_M(x_i, x_j)}{\sum_{x_i \in A, x_j \in B} D_M(x_i, x_j)} \quad (6)$$

#### IV. EXPERIMENTS

We perform a comprehensive experimental evaluation to demonstrate the effectiveness of our proposed HT detection scheme. First, we describe the experimental setup including platform and evaluation methods. Next, we provide performance measurements and the comparison between proposed

approach and state-of-the-art methods. Finally, we present the overhead and robustness analysis of proposed method.

##### 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. We evaluate our framework using diverse benchmarks from Trust-Hub [31], ISCAS (both ISCAS’85 and ISCAS’89), which are widely used by state-of-art approaches [15], [27], [32]. Specifically, the statistics about the benchmark circuits including sizes and lines of codes (LOC) are shown in Table II.

TABLE II: Sizes of benchmarks from ISCAS [33], [34] and Trust-Hub [31].

Benchmarks	Inputs	Outputs	# of Signals	LOC
AES-T1100	128	128	284	92
AES-T1200	128	128	289	96
AES-T1600	128	128	292	92
AES-T1700	128	128	295	104
c2670	233	140	1193	1482
c3540	50	22	1669	1877
c5315	178	123	2406	2615
c6288	32	32	2406	2685
c7552	207	108	3512	3944
s13207	31	121	7951	3681
s15850	14	87	9772	4010
s35932	35	320	16065	39484

To enable comprehensive evaluation, we also utilize synthetic benchmarks, as discussed in Section IV-B. The circuit-graph [35] library is utilized to translate hardware designs into desired graph structures and craft synthetic designs. Knowledge extracted from benchmarks and hidden layer outputs are formatted into PyTorch tensors to make it compatible with ML models requiring tensor inputs.

We explored a wide range of Trojan types with combinational, sequential, as well as hybrid triggers. Specifically, the triggers are constructed using one or more of the following attributes:

- Rare signals
- Non-rare signals
- Rare branches
- Rare FSM states
- Rare FSM transitions
- Synchronous counter (incremented by clock)
- Asynchronous counter (increment by events)
- Both synchronous and asynchronous counters
- Sequences of rare events (e.g., nested rare branches)

Figure 11 shows an example Trojan with a hybrid trigger, which consists of both combinational (rare and non-rare signals) and sequential (synchronous and asynchronous counters) events.

##### B. Evaluation Method

To construct the benign and malicious sample dataset, traditional approaches randomly sample rare trigger conditions



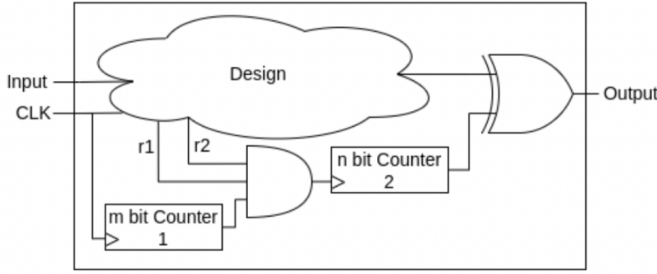


Fig. 11: A sample Trojan from our experiment, it uses hybrid trigger with rare signal (r1), non-rare signal (r2), synchronous counter (Counter 1) and asynchronous counter (Counter 2). A specific value of Counter 2 is the trigger for this Trojan.

of each benchmark using ATPG, and individually integrate these conditions into the original design to construct a design under test (DUT). However, such an imbalanced dataset can lead to data bias problems. For example, if we generate 1000 DUTs with one trigger condition for each benchmark, even a naive ML model (always-true prediction) can reach  $> 99.99\%$  accuracy, since most of the inputs are designs with implants. It indicates that accuracy is not sufficient to demonstrate the effectiveness. To overcome this problem, we employ the following two strategies. First, rather than relying on accuracy as the metric, we consider the following four metrics:

- **Recall:**  $\frac{tp}{tp+fn}$
- **Precision:**  $\frac{tp}{tp+fp}$
- **Accuracy:**  $\frac{tp+tn}{tp+tn+fp+fn}$
- **F1 Score:**  $\frac{tp}{tp+\frac{1}{2}(fp+fn)}$

where  $tp$ ,  $tn$ ,  $fp$  and  $fn$  are the number of true positive, true negative, false positive and false negative accordingly. Intuitively, recall is a measure of a classifier’s exactness, while precision is a measure of a classifiers completeness, and F1 score is the harmonic mean of recall and precision. Our second strategy is to generate synthetic samples. We use circuitgraph [35] to automatically synthesize benign circuit netlists. In order to ensure that the dataset has fifty-fifty distribution of benign and malicious samples, we randomly sample designs from both standard and synthesized benchmarks to integrate trigger conditions to construct DUTs. Although the nodes of Trojans are significantly smaller than the golden circuits, the extracted features from golden and malicious benchmarks are significantly different. This is due to the fact that when Trojans are injected into circuits, the connectivity and the testability of all reachable nodes are changed. In summary, our proposed framework mitigates the data bias by maintaining balanced distributions over feature, category, as well as evaluation metrics.

Given the above configurations, we evaluate the performance of the following four ML-based HT detection schemes:

- **SVM [24]:** Supervised learning based HT detection utilizing support vector machine (SVM).
- **RF [25]:** Supervised learning based HT detection utilizing random forest (RF).
- **HTNet [27]:** State-of-the-art approach based on transfer learning that uses unsupervised learning (KNN) to achieve golden-free detection .

- **COTD [15]:** Unsupervised learning based HT detection utilizing SCOAP parameters and k-means clustering.
- **TD-Zero:** Our proposed HT detection technique.

We also compare with the following three Trojan detection approaches that do not use machine learning.

- **VeriTrust [36]:** A hardware Trust verification framework based on examining verification corners.
- **ICAS [37]:** An extensible framework for estimating the susceptibility of IC layouts to additive Trojans
- **FANCI [38]:** An algorithm for identifying malicious logic using Boolean functional analysis

Since our proposed method relies on structural features of the circuits, we also compare with the following state-of-the-art structural HT detection methods:

- **SFHC [39]:** A HT detection method based on structural features and host circuits.
- **RF [40]:** A HT detection method based on the structural features using random forests.
- **LGB [41]:** HT detection using Light Gradient Boosting based on structural features and SCOAP values.

### C. Comparison of Detection Performance

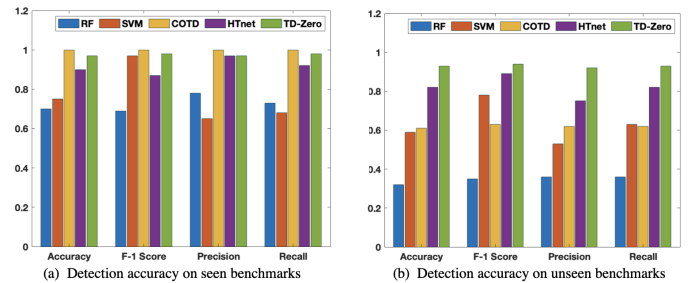


Fig. 12: Comparison of HT detection performance by applying various methods on both (a) seen and (b) unseen benchmarks.

Table III compares the performance of various detection schemes using accuracy (ACC), false positive rate (FPR), and false negative rate (FNR). Figure 12 presents the bar plots to record the accuracy, recall, precision and F1 scores of each method. To better evaluate the transferability of each approach, two different test datasets are provided. One is composed of known (seen) benchmarks during training phase, while another consists of unseen (unknown) benchmarks.

By “unseen”, we refer to benchmarks with different sources as well as functionalities. For example, we trained our model using s13207 from ISCAS89 [34] dataset with injected Trojan that causes functional corruption, and tested it using AES-T1100 benchmark from Trust-Hub [31] implanted with Trojan that causes information leakage. This strategy can remove bias from our model. It also tests the speculation capacity of the proposed zero-shot learning method.

As we can see from Table III, the SVM model achieves 75% accuracy for known samples, and RF model achieves 70%. Their performances falls behind HTNet and proposed method. This observation is supported by the intrinsics of RF. RF’s worst performance is expected since it makes decisions based

TABLE III: Comparison of HT detection accuracy (ACC), false positive rate (FPR), and false negative rate (FNR). While SVM and RF require golden models, HTNet, COTD and TD-Zero are golden-free HT detection methods.

Methods	SVM [24]			RF [25]			HTNet [27]			COTD [15]			TD-Zero			improv
	ACC	FPR	FNR	ACC	FPR	FNR	ACC	FPR	FNR	ACC	FPR	FNR	ACC	FPR	FNR	
Known	0.75	0.13	0.12	0.70	0.25	0.05	0.90	0.07	0.03	<b>1.00</b>	0.0	0.0	0.97	0.02	0.01	-0.03%
Unknown	0.59	0.27	0.14	0.32	0.33	0.035	<b>0.82</b>	0.07	0.11	0.61	0.23	0.16	0.93	0.05	0.02	13.4%
Average	0.67	0.20	0.13	0.51	0.29	0.20	<b>0.86</b>	0.07	0.07	0.80	0.12	0.08	0.95	0.35	0.15	10.5%

on a sequence of logical selections which is unreliable for HT detection and vulnerable to obfuscation. Their problems are revealed by their relatively low *recall* scores, which are  $< 70\%$ . Intuitively, a low *recall* score inflicts high proportion of false negative which demonstrates a high chance to bypass HT detection. This limitation comes from their poor expressive ability compared to HTNet and proposed method. When it comes to unknown benchmarks, these problems are further aggravated and their performance declines (even below 60%). This is expected as both SVM and RF rely on local knowledge from training samples to make decisions. When it comes to unknown benchmarks, any new features will appear ambiguous to them. Both SVM and RF possess a very low *precision* for unseen samples. It represents the problem that they tend to predict benign inputs as malicious. Features from unseen benchmarks are highly-likely to be classified as malicious since they are different from those of known samples.

A limiting factor for SVM and RF based Trojan detection is that they require golden reference models. Specifically, these two supervised ML methods requires Trojan-free (golden) designs during the training phase. This is unrealistic in IP-based SoC design framework since the SoC design house has only one version of each IP, which may or may not have Trojan. Our golden-free solution using zero-shot learning addresses this fundamental challenge. Although golden designs are necessary during the training phase, our zero-shot learning framework does not require any golden model during the testing phase. As discussed in Section II-D, any structural alternation or unnatural testability changes due to malicious implants are extracted as general knowledge in zero-shot learning. So when a new class that was not present in the training data needs to be recognized, the model uses its understanding of the general knowledge to make predictions. By comparing the attributes of the new class to those of the known classes, the model can infer which category the new class is closer to.

HTNet address the challenge of golden-free in a different way. HTNet applies transfer-learning, where lightweight re-training is performed for each unseen benchmark, so that they can maintain a decent 87% accuracy. However, it still suffers from relatively low 0.75 *recall* score for unknown samples. In contract, the proposed method applies zero-shot learning, which requires no re-training to obtain an accuracy as high as 93%, with *recall* of 0.92 and *precision* of 0.93, respectively.

COTD is another golden-free detetcion framework, which applies k-means clustering algorithm for classification based on *Euclidean distance*. As discussed in Section II-C, Euclidean distance is not beneficial for Trojan detection. Though COTD achieves 100% accuracy for known benchmark circuits, however, it fails miserably (61% accuracy) for unseen/unknown benchmarks. This is primarily due to the fact that COTD directly applies the tuple of SCOAP parameters for detection

without preprocessing.

TABLE IV: Accuracy comparison between TD-Zero and non-ML methods.

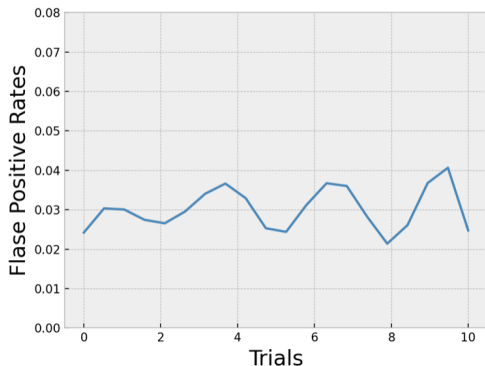
Methods	Acc (Seen)	Acc (Unseen)	Time (s)	Golden-Chips
VeriTrust	0.96	0.93	128.8	Yes
ICAS	0.92	0.47	13.2	Yes
FANCI	0.99	0.28	327.6	No
TD-Zero	0.97	0.93	44.6	No

Table IV compares the detection accuracy of our approach (TD-Zero) with the three non ML-based methods on seen and unseen benchmarks. The table also shows the requirement of golden reference model (chip) as well as the detection time (seconds). As we can see from the results, our proposed method achieves the overall best performance. VeriTrust also has good performance on unseen benchmarks since it is insensitive to the implementation style of HTs, but it requires the golden-reference model. ICAS is the fastest algorithm but is fragile on unseen benchmarks. This is due to the fact that ICAS is only designed and evaluated with three representative attacks from the literature as described in [37]. FANCI possesses the best accuracy for seen benchmarks but also lags behind on unseen benchmarks. The Boolean functional analysis restricts its applicability for Trojans with functional payload. Moreover, it identifies many normal signals as potentially suspicious signals. It is also most expensive in terms of detection time.

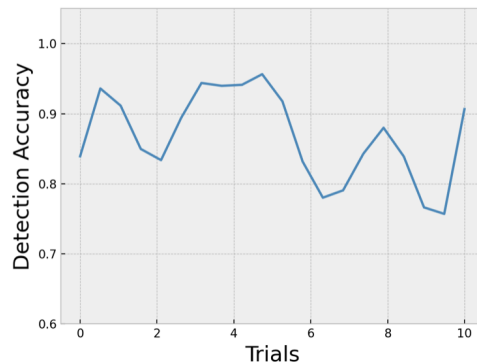
We have also performed evaluation for Trojan detection with structural feature-based approaches, including both normal and adversarial conditions. For normal detection, we composed our test set containing 1000 benchmarks as a hybrid of evenly distributed golden circuits and Trojan-implanted ones. For adversarial detection, we generated adversarial samples, i.e., Trojans with obfuscation, and we follow the routine in [42] by mixing 35% of adversarial samples into the test set to evaluate the robustness of all methods against HT with obfuscations. Table V shows that with respect to normal detection, our approach and the LGB model yield comparable results, both achieving an accuracy of 99%. However, the scenario takes a distinctive turn when exposed to adversarial samples, as a noticeable decline in performance becomes evident for the remaining three methods. Notably, the accuracy of both SHFC and RF plummets to levels lower than 60%, reminiscent of random guess. SHFC is a signature-based detection method relying on a pre-established library of templates, and the RF-based technique is a pattern recognition framework, they similarly struggle to distinguish between obfuscated patterns and authentic golden samples, incurring a substantial false negative rate in their evaluation results. Even the LGB model with ensemble learning strategy, the overall accuracy reduces to 64% , reflecting a notable drop in performance when grappling with the complexities introduced by adversarial patterns. In contrast, our proposed method retains acceptable accuracy (85%).

TABLE V: Detection performance for hardware Trojans with Structural Feature based Methods

Evaluation Measures	Normal Detection				Adversarial Detection			
	SFHC [39]	RF [40]	LGB [41]	Proposed	SFHC	RF	LGB	Proposed
Accuracy	0.79	0.85	0.93	0.93	0.55	0.58	0.64	0.85
Precision	0.82	0.87	0.94	0.90	0.60	0.77	0.75	0.88
Recall	0.78	0.84	0.92	0.97	0.54	0.56	0.62	0.83
F-Score	0.80	0.86	0.93	0.99	0.57	0.64	0.67	0.85
TPR	0.82	0.87	0.94	0.90	0.61	0.77	0.75	0.88
TNR	0.77	0.84	0.92	0.96	0.51	0.40	0.54	0.83
FPR	0.18	0.13	0.06	0.10	0.39	0.23	0.25	0.12
FNR	0.23	0.16	0.08	0.04	0.49	0.60	0.46	0.17



(a) The false positive rates of proposed method



(b) The detection accuracy against adversarial samples.

Fig. 13: The performance of our proposed method when dealing with non-malicious modifications. (a) The false positive rates when facing non-malicious modifications, and (b) the detection accuracy of proposed method against adversarial attacks including non-functional obfuscation modifications.

#### D. False Positive Evaluation

The false positive rates (FPRs) should be taken into consideration when dealing with non-malicious (non-Trojan) modifications. To evaluate this, we have injected extra gates to craft non-malicious samples. We also adopted the idea from [43] to craft adversarial samples, which can be considered as a combination of malicious implants and non-malicious modification for obfuscation purposes. We repeat the testing process for 10 trials to obtain comprehensive evaluation. The false positive rate is presented in Figure 13(a). For adversarial samples, the detection rate is presented in Figure 13(b).

As shown in the figure, the FPRs of our proposed method is constantly below 5%. This behavior highlights our method’s ability to accurately discriminate between malicious and non-malicious modifications. In terms of adversarial samples, our evaluation demonstrates the remarkable resilience of our model against the adversarial Trojans proposed in [43]. The attack works by introducing non-Trojan modification to mess up the feature patterns to confuse ML algorithms. However, the figure demonstrates that the performance of our model (detection accuracy) remains high even under various obfuscation related modifications. The success of our framework comes from the fact that it does directly compare the circuit structures, instead it extracts general knowledge of numerous malicious and benign benchmarks to grasp the key differences between them. In fact, the information of circuits are carried out and processed through the forward pass (FP) of graph convolutional network (GCN). For non-malicious modifications,

they cause little contribution to the feature map in hidden layers, therefore doesn’t significantly affect the final outputs.

To illustrate the impact of malicious as well as non-malicious modifications, we show the feature map spectrum of four circuits: the original circuit, the one with non-malicious modification, the one with malicious modification, and another malicious modification with obfuscation patterns. We also compute the similarity distance between each benchmark and the original one based on the Mahalanobis distance as described in Section III-C. For non-malicious changes (Figure 14(b)), the result circuit is much closer to the original one compared with Trojan implanted one (Figure 14(c)). In terms of adversarial samples (Figure 14(d)), since the malicious modification has major impact over the circuit, the similarity distance (Mahalanobis distance) from it to the original circuit is still far.

#### E. Knowledge Extraction Evaluation

The effectiveness of TD-Zero is also demonstrated by exploring its ability to extract general knowledge from benchmarks. We categorize data samples into four types, benign and malicious benchmarks that are already-seen during the training phase as well as the same distribution of those unseen benchmarks. Next, we gather the extracted knowledge for each of them. Figure 6 shows the extracted knowledge from our GCN flow is converted to a compact form of 32 features by dense layers. To better visualize the features, we first compute

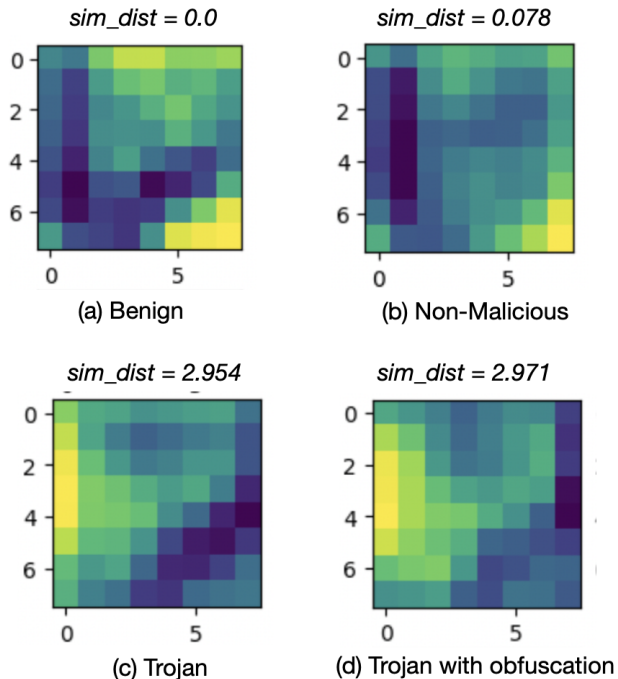


Fig. 14: The inner feature map of four different benchmark circuits. (a) The original benign benchmark, (b) benchmark with non-malicious modification, (c) benchmark with malicious modification (Trojan implanted), and (d) benchmark with Trojan implanted and non-Trojan obfuscation. The similarity distances are computed based on the Mahalanobis distance as outlined in Section III-C.

a real-valued parameter matrix  $L$  based on our learned metric  $M$  so that  $M = L^T L$ . Then notice:

$$\sqrt{(x - x')^T M (x - x')} = \sqrt{(Lx - Lx')^T (Lx - Lx')}$$

It indicates that, the Euclidean distance between  $Lx, Lx'$ , is equivalent to the Mahalanobis distance between  $x$  and  $x'$ . Therefore, the clusters and boundaries of knowledge can be intuitively visualized by plotting  $Lx$  values. Next, we apply *Principal component analysis* (PCA) to further reduce feature number to 3. PCA is a common data analysis method used for dimension reduction of high-dimensional data, and can be used to extract the main features for better illustration.

The visualization of features are presented in Figure 15. There is a clear separation for benign and malicious samples. Even for unseen benchmarks, our scheme remains functional. The extracted features from unseen malicious samples are much closer to seen malicious ones, and the same for benign benchmarks. This proves that TD-Zero can indeed extract general knowledge rather than individually-specific features, which explains the promising performance of TD-Zero for unseen benchmarks as shown in Table III.

#### F. Metric Learning Evaluation

With general knowledge obtained in previous steps, we evaluate the effectiveness of our metric learning scheme by comparing the HT detection performance with different metrics. In our evaluation, we take  $p = 1$  Minkowski (Manhattan),  $p = 2$  Minkowski (Euclidean), Chebyshev and Cosine

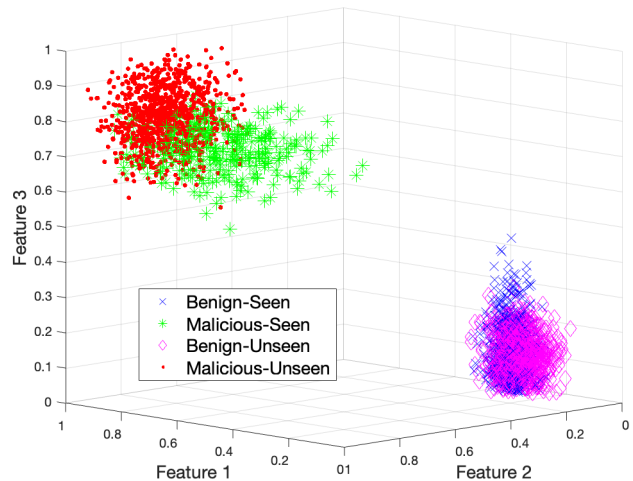


Fig. 15: The distribution of ternary features for four different class of samples, generated by performing dimension reduction (PCA) on extracted knowledge.

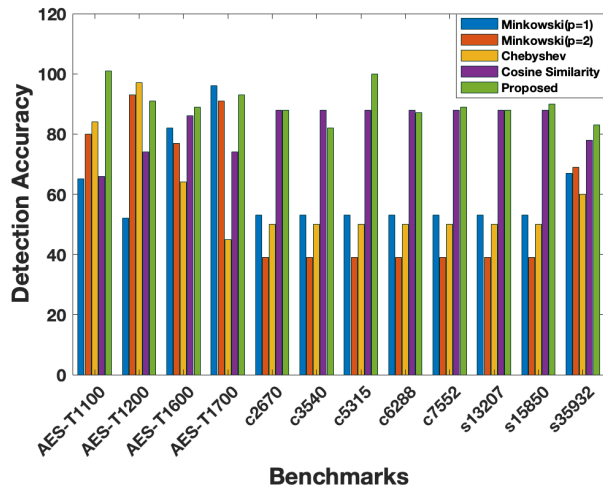


Fig. 16: Comparison of HT detection accuracy by applying various metrics on GCN’s extracted knowledge from several all involved benchmarks. Our proposed detection framework with metric learning provides averagely more than 90% accuracy for all tested benchmarks, including large scale circuits such as s15850 and s35932.

similarity. Results are presented in Figure 16. Even with extracted knowledge from GCN, the performance of the first four approaches suffer from instability. Instead, our proposed detection framework with metric learning scheme provides more than 90% accuracy for all tested benchmarks, including large scale circuits such as s15850 and s35932..

#### G. ML Overhead Analysis

Table VI compares the average overhead of various detection schemes. We present the required time, memory resources and CPU efficiency during training phase, along with the necessity for golden-chips during testing phase. The SVM approach is the most economic in terms of training cost. It



can be trained within 2 hours and only requires 17.55 W of power consumption. In contrast, HTNet is very costly, it needs more than 6 hours to complete training phase and it introduces highest power consumption. Our proposed method (TD-Zero) requires more memory space during training phase since it builds complicated structure than the others. However, TD-Zero performs better than HTNet in terms of training time as well as power consumption.

TABLE VI: Comparison of Training Cost and Data Resources.

Models	Time(s)	Memory(MB)	Power(W)	Golden-Chips
SVM	6133.1	67.8	17.55	Yes
RF	8074.2	299.9	32.55	Yes
COTD	9986.7	128.5	28.96	No
HTNET	44215.0	1024.0	54.16	No
TD-Zero	11377.6	1377.2	23.16	No

#### H. Scalability Evaluation

Scalability is another important aspect of performance evaluation, as scalability closely affects the usability of proposed method. Consequently, designers must ensure their HT detection scheme provides acceptable accuracy when deployed on large-scale of benchmarks. Figure 17 shows the variation of 4 different HT detection over each testing benchmarks. The number of signals of each benchmark is provided in Section IV-A for reference. To better evaluate the effect of scales, both small-scale designs (e.g., c2670) and large-scale designs (e.g., S38417) are considered.

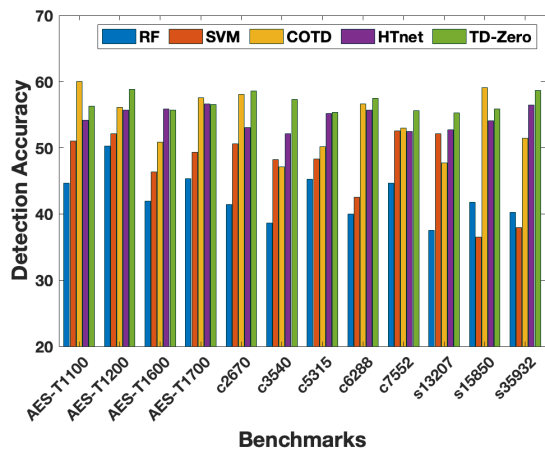


Fig. 17: Relative performance of RF (blue), SVM (orange), COTD (yellow), HTnet (purple), and TD-Zero (green) over different benchmarks.

As we can see from the histogram, model performance varies across designs. There are two major trends we can observe from the statistics. The first one is the inevitable decrease of model accuracy of light-weight models (SVM, RF). The simple structure of these two models is the bottleneck for feature extraction and expression. Both SVM and RF are based on sequential space separation, so only limited pattern of features can be recognized. When dealing with large benchmarks with overwhelming knowledge space, the complexity and amount of features exceed their capacity. The second observation is the relative stable performance of HTnet and TD-Zero. For HTnet, the transfer learning process guarantees the ML model always gets familiar with new features from unseen benchmarks. While for TD-Zero, we consider the idea

of extracting general knowledge as the essential advantage. In this way, ML models are always looking for key information from input, regardless of the unrelated parts of the design. Also, TD-Zero provides better overall performance compared to HTnet. Moreover, TD-Zero shows a reverse trend over SVM and RF, where its performance goes up when dealing with some of the large designs. We consider the common knowledge as the critical reason for this scenario.

#### V. CONCLUSION

While machine learning (ML) techniques are widely applied in hardware Trojan (HT) detection, they suffer from challenges such as requirement of golden chips, expert knowledge based feature selection, and unreliable distance metric. In this paper, we propose a novel hardware Trojan (HT) detection scheme using zero-shot learning, which requires no golden reference during testing (detection) phase. The trained model can be utilized for unseen benchmarks without any further re-training. Moreover, it provides stable accuracy for diverse benchmarks, and is able to adjust its distance measure through metric learning for improved performance. Extensive experimental evaluation using a wide variety of benchmarks demonstrated that our approach can achieve high detection accuracy while maintaining reliable recall and precision score. Our studies also reveal that our proposed framework outperforms state-of-the-art approaches in terms of detection accuracy (10.5% on average), stability as well as transferability, making it suitable for detecting unknown HTs in unseen benchmarks.

#### ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation (NSF) grant CCF-1908131.

#### REFERENCES

- [1] J. Lowdermilk *et al.*, "Towards zero trust: An experience report," in *2021 IEEE Secure Development Conference (SecDev)*, 2021, pp. 79–85.
- [2] Ma *et al.*, "Test generation for sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 10, pp. 1081–1093, 1988.
- [3] M. T. Ibn Ziad, M. A. Arroyo, E. Manzhosov, R. Piersma, and S. Sethumadhavan, "No-fat: Architectural support for low overhead memory safety checks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 916–929.
- [4] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3332–3344, 2016.
- [5] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 3–25.
- [6] Z. Pan, J. Sheldon, and P. Mishra, "Test generation using reinforcement learning for delay-based side-channel analysis," in *ICCAD*, 2020.
- [7] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 408–413.
- [8] Z. Pan, J. Sheldon, C. Sudusinghe, S. Charles, and P. Mishra, "Hardware-assisted malware detection using machine learning," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1775–1780.
- [9] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty, and S. Bhunia, "Tesor: A robust temporal self-referencing approach for hardware trojan detection," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2011, pp. 71–74.



- [10] Y. Zheng, S. Yang, and S. Bhunia, "Semia: Self-similarity-based ic integrity analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, pp. 37–48, 2016.
- [11] Y. Liu, K. Huang, and Y. Makris, "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.
- [12] T. Hoque, S. Narasimhan, X. Wang, S. Mal-Sarkar, and S. Bhunia, "Golden-free hardware trojan detection with high sensitivity under process noise," *Journal of Electronic Testing*, vol. 33, no. 1, pp. 107–124, 2017.
- [13] A. Vakil, "Golden-chip free side channel delay analysis test for hardware trojan and recycled ic detection," Ph.D. dissertation, George Mason University, 2021.
- [14] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: Identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of ACM SIGSAC Conference on Computer Communications Security*. New York, NY, USA: ACM, 2013, pp. 697–708.
- [15] H. Salmani, "Cotd: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 338–350, 2016.
- [16] B. Cakir and S. Malik, "Hardware trojan detection for gate-level ics using signal correlation based clustering," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 471–476.
- [17] Chen *et al.*, "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1370–1383, 2018.
- [18] Y. Jin, D. Maliuk, and Y. Makris, "Post-deployment trust evaluation in wireless cryptographic ics," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 965–970.
- [19] X. Chen *et al.*, "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1370–1383, 2017.
- [20] X. Chen, L. Wang, Y. Wang, Y. Liu, and H. Yang, "A general framework for hardware trojan detection in digital circuits by statistical learning algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1633–1646, 2016.
- [21] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, "Detection technique for hardware trojans using machine learning in frequency domain," in *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*. IEEE, 2015, pp. 185–186.
- [22] R. Elnaggar, K. Chakrabarty, and M. B. Tahoori, "Run-time hardware trojan detection using performance counters," in *2017 IEEE International Test Conference (ITC)*. IEEE, 2017, pp. 1–10.
- [23] E.-R. Zhou, S.-Q. Li, J.-H. Chen, L. Ni, Z.-X. Zhao, and J. Li, "A novel detection method for hardware trojan in third party ip cores," in *2016 International Conference on Information System and Artificial Intelligence (ISAI)*. IEEE, 2016, pp. 528–532.
- [24] K. Hasegawa, M. Yanagisawa, and N. Togawa, "A hardware-trojan classification method using machine learning at gate-level netlists based on trojan features," *IEICE*, vol. 100, no. 7, pp. 1427–1438, 2017.
- [25] —, "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.
- [26] A. Kulkarni, Y. Pino, M. French, and T. Mohsenin, "Real-time anomaly detection framework for many-core router through machine-learning techniques," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 1, pp. 1–22, 2016.
- [27] S. Faezi, R. Yasaei, and M. A. Al Faruque, "Htnet: Transfer learning for golden chip-free hardware trojan detection," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1484–1489.
- [28] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.
- [29] G. J. McLachlan, "Mahalanobis distance," *Resonance*, vol. 4, no. 6, pp. 20–26, 1999.
- [30] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.
- [31] "TrustHub.org: Trust-HUB,," <http://trust-hub.org/benchmarks/trojan>.
- [32] Y. Lyu and P. Mishra, "Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling," *TCAD*, 2021.
- [33] "ISCAS85 benchmarks," <https://filebox.ece.vt.edu/~mhsiao/iscas85.html>.
- [34] "ISCAS benchmarks," <https://filebox.ece.vt.edu/~mhsiao/iscas89.html>.
- [35] Sweeney *et al.*, "Circuitgraph: A python package for boolean circuits," *Journal of Open Source Software*, 2020.
- [36] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "Veritrust: Verification for hardware trust," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–8.
- [37] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "Icas: an extensible framework for estimating the susceptibility of ic layouts to additive trojans," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1742–1759.
- [38] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 697–708.
- [39] Q. Liu, P. Zhao, and F. Chen, "A hardware trojan detection method based on structural features of trojan and host circuits," *IEEE Access*, vol. 7, pp. 44 632–44 644, 2019.
- [40] T. Kurihara and N. Togawa, "Hardware-trojan detection based on the structural features of trojan circuits using random forests," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 105, no. 7, pp. 1049–1060, 2022.
- [41] R. Sharma, G. Sharma, M. Pattanaik, and V. Prashant, "Structural and scoop features based approach for hardware trojan detection using shap and light gradient boosting model," 2023.
- [42] S. Zhou, C. Liu, D. Ye, T. Zhu, W. Zhou, and P. S. Yu, "Adversarial attacks and defenses in deep learning: From a perspective of cybersecurity," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–39, 2022.
- [43] Nozawa *et al.*, "Generating adversarial examples for hardware-trojan detection at gate-level netlists," *Journal of Information Processing*, vol. 29, pp. 236–246, 2021.



**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.