Automated Hardware Trojan Detection at LUT Using Explainable Graph Neural Networks

Lingjuan Wu[†], Hao Su[‡], Xuelin Zhang[†], Yu Tai[‡], Han Li[†] and Wei Hu[*‡]
[†]*College of Informatics, Huazhong Agricultural University, Wuhan 430070, China*
[‡]*School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China*

***Abstract*—Trojan horses represent a major threat to hardware security and trust. In this work, we propose a novel hardware Trojan detection method based on explainable graph neural networks (GNNs) targeting FPGA netlists. We leverage the rich explicit structural features and behavioral characteristics at LUT, which offers an ideal abstraction level and granularity for Trojan detection. A GNN model with optimized class-balanced focal loss is trained for automated Trojan feature extraction and classification. Based on the Granger causality theory, we develop an interpretable approach to explain the decision mechanism of our GNN model. Experimental evaluations using 927 Trust-Hub hardware Trojan benchmarks and 262 Trojan free open source IP cores show that the proposed method provides promising detection results with accuracy, precision and F1-measure of 98.78%, 99.69% and 99.23% for Xilinx FPGA netlists while 97.93%, 97.87% and 98.51% for Intel FPGA netlists respectively. The experiment results have demonstrated that the proposed explainable approach can successfully identify the essential components that contribute to accurate Trojan classification and provide interpretable explanation for the GNN model.**

***Index Terms*—Hardware security, hardware Trojan detection, graph neural networks, FPGA netlist, look-up-table.**

I. Introduction

Under a globalized semi-conductor supply chain, modern integrated circuit (IC) and system-on-chip (SoC) designs typically employ off-the-shelf IP components to reduce design effort and cost. These IP products are usually delivered by vendors of different trust levels and may contain undocumented malicious design modification or unspecified functionality known as hardware Trojans. Trojans are oftentimes carefully crafted to prevent detection and activated only under rare events. When triggered, Trojans can create covert channels to leak secrets, downgrade performance or cause service interruption, which pose severe threats to hardware security and dependability. Literature has reported that hardware Trojans can even reside in highly critical hardware designs used in military and industrial infrastructures [1], [2].

Various hardware Trojan detection techniques have been developed by the hardware security research community, such as side-channel analysis (SCA), logic testing, reverse engineering, switching probability analysis and formal security verification. However, SCA based approaches [3], [4] assume availability of hard-to-obtain golden chips and are very sensitive to the size of the Trojan as well as process variations. Logic testing based techniques [5], [6] see shortcomings in having to explore the inordinately large design space to activate the Trojan. Destructive reverse engineering based approaches [7] require the support of specialized equipment and thus are very expensive. Switching probability analysis based methods [8]–[10] can see high false positive rates and may fail to detect the multiple signal triggered Trojans. Formal security verification based solutions [11]–[13] fall short in relying on the quality of assertion properties and may suffer from scalability issues due to state space explosion. Since security flaws like Trojans are extremely difficult to patch after chip fabrication, it is crucial to detect and eliminate them early in the design phase.

Recently, a clear trend in hardware Trojan research is employing machine learning and artificial intelligence techniques for both Trojan design [14], [15] and detection [16]–[19]. Various machine learning networks and classification algorithms, e.g., multi-layer neural networks [16], clustering algorithms [17], gradient boosting algorithms [18], probabilistic neural networks [19], have been developed for Trojan detection at RTL and gate-level. However, these works usually require expert knowledge for manual feature selection, extraction and quantification, which brings challenges in automating the detection process. GNN models are deep neural networks and have achieved remarkable success in the domain of graph data analysis [20]. A recent work leverages GNNs to extract Trojan features [21], in which two GNN models are trained for RTL and gate-level Trojan detection respectively. To the best of our knowledge, existing machine learning based Trojan detection methods typically focus on ASIC design flow. The Trojan features at LUT are overlooked, considering that it requires more effort to dump FPGA netlists. Moreover, the neural networks used in the above mentioned solutions are lack of explainability, where the networks are often regarded as black box and the Trojan detection results cannot be interpreted in a meaningful way.

In this work, we investigate the hardware Trojan detection problem from the viewpoint of FPGA netlist using explainable GNNs. We automate the adjacency matrix and node embedding generation process from FPGA netlist, which carries rich structural features and behavioral characteristics. We further develop an explainable GNN model for automated hardware Trojan feature extraction and classification. Specifically, we explore the Granger causality theory to analyze the contribution of each node to the Trojan classification result, which

Corresponding author: Wei Hu, email: weihu@nwpu.edu.cn. This research is supported in part by the National Key R&D Program of China under grant 2021YFB3100900, the National Natural Science Foundation of China under grant 62074131 and the Fundamental Research Funds for the Central Universities of China under Grant 2662022XXYJ005.

helps to explain the prediction mechanism of our GNN model. Experimental evaluations using over 900 Trust-Hub benchmarks show that the proposed method provides promising hardware Trojan detection results and interpretable explanation for the GNN model.

The major difference that distinguishes this work from existing solutions lies in that we perform Trojan detection at the level of look-up-table (LUT), where the Trojan features are more explicit than those in RTL code and Boolean gate netlist. In addition, this work makes the first endeavor to use explainable GNN model for hardware Trojan detection. Specifically, this work makes the following contributions:

- Proposing a novel machine learning based hardware Trojan detection method targeting FPGA netlists;
- Developing an efficient GNN model with optimized class-balanced focal loss for automated hardware Trojan feature extraction and classification;
- Providing a Granger causality based explainable approach for the GNN model, which leads to interpretable Trojan detection results;
- Presenting experimental results to demonstrate the effectiveness of the proposed method in terms of hardware Trojan detection and GNN model explanation.

The remainder of this paper is organized as follows. Section II reviews related work in hardware Trojan detection. In Section III, we describe our threat model. Section IV proposes an efficient and explainable method for automated hardware Trojan detection leveraging GNN at LUT level. We present experimental results using Trust-Hub benchmarks in Section V and conclude the paper in Section VI.

II. Related Work

In this section, we briefly review the related work in design-phase hardware Trojan detection.

Functional verification based approaches aim to generate test vectors to maximize the probability of activating the Trojan [22], [23]. Switching probability analysis based methods perform Trojan detection through structural and behavioral analysis leveraging the fundamental property that Trojans are rarely triggered. FANCI [8] uses Boolean functional analysis to identify the malicious circuit based on the observation that hardware Trojans usually incorporate stealthy logic that almost never influences the primary output. VeriTrust [9] detects the parasite-based Trojans by identifying the dedicated triggers through examining verification corners. AVFSM [10] employs automatic test pattern generation to extract the state transition graph to detect the Trojans inserted to finite state machines. The existing solutions exploit the low switching activity characteristic for Trojan detection. However, these methods can be sensitive to the implementation style of the Trojans and may fail to detect the Trojans with multiple discrete triggers.

Formal verification based techniques detect Trojans by developing hardware security models and capturing the malicious behaviors that violate the security property such as integrity or confidentiality. For example, researchers developed information flow tracking models at RTL and gate-level for Trojan detection [11], [12]. Jin *et al.* proposed the enhanced proof-carrying hardware IP framework with secrecy tags to provide mechanisms for security verification [13], in which the Trojans are detected by formally proving that sensitive information does not reach undesired sites. However, the existing approaches rely on manually defined properties, in which the quality and completeness of the assertion properties can heavily influence the Trojan detection performance. Another limitation is that formal methods may suffer from scalability issues and cannot be easily used for large-scale circuits.

Recently, a large body of works employ machine learning for hardware security in which the Trojan detection process is considered as a binary classification problem and the IP cores under test are classified into Trojan-free and Trojan-infected ones. Hasegawa *et al.* proposed a multi-layer neural network based method by extracting five-dimensional features in gate-level netlist and classifying the unknown nets into Trojan and normal ones [16]. Salmani employed the unsupervised clustering algorithm for gate-level Trojan detection based on the controllability and observability feature [17]. Researchers also developed the gradient boosting algorithm [18] and the probabilistic neural networks [19] for Trojan detection at the design level of RTL leveraging the abstract syntax tree and control flow graph respectively. Although these solutions can successfully detect specific types of Trojans, they suffer from several shortcomings such as heavily reliance on expert knowledge for Trojan feature extraction, requiring reference designs and unable to detect new Trojans. A more recent work utilizes GNN to automatically extract circuit structural features for Trojan identification [21]. However, this method requires tedious work in converting the original IP cores to data flow graph through semantic analysis. Moreover, the existing solutions provide Trojan detection results without interpreting them in a human understandable way, which hinders theoretical reasoning about their effectiveness.

Explainable machine learning techniques aim to provide interpretable explanation for the classification results by illustrating why an instance is classified into a specific class. Recently, some efforts have been made in the security vulnerability detection domain using explainable approaches. Pan *et al.* developed an explainable model for Spectre and Meltdown detection by exploiting the temporal differences of sequential hardware events [24]. A linear regression based explainable approach was proposed for hardware-assisted malware detection and malicious behavior localization [25]. Another work developed an explainable hardware Trojan detection framework based on the Shapley Ensemble Boosting algorithm, in which Shapley analysis was performed to evaluate the impact of each feature towards the prediction result [26]. However, that work requires manual Trojan feature selection, which makes the explanation results dependent on expert knowledge.

To overcome these limitations, we propose a novel explainable machine learning based hardware Trojan detection method targeting FPGA netlists. The key observation is that at the level of LUT, the topological graph and initialization

value represent explicit structural features and behavioral characteristics, which offers an ideal abstraction level for Trojan detection. A GNN model with optimized class-balanced focal loss is developed for automated Trojan feature extraction and classification. We further employ the Granger causality theory [27] to provide interpretable explanation for the classification mechanism of the GNN model. To the best of our knowledge, this is the first work that creates an explainable GNN model for pinpointing hardware Trojans and achieves interpretable detection results.

III. Threat Model

We consider a typical IC or SoC design flow, where IP cores from global third-parties are integrated to reduce the design time and effort. We assume that the untrusted IP designers or electronic design automation (EDA) tools may intentionally make malicious modifications that are out of the specification of the IP product. We assume that the hardware Trojans are carefully crafted and triggered only under rare events to escape from being detected; they are inserted into the IP cores and delivered to the consumers in the form of RTL code or gate-level netlist. We assume that the consumers have access to the design source of the IP cores and can perform functional verification as well as various hardware Trojan detection techniques according to the specification.

We focus on Trojan detection in the early design phase and assume that we have access to the RTL source code or gate-level netlist and its technology library of the IP cores with Trojan infected. We can synthesize the IP cores using FPGA synthesis tools such as the *Xilinx Vivado* or open source *Yosys* toolkit in order to dump the FPGA netlist for further Trojan detection analysis. Our threat model is similar to those employed in previous works in design level hardware Trojan detection [9], [12], [21].

IV. Automated Trojan Detection Method

This section starts from a high-level overview of the proposed hardware Trojan detection method. It then elaborates the topological graph analysis of the FPGA netlist, followed by the GNN model design, the loss function optimization and the explanation method respectively.

A. Overview

This work employs GNN for automated hardware Trojan feature extraction and detection, which involves solving a binary classification problem at the level of LUT. Fig. 1 illustrates the general flow of the proposed Trojan detection method. Firstly, the IP cores in the form of RTL code or gate-level nestlist are flattened and synthesized into FPGA netlists using an FPGA synthesis tool. Secondly, we analyze the topological graph of the FPGA netlists automatically using our self-developed script, build the adjacency matrix and generate node embedding to represent the connections and node types. Thirdly, the matrix created is processed by the pre-trained GNN model and the IP core under test is classified as Trojan-free or Trojan-infected.

B. Adjacency Matrix Construction at LUT

In this paper, we consider an FPGA design flow and use the open source synthesis tool *Yosys* to synthesize IP cores to FPGA netlists for graph analysis. The observation is that mapping RTL source code or gate-level netlists to LUT networks will make both the circuit structural features and behavioral characteristics explicit, which can significantly improve the Trojan detection performance. Specifically, the initialization value of an LUT reflects its design functionality, internal design structure and switching behaviors. In addition, FPGA netlists have much smaller number of nodes as compared to the gate-level netlists, which can reduce data analysis complexity and improve scalability. For example, the AES-T400 IP core in *Trust-Hub.org* consists of 120912 and 25900 nodes when synthesized to Boolean gates and Xilinx FPGA netlists respectively.

A hardware design is a non-Euclidean structure and can be represented as a directed graph consisting of nodes and edges. We develop a script to parse the FPGA netlist, build the adjacency matrix and generate node embedding automatically. Our script first constructs a list consisting of all the cells in the netlist. It then performs topological graph optimization to eliminate those nodes with only one-bit input and output signal such as IBUFs and OBUFs. This reduces the number of nodes without affecting the circuit graph structure. Finally, we traverse the optimized graph and assign a unique label for each node. The adjacency matrix is generated according to the unique label and the connection of the nodes. We further assign a type label to each node based on its type and initialization value. For example, the unique and type labels of LUT4 with initialization values of 16'h0001, 16'h0010 and 16'h0010 are set to 0 0, 1 1 and 2 1 respectively. The node embedding is in one-hot encoding and is generated based on the unique and type labels.

C. GNN Model

With the adjacency matrix and node embedding generated for each IP core, we train a GNN model for automated hardware Trojan feature extraction and classification. As illustrated in Fig. 1, the GNN model employed in this paper contains four major layers including the graph convolution and activation, self-attention pooling, readout and fully connected layers.

Given the original input graph G, we first obtain its adjacency matrix $A \in \mathbb{R}^{N\times N}$ and initial node embedding $X \in \mathbb{R}^{N\times P}$, where N is the number of nodes and P is the number of node types. Denote $\hat{y}$ as the classification result of the GNN model:

$$\hat{y} = \begin{cases} (0,1), & \text{G is Trojan-infected} \\ (1,0), & \text{G is Trojan-free} \end{cases}. \tag{1}$$

The four major layers and operations of the GNN model are described in the following.

Graph Convolution and Activation: The first step can be considered as node embedding operation through three layers of graph convolution and activation. Denote $X^{(i)}$ as the node

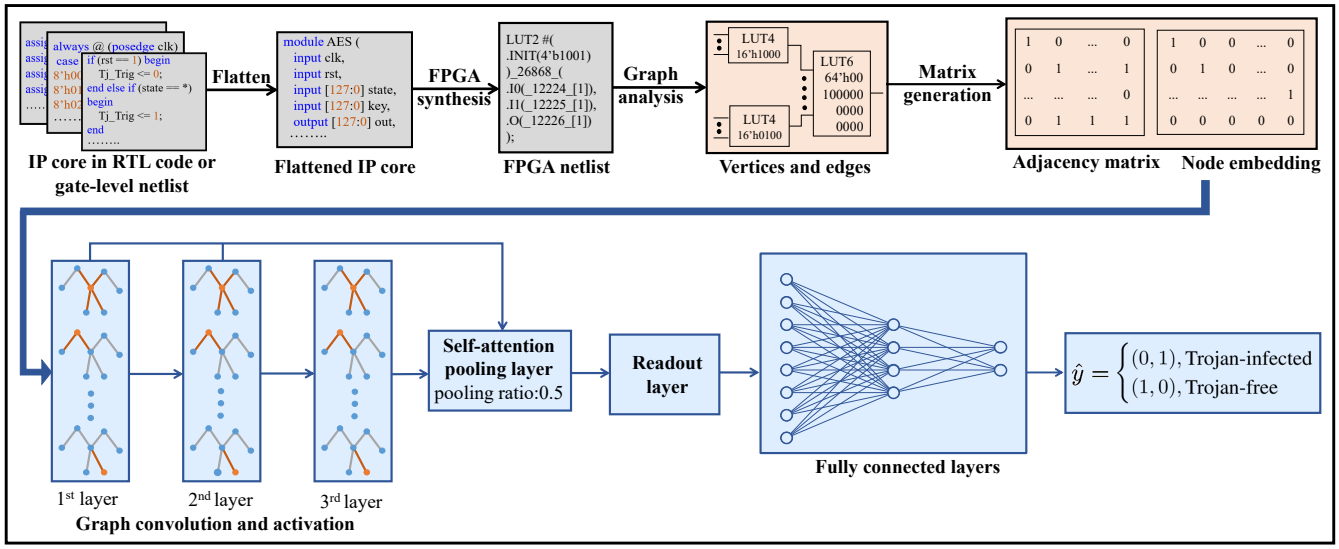

Fig. 1. Automated hardware Trojan detection flow leveraging GNN at the level of LUT.

embedding of the i^{th} layer, where $i \in [1,2,3]$, $X^{(0)} = X$ and $I \in \mathbb{R}^{N \times N}$ is an identity matrix. Then we have

$$X^{(i)} = \delta\left(D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}X^{(i-1)}\theta^{(i)}\right) \tag{2}$$

and

$$X_{conv} = X^{(1)} \| X^{(2)} \| X^{(3)}, \tag{3}$$

where $\|$ denotes the concatenation operation, δ is the $ReLU$ activation function, D is the diagonal degree matrix of $A+I$ and $\theta^{(i)}$ is the learnable weight variable in the i^{th} graph convolution layer. $A+I$ is the self-connected adjacency matrix where the identity matrix I allows the features calculated in previous layer to be transferred to the current layer.

Self-attention Pooling: We further process the node embedding X_{conv} through the self-attention pooling layer to select the nodes with important features and reduce the number of parameters in the GNN model.

Denote $IndexTop([a,b,c],k)$ as the index searching function to find the corresponding index set of the top-k elements in $[a,b,c]$. Given a ratio η (0.5 in our model), we adopt the top-k ($k = \lceil \eta N \rceil$) node selection strategy based on the ranking of the self-attention scores $Z = [Z_1, Z_2, \cdots, Z_N] \in \mathbb{R}^N$ in the graph convolution layer:

$$\begin{aligned} Z &= \sigma\left(D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}X_{conv}\Theta\right), \\ \text{IndexSet} &= IndexTop(Z, \lceil \eta N \rceil), \end{aligned} \tag{4}$$

where σ is the $Tanh$ activation function, Θ is the learnable weight variable in the pooling layer. We then obtain the updated adjacency matrix A_{pool} and node embedding X_{pool} calculated as:

$$\begin{aligned} A_{pool} &= \{A[i,j] | i \in \text{IndexSet and } j \in \text{IndexSet}\}, \\ X_{pool} &= \{X_{conv}[i,:] | i \in \text{IndexSet}\}. \end{aligned} \tag{5}$$

Readout Layer: In order to reduce the features of graphs with different sizes to a fixed dimension and perform graph classification, we further implement the global pooling operation with mean-pooling and max-pooling strategies [28], which is described as follows:

$$X_{readout} = \frac{1}{N^*}\sum_{i=1}^{N^*}\left[X_{pool}[i,:] \| \max X_{pool}[i,:]\right], \tag{6}$$

where $X_{pool}[i:]$ is the embedding vector of the i-th node (the i-th row in X_{pool}), and $N^* = \lceil \eta N \rceil$ is the number of rows in X_{pool}. The transformed node embedding $X_{readout}$ contains the average and maximum value from the prior layer.

Fully Connected Layer and Classification: In the last step, the node embedding $X_{readout}$ is processed through three fully connected layers followed by the activation operation for classification. A two-dimensional vector $\hat{y}$ is calculated to represent the predicted probabilities of two classes, i.e., Trojan-infected or Trojan-free.

$$\begin{aligned} \hat{y} &= \phi\left(g_3(g_2(g_1(X_{readout})))\right), \\ g_i(X_i) &= X_i W_i + b_i, \end{aligned} \tag{7}$$

where ϕ is the Softmax activation function and g_i is the linear transform with bias for $i \in [1,2,3]$. W_i and b_i are the target variables for optimization. Finally, after the Softmax normalization, the category with higher prediction probability will be considered as the classification result.

D. Class-Balanced Focal Loss

Since the distributions of Trojan-infected and Trojan-free datasets are usually imbalanced, we employ the advanced cost-sensitive class-balanced (CB) focal loss function [29] for GNN model training to address the class imbalance problem. Compared to the classical re-balancing strategies such as up-sampling and down-sampling, which may cause overfitting or lose important samples, CB focal loss achieves significant improvements in classification of imbalanced datasets by assigning larger weights to the minor classes.

Given an input graph G with class label y, let C be the total number of classes ($C = 2$ for binary classification as in this paper) and n_y be the number of samples in class y where $y \in \{1, 2, \cdots, C\}$. Suppose the predicted probabilities of different classes are $\mathbf{z} = [z_1, z_2, \ldots, z_i, \ldots, z_C]^\top$, where $z_i \in [0, 1]$. We define z_i^t as:

$$z_i^t = \begin{cases} z_i, & \text{if } i = y \\ -z_i, & \text{otherwise} \end{cases}. \tag{8}$$

Denote $p_i^t = \text{sigmoid}\,(z_i^t) = 1/(1 + \exp(-z_i^t))$. The mathematical formulation of the CB focal loss is as follows:

$$\text{CB}_{\text{focal}}\,(\mathbf{z}, y) = -\frac{1-\beta}{1-\beta^{n_y}} \sum_{i=1}^{C} \left(1 - p_i^t\right)^\gamma \log\left(p_i^t\right), \tag{9}$$

where $\beta \in [0, 1)$ and $\gamma \geq 0$ are the tuning parameters. Specifically, β controls the weights of imbalanced classes and γ smoothly adjusts the weights of hard samples that can be easily misclassified [29]. Therefore, in addition to the class imbalance problem, CB focal loss can also deal with hard samples by assigning them larger weights, which improves the robustness of our GNN model.

E. Explanation Method

In this work, we develop a method for GNN model explanation from the perspective of Granger causality, which reflects the relation between two variables when one leads to the other. Specifically, by applying the concept of Granger causality to the graph analysis domain, we say that there is Granger-cause between an arbitrary node N_i and the GNN model prediction variable $\hat{y}$, by comparing the prediction results with and without node N_i. We calculate the causal contribution of each node and create a compact subgraph, which is the most relevant portion towards the prediction result and thus provides explanations for the decisions of the GNN model. The Granger causality based explanation method can be mathematically formalized as follows.

For a hardware design represented as graph G with true label y, we use $G_{/N_i}$ to represent the new graph after removing node N_i and its associated edges from G, where $N_i \in G$. Given a pre-trained GNN model, we use $e(G)$ to denote the prediction error of the model with respect to G. Specifically, the causal contribution of node N_i to the output of the GNN model, denoted as $E(N_i)$, is defined as the decrease in prediction error between $e(G_{/N_i})$ and $e(G)$, formulated as:

$$E(N_i) = e(G_{/N_i}) - e(G). \tag{10}$$

We use $f(\cdot)$ to denote the prediction function of the pre-trained GNN model. Then, the corresponding prediction results of G and $G_{/N_i}$ are $f\,(G)$ and $f\left(G_{/N_i}\right)$ respectively. Let $\mathcal{L}(y, \hat{y})$ be the loss function for measuring the distance between the true label y and the prediction result $\hat{y}$. The prediction errors of G and $G_{/N_i}$ can be formulated as Eq. (11) and Eq. (12) respectively:

$$e(G) = \mathcal{L}\left(y, f(G)\right), \tag{11}$$

$$e(G_{/N_i}) = \mathcal{L}\left(y, f(G_{/N_i})\right). \tag{12}$$

Thus, the causal contribution of node N_i can be measured by the difference in loss between G and $G_{/N_i}$.

It should be noted that a large positive value of $E(N_i)$ indicates that node N_i is critical for the GNN model to make an accurate prediction, while a small $E(N_i)$ implies the insignificance of N_i. By traversing the nodes in G and ranking their causal contribution values, we can obtain the *Top-k* most important nodes, which compose a compact subgraph and provide interpretable explanations for the GNN model.

V. Experimental Results

In this section, we present evaluation results of the proposed hardware Trojan detection method. We first describe our experimental setup. We then report the GNN model training and Trojan classification results in Sections V-B and V-C respectively. We provide the Granger causality-based GNN model explanation results in Section V-D, and compare our method with state-of-the-art machine learning based Trojan detection techniques in Section V-E.

A. Experimental Setup

We collect a comprehensive dataset with 927 Trojan-infected and 262 Trojan-free samples, which are regarded as the positive and negative classes with labels of 1 and 0 respectively. The Trojan-infected samples are *Trust-Hub.org* benchmarks with Trojans inserted into AES, RS232, PIC, c2670, c3540, c5315, c6288, s1423, s13207, s15850 and s35932 base circuits, where the AES, RS232 and PIC series benchmarks are in the form of RTL code and the others are in gate-level netlist. The activation mechanism and payload of the Trojan-infected samples are described in Table I, where TRIT-TC represents 574 conditionally triggered combinational Trojans and TRIT-TS represents 329 FSM and counter triggered Trojans respectively. The Trojan-free samples are open source IP cores from *opencores.org* and different from the Trojan-infected base circuits. We randomly divide the dataset into training and testing sets with the ratio of 4:1.

In our test, we perform the GNN model training and Trojan detection targeting two different FPGA platforms, i.e., Xilinx and Intel FPGAs, to demonstrate the effectiveness of the proposed method. We use the *synth_xilinx* and *synth_intel* synthesis scripts in *Yosys* with the *flatten* and *nobram* options to map each benchmark to FPGA netlists targeting the Xilinx Virtex-7 and Intel MAX10 FPGAs respectively. We then invoke our self-developed script to analyze the generated FPGA netlists, build the adjacency matrix and generate node embedding for each benchmark, which are further used for model training or Trojan classification.

All the experiments are performed on a PC with an Intel i5 CPU and an Nvidia GeForce RTX 3060 graphics card with 12GB memory.

B. Training Results

We train the GNN model with CB focal loss through grid search and four-fold cross validation on the training

TABLE I
DESCRIPTION OF TROJAN-INFECTED SAMPLES.

Benchmark	Activation mechanism	Payload
AES-T500	A specific sequence of plaintext	Increase power
AES-T600	A specific input plaintext	Leak information
AES-T1200	Counter reaches a specific value	Leak information
AES-T1300	A specific input plaintext	Leak information
AES-T1400	A specific sequence of plaintext	Leak information
AES-T1500	Counter reaches a specific value	Leak information
AES-T1800	A specific input plaintext	Denial of service
AES-T1900	Counter reaches a specific value	Denial of service
AES-T2000	A specific sequence of plaintext	Leak information
AES-T2100	A specific number of encryption	Leak information
RS232-T100	A comparator over 19 signals	Denial of service
RS232-T200	A inserted counter triggered	Reduce design reliability
RS232-T300	A specific number of transmission	Leak information
RS232-T400	Data equals to a specific value	Leak information
RS232-T500	A specific number of transmission	Denial of service
RS232-T600	A specific sequence of states	Leak information and denial of service
RS232-T700	A specific sequence of states	Denial of service
RS232-T800	A comparator over 19 signals	Denial of service
RS232-T900	A specific sequence of states	Denial of service
RS232-T901	A specific sequence of states	Denial of service
PIC-T100	Execution of specific instructions	Change the address to program memory
PIC-T200	Execution of specific instructions	Replace the instruction register
PIC-T300	Execution of specific instructions	Change data lines to external EEPROM
PIC-T400	Execution of specific instructions	Change address lines to 0
TRIT-TC	Rare internal nodes activation	Alter randomly selected signals
TRIT-TS	Multiple trigger conditions required	Alter randomly selected signals

dataset. In our tests, we evaluate the performance of the model under the criteria of true positive rate (TPR), true negative rate (TNR), accuracy, precision and F1-measure. We perform hyperparameter selection under the criterion of maximum F1-measure. The main hyperparameter selection strategies and the optimal values of the GNN model trained for Xilinx and Intel FPGA netlists are described in Table II.

TABLE II
HYPERPARAMETER SELECTION OF THE GNN MODEL WITH CB FOCAL LOSS FOR XILINX AND INTEL FPGA NETLISTS.

Hyperparameter	Search range	Optimal value on Xilinx FPGA	Optimal value on Intel FPGA
learning rate	{0.01, 0.001, 0.0001}	0.001	0.01
hidden size	{16, 32}	32	32
batch size	{16, 32}	16	16
β of CB focal loss	{0.999, 0.99, 0.9}	0.99	0.9
γ of CB focal loss	{0.5, 1.0, 2.0}	0.5	0.5
weight decay	{0.0001, 0.001}	0.0001	0.0001
pooling ratio	{0.5, 0.8}	0.5	0.5

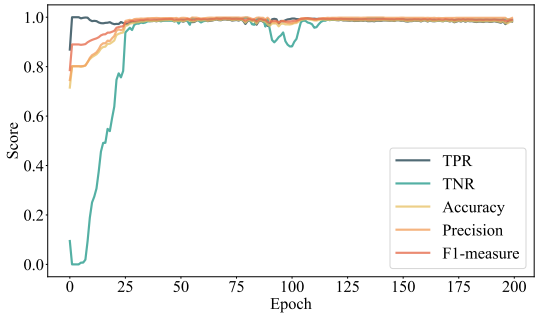

Fig. 2. The training process of the GNN model with CB focal loss under the optimal hyperparameters for Xilinx FPGA netlists.

Afterwards, we retrain the GNN model with CB focal loss under the optimal hyperparameters and perform evaluations on the validation dataset. The training process of the GNN model with CB focal loss for Xilinx and Intel FPGA netlists are shown in Fig. 2 and Fig. 3 respectively, where both GNN models converge fast within 200 epochs. We also train another two GNN models with classical cross-entropy (CE) loss following the parameter selection procedure of CB focal loss for Xilinx and Intel FPGA netlists respectively for comparison.

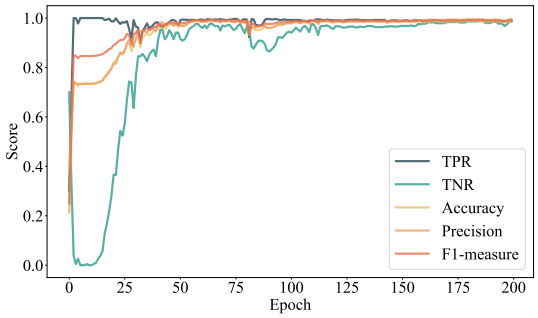

Fig. 3. The training process of the GNN model with CB focal loss under the optimal hyperparameters for Intel FPGA netlists.

The validation results are shown in Table III. Each experiment is independently repeated 10 times and the average value is reported.

TABLE III
VALIDATION RESULTS OF THE TRAINED GNN MODEL.

GNN model validation results for Xilinx FPGA netlists						
Loss function	Time (ms)	TPR	TNR	Accuracy	Precision	F1-measure
CE loss	1.57	98.46%	96.24%	97.90%	98.76%	98.61%
CB focal loss	1.39	99.32%	98.72%	99.20%	99.69%	99.50%
GNN model validation results for Intel FPGA netlists						
Loss function	Time (ms)	TPR	TNR	Accuracy	Precision	F1-measure
CE loss	2.57	99.46%	95.07%	98.31%	98.24%	98.85%
CB focal loss	2.55	99.48%	97.45%	98.93%	99.07%	99.26%

From Table III, the GNN model with CB focal loss achieves better Trojan classification performance with accuracy, precision and F1-measure of 99.20%, 99.69% and 99.50% for Xilinx FPGA netlists and 98.93%, 99.07% and 99.26% for Intel FPGA netlists respectively. These results demonstrate the effectiveness of the proposed loss function optimization strategy for the GNN model.

C. Classification Results

The classification results on the testing dataset using the trained optimal GNN model with CB focal loss are presented in Table IV. These results show that the proposed method can successfully detect the Trojans with accuracy, precision and F1-measure of 98.78%, 99.69% and 99.23% for Xilinx FPGA netlists while 97.93%, 97.87% and 98.51% for Intel FPGA netlists respectively on average.

The TPR (recall), which reflects the model's ability to correctly identify the Trojans, is calculated according to the parasite base circuits and shown in Fig. 4. For Xilinx FPGA netlists, the trained GNN model achieves the maximum TPR of 1 except the s13207 and s15850 benchmarks, for which

TABLE IV
HARDWARE TROJAN CLASSIFICATION RESULTS.

Trojan classification results for Xilinx FPGA netlists					
Time (ms)	TPR	TNR	Accuracy	Precision	F1-measure
1.42	98.79%	98.72%	98.78%	99.69%	99.23%
Trojan classification results for Intel FPGA netlists					
Time (ms)	TPR	TNR	Accuracy	Precision	F1-measure
2.69	99.17%	95.09%	97.93%	97.87%	98.51%

the TPRs are 0.9643 and 0.9655 respectively. For Intel FPGA netlists, the trained GNN model achieves the maximum TPR of 1 except the AES, RS232, c2670 and c3540 benchmarks, for which the TPRs are 0.95, 0.95, 0.9875 and 0.9833 respectively. For both FPGAs, the proposed method can successfully identify the Trojans with high TPR. These results validate our findings that LUTs carry rich explicit circuit structural features and behavioral characteristics, which provide an ideal abstraction level and granularity for automated Trojan feature extraction and classification.

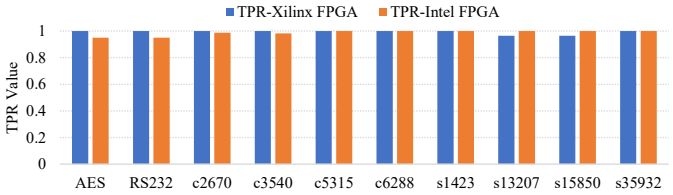

Fig. 4. The hardware Trojan classification results on various parasite base circuits under the criterion of TPR for Xilinx and Intel FPGA netlists.

D. Explanation Results

In this subsection, we report the GNN model explanation results and seek to answer the question: what are the key features and components that dominate the Trojan detection results of our GNN model. Using the Granger causality based explanation method introduced in Section IV-E, we calculate the casual contribution of each node in the testing dataset to identify the top-ranked subgraphs leading to the hardware Trojan classification results. Considering that a primary goal of explanation is to generate more compact subgraphs while maintaining the prediction result, the generated subgraphs are further fed into the pre-trained GNN model to evaluate the explanation performance. The prediction results of the GNN model on the explanation subgraphs for Xilinx and Intel FPGA netlists are shown in Table V, where the parameter *Top-k* represents the percentage of nodes selected according to the causal contribution value and TPR is reported as the evaluation criterion to measure the hardware Trojan characterization capability of the explanation results.

TABLE V
EXPLANATION TPR RESULTS UNDER DIFFERENT *Top-k* SETTINGS.

Top-k	5%	6%	7%	8%	9%	10%
TPR-Xilinx	42.45%	54.18%	57.54%	59.21%	64.24%	74.86%
TPR-Intel	54.31%	60.18%	68.07%	68.80%	71.74%	73.94%

As shown in Table V, the TPR consistently increases with the increase of *Top-k*, while more nodes with significant causal contribution are selected and fed into the GNN model. In particular, the TPRs reach 74.86% and 73.94% for Xilinx and Intel FPGA netlists respectively, when only the top 10% of nodes are selected for prediction. These results indicate that the proposed explanation method can successfully identify the key components in the entire FPGA netlists that lead to the prediction results of the GNN model.

To further check the interpretability of the explanation results, we illustrate the explanation subgraphs of the AES-T1300, AES-T1200, AES-T500 and c2670-T000 benchmarks from the Intel FPGA netlists with *Top-k* of 1%, 1%, 1% and 7% respectively. The specific values of *Top-k* are selected to generate meaningful compact subgraphs while maintaining the classification results of our GNN model. Note that an overly small node number may incur meaningless explanation. Therefore, 7% is selected for c2670-T000 since this benchmark is a tiny circuit with only 177 nodes when synthesized to Intel FPGA netlists. The explanation results including the subgraphs and Trojan triggers are shown in Fig. 5. It should be pointed out that the subgraph with connected nodes is visualized for each benchmark. The detailed information of each node is reported in Table VI, in which the type label is assigned when we build the adjacency matrix.

From Fig. 5, we can see that the key components that contribute to the prediction results of AES-T1300, AES-T1200 and AES-T500 are a cascade of LUTs with significantly biased initialization values. For example, the explanation subgraph that influences the prediction of AES-T1300, is a fanin cone of LUTs with initialization values of `16'b1000_0000_0000_0000` and `16'b0000_0000_0000_0001`. For c2670-T000, after carefully checking the compact subgraph and the Trojan logic, we find that nodes `A,B,C,K` are Trojan-related, which are connected with edges in black in Fig. 5(d). The rest of the subgraph connected with edges in gray are normal logic. The initialization value of LUT provides useful information for estimating the switching probability of its output signal. For example, the output signal of a LUT with the initialization value of `16'b1000_0000_0000_0000` only has the possibility of 1/16 to take the value of logical `1` when the address line inputs follow a unified distribution. A cascade of LUTs with significantly biased initialization values would render an extremely low switching signal. Therefore, these explanation results are consistent with the well-accepted fact that the rarely activated Trojans exhibit low switching activity. Specifically, we obtain the same explanation results for Xilinx FPGA netlists that the LUTs with significantly biased initialization values are the dominating components for Trojan classification in the GNN model.

These results demonstrate that the proposed Granger causality based explanation method can successfully identify the key components that contribute to accurate hardware Trojan classification and provide interpretable explanation for our GNN model.

TABLE VII
COMPARISON OF VARIOUS MACHINE LEARNING BASED HARDWARE TROJAN DETECTION METHODS.

Method	Target	Precision	TPR	F1-measure	Detection time (s)	Automated feature extraction	Reference free	Interpretable
Multi-layer neural network [16]	gate-level netlist	NA	85.3%	NA	NA	×	×	×
Unsupervised clustering [17]	gate-level netlist	NA	100%	NA	0.28	×	✓	×
Gradient boosting algorithm [18]	RTL code	NA	100%	NA	0.0017	×	×	×
Probabilistic neural network [19]	RTL code	NA	100%	NA	1.32	×	✓	×
Graph neural network [21]	RTL code	92.3%	96.6%	94.0%	0.026	✓	✓	×
	gate-level netlist	91.2%	84.1%	86.0%	13.72	✓	✓	×
Artificial immune system [30]	RTL code	87%	85%	86%	NA	×	✓	×
Graph neural network (our work)	**LUT netlist (Xilinx)**	**99.69%**	**98.79%**	**99.23%**	**0.0014**	✓	✓	✓
	LUT netlist (Intel)	**97.87%**	**99.17%**	**98.51%**	**0.0027**	✓	✓	✓

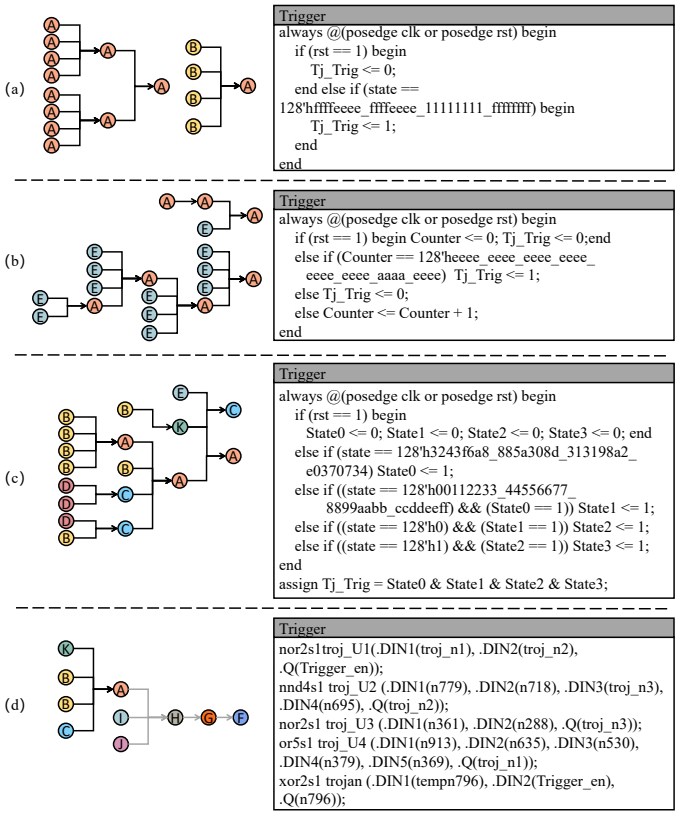

Fig. 5. Explanation results of various hardware Trojans for Intel FPGA netlists and the corresponding Trojan trigger logic. (a) Explanation result and trigger logic of AES-T1300. (b) Explanation result and trigger logic of AES-T1200. (c) Explanation result and trigger logic of AES-T500. (d) Explanation result and trigger logic of c2670-T000.

TABLE VI
DESCRIPTION OF THE NODES SHOWN IN FIGURE 5.

Node	Type label	Node type	INIT (Binary value)
A	32772	LUT	1000_0000_0000_0000
B	5	LUT	0000_0000_0000_0001
C	4100	LUT	0001_0000_0000_0000
D	4373	LUT	0001_0001_0001_0001
E	4	DFF	–
F	1	OUTPUT	–
G	45236	LUT	1011_0000_1011_0000
H	63492	LUT	1111_1000_0000_0000
I	45236	LUT	1011_0000_1011_0000
J	48063	LUT	1011_1011_1011_1111
K	16388	LUT	0100_0000_0000_0000

E. Comparison and Analysis

We further compare our method with state-of-the-art machine learning based hardware Trojan detection techniques as summarized in Table VII. The average detection time is also considered as a metric. Although these methods are implemented on different hardware platforms, the detection time measurements can give us a hint about the performance. Compared to the related works in [16]–[19], [30], which require manually defining Trojan features, our method provides automated Trojan feature extraction. Moreover, our method is golden reference free and can by easily used for new Trojan detection. Compared to the existing GNN based Trojan detection methods at RTL and gate-level, our method targets LUTs and achieves higher precision, TPR and F1-measure as well as human-interpretable explanation.

VI. CONCLUSION

This paper proposes an effective and interpretable approach to thwarting hardware Trojan threats in third-party IP cores. IP cores are synthesized to FPGA netlists and a GNN model with optimized CB focal loss is developed for automated Trojan feature extraction and classification. The Granger causality based explanation method is employed to understand the Trojan classification principle of our GNN model. The compact subgraphs generated are interpretable and consistent with the well-accepted fact that the conditionally triggered Trojans exhibit low switching activity. Experimental results using 927 Trust-Hub benchmarks and 262 Trojan-free open source IP designs show that the proposed method can precisely pinpoint hardware Trojans with high accuracy and performance. This work expands the spectrum of machine learning based hardware Trojan detection techniques as well as FPGA security design tools.

REFERENCES

[1] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware Trojans," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 197–214.

[2] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 23–40.

[3] M. Sabri, A. Shabani, and B. Alizadeh, "SAT-based integrated hardware Trojan detection and localization approach through path-delay analysis," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2850–2854, 2021.

[4] X. Cui, E. Koopahi, K. Wu, and R. Karri, "Hardware Trojan detection using the order of path delay," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 3, pp. 1–23, 2018.

[5] M. Nourian, M. Fazeli, and D. Hély, "Hardware Trojan detection using an advised genetic algorithm based logic testing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 461–470, 2018.

[6] Y. Lyu and P. Mishra, "Automated trigger activation by repeated maximal clique sampling," in *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 482–487.

[7] C. Bao, D. Forte, and A. Srivastava, "On reverse engineering-based hardware Trojan detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 49–57, 2015.

[8] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 697–708.

[9] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, 2015.

[10] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[11] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1691–1696.

[12] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting hardware Trojans with gate-level information-flow tracking," *Computer*, vol. 49, no. 8, pp. 44–52, 2016.

[13] Y. Jin, X. Guo, R. G. Dutta, M.-M. Bidmeshki, and Y. Makris, "Data secrecy protection through information flow tracking in proof-carrying hardware IP—Part I: Framework fundamentals," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2416–2429, 2017.

[14] J. Cruz, P. Gaikwad, A. Nair, P. Chakraborty, and S. Bhunia, "Automatic hardware Trojan insertion using machine learning," *arXiv preprint arXiv:2204.08580*, 2022.

[15] J. Clements and Y. Lao, "Hardware Trojan design on neural networks," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[16] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists using multi-layer neural networks," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2017, pp. 227–232.

[17] 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.

[18] T. Han, Y. Wang, and P. Liu, "Hardware Trojans detection at register transfer level based on machine learning," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[19] F. Demrozi, R. Zucchelli, and G. Pravadelli, "Exploiting sub-graph isomorphism and probabilistic neural networks for the detection of hardware Trojans at RTL," in *IEEE International High Level Design Validation and Test Workshop*. IEEE, 2017, pp. 67–73.

[20] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[21] R. Yasaei, L. Chen, S.-Y. Yu, and M. A. Al Faruque, "Hardware Trojan detection using graph neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[22] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware Trojan detection and reducing Trojan activation time," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 1, pp. 112–125, 2011.

[23] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "Mero: A statistical approach for hardware Trojan detection," in *Cryptographic Hardware and Embedded Systems-CHES 2009: 11th International Workshop*. Springer, 2009, pp. 396–410.

[24] Z. Pan and P. Mishra, "Automated detection of spectre and meltdown attacks using explainable machine learning," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 24–34.

[25] Z. Pan, J. Sheldon, and P. Mishra, "Hardware-assisted malware detection using explainable machine learning," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 663–666.

[26] Z. Pan and P. Mishra, "Hardware Trojan detection using Shapley ensemble boosting," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 496–503.

[27] W. Lin, H. Lan, and B. Li, "Generative causal explanations for graph neural networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6666–6679.

[28] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3734–3743.

[29] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *IEEE/CVF Conference on Computer Cision and Pattern Recognition*, 2019, pp. 9268–9277.

[30] F. Zareen and R. Karam, "Detecting RTL Trojans using artificial immune systems and high level behavior classification," in *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018, pp. 68–73.