

Explainable Hardware Trojan Detection at RTL using Attention Mechanism

Siyu Tian¹, Wei Hu³, Lingjuan Wu^{*1,2}, Tianle You¹, Hao Su³, Jiacheng Zhu³

¹College of Informatics, Huazhong Agricultural University, Wuhan 430070, China

²State Key Laboratory of Integrated Chips and Systems, Shanghai 200433, China

³School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China

Abstract—Hardware Trojans pose a severe threat to the security and trustworthiness of integrated circuit (IC) designs. Existing Trojan detection techniques see shortcomings in low accuracy, reliance on expert knowledge for manual feature extraction and lack of explainability. This work proposes a novel explainable hardware Trojan detection solution at register transfer level (RTL) using attention mechanism. The proposed method captures rich circuit structural and semantic features from the path-contexts in the abstract syntax tree representation of RTL design. A code2vec deep learning model with attention mechanism is trained for automated feature extraction and distinguishing Trojan-infected designs from Trojan-free ones. It further develops an explainable approach by analyzing the attention weights in classification to understand the decision-making mechanism of the trained model. Experimental evaluations using 63 Trust-Hub Trojan benchmarks and 147 Trojan-free design cores demonstrate that the proposed method achieves promising detection results with TPR, TNR and F1-score of 100%, 99.43% and 99.45% respectively. Explanation results show that the decision mechanism of our code2vec model is closely related to the hardware Trojan trigger logic.

Index Terms—Hardware security, hardware Trojan detection, RTL code, abstract syntax tree, attention mechanism.

I. INTRODUCTION

Trojans and backdoors represent a significant threat to hardware security and trustworthiness under a globalized semiconductor design and supply chain. They are carefully designed to stay dormant for most of the time and triggered only under rare conditions to escape from being detected. Once triggered, hardware Trojans can create attack surfaces that can lead to information leakage, functionality change or even denial of service [1]. Existing literature has spotted clues of such stealthy malicious design modifications even in chips involved in critical infrastructures [2] and military defense systems [3]. Since security vulnerabilities are difficult to patch after chip fabrication, it is of great importance to identify and prevent Trojan threats in the early design stage.

Researchers have developed various pre-silicon hardware Trojan detection techniques [4]–[8]. Functional verification techniques try to activate the Trojan and observe malicious behavior [4]. However, it can be a desperate task under exponential scale state space. Switching probability analysis

based methods [5], [6] leverage the extremely low-switching feature of Trojan trigger signals as a clue to spot stealthy malicious logic. However, these methods can see a high false positive rate and may miss Trojans controlled by multiple discrete triggers. Formal verification approaches [7], [8] are quite effective in identifying Trojans that can cause security property violations. However, property specification is still largely a manual process that does not provide any guarantee of quality or completeness.

Recently, artificial intelligence (AI) techniques have drawn numerous attention in the hardware security community for Trojan detection. Researchers have extracted structural and behavioral features and trained machine learning classifiers, such as random forest [9], [10] and decision tree [11], to identify the Trojan-infected hardware designs or circuit nodes. However, these methods usually rely on specialized expertise for feature engineering and see shortcomings in detecting the newly designed Trojans. To overcome these challenges, recent works leverage graph convolutional networks (GCNs) [12], [13] and multimodal learning [14] for automated feature extraction and Trojan classification at the gate level and RTL. However, most of the current deep learning based approaches overlook semantic features in the RTL code, which are essential characteristics for identifying hardware Trojans. In addition, these works lack explainability, making it difficult to intuitively understand the decision mechanism of the deep learning model and ensure that the features learned are closely associated with the hardware Trojan logic.

In this work, we propose an explainable hardware Trojan detection method at RTL by training a deep learning model with attention mechanism. We convert the RTL code of hardware designs to abstract syntax tree (AST), and then extract path-contexts between leaf nodes, which preserve rich circuit structural information and semantic characteristics. The path-contexts are then used to train a code2vec model with attention mechanism [15] for hardware design classification and Trojan detection. We further analyze the attention weights in classification to understand the decision mechanism of the trained code2vec model. Evaluations using 210 RTL benchmarks from *Trust-Hub.org* and *OpenCores.org* show that the proposed method provides promising Trojan detection results as well as interpretable explanations for our model. The contributions of this paper are summarized as follows:

- Proposing a novel hardware Trojan detection method at

Corresponding author: Lingjuan Wu, email: wulj@mail.hzau.edu.cn. This work was supported in part by the National Natural Science Foundation of China under Grants 62404082 and U23B2041, and SKLICS Project Fund under Grant SKLICS-G202504.

RTL through training a code2vec model with attention mechanism for automated feature extraction and hardware design classification;

- Providing an RTL code representation approach by extracting path-contexts from the AST of RTL code, which captures rich circuit structural and semantic information;
- Developing an explainable method based on attention mechanism, which identifies the key logic leading to the Trojan detection results and thus provides interpretable explanations for the decision-making rule of our model;
- Presenting experimental results to demonstrate the effectiveness of the proposed method in terms of RTL hardware Trojan detection and model explanation.

The remainder of this paper is organized as follows. Section II reviews related work in pre-silicon hardware Trojan detection using AI techniques. The threat model is described in Section III. Section IV introduces the proposed explainable hardware Trojan detection method. We present experimental results in Section V and conclude the paper in Section VI.

II. RELATED WORK

A. AI based Hardware Trojan Detection

Early efforts in hardware Trojan detection primarily rely on domain expertise to manually extract circuit structural and/or behavioral features from gate-level netlist or RTL code, which are then used to train machine learning or deep learning classifiers to distinguish between Trojan-infected and benign circuits. Kurihara *et al.* [9] extracted 36-dimensional structural features at the gate level and employed the random forest model to classify the circuit nets as Trojan-infected or normal ones. Yang *et al.* [10] extracted the circuit structure and signal characteristics related features, and trained a random forest classifier to identify Trojan nodes in the RTL code. Another work [11] proposed a Trojan propagation probability method to re-identify Trojan nodes initially classified by a decision tree model, thereby improving classification performance. Although manual feature extraction methods offer simplicity, the feature selection process is highly dependent on expert knowledge and lacks scalability, which hinders their effectiveness in detecting previously unseen Trojan variants.

Recent works employed deep learning models for automated feature extraction and Trojan identification. Researchers have converted the gate-level netlist and RTL code to data flow graph and trained GCNs for Trojan detection [12], [13]. To bridge the gap between graph representation learning and feature-based Trojan detection, node features obtained from domain knowledge were used in graph neural networks (GNNs) and GCN model training [16]–[19]. Another work employed generative adversarial networks to amplify the Trojan dataset and used multimodal deep learning to improve the detection performance and robustness [14]. However, the deep learning models used in these works are regarded as black-box, and the rationale behind the decision-making process cannot be interpreted.

B. Explainable AI Techniques

Explainable AI techniques aim to provide interpretable explanations for the decision-making rule of the models by illustrating why a sample is classified to a specific category. Recently, some efforts have been made in explainable hardware Trojan detection. Mahfuz *et al.* [20] leveraged the Captum-Explainer algorithm to identify influential nodes and edges in GNNs for gate-level Trojan detection. Pan *et al.* [21] developed a Shapley analysis and boosting combined framework to generate the spectrum of impact for each feature towards Trojan classification. Su *et al.* [22] developed a Granger causality theory based explainable method for hardware Trojan localization in FPGA netlists using GNN model. Vishwakarma *et al.* [23] explained the phenomenon of the model outputting a NULL set during hardware Trojan detection, arguing that when the confidence scores of predictions for all classes fall below 0.5, returning a NULL set is more reasonable than providing untrustworthy results. Kundu *et al.* [24] employed the large language model GPT-3.5 to extract the line numbers, wire names and Trojan triggers within the hardware design code for Trojan detection.

These works [20], [22]–[24] provide post-hoc explainability for the trained model and try to identify the key features leading to Trojan detection results. Although [21] identifies the key features before model training, the explanation results are heavily dependent on expert knowledge for manual feature extraction. Compared to the aforementioned research, this work proposes a novel explainable hardware Trojan detection method using attention mechanism, which provides inherent real-time explainability for the decision-making rule of the trained model.

III. THREAT MODEL

We consider typical IC design scenarios, where designers usually integrate IP cores from untrusted third-party suppliers to reduce design effort and shorten time-to-market. We assume that during the design phase, some grudging staff or electronic design automation (EDA) tools will deliberately make illegal modifications, that are out of the original specifications, to the IP product [22]. We assume that hardware Trojans are inserted into the IP cores in the form of RTL code, and the Trojans are conditionally triggered for malicious purpose. In this work, we perform hardware Trojan detection at the pre-silicon stage and assume that we have access to the RTL code of the IP cores.

IV. EXPLAINABLE TROJAN DETECTION METHOD

A. RTL Code Representation

In this work, we represent the hardware design in the form of RTL code as path-contexts to preserve the circuit structural and semantic features. As shown in Fig. 1, the RTL source code is flattened and converted to AST using Pyverilog toolkit [25], after which the connecting paths between leaf nodes in the AST are extracted to generate the path-contexts. Specifically, the RTL code representation process mainly involves the following two steps.

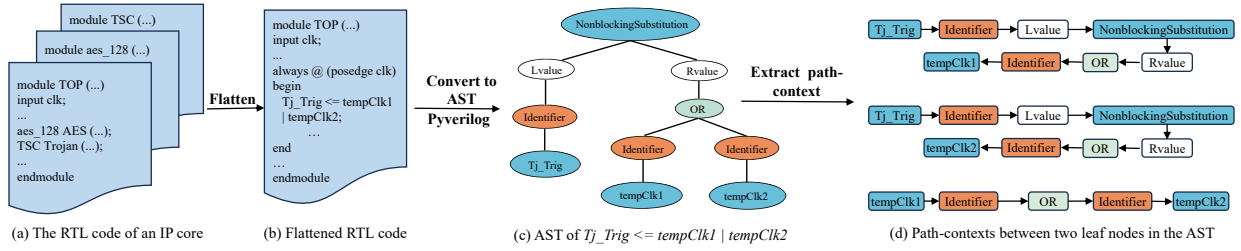


Fig. 1. The design flow of RTL code representation method.

1) *RTL code to AST conversion*: We start with analyzing and flattening the RTL code of an IP core into a single module and then employ the Pyverilog toolkit to parse the code and generate its AST. AST is an intermediate representation form that captures the data flow of the flattened RTL code in a tree structure, in which each node is an element in the RTL code.

Fig. 1 (c) shows the AST generated from the RTL source code of $Tj_Trig \leq tempClk1 | tempClk2$, where each item within an oval represents a token. Through non-blocking assignment, the expression is divided into two parts: the left part represents the identifier of Tj_Trig , and the right part consists of the identifiers of $tempClk1$ and $tempClk2$ performing a bitwise OR operation.

2) *Path-context generation*: The path-context is composed of two leaf nodes in the AST and their connecting path. For a hardware design with m path-contexts extracted, the i^{th} path-context can be represented as:

$$path_context_i = t_1, t_2, \dots, t_{n-1}, t_n \quad (1)$$

where t_1 and t_n are the leaf nodes, and t_2, \dots, t_{n-1} represents the connecting path between them, denoted as q_i . In our RTL code representation, each node in the AST is considered as a token, such as $t_1, t_2, \dots, t_{n-1}, t_n$. Specifically, a token is the basic unit that constitutes the AST, and can be classified into categories such as identifiers, keywords, constants, and operators. The path-context representation captures both the circuit structural information in the data flow as well as the semantic characteristics such as signal names and keywords in the RTL code. It is worth mentioning that in order to prevent the training dataset from becoming overly large, repeating path-contexts will not be retained.

B. Code2vec Model

With a collection of path-contexts generated for each IP core, we train the code2vec model [15] with attention mechanism for automated feature extraction, quantification and hardware Trojan detection, in which the IP core under test is classified as Trojan-infected or Trojan-free. The code2vec model can effectively learn code embeddings – continuous distributed vector representations of the RTL code, which enables us to model the hardware design in a natural and effective manner. The code2vec model encodes path-contexts extracted from the AST into vector representations and employs the attention mechanism to perform weighted fusion

on the embedding vectors. This operation enables automated extraction of structural and semantic features highly related to hardware Trojans, thereby improving detection accuracy and model explanation. As shown in Fig. 2, the code2vec model mainly includes four major layers and operations as described in the following.

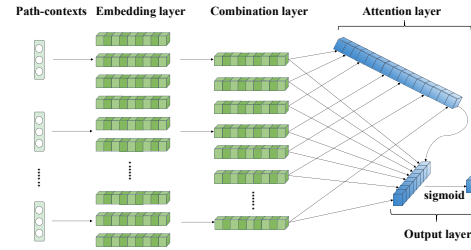


Fig. 2. The framework of the code2vec model with attention mechanism.

Embedding layer: In this layer, each path-context is transformed into vector representation through looking into two embedding matrices:

$$\begin{aligned} leaves_matrix &\in \mathbb{R}^{|X| \times d} \\ path_matrix &\in \mathbb{R}^{|A| \times d} \end{aligned} \quad (2)$$

where X is the set of AST leaf nodes observed during the training process, A represents the set of connecting paths, and d denotes the dimension of the embedding vector in the feature space. These two matrices are initialized with random values and then learned empirically in the code2vec model training. The embedding vector of $path_context_i$ is formulated as:

$$\begin{aligned} p_i &= embedding(\langle t_1, q_i, t_n \rangle) \\ &= [leaves_matrix_u; path_matrix_s; leaves_matrix_v] \end{aligned} \quad (3)$$

where $p_i \in \mathbb{R}^{3d}$, u and v are the positions corresponding to the leaves of t_1 and t_n in the $leaves_matrix$ respectively, while s is the position corresponding to q_i in the $path_matrix$.

Combination layer: This layer mainly includes a fully connected operation to combine the path-context p_i with three independent vectors into a single one. The combined embedding \tilde{p}_i is calculated as:

$$\tilde{p}_i = \tanh(W \cdot p_i) \in \mathbb{R}^d \quad (4)$$

where $W \in \mathbb{R}^{d \times 3d}$ is the learnable weight matrix, and $\tanh(\cdot)$ is the hyperbolic tangent function.

Attention layer: This layer introduces an attention mechanism to assign different weights to the combined embeddings. A global attention vector $\mathbf{a} \in \mathbb{R}^d$ is randomly initialized, and optimized in the training process of the code2vec model. The attention weight α_i of \tilde{p}_i can be calculated through the normalized inner product between the combined embedding itself and the global attention vector \mathbf{a} :

$$\alpha_i = \frac{\exp(\tilde{p}_i^T \cdot \mathbf{a})}{\sum_{j=1}^m \exp(\tilde{p}_j^T \cdot \mathbf{a})} \quad (5)$$

where the exponential function $\exp(\cdot)$ is used to make the attention weights positive, and the normalization ensures that the sum of α_i is 1. Then the path-context embeddings in the IP core are linearly aggregated based on their corresponding attention weights:

$$v = \sum_{i=1}^m \alpha_i \cdot \tilde{p}_i \quad (6)$$

where $v \in \mathbb{R}^d$ is a vector that represents the entire IP core.

Output layer: The output layer employs the fully connected operation to obtain a probability value between 0 and 1, which can be formulated as:

$$y_{pred} = \sigma(\Theta v + b) = \frac{1}{1 + e^{-(\Theta v + b)}} \quad (7)$$

where σ is the sigmoid activation function, Θ and b are the learnable weight and bias respectively. In our tests, if y_{pred} is greater than the threshold of 0.5, the IP core under test is classified as Trojan-infected, or Trojan-free otherwise.

C. Explanation Method

Deep learning models with attention mechanism can record the attention weight assigned to each feature when performing classification, thereby revealing the key features that the model focuses on during the decision-making process. Specifically, in our code2vec model, the attention mechanism is used to weight the importance of different path-contexts, enabling us to observe the degree of attention the model pays to each path-context in classification. According to the previously mentioned attention weight calculation method as illustrated in Eq. (5), the attention score for each path-context in the IP core under test is obtained from the attention layer of the trained code2vec model, formulated as:

$$\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \dots, \alpha_{m-1}, \alpha_m\} \quad (8)$$

where α_i is the attention weight corresponding to the path-context \tilde{p}_i . After ranking the attention scores in descending order, we obtain:

$$\hat{\boldsymbol{\alpha}} = \{\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_{m-1}, \hat{\alpha}_m\} \quad (9)$$

where $\hat{\alpha}_1$ is the highest attention score. Specifically, by ranking the attention scores output by the attention layer, we can identify the most relevant path-contexts, i.e., the key logic in the RTL code, that contribute to the prediction result and thus provide explanations for the decision mechanism of the trained code2vec model. The attention mechanism based explanation

method not only provides interpretability for the hardware Trojan detection results, but also enhances the transparency of the code2vec model.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

Experimental platform: The experiments are performed on a PC with an Intel i7-12700 CPU and an Nvidia GeForce RTX 4060Ti graphics card with 16GB memory.

Dataset: We collect a dataset with 63 Trojan-infected and 147 Trojan-free hardware designs, which are regarded as the positive and negative classes and assigned with the label of 1 and 0 respectively. The Trojan-infected samples consist of various types of Trojans with different activation mechanisms and payload effects, as reported in Table I. Among them, 45 Trojan benchmarks are downloaded from *Trust-Hub.org*, which are inserted into the base circuits of AES, RS232 and PIC. We also implant 18 conditionally triggered Trojans into a DES IP core. The Trojan-free samples are publicly available open source IP cores from *OpenCores.org*, which are different from the above mentioned base circuits and mainly perform arithmetic operations and digital signal processing.

TABLE I
DESCRIPTION OF THE HARDWARE TROJAN BENCHMARKS.

Trojan benchmarks	Quantity	Payload effect
AES-Txx	27	Leak information, Denial of service, Change functionality
RS232-Txx	14	Denial of service, Reduce design reliability, Leak information
PIC16F84-Txx	4	Denial of service, Leak information
DES-Txx	18	Leak information, Denial of service

Hyperparameter setting: During the code2vec model training process, we employ grid search to minimize the cross-entropy loss and determine the optimal combination of hyperparameters. Table II shows the search range and the optimal values for the main hyperparameters. In order to mitigate overfitting, Dropout [26] is incorporated as a regularization strategy, while the Adam optimizer [27] is utilized to dynamically adjust the learning rate and enhance the convergence speed of training.

TABLE II
HYPERPARAMETER SELECTION IN MODEL TRAINING.

Hyperparameter	Search range	Optimal value
Epoch	{50, 100, 200}	100
Dropout ratio	{0.1, 0.2, 0.5}	0.2
Batch size	{8, 16, 32}	8
Embedding dimension	{64, 128, 256}	128
Learning rate	{0.001, 0.005, 0.1}	0.001

Evaluation metrics: We use the evaluation metrics of true positive rate (TPR), true negative rate (TNR), precision, accuracy and F1-score to comprehensively evaluate the hardware Trojan detection performance of the code2vec model.

B. Hardware Trojan Detection Results

In our test, we randomly divide the dataset composed of hardware Trojan and OpenCores benchmarks into training and

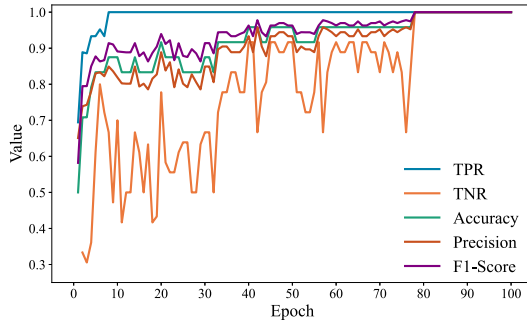


Fig. 3. The training process of the code2vec model under the optimal hyperparameters.

testing sets at a ratio of 3:1. We further employ 4-fold cross-validation to evaluate the performance of the trained code2vec model. The training process is shown in Fig. 3, where the code2vec model converges rapidly within 80 epochs. These results show that the code2vec model not only has strong learning capabilities, but also maintains stable and reliable performance in detecting hardware Trojans. To eliminate the influence of random seeds on the experimental results, we perform 20 rounds of independently repeated experiments. The hardware Trojan classification and detection results of the trained code2vec model in each round are reported in Table III.

Table III shows that the code2vec model can successfully detect the hardware Trojans with TPR of 100% in each round, while achieving TNR, accuracy and F1-score of 99.43%, 99.62% and 99.45% respectively on average. These results show that the trained code2vec model can accurately distinguish the Trojan-infected hardware designs from Trojan-free ones, demonstrating the effectiveness of the proposed RTL code representation and hardware Trojan detection method.

TABLE III
HARDWARE TROJAN DETECTION RESULTS OF THE CODE2VEC MODEL.

Round	TPR	TNR	Precision	Accuracy	F1-score
1	100.00%	100.00%	100.00%	100.00%	100.00%
2	100.00%	100.00%	100.00%	100.00%	100.00%
3	100.00%	100.00%	100.00%	100.00%	100.00%
4	100.00%	100.00%	100.00%	100.00%	100.00%
5	100.00%	100.00%	100.00%	100.00%	100.00%
6	100.00%	93.94%	90.91%	96.23%	95.24%
7	100.00%	100.00%	100.00%	100.00%	100.00%
8	100.00%	100.00%	100.00%	100.00%	100.00%
9	100.00%	100.00%	100.00%	100.00%	100.00%
10	100.00%	100.00%	100.00%	100.00%	100.00%
11	100.00%	100.00%	100.00%	100.00%	100.00%
12	100.00%	100.00%	100.00%	100.00%	100.00%
13	100.00%	100.00%	100.00%	100.00%	100.00%
14	100.00%	100.00%	100.00%	100.00%	100.00%
15	100.00%	100.00%	100.00%	100.00%	100.00%
16	100.00%	97.22%	94.44%	98.11%	97.14%
17	100.00%	100.00%	100.00%	100.00%	100.00%
18	100.00%	100.00%	100.00%	100.00%	100.00%
19	100.00%	97.44%	93.33%	98.11%	96.55%
20	100.00%	100.00%	100.00%	100.00%	100.00%
Average	100.00%	99.43%	98.93%	99.62%	99.45%

C. Explanation Results

In this subsection, we report the code2vec model explanation results using attention mechanism. Based on the ex-

planation method introduced in Section IV-C, we record the attention weight assigned to each path-context in the attention layer of the code2vec model. We then identify the *top-k* ranked ones, which are the most relevant portions in the RTL code towards the hardware Trojan detection results and thus provide insights about the decision-making rule of the code2vec model.

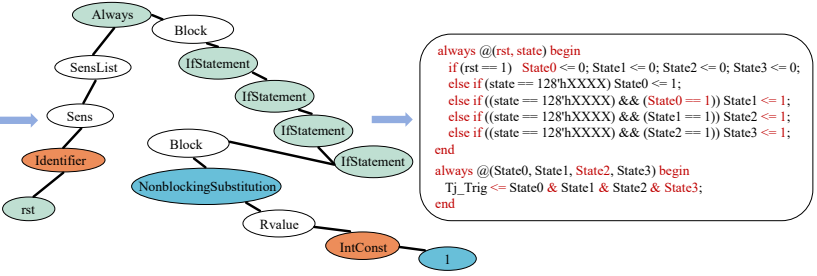
Fig. 4 illustrates the explanation results for the randomly selected hardware Trojan benchmarks of AES-T500 and AES-T900. For each benchmark, the left panel represents the top-5 path-contexts obtained by ranking the attention scores. These path-contexts capture the circuit structural and semantic characteristics in the original RTL code of the hardware design under test. Specifically, in these path-contexts extracted from AST, *Always* denotes an always block, *IfStatement* represents an if statement, *NonblockingSubstitution* indicates a non-blocking assignment, *Identifier* refers to a variable identifier, and *IntConst* stands for an integer constant. The other items are variable names extracted from the RTL code, which represent semantic information, such as *State0*, *State2* and *Tj_Trig*. The subgraph in the middle panel shows the AST reconstructed from the path-context with the highest attention score. The right panel is the recovered RTL code, where the code in red represents the key statements and variables identified in the top-5 path-contexts.

Specifically, for AES-T500, the top-5 path-contexts are within the hardware Trojan module of *TSC*, and the recovered RTL code reveals the activation mechanism of the trigger signal *Tj_Trig*. For AES-T900, the extracted path-contexts are also correlated with the Trojan trigger logic, in which the *Tj_Trig* signal is activated when *Counter* reaches the specified value. The attention mechanism based explanation method can effectively identify key components in the RTL code that make significant contributions to the predictions of the code2vec model, which is closely related to the Trojan trigger logic. Consequently, our detection approach is capable of precisely localizing Trojan-related patterns, allowing accurate identification of malicious logic and achieving a TPR of 100%. These results demonstrate the effectiveness of the proposed explanation method and provide interpretable explanations for the decision-making mechanism of the code2vec model.

D. Comparison and Discussion

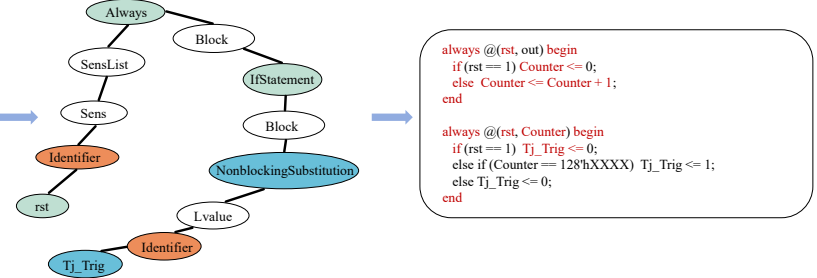
Table IV summarizes the comparison of our method with state-of-the-art hardware Trojan detection approaches at the pre-silicon stage using AI techniques. From a quantitative perspective, our method outperforms the related works with superior Trojan detection performance. Notably, in our work, the average TPR, precision, and F1-score are 100%, 98.93%, and 99.45% respectively, where TPR reflects the model's ability to correctly identify the Trojans. Moreover, the average detection time (i.e., testing time) for an IP core is 0.0081s, demonstrating the superior performance and speed of our method as compared to the other works. From a qualitative perspective, our method enables fully automated feature extraction, which not only minimizes reliance on domain expertise but also enhances the efficiency and generalizability.

The top-5 path-contexts ranked by attention scores in AES-T500	
1 st	rst,Identifier Sens SensList Always Block IfStatement IfStatement IfStatement Block NonlockingSubstitution Rvalue IntConst,1
2 nd	State2,Identifier Sens SensList Always Block NonlockingSubstitution Rvalue And Identifier,State3
3 rd	rst,Identifier Sens SensList Always Block IfStatement IfStatement IfStatement Block NonlockingSubstitution Rvalue IntConst,1
4 th	level,Sens SensList Always Block IfStatement IfStatement IfStatement Land Eq Identifier,State0
5 th	rst,Identifier Sens SensList Always Block IfStatement Block NonlockingSubstitution Lvalue Identifier,State0



(a) Explanation results for the hardware Trojan benchmark of AES-T500

The top-5 path-contexts ranked by attention scores in AES-T900	
1 st	rst,Identifier Sens SensList Always Block IfStatement Block NonlockingSubstitution Lvalue Identifier,Tj_Trig
2 nd	rst,Identifier Sens SensList Always Block IfStatement Block NonlockingSubstitution Rvalue Plus IntConst,1
3 rd	Counter,Identifier Sens SensList Always Block IfStatement Block NonlockingSubstitution Lvalue Identifier,Tj_Trig
4 th	level,Sens SensList Always Block IfStatement Block NonlockingSubstitution Rvalue Plus Identifier,Counter
5 th	level,Sens SensList Always Block IfStatement Block NonlockingSubstitution Lvalue Identifier,Counter



(b) Explanation results for the hardware Trojan benchmark of AES-T900

Fig. 4. Explanation results for the hardware Trojan benchmarks of (a) AES-T500 and (b) AES-T900.

TABLE IV
COMPARISON OF OUR METHOD WITH THE EXISTING MACHINE LEARNING BASED HARDWARE TROJAN DETECTION APPROACHES.

Method	Abstract level	Number of benchmarks	TPR	TNR	Precision	Accuracy	F1-score	Automated feature extraction	Golden reference-free	Explainability	Detection time (s)
Decision Tree [11]	Gate level	32	89.00%	99.70%	88.00%	99.60%	86.10%	×	×	×	–
GNN [16]	Gate level	24	89.00%	–	97.80%	99.80%	92.10%	✓	✓	×	–
GNN [17]	Gate level	17	78.00%	85.00%	–	–	–	✓	✓	×	–
GCN [18]	Gate level	39	96.80%	99.90%	94.50%	99.80%	95.40%	✓	✓	×	–
GAT [20]	Gate level	16	98.47%	98.14%	–	–	–	✓	✓	✓	–
Decision tree [21]	Gate level	10	99.50%	–	99.70%	99.93%	99.60%	×	×	✓	1339.00
GAN [23]	Gate level	–	–	–	–	90.00%	–	×	×	✓	–
GPT-3.5 [24]	Gate level / RTL	8	89.00%	–	97.00%	93.00%	91.00%	✓	×	✓	–
GNN [22]	LUT level	183	95.14%	95.78%	–	95.71%	–	✓	✓	✓	–
GCN [12]	RTL	49	88.40%	–	98.90%	99.60%	93.10%	✓	✓	×	0.50
GNN [19]	RTL	30	99.10%	–	94.50%	99.30%	96.70%	✓	✓	×	–
GCN [13]	RTL	36	100.00%	–	94.40%	91.70%	96.70%	✓	✓	×	0.2840
Code2vec (Ours)	RTL	210	100.00%	99.43%	98.93%	99.62%	99.45%	✓	✓	✓	0.0081

Moreover, our method is golden reference-free, which is of great importance since the golden designs are hard to obtain in practical applications.

Furthermore, our approach provides interpretable explanations for the decision mechanism of our model. Compared to the existing explainable Trojan detection works [20], [22]–[24], which provide post-hoc explainability, our method provides built-in explainability for the model. Although [21] performs feature selection to identify the key features before model training, it requires manual feature engineering, which makes the explanation results heavily dependent on expert knowledge. In contrast, in our work, the code2vec model with attention mechanism is inherently interpretable, which allows for direct understanding of the decision-making process during model execution. By leveraging the attention scores in the attention layer of the model, we are able to directly identify the key logic in the hardware design that the model focuses

on during classification, thereby significantly improving the timeliness and reliability of the explanation results.

VI. CONCLUSION

This work makes an important step towards explainable hardware Trojan detection at the early design phase of integrated circuits. We represent the RTL code of a hardware design as AST and then extract the path-contexts between leaf nodes, which carry rich circuit structural and semantic features. We then train the code2vec model with attention mechanism for automated feature extraction and hardware Trojan detection. We further analyze the attention scores of path-contexts in the attention layer, which identify the most relevant logic in the RTL code leading to the classification results, and thus provide human-understandable explanations for the code2vec model. Evaluation results demonstrate the effectiveness of the proposed method in hardware Trojan detection as well as interpretable model explanation.

REFERENCES

- [1] W. Hu, B. Li, L. Wu, Y. Li, X. Li, and L. Hong, "Design for assurance: Employing functional verification tools for thwarting hardware Trojan threat in 3PIPs," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–17, 2025.
- [2] H. Riggs, S. Tufail, I. Parvez, M. Tariq, M. A. Khan, A. Amir, K. V. Vuda, and A. I. Sarwat, "Impact, vulnerabilities, and mitigation strategies for cyber-secure critical infrastructure," *Sensors*, vol. 23, no. 8, p. 4060, 2023.
- [3] M. Ender, A. Moradi, and C. Paar, "The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series FPGAs," in *29th USENIX Security Symposium (USENIX Security 2020)*, 2020, pp. 1803–1819.
- [4] H. E. Taheri and M. Mirhassani, "A pre-activation, golden IC free, hardware Trojan detection approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 3, pp. 315–324, 2022.
- [5] K. Hasegawa, S. Hidano, K. Nozawa, S. Kiyomoto, and N. Togawa, "R-HTDetector: Robust hardware-Trojan detection based on adversarial training," *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 333–345, 2023.
- [6] A. Sarihi, P. Jamieson, A. Patooghy, and A.-H. A. Badawy, "Trojan-Forge: Generating adversarial hardware Trojan examples using reinforcement learning," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024, pp. 1–7.
- [7] M. Qin, J. Li, J. Yan, Z. Hao, W. Hu, and B. Liu, "HT-PGFV: Security-aware hardware Trojan security property generation and formal security verification scheme," *Electronics*, vol. 13, no. 21, p. 4286, 2024.
- [8] J. Zhu, W. Hu, L. Wu, D. Zhu, and D. Mu, "LUT level information flow tracking for FPGA design security verification," in *2024 IEEE Global Communications Conference*. IEEE, 2024, pp. 3093–3098.
- [9] T. Kurihara and N. Togawa, "Hardware-Trojan classification based on the structure of trigger circuits utilizing random forests," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2021, pp. 1–4.
- [10] J. Yang, Y. Zhang, Y. Hua, J. Yao, Z. Mao, and X. Chen, "Hardware Trojans detection through RTL features extraction and machine learning," in *2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2021, pp. 1–4.
- [11] R. Negishi, T. Kurihara, and N. Togawa, "Hardware-Trojan detection at gate-level netlists using a gradient boosting decision tree model and its extension using Trojan probability propagation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 107, no. 1, pp. 63–74, 2024.
- [12] R. Yasaei, S. Faezi, and M. A. Al Faruque, "Golden reference-free hardware Trojan localization using graph convolutional network," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 10, pp. 1401–1411, 2022.
- [13] Y. Li, S. Li, and H. Shen, "HTrans: Transformer-based method for hardware Trojan detection and localization," in *2023 IEEE 32nd Asian Test Symposium (ATS)*. IEEE, 2023, pp. 1–6.
- [14] R. Vishwakarma and A. Rezaei, "Uncertainty-aware hardware Trojan detection using multimodal deep learning," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.
- [15] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, pp. 1–29, 2019.
- [16] K. Hasegawa, K. Yamashita, S. Hidano, K. Fukushima, K. Hashimoto, and N. Togawa, "Node-wise hardware Trojan detection based on graph learning," *IEEE Transactions on Computers*, vol. 74, no. 3, pp. 749–761, 2025.
- [17] H. Lashen, L. Alrahis, J. Knechtel, and O. Sinanoglu, "TrojanSAINT: Gate-level netlist sampling-based inductive learning for hardware Trojan detection," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.
- [18] J. Xiao, S. Chai, Y. Gao, Y. Huang, F. Zhang, and T. Chen, "HTs-GCN: Identifying hardware Trojan nodes in integrated circuits using a graph convolutional network," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 6, pp. 2353–2366, 2025.
- [19] P. Ma, G. Shang, H. Liu, J. Shi, W. Pan, Y. Zhang, and Y. Hao, "GNN-Based hardware Trojan detection at register transfer level leveraging multiple-category features," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 3, pp. 831–840, 2025.
- [20] T. Mahfuz, P. Gaikwad, T. Suha, S. Bhunia, and P. Chakraborty, "SALTY: Explainable artificial intelligence guided structural analysis for hardware Trojan detection," in *2025 IEEE 43rd VLSI Test Symposium (VTS)*. IEEE, 2025, pp. 1–7.
- [21] 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.
- [22] H. Su, W. Hu, X. Zhang, D. Zhu, and L. Wu, "Toward precise and explainable hardware Trojan localization at LUT level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 7, pp. 2817–2821, 2025.
- [23] R. Vishwakarma and A. Rezaei, "Risk-aware and explainable framework for ensuring guaranteed coverage in evolving hardware Trojan detection," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [24] R. K. Kundu, K. Khalil, E. Garcia, E. Grassia, P. Calyam, and K. A. Hoque, "PEARL: An adaptive and explainable hardware Trojan detection using open source and enterprise large language models," *IEEE Access*, vol. 13, pp. 133 755–133 772, 2025.
- [25] S. Takamaeda Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for Verilog HDL," in *Applied Reconfigurable Computing: 11th International Symposium, ARC 2015*. Springer, 2015, pp. 451–460.
- [26] I. Salehin and D.-K. Kang, "A review on dropout regularization approaches for deep neural networks within the scholarly domain," *Electronics*, vol. 12, no. 14, p. 3106, 2023.
- [27] R. O. Ogundokun, R. Maskeliunas, S. Misra, and R. Damaševičius, "Improved CNN based on batch normalization and Adam optimizer," in *International Conference on Computational Science and Its Applications*. Springer, 2022, pp. 593–604.