GNN4Gate: A Bi-Directional Graph Neural Network for Gate-Level Hardware Trojan Detection

Dong Cheng[1], Chen Dong[1,4,*], Wenwu He[2], Zhenyi Chen[3], Yi Xu[1]
[1]*College of Computer and Data Science, Fuzhou University, Fuzhou, China*
[2]*School of Computer Science and Mathematics, Fujian University of Technology, Fuzhou, China*
[3]*Department of Computer Science and Engineering, University of South Floride, Tampa, USA*
[4]*Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou, China*
chengdong2021@foxmail.com, *dongchen@fzu.edu.cn, hwwhbb@163.com, zhenyichen@usf.edu, xuyilaser@foxmail.com

***Abstract*—Hardware is the physical foundation of cyberspace, and chips are the core components. The security risk of the chip will bring disaster to the entire world. Hardware Trojans (HTs) are malicious circuits, which are the primary security issue of chip. Recently, a series of machine learning-based HT detection methods were proposed. However, some shortcomings still deserve further consideration, such as relying too much on manual feature extraction, losing some signal propagation structure information, being hard to track the HTs' location and adapt them to various types of HTs. To address the above challenges, this paper proposes a gate-level HT detection method based on Graph Neural Network (GNN), named GNN4Gate, which is a golden-free Trojan-gate identification technology. Specifically, a special coding method combining logic gate type and port connection information is developed for circuit graph modeling. Based on this, taking logic gates as the classification object, an automatic GNN detection architecture based on Bi-directional Graph Convolutional Network (Bi-GCN) is developed to aggregate both the circuit signal propagation (forward) and dispersion (backward) structure features from the circuit graph. The proposed method is evaluated by Trusthub benchmarks with different functional HTs, the average True Positive Rate (Recall) is 87.14%, and the average True Negative Rate is 99.73%. The experimental results demonstrate that GNN4Gate is sufficiently accurate compared to the state-of-the-art detection works at gate-level.**

***Index Terms*—Hardware Trojan, Static Detection, Gate-Level, Trojan-Gate, Graph Neural Network, Golden-free**

I. Introduction

Hardware is the foundation of the entire cyberspace, and chips are the core components, chips' security risks mean the security risks of cyberspace. Due to high profit and competitive pressure, the design and manufacturing process of System-on-Chip (SoC) usually involves multiple third parties [1], which makes it easy for attackers to implant malicious circuits into a large-scale integrated circuit (IC) to perform specific malicious operations (denial-of-service, information leakage etc.) [2]. These malicious circuits are called Hardware Trojans (HTs), like malicious code, detecting and handling of the HTs is the most effective measure to guarantee the security of the chip.

Because of the small proportion of HT circuits [3], rare triggers [4], and difficulty in obtaining golden chip [5], traditional methods are not suitable for detecting ultra-large-scale circuits, such as facilitate detection and side-channel signal analysis. As shown in Figure 1(a), the manufacturing stage is more difficult than the design stage for attackers to insert HTs, so pre-silicon detection is very meaningful for protecting the ICs' security. As shown in Figure 1(b), since each level of the hierarchical design may has risks, compared with RTL, gate-level design

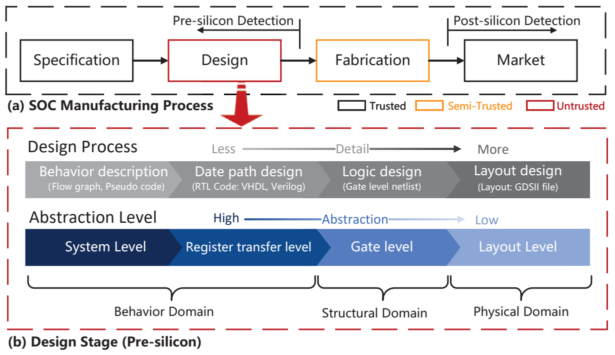

Fig. 1. The production process of SoC.

stage can reflect the structure of the underlying components, detecting and handling of HTs at gate-level has great research value for thoroughly understanding and controlling HTs.

Benefiting from Machine Learning (ML), static detection [6] methods has shown great potential in recent works [7], [8], [9], which can be applied to very large scale integrated (VLSI) circuit while maintaining high efficiency without golden reference chip. Most works are based on manual feature statistical analysis, among which features based on net [10] or controllability and observability [11] are currently the most used features. More manual features are defined and learned with a wide range of machine learning methods, which is time-consuming and labor-intensive, and can't represent the circuit well. Some recent studies [9], [12] have tried to use Natural Language Processing (NLP) methods to process structural information. They extracted gate paths from circuit topology graphs as a corpus to train NLP models, and successfully got rid of artificial feature extraction and achieved good results. But, their methods are subjective in the selection of the structure, and a lot of circuit structural information is inevitably lost.

In recent years, the processing of graph data has become a hot topic in ML [13]. [14] has explored the application of graph similarity in hardware security and used it to detect HTs in reverse engineering. GNN4IP [15] has also explored the effectiveness of Graph Neural Networks (GNN) for IP protection. GNN4HT [16] converts RTL-level netlists into specific data flow graphs, detects HTs in terms of functional logic, and classifies them based on the entire graph. HW2VEC [17] has further developed graph learning tool for automating hardware security. But for the gate-level design stage, there is still no effective and easy-to-expandable detection scheme.

Apart from this, [18] first proposed the locating based on the works of net classification [7], [8], [10]. For further understanding and controlling HTs, more fine-grained detecting and locating method is worth exploring. This paper proposes a novel static detection and location method based on GNN to learn the representation features of the gates. Because most of the existing GNNs are not efficiently used in directed graphs, we find the properties similar to circuit signal propagation in rumor detection [19], and explore the effectiveness of the Bi-directional Graph Convolutional Network (Bi-GCN) framework in circuit structure. Our contributions are as follows:

- Using GNN to learn structure features automatically for gate classification. This is the first work to classify gates and employ GNN at gate-level HTs detection and location.
- A special coding method is proposed to model the circuit graph, encode the special port connection information into node features, and create a gate-level circuit graph dataset from the benchmarks of TrustHub.
- A BiGCN-based automatic detection framework is designed, which can effectively aggregate circuit signal propagation and dispersion features to learn gate representation, and remove the workload of manual feature extraction and dependence on the golden model.
- The detection granularity of the existing works is summarized, the state-of-the-art is analyzed, and a comprehensive comparison is made.

The rest of the paper is organized as follows. Section II introduces the challenges and motivation. Section III presents the GNN4Gate detailedly. The experimental results and conclusion are shown in Section IV and V respectively.

II. BACKGROUND AND MOTIVATION

In this section, existing static detection based on gate-level netlists are summarized, and the current challenges are listed.

A. Existing Works

Compared with dynamic detection, static detection does not need to run analog circuits or actual circuits, nor does it require additional overhead, and it attracts extensive attention. The current static detection work based on gate-level netlist mainly includes threshold-based and ML-based methods [6].

The threshold-based static detection method [4], [20] quantifies structural features and uses thresholds to classify normal and malicious circuits. [4] created a HT template library for matching suspicious structures, using the SCOAP method to calculate rare values, and generating HT candidates through outlier algorithms. It is difficult to determine an appropriate threshold, because the choice of the threshold has a great influence on the detection results.

Machine learning methods can automatically learn new patterns, process large-scale data, and achieve high performance. Hasegawa et al. [10] first defined the features related to the nets, classified them through Support Vector Machine (SVM), and copied the HT samples to balance the samples. [7], [8], [21], [22] further extended the work of net classification. [22] is a hybrid detection method that used both static and dynamic features. The net classification methods are the finest detection method at present, which can effectively locate the structure of HTs. The related locating work is worthy of further exploration. Features based on controllability and observability analysis are also widely used in detection. [11] and [23] used the SCOPA method to obtain features, and [23] introduced the adaptive synthetic sampling technology (ADASYN) method to balance the samples. Most existing works focused on manual features and highly relied on design experience. Some recent studies [9], [12] have used NLP technology to process structural information to extract features automatically. GramsDet [9] used n-gram technology to extract the paths as corpus, and trained a language model based on LSTM to predict suspicious path structure.

B. Hardware Trojan Detection Granularity

Currently, gate-level HT detection methods have developed into three granularity levels of recognition:

- **Circuit level**. The whole circuit is evaluated and classified, and the circuits containing Trojans are listed.
- **Path level**. The suspicious path structure is classified by checking the path sequence.
- **Component level**. The existing component classification work mainly focuses on net classification, which is the most fine detection level. Classifying suspicious components can further determine the overall structure of HT.

The finer the granularity of detection, the easier it is to distinguish different structures and functions in the circuits, and help people further understand, locate and control HTs.

C. Challenges of Hardware Trojan Detection

In general, the current HTs detection technologies for gate-level netlists have the following challenges:

- **Golden-free**. Golden chips are very difficult to obtain, or even non-existent, so a Golden-free detection model can eliminate many limitations.
- **Features representation**. In machine learning, the definition, selection, and extraction of features are critical to downstream classification tasks.
- **Unbalanced samples**. HTs are usually only a very small part of the circuit, and the use of unbalanced samples often leads to biased training models, which is very unfavorable for anomaly detection.
- **Location**. The locating of HTs is of extraordinary significance to the review of large-scale circuits, and is a prerequisite for people to deeply understand and control HTs in the future.
- **Scalability**. Like antivirus software, a framework that can be applied to detect different types (purposes) of HTs is very necessary and effective, even can save a lot of costs.
- **Automated detection and prevention framework**. A more superior process is that from the original data to the target model application is as automated as possible, with as little manual involvement as possible.

Dedicated methods to meet challenges listed above are worth our efforts. This paper proposes a golden-free HT detection method at gate-level to learn the gate representation automatically for Trojan-gate classification.

III. GNN4Gate: Graph Neural Network for Gate-level Hardware Trojan Detection

In this section, we formally introduce the Trojan-gate classification and GNN4Gate detection framework in detail.

A. Problem Model

The detection is based on the gate-level netlist, which can be provided by the supplier or obtained through the schematic read-back operation [24] of reverse engineering . It is difficult to obtain a complete netlist in many practical use cases for HT detection, so incomplete netlists containing partial structures are also considered in this work. The logic gate i in the netlist can be coded as $\mathbf{X_i} \in \mathbb{R}^d$, where d is the dimension of the coding, and as at whole, the netlist can be represented as $\mathbf{X} \in \mathbb{R}^{n \times d}$. We formulate the HT detection task as a Trojan-gate classification problem as follows:

$$y_i = f(X_i) = \begin{cases} (0,1), & \text{if the gate belongs to a HT} \\ (1,0), & \text{otherwise.} \end{cases} \qquad (1)$$

where f is a classification function. In this paper, we propose GNN4Gate to learn a model to approximate the function f.

B. Detection Framework

The overall architecture of GNN4Gate is shown in Figure 2. In graph modeling stage, all gate types are collected from the gate-level netlists to create a gate library. Then we encode the gate features, and model the forward circuit graph following the signal propagation direction, described in section III-C. In the model training stage, a backward graph is obtained by reversing the edge direction to represent the signal dispersion structure. The forward and backward graph are sent to a Bi-GCN architecture to train a model to classify all gates, described in section III-E (Some technical details about Graph Convolutional Network are described in Section III-D). In the review stage, suspicious gates are located according to the classification results, which facilitates manual review for further HT removal and circuit redesign.

C. Graph Modeling with Port Information

As shown in Figure 3, a directed circuit graph $G =< V, E >$ following the signal propagation direction is created from the netlist, where the node set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ contains n logic gates of the netlist, and edge set $E = \{e_{st} | s, t = 0, \ldots, n-1\}$ represents the connection relationship of the gates. We define $e_{st} \in E$ if the logic gate v_s output port is wired to the gate v_t input port. The adjacency matrix is defined as $\mathbf{A} \in \{0,1\}^{n \times n}$, if $e_{st} \in E$, $a_{st} = 1$, otherwise it is 0. To preserve the circuit structure information as much as possible, three kinds of information are encoded as the features $\mathbf{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ of n nodes, as shown in Figure 3(c):

*1) **Gate Type:*** The primary feature of the node is the type of gate. All the gate types in the netlists are obtained to create a gate library, and then one-hot coding is used to encode the gate types as initial node features.

*2) **Port Connection:*** The gate may have different port types. As shown in Figure 3(a), the flip-flop contains several types of input ports, including Data (D), Clock (CLK) and Reset

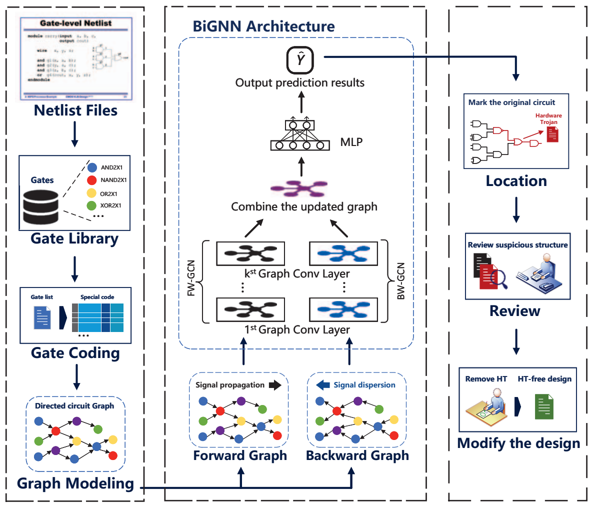

Fig. 2. The overall workflow of GNN4Gate. The whole process includes three stages: circuit graph modeling, GNN model training and HT review.

ports. If only the gate type code is used to make a simple abstraction of the circuit, there will be no difference between all predecessor connections of a node, and the port connection information will be lost.

To solve this problem, a special coding method is proposed to embed the connection information on the edge into the node features. Specifically, four special ports are considered, including SI, SE, SN, and RN ports. The connection port is coded as a feature of the predecessor node. In other words, if the output of the gate is connected to a special port, the corresponding feature bit is set to 1, otherwise is 0.

*3) **Primary Input and Output:*** The primary input and primary output are treated as special port connections and encoded in the same way as above. If a gate contains the input wires, the corresponding feature bit is coded as 1, otherwise 0. The gate containing the output wires is coded in the same way.

D. Graph Convolutional Networks

The node-based Graph Convolutional Networks (GCN) [25] exchanges information between neighboring nodes through message propagation as follows:

$$\mathbf{H}_k = M(\mathbf{A}, \mathbf{H}_{k-1}; \mathbf{W}_{k-1}), \qquad (2)$$

where $\mathbf{H} \in \mathbb{R}^{n \times v_k}$ is the hidden layer feature matrix obtained through the $k\!-\!th$ Graph Convolutional Layer (GCL) operation, and v_k is the number of hidden units of the corresponding GCL. $M(\cdot)$ is a message propagation function with a learnable parameter $\mathbf{W}_{k-1}$. The node information can be aggregated through the adjacency matrix $\mathbf{A}$ and the feature matrix $\mathbf{H}_{k-1}$ of the previous layer. The most popular message propagation function $M(\cdot)$ in existing works is the following form [25]:

$$\mathbf{H}_k = \sigma(\hat{\mathbf{D}}^{\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}_{k-1} \mathbf{W}_{k-1}), \qquad (3)$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-connection added. $\hat{\mathbf{D}}$ is the degree matrix calculated by the adjacency matrix $\hat{\mathbf{A}}$. $\sigma(\cdot)$ is the activation function.

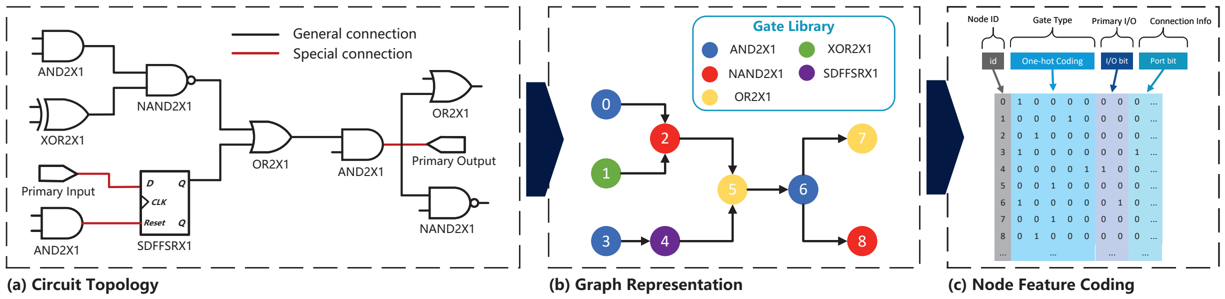

Fig. 3. The circuit graph modeling process of the gate-level netlist.

E. GNN4Gate

Based on Bi-GCN [19], GNN4Gate can learn high-level representations from circuit signal propagation and signal dispersion structure. Specifically, the HT detection process can be divided into 3 steps.

1) ***Construct Forward and Backward Graphs:*** According to the gate-level netlist, the directed circuit graph $G =< V, E >$ is created, $\mathbf{A}$ and $\mathbf{X}$ are the adjacency matrix and the node feature matrix of G, respectively. To classify a gate, its local structure information needs to be considered. In other words, it is necessary to aggregate the features of successor nodes. However, in a directed graph, the node features can only be aggregated along the edge direction. One may consider to use an undirected graph to aggregate the features of the bi-directional node information at the same time, but the propagation sequence information will be lost. To this end, in addition to the forward graph, the backward graph is introduced by reversing the edge direction of the forward graph, which can aggregate the features of successor nodes in the original forward graph along the reverse edge. In the forward graph, the adjacency matrix $\mathbf{A}^{FW} = \mathbf{A}$, and in the backward graph, the adjacency matrix $\mathbf{A}^{BW} = \mathbf{A}^{\top}$. The two graphs share the same feature matrix $\mathbf{X}$.

2) ***Gate Representation Learning:*** GNN4Gate uses two independent two-layer GCL backbones as the basic components of the feature extraction for forward and backward graph, respectively named FW-GCN and BW-GCN, as shown in Figure 2(b). We use Eq.(2) to describe the process of feature extraction as below:

$$\mathbf{H}_1^{FW} = M(\mathbf{A}^{FW}, \mathbf{X}; \mathbf{W}_0^{FW}), \tag{4}$$

$$\mathbf{H}_2^{FW} = M(\mathbf{A}^{FW}, \mathbf{H}_1^{FW}; \mathbf{W}_1^{FW}), \tag{5}$$

where the specific operation of the propagation function M follows Eq.(3), and ReLU is used as the activation function σ. The input of the first layer GCL is the feature matrix $\mathbf{X}$. $\mathbf{H}_1^{FW}$ and $\mathbf{H}_2^{FW}$ are the hidden features corresponding to the two layers of FW-GCN, and $\mathbf{W}_0^{FW}$,$\mathbf{W}_1^{FW}$ are the weights of corresponding layers. We use the same settings in BW-GCN and get the corresponding feature $\mathbf{H}_1^{BW}$ and $\mathbf{H}_2^{BW}$. The obtained signal propagation and dispersion structure features are concatenated into the final node features, denoted as

$$\mathbf{H}^C = CONCAT(\mathbf{H}_2^{FW}, \mathbf{H}_2^{BW}). \tag{6}$$

The two-layer GCN makes the gate nodes focus on local structural features, so GNN4Gate is adapted to subgraphs modeled by incomplete netlists that contain partial structures.

3) ***HT Classification:*** In order to get the final classification output, a fully connected layer and a Softmax layer are used to learn the prediction function, specifically,

$$\hat{\mathbf{Y}} = Softmax(FC(\mathbf{H}^C)), \tag{7}$$

where $\hat{\mathbf{Y}}$ is a matrix of predicted probabilities for all gates, used to predict the label of each gate. In order to solve the problem of sample imbalance, the weighted cross-entropy loss function is used for HT classification as follows:

$$Loss_{CE} = \frac{1}{N}\sum_{i=1}^{N} w_p \cdot y_i \cdot \log \hat{y_i} + w_n \cdot (1-y_i) \cdot \log(1-\hat{y_i}), \tag{8}$$

where N is the number of gates, y_i is the predicted output value of the $i-th$ gate, $\hat{y_i}$ is the actual value, w_p and w_n are the weights of positive and negative categories respectively. The weight vector $(w_p,\ w_n)$ is set according to the approximate ratio of the positive and negative samples in each netlist, and is not used as a hyperparameter. For example, the ratio of the Trojan gate to the normal gate for RS232-T1000 is 13:202 and thus the weight vector $(w_p,\ w_n)$ is set to (0.0644, 1).

IV. EXPERIMENT SETTING AND RESULTS

A. Datasets and Settings

Datasets Creation: Based on the benchmark provided by Trusthub [26], we use the method introduced in Section III-C to create the circuit graph datasets. Each gate is encoded into a 65-dimensional feature representation (59 one-hot code bits and 6 special port bits). The basic information of the resultant 17 datasets is summarized in Table II.

Experimental Setup: The setting of this experiment refers to the general setting of GNN training. The number of hidden units in each GCL layer is recommended to be 32 or 64 (set to a multiple of 2 is good for calculation efficiency), and set it to 32 in this experiment. GNN4Gate uses one fully connected layer to reduce the number of hidden units to 2 to predict the result of HT detection. In addition, a Dropout layer with the rate of 0.5 is added after each GCN layer to avoid overfitting. We use the Batch Gradient Descent algorithm with the learning rate of 0.01 to update the parameters of GNN4Gate for 200 epochs, and use the Adam algorithm to optimize the model.

B. HT Detection

In order to evaluate the performance of GNN4Gate, we use the same leave-one-out cross validation method as used in [7].

TABLE I
Compare the performance of GNN4Gate with the state-of-the-art HT detection method at gate-level.

Paper (Year)	Classification Object	Technology Type - Method	Data Balancing	TPR	TNR	ACC	Automatic feature extraction	Location
[4](2019)	Circuit	Threshold - Template library	-	NA	NA	78%	✗	✗
[12](2017)	Path	ML - Language Model	-	96.5%	97.7%	NA	✓	✗
[9](2019)	Path	ML - Recurrent Neural Network	-	80.7%	95.5%	NA	✓	✗
[7](2020)	Net	ML - Neural Networks	Duplication	84.6%	95.1%	NA	✗	✓
[22](2020)	Net	ML - XGBoost (Hybrid detection)	-	90.9%	99.0%	NA	✗	✓
[23](2019)	Net	ML - Multi-Model	ADASYN	82.46%	98.99%	98.26%	✗	✓
GNN4Gate	Gate	ML - Graph Neural Network	Weighted Loss Function	87.14%	**99.73%**	**99.61%**	✓	✓

TABLE II
Compare the performance of GNN4Gate and net classification methods on each netlist

Benchmark [26]	Normal Gate	Trojan Gate	Effect Tpye	GNN4Gate						[7](2020)		[22](2020)		[23](2019)	
				TN	FP	FN	TP	TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR
RS232-T1000	202	13	Change Function	202	0	0	13	100.00	100.00	100.00	96.80	100.00	97.50	84.09	99.25
RS232-T1100	204	12	Change Function	204	0	0	12	100.00	100.00	100.00	96.40	80.00	97.60	80.95	100.00
RS232-T1200	202	14	Change Function	202	0	0	14	100.00	100.00	100.00	97.10	81.80	98.00	78.79	95.39
RS232-T1300	204	9	Change Function	204	0	0	9	100.00	100.00	100.00	93.90	100.00	97.60	87.10	98.19
RS232-T1400	202	13	Change Function	202	0	0	13	100.00	100.00	100.00	98.00	97.70	100.00	86.96	99.62
RS232-T1500	202	14	Change Function	202	0	0	14	100.00	100.00	100.00	94.90	100.00	99.20	93.33	96.48
RS232-T1600	202	12	Change Function	202	0	0	12	100.00	100.00	100.00	96.80	100.00	97.20	80.00	95.74
s35932-T100	5426	15	Leak Information	5425	1	1	14	93.33	99.98	57.10	98.40	76.40	100.00	80.00	100.00
s35932-T200	5422	16	Denial of Service	5422	0	5	11	68.75	100.00	41.70	94.50	58.30	100.00	83.33	100.00
s35932-T300	5426	36	Degrade Performance	5426	0	24	12	33.33	100.00	100.00	99.40	94.40	100.00	30.56	100.00
s38417-T100	5329	12	Denial of Service	5325	4	0	12	100.00	99.92	81.80	98.50	100.00	100.00	91.67	99.95
s38417-T200	5329	15	Denial of Service	5321	8	3	12	80.00	99.85	90.90	94.60	100.00	99.80	80.00	99.95
s38417-T300	5358	15	Denial of Service	5329	29	0	15	100.00	99.46	100.00	94.50	93.60	100.00	100.00	99.95
Average	-	-	-	-	-	-	-	90.42	**99.94**	90.12	96.45	**90.94**	98.99	81.29	98.81
s15850-T100	2155	27	Denial of Service	2084	71	3	24	88.89	96.71	80.80	92.60	-	-	88.89	99.33
s38584-T100	6473	9	Denial of Service	6467	6	6	3	33.33	99.91	15.80	99.00	-	-	73.68	99.99
s38584-T200	6473	83	Leak Information	6463	10	4	79	95.18	99.85	59.80	88.00	-	-	-	-
s38584-T300	6473	731	Leak Information	6457	16	84	647	88.51	99.75	85.90	94.80	-	-	-	-
Total Average	-	-	-	-	-	-	-	**87.14**	**99.73**	83.55	95.81	-	-	81.29	98.92

Each time one of the netlists in Table II is selected as the test set (unknown netlist), and the other netlists are used as the train set (known netlists). In the experiments, the classification results of each netlist are recorded, and True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) are counted. Three ML indicators are used to evaluate the related models, including True Positive Rate (TPR, also known as Recall), True Negative Rate (TNR) and Accuracy (ACC), that can be calculated as follows:

$$TPR = \frac{TP}{TP + FN}, TNR = \frac{TN}{FP + TN}, ACC = \frac{TP + TN}{TP + TN + FP + FN}.$$

Table I and II show the performances of the state-of-the-art and GNN4Gate at gate-level. From Table I, one can find that, the average TPR and TNR of GNN4Gate reach 87.14% and 99.73% respectively, which are highly competitive to others. More importantly, GNN4Gate can automatically extract features and locate relevant HT structures, showing great potential in gate-level detection. We compare the performance of GNN4Gate with the lasted net classification works on each netlist in Table II. Significantly, GNN4Gate achieved very high TNR on all data sets. When compared with the results of [22], the average TPR of GNN4Gate on the corresponding data set reaches 90.42%, which is very close to [22]. It is worth noting that [22] is a hybrid detection, which uses dynamic information in addition to the static structure features. Therefore, we expect that the structural features extracted by GNN can be further combined with dynamic detection to achieve better results. Moreover, due to the lack of sample support for similar Trojan structures, the test results of GNN4Gate on the s35932-T300 and s38584-T100 netlists are not good. One can see similar abnormal results on the s35932-T300 of [23] and the s38584-T100 of [7]. Compared with [7] using a wide range of local structure information (e.g. the number of up to 5-level loops on the output side), GNN4Gate focuses more on the neighbor structure information of the gate (up to 2-level neighbors), so it is more adapt to incomplete netlists.

C. Ablation Study

In order to further analyze the effectiveness of graph modeling with port information and Bi-GCN framework for HT detection, we compare the resultant variants with the standard model GNN4Gate in two scenarios (with or without port coding). We use the same experimental settings as in Section IV-B and report the average performance of each model on 17 netlists. The result is shown in Figure 4, where FW-GCN and UD-GCN represent GCN detection model using forward and undirected graph structures respectively. Under the precise weighted sample balance described in Section III-E, the TPR of all the relevant models has been improved in the port coding scenario, and the TNR has been maintained at a high level. From Figure 4, one can find that, by aggregating both the forward and backward neighbors' information, UD-GCN achieves better performances than FW-GCN does, but it loses the sequence information of signal propagation, which leads to worse performances compared with the standard model

GNN4Gate. Obviously, UD-GCN is also comparable to existing works, but the port information brings relatively little improvement. Because the port connection has a certain directionality, and can be fully utilized in the directed graph.

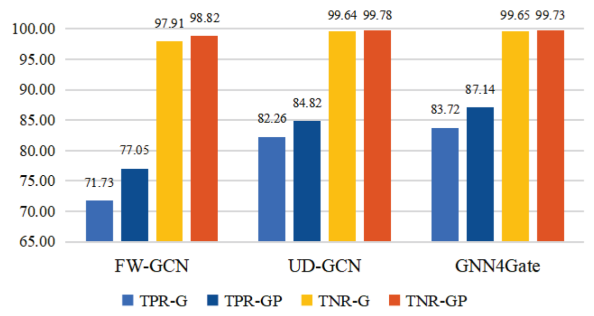

Fig. 4. The HT detection performance of the GCN-based methods on two kinds of coded datasets (G means only gate type coding, GP means coding combined with port information).

TABLE III
RESULTS OF GNN4GATE ON NETLIST S35932-T200

Result	TN: 5422, FP: 0, FN: 5, TP: 11
HT gates	U5548, U5566, U6740, U6802, Trojan1, Trojan2, Trojan3, Trojan4, Trojan1234_NOT, Trojan5, Trojan6, Trojan7, Trojan8, Trojan5678_NOT, INVtest_se, Trojan_Trigger
Suspicious gates	**Trojan1, Trojan2, Trojan3, Trojan4, Trojan1234_NOT, Trojan5, Trojan6, Trojan7, Trojan8, Trojan5678_NOT, INVtest_se**

D. HT Location

The goal of HTs location is to determine the HTs' structure from large-scale circuits, so that people can better understand and control HT. At present, there is very little work on HTs location. Although, both net classification and gate classification are the finest HT detection methods at gate-level, more efforts are necessary for the former to determine the relevant gates to have a comprehensive understanding and better control on HTs. The latter, such as GNN4Gate, can get the location of Trojan-gates directly.

As shown in Table III, GNN4Gate gives the prediction results of gate classification and lists the suspicious logic gates on the s35932-T200 netlist, the complete HT structure includes 16 Trojan-gates, of which 11 are detected by GNN4Gate. Sometimes, some normal gates are mistakenly classified as Trojan-gates, but GNN4Gate has a high TNR, and the number of misclassifications will be very small compared to the whole circuit. Professionals can directly review the local structure of the suspicious gates without reviewing the whole circuit. HT location can be used as a further work of HT detection, which has greater application needs.

V. CONCLUSION

This paper presents a Gold-free HT detection method to identify Trojan-gate. The netlist is modeled as a directed graph using the proposed coding method combining port information. Based on Bi-GCN, the proposed GNN4Gate model automatically extracts features, learns gate representation, realizes gate classification for the first time, and detects and locates HTs. Experimental results show that the proposed model can achieve comparable performance to the state-of-the-art works.

ACKNOWLEDGMENT

The work was supported by the Natural Science Foundation of Fujian Province, China (No.2018J01793, No.2018J01800, No.2020J01500), the Foundation of the Education Department of Fujian Province, China (No.JAT190025).

REFERENCES

[1] C. Dong *et al.*, "A multi-layer hardware trojan protection framework for iot chips," *IEEE Access*, 2019.
[2] B. Shakya *et al.*, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, 2017.
[3] C. Dong *et al.*, "Hardware trojans in chips: a survey for detection and prevention," *Sensors*, 2020.
[4] Q. Liu, P. Zhao, and F. Chen, "A hardware trojan detection method based on structural features of trojan and host circuits," *IEEE Access*, 2019.
[5] S. Faezi *et al.*, "Htnet: Transfer learning for golden chip-free hardware trojan detection," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
[6] Y. Yang *et al.*, "Survey: Hardware trojan detection for netlist," in *IEEE 29th Asian Test Symposium (ATS)*, 2020.
[7] T. Kurihara *et al.*, "Evaluation on hardware-trojan detection at gate-level ip cores utilizing machine learning methods," in *IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2020.
[8] K. Hasegawa *et al.*, "Hardware trojans classification for gate-level netlists using multi-layer neural networks," in *IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017.
[9] R. Lu *et al.*, "Gramsdet: Hardware trojan detection based on recurrent neural network," in *IEEE 28th Asian Test Symposium (ATS)*, 2019.
[10] K. Hasegawa *et al.*, "Hardware trojans classification for gate-level netlists based on machine learning," in *IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016.
[11] 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*, 2016.
[12] H. Shen *et al.*, "Lmdet: A "naturalness" statistical method for hardware trojan detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.
[13] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[14] M. Fyrbiak *et al.*, "Graph similarity and its applications to hardware security," *IEEE Transactions on Computers*, 2019.
[15] R. Yasaei *et al.*, "Gnn4ip: Graph neural network for hardware intellectual property piracy detection," *arXiv preprint arXiv:2107.09130*, 2021.
[16] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque, "Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
[17] S.-Y. Yu *et al.*, "Hw2vec: A graph learning tool for automating hardware security," *arXiv preprint arXiv:2107.12328*, 2021.
[18] C. Dong *et al.*, "A locating method for multi-purposes hts based on the boundary network," *IEEE Access*, 2019.
[19] T. Bian *et al.*, "Rumor detection on social media with bi-directional graph convolutional networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
[20] S. Yao *et al.*, "Fastrust: Feature analysis for third-party ip trust verification," in *IEEE International Test Conference (ITC)*, 2015.
[21] K. Hasegawa *et al.*, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.
[22] S. Li *et al.*, "A xgboost based hybrid detection scheme for gate-level hardware trojan," in *IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 2020.
[23] C. H. Kok *et al.*, "Classification of trojan nets based on scoap values using supervised learning," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.
[24] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proceedings of the 48th Design Automation Conference*, 2011.
[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[26] "Trusthub," *Available on-line: https://www. trust-hub. org*, 2016.