GNN4IP: Graph Neural Network for Hardware Intellectual Property Piracy Detection

Rozhin Yasaei*, Shih-Yuan Yu*, Emad Kasaeyan Naeini, Mohammad Abdullah Al Faruque
Department of Electrical Engineering and Computer Science
University of California, Irvine, California, USA
ryasaei, shihyuay, ekasaeya, alfaruqu@uci.edu

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586150

***Abstract*—Aggressive time-to-market constraints and enormous hardware design and fabrication costs have pushed the semiconductor industry toward hardware Intellectual Properties (IP) core design. However, the globalization of the integrated circuits (IC) supply chain exposes IP providers to theft and illegal redistribution of IPs. Watermarking and fingerprinting are proposed to detect IP piracy. Nevertheless, they come with additional hardware overhead and cannot guarantee IP security as advanced attacks are reported to remove the watermark, forge, or bypass it. In this work, we propose a novel methodology, GNN4IP, to assess similarities between circuits and detect IP piracy. We model the hardware design as a graph and construct a graph neural network model to learn its behavior using the comprehensive dataset of register transfer level codes and gate-level netlists that we have gathered. GNN4IP detects IP piracy with 96% accuracy in our dataset and recognizes the original IP in its obfuscated version with 100% accuracy.**

***Index Terms*—Hardware intellectual property- IP piracy - Graph convolutional network - Data flow graph**

I. Introduction

The integrated circuits (IC) manufacturing industry has developed significantly and scaled down to 7nm technology that has made the integration of numerous transistors possible. However, the hardware engineers could not keep up with rapid advancement in the fabrication technology, and they fail to use all of the available transistors in the die. To close this productivity gap under time-to-market pressure, hardware Intellectual Property (IP) core design has grabbed substantial attention from the semiconductor industry and has dramatically reduced the design and verification cost [1].

The globalization of the IC supply chain poses a high risk of theft for design companies that share their most valuable assets, IPs, with other entities. IP piracy is a serious issue in the current economy, with a drastic need for an effective detection method. According to the U.S. Department of Commerce study, 38% of the American economy is composed of IP-intensive industries [2] that lose between $225 billion to $600 billion annually because of Chinese companies stealing American IPs mainly in the semiconductor industry, based on the U.S. Trade Representative report [3].

Hardware IP is considered as any stand-alone component of a system-on-chip design that is classified into three categories based on the level of abstraction: Soft IP (i.e., synthesizable HDL source code), Firm IP (i.e., netlists and placed RTL block), and Hard IP (i.e., GDSII and physical layout) [4].

* Both are equal contributing authors. Yasaei is the corresponding author.

Conventionally, the IP protection techniques fall into preventive (i.e., logic encryption, camouflaging, metering, and split manufacturing) and detective (i.e., digital signature) methods. All these methods add excessive implementation overhead to the hardware design that limits their applications in practice. Moreover, they mainly focus on security at the IC level, while many commercial IPs comprise the soft IPs due to flexibility, independence of platform technology, portability, and easy integration with other components. The high level of abstraction makes the IP protection more challenging since it is easier for an adversary to slightly change the source code and redistribute it illegally at the lower levels of abstraction. Although the existing preventive countermeasures deter IP theft, they cannot guarantee IP security as the adversaries keep developing more sophisticated attacks to bypass them. **Therefore, an effective IP piracy detection method is crucial for IP providers to disclose the theft.**

To this end, the state-of-the-art piracy detection method is embedding signatures of IP owner known as watermark and legal IP user known as a fingerprint in the circuit design to assure authorship and trace legal/illegal IP usage. IP watermarking and fingerprinting are prone to removal, masking, or forging attacks that attempt to omit the watermark, distort its extraction process, or embed another watermark in IP [4].

In this work, we propose a novel methodology for IP piracy detection that, instead of insertion and extraction of a signature to prove the ownership, models the circuits and assess the similarity between IP designs. Therefore, our method does not require additional hardware overhead as the signature and is not vulnerable to removal, masking, or forging attacks. It also effectively expose the infringement between two IPs when the adversary complicates the original IP to deceive the IP owner. Modeling the hardware design is challenging since it is a structural non-Euclidean data type, despite most modeling techniques. Thus, similar to [5], we represent the circuit as a data-flow graph (DFG) due to similar data types and properties. Afterward, we model it using state-of-the-art graph learning method.

A. Motivational Example

We study the concept of piracy and similarity among hardware designs in a test case of two different variations of the full adder circuit. As shown in Figure 1, although the Verilog codes for adder 1 and 2 are different, they both have fundamentally

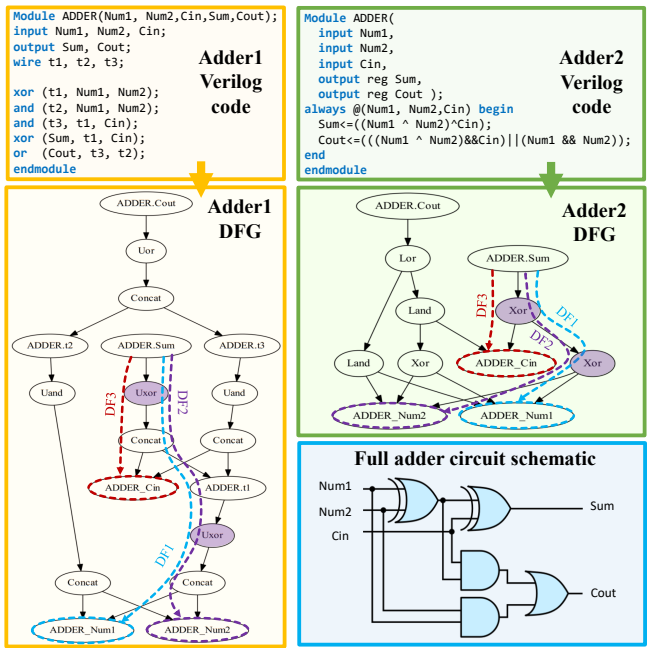

Fig. 1. The circuit schematic, Verilog codes, and DFGs of full adder circuits.

the same design, as depicted in the schematic figure. We unveil the similarity between two adders using DFG, which expresses the signals dependency and computational structure. At first glance, the generated DFGs for adders seem varied, but a deep look into data flows (DF) indicates the same signal relations. For instance, the output signal *Sum* depends on *Num1*, *Num2*, and *Cin* input signals through the DF1, DF2, and DF3, respectively. Suppose we focus on critical nodes in the flow (XOR nodes) and ignore the excessive nodes related to concatenation and internal signals. In that case, the DFs in both DFGs represent the same operations.

B. Research Challenges

The development of an effective IP piracy detection method poses paramount research challenges as follows:

- **Hardware overhead**: All existing piracy detection methods add hardware overhead to IP design.
- **Attacks**: Signatures-based countermeasures are vulnerable to removal, forging, and masking attacks.
- **Same behaviors, different topologies**: As the case study exemplified, varied HDL codes generate different DFGs even if they represent the same hardware design. The different typologies in DFGs can easily fool the standard graph similarity algorithms, and behavioral analysis of graphs is required to learn circuit design.
- **Scalability**: The manual review of hardware design is not feasible in practice. The graph similarity is an NP-complete problem, and existing algorithms [6] suffer from high complexity and are not scalable to large designs and industrial-level IPs with thousands of code lines.

C. Our contribution

We propose a novel methodology based on graph learning to surmount research challenges and propose these contributions:

- To overcome the shortcomings of current IP piracy detection methods, we propose a novel countermeasure based on hardware design analysis that does not require adding any signature and overhead to IP design.
- We develop a scalable, automated framework called *hw2vec* that generates the DFG for hardware designs and assigns a embedding to them such that the proximity in the embeddings indicates similarity between circuits.
- We construct a Graph Neural Network (GNN) model to learn the circuit's behavior and assess the similarity between a pair of IPs according to graph embeddings.
- We gather a dataset of hardware designs in RTL and gate-level netlist to develop and assess our methodology.

II. Backgrounds and Related Works

A. Hardware IP Security

The hardware is susceptible to security threats such as IP piracy (unlicensed usage of IP), overbuilding, counterfeiting (producing a faithful copy of circuit), reverse engineering, hardware Trojan (malicious modification of circuit) [5], [7], [8], and side-channel attacks [9]. The IP protection methods proposed in the literature can be classified as follows:

Watermarking and fingerprinting [10]: The IP owner and legal IP user's signatures, known as watermark and fingerprint, are added to circuit to prove infringement.

Hardware metering [11]: The designer assigns a unique tag to each chip, which can be used for chip identification (passive tag) or enabling/disabling the chip (active tag).

Obfuscation [1]: There are two obfuscation methodologies; **logic locking** (encryption) [12] and **IC camouflaging** [13]. In logic locking, additional gates such as XOR are inserted in non-critical wires. The circuit would be functional only if the correct key is provided which is stored in a secure memory out of reach of the attacker. Camouflaging modifies the design such that cells with different functionalities look similar to the attacker and confuses the reverse engineering process.

Split manufacturing [14]: IP house split the design to separate ICs and have them fabricated in different foundries. Thus, none of the foundries have access to the whole design to overbuild, reverse engineer, or perform malicious activities.

The existing defenses suffer from a large overhead on area, power, and timing that restrict their application. As the new countermeasures are developed, the attacks are advanced to bypass them. SAT attack is a powerful method used to formulate and solve a sequence of SAT formulas iteratively to unlock the encrypted circuit, reverse engineer the Boolean functionalities of camouflaged gates [15], or reconstruct the missing wire in 2.5D split manufactured ICs [16]. Anti-SAT [17], and AND-tree insertion [18] obfuscation techniques are proposed to mitigate SAT attack. However, signal probability skew attack, AppSAT guided removal attack, and sensitization guided SAT attack [19] break them. Proximity attack [20] is another attack against 2.5D split manufacturing that iteratively connects the inputs to outputs in two IC partitions until a loop is formed. Removal, masking, and forging attacks bypass watermarking by eliminating, distorting, or embedding a ghost watermark [4]. There is a rising trend in machine learning-based defenses [5], [21] and the recent advances made the models even resistant against adversarial attacks [22].

B. Graph Neural Networks (GNNs)

In *GNN4IP*, we leverage GNN, a deep learning methodology that tackles graph data [23]. Several works in the literature have shown the effectiveness of GNN in identifying software clones and detecting binary code similarity [24], [25]. Our architecture is inspired by the Spatial-based Graph Convolution Neural Network, which defines the convolution operation based on a node's spatial relations with the following phases: (i) *message propagation phase* and (ii) the *read-out phase*. The *message propagation* phase involves two sub-functions: **AGGREGATE** and **COMBINE**, given by,

$$a_v^{(k)} = \textbf{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in N(v)\}), \quad (1)$$

$$h_v^{(k)} = \textbf{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k)}), \quad (2)$$

where $h_v^{(k)} \in R^{C^k}$ denotes the node embedding after k iterations for the v_{th} node. Essentially, the **AGGREGATE** function collects the features of the neighboring nodes to extract an aggregated embedding $a_v^{(k)}$ for the layer k, and the **COMBINE** function combines the previous node features $h_v^{(k-1)}$ with $a_v^{(k)}$ to output next embedding $h_v^{(k)}$. This message propagation is carried out for a pre-determined number of iterations k. Next, in the *read-out* phase, the overall graph-level embedding extraction is carried out by either summing up or averaging up the node embeddings in each iteration. The graph-level embedding is denoted as $h_G^{(k)}$ and is defined as,

$$h_G^{(k)} = \textit{READOUT}(\{h_v^{(k-1)} : v \in G\}) \quad (3)$$

In our work, we use $h_G^{(k)}$ as the hardware design embedding to assess the similarity between circuits and discover piracy.

III. GNN4IP Methodology

In this work, we formulate the problem of IP piracy detection as finding the similarity between two hardware designs. We assume the existence of a feed-forward function f that outputs whether two circuits p_A and p_B are subject to piracy or not through a binary label y as given in Equation 4.

$$y = f(p_A, p_B) = \begin{cases} (1,0) & \text{if piracy in } p_A, p_B \\ (0,1) & \text{if no-piracy in } p_A, p_B \end{cases} \quad (4)$$

To approximate f, we extract the DFGs G_A and G_B from circuits pair (p_A, p_B) using the *DFG generation pipeline* and pass it to a graph embedding layer, *hw2vec*, to acquire embeddings (h_{G_A}, h_{G_B}). Lastly, our model infer the piracy label, $\hat{Y}$, by computing the cosine similarity between (h_{G_A}, h_{G_B}).

A. Threat Model

In our threat model, we examine the IP designs in RTL or gate-level netlist to discover piracy. We assume that the design is a soft IP, firm IP, or derived by reverse engineering a hard IP or IC. The adversary can be a hardware designer, competitor company, or the fabrication foundry who present the stolen IP as their genuine design and sell it as an IC or an IP at the same or lower level of abstraction. The attack scenario may involve modification of IP design to tamper piracy detection. The attacker can get access to the original IP through one of these means: I) purchase the IP for limited usage, II) leaked through a rogue employee in the design house, or III) reverse engineered the physical layout or IC.

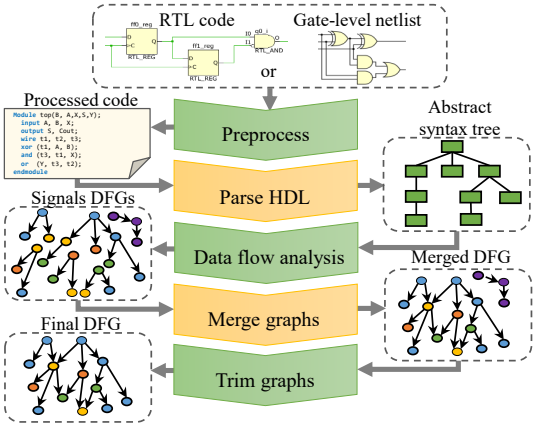

Fig. 2. Data flow graph generation pipeline for RTL code and netlist.

B. Hardware Data Flow Graph Extraction

Hardware design is non-Eulicidian structural data that shares similar properties with a graph. We generate DFGs from either RTL code or gate-level netlist as the first step to model it. The DFG is a rooted directed graph illustrating the computation structure and the data flow from the circuit's output signals (the root nodes) to the input signals (the leaf nodes). It is defined as graph $G = (V, E)$ where $V = \{v_1, v_2, ..., v_n\}$ is the vertices set and each node v_i represents a signal, constant value, or operations such as concatenation, branch, Boolean operators, etc. We define set of directed edges $E = e_{ij}$ for all i, j such that $e_{ij} \in E$ if the operation v_j is applied on v_i or the value of v_i depends on the value of v_j.

To extract DFG, we develop an automated framework using a hardware design toolkit called Pyverilog [26]. Figure 2 demonstrates our DFG generation pipeline that is consisted of five phases: preprocess, parser, data flow analysis, merge, and trim. The procedure begins with preprocessing the RTL code or gate-level netlist in Verilog format to flatten the modular codes and resolve incompatibilities and syntax errors. Afterward, the parser scans the code and produces the corresponding abstract syntax tree used by the data flow analyzer to generate a data flow tree per signal. Next, the signal's trees are merged to construct one main DFG for the whole design. Eventually, the redundant nodes and disconnected subgraphs are trimmed, and the final DFG is generated.

C. Hardware IP Piracy Detection Algorithm

Our IP piracy detection algorithm is shown in Algorithm 1. In the algorithm, *GNN4IP* refers to approximating function f, which can yield the inference of whether two circuits p_1 and p_2 are subject to IP piracy. Applying typical machine learning methodologies to hardware designs, which are non-euclidean in nature, usually requires feature engineering and immense expert knowledge in hardware design. Thus, we propose our scalable, automated IP piracy detection framework, *hw2vec* with an architecture depicted in Figure 3.

The *hw2vec* uses the DFG generation pipeline and acquires the corresponding graph G for circuit p in the form of $(\mathbf{X}, \mathbf{A})$ where $\mathbf{X}$ represents the initial list of node embeddings and $\mathbf{A}$ stands for the adjacency information of G. Next, the

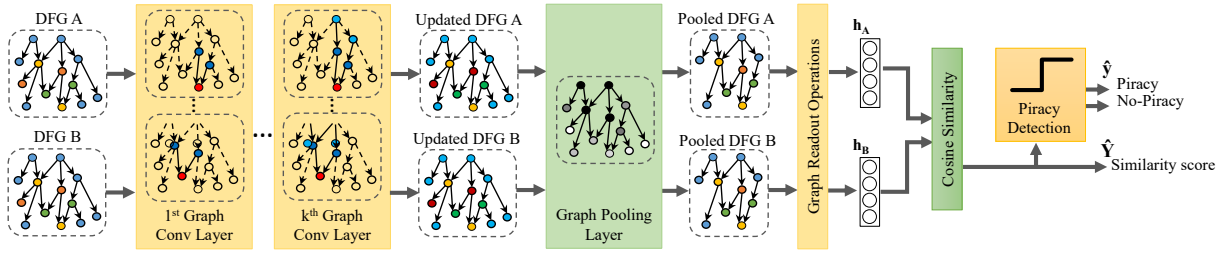

Fig. 3. The overall architecture of *GNN4IP* for hardware IP piracy detection.

hw2vec begins the message propagation phase, denoted as Graph_Conv in the algorithm, which is Graph Convolution Network (GCN) [27]. In each iteration l of message propagation, the node embeddings $\mathbf{X}^{l+1}$ will be updated as follows,

$$\mathbf{X^{(l+1)}} = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}\mathbf{X^{(l)}}W^{(l)}) \tag{5}$$

where W^l is a trainable weight used in the GCN layer. $\widehat{A} = A + I$ is the adjacency matrix of G used in the layer for aggregating the feature vectors of the neighboring nodes where I is an identity matrix that adds the self-loop connection to make sure the features calculated in the previous iteration will also be considered in the current iteration. $\widehat{D}$ is the diagonal degree matrix used for normalizing $\widehat{A}$. $\sigma(.)$ is the activation function such as Rectified Linear Unit (ReLU). Here, we denote the initial node embedding as $X^{(0)}$ and initialize each node embedding $\mathbf{X}_i^{(0)}$, $\forall i \in V$, by directly converting the node's name to its corresponding one-hot vector. We denote the final propagation node embedding $\mathbf{X}^{(l)}$ as $\mathbf{X}^{prop}$, and denote the corresponding adjacency matrix as $\mathbf{A}^{prop}$.

Once propagated the information on G, the resultant node embedding $\mathbf{X}^{prop}$ is further processed with an attention-based graph pooling layer Graph_Pool. Denote the collection of all node embeddings of G after passing through L layers of GCN as $\mathbf{X}^{prop}$. The $\mathbf{X}^{prop}$ is passed to a self-attention graph pooling layer that learns to filter out irrelevant nodes from the graph, creating the pooled set of node embeddings $\mathbf{X}^{pool}$ and their edges $\mathbf{A}^{pool}$. In this layer, we use a graph convolution layer to predict the scoring $\alpha = \mathbf{SCORE}(\mathbf{X}^{prop}, \mathbf{A}^{prop})$ and use α to perform *top-k* filtering over the nodes in the DFG [28].

Algorithm 1: Hardware IP Piracy Detection Algorithm

Input: Hardware design programs p_1, p_2.
Output: A label indicating whether p_1, p_2 is piracy.
def `hw2vec(p)`**:**
  $X, A \leftarrow$ GraphExtraction(p)
  $X^{prop}, A^{prop} \leftarrow$ Graph_Conv(X, A)
  $X^{pool}, A^{pool} \leftarrow$ Graph_Pool(X^{prop}, A^{prop})
  $h_G \leftarrow$ Graph_Readout(X^{pool})
  return h_G
def `gnn4ip(p1, p2)`**:**
  $h_{p_1}, h_{p_2} \leftarrow$ *hw2vec*(p_1), *hw2vec*(p_2)
  $\hat{Y} \leftarrow$ Cosine_Sim(h_{G_1}, h_{G_2})
  if $\hat{Y} > \delta$ **then**
    return 1
  else
    return 0
gnn4ip(p_1, p_2) // run the GNN4IP check.

Then, the Graph_Readout in our algorithm aggregates the node embeddings $\mathbf{X}^{pool}$ to acquire the graph-level embedding $\mathbf{h}_G$ for the DFG G using this formula $\mathbf{h}_G = \mathbf{READOUT}(\mathbf{X}^{pool})$. The **READOUT** operation can be either summation, averaging, or selecting the maximum of each feature dimension over all the node embeddings, denoted as *sum-pooling*, *mean-pooling*, or *max-pooling* respectively. Lastly, *hw2vec* returns the embedding $\mathbf{h}_G$ of each hardware.

The *gnn4ip* utilizes *hw2vec* to transform p_1 and p_2 into the corresponding DFG embeddings, denoted as h_{p_1} and h_{p_2}. Then, it calculates the cosine similarity of h_{p_1} and h_{p_2} to produce the final IP piracy prediction, denoted as $\hat{Y} \in [-1, 1]$. The formula can be written as follows,

$$\hat{Y} = \text{Cosine\_sim}(h_{p_1}, h_{p_2}) = \frac{h_{p_1} \cdot h_{p_2}}{|h_{p_1}||h_{p_2}|} \tag{6}$$

Finally, our *gnn4ip* utilizes predefined decision boundary δ and $\hat{Y}$ to judge whether two programs p_1 and p_2 are a piracy to one another as described in Algorithm 1 and to return the results of IP piracy detection using a binary label (0 or 1).

As both *gnn4ip* and *hw2vec* include several trainable parameters, we need to train these parameters for IP piracy detection via computing the cosine embedding loss function, denoted as H, between true label Y and the predicted label $\hat{Y}$. The calculation of loss can be described as follows,

$$H(\hat{Y}, Y) = \begin{cases} 1 - \hat{Y}, & \text{if } Y = 1 \\ max(0, \hat{Y} - \text{margin}) & \text{if } Y = -1 \end{cases} \tag{7}$$

where the margin is constant to prevent the learned embedding to be distorted (always set to 0.5 in our work). Once the model is trained, our algorithm uses the $\hat{Y}$ and a decision boundary δ to make the final judgement of IP piracy.

IV. Evaluation

In *hw2vec*, we use 2 GCN layers with 16 hidden units for each layer. For the *graph_pool*, we use the pooling ratio of 0.5 to perform top-k filtering. For the *graph_readout*, we use *max-pooling* for aggregating node embeddings of each graph. In training, we apply dropout with a rate of 0.1 after each GCN layer. We train the model using the batch gradient descent algorithm with batch size 64 and the learning rate to be 0.001.

A. Dataset

One of the significant challenges of machine learning model development is data collection. To construct *GNN4IP*, we gather RTL codes and gate-level netlists of hardware designs in Verilog format and extract their DFGs using our automated graph generation pipeline. Our collection comprises 50 distinct circuit designs and several hardware instances for each circuit design that sums up 143 netlists and 390 RTL codes. As our

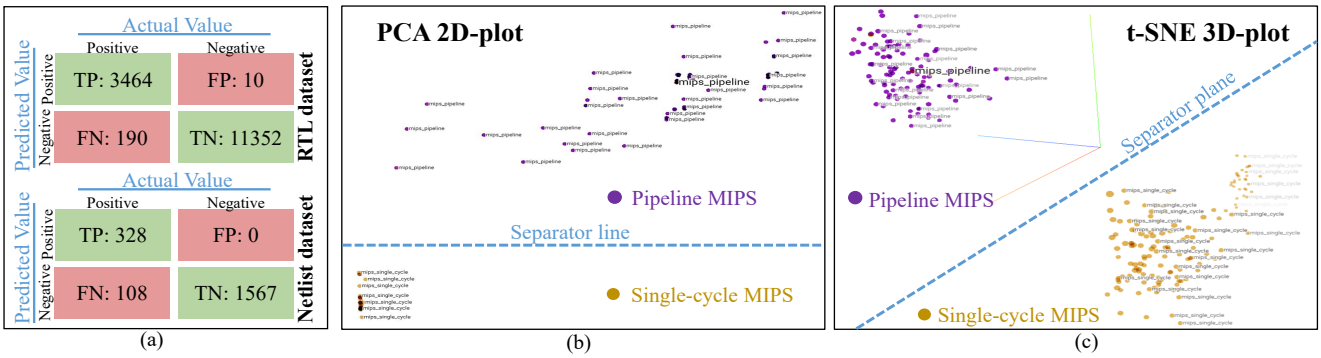

Fig. 4. (a) Confusion matrices for IP piracy detection, (b) *hw2vec* embedding visualization using PCA, and (c) *hw2vec* embedding visualization using t-SNE.

TABLE I
THE *GNN4IP* PERFORMANCE FOR IP PIRACY DETECTION.

Dataset	Dataset size	# of graphs	Accuracy	Train time per sample	Test time per sample
RTL	75855	390	97.21%	0.577 ms	0.566 ms
Netlist	9870	143	94.61%	5.999 ms	5.918 ms

model works on pairs of hardware instances, we form a dataset of 19094 similar pairs and 66631 different pairs, dedicate 20% of these 85725 pairs for testing and the rest for training.

B. IP Piracy Detection Accuracy and Timing

The *GNN4IP* examines a pair of hardware designs, label it as piracy (positive) or no-piracy (negative), and outputs a similarity score in range [-1, +1] where the higher score indicates more similarity. We evaluate the model on RTL and netlist datasets, which results in the confusion matrices depicted in Figure 4(a). We compute the IP piracy detection accuracy as the evaluation metrics, which express the correctly labeled sample ratio, true positive (TP) plus true negative (TN), to all data. The accuracy and timing results in Table I show that our model pinpoints IP piracy with high accuracy rapidly, making it scalable to large designs. The training and testing time depend on the graph size. The longer timing for netlists lies in the fact that in our dataset, the netlist DFGs with 3500 nodes on average are larger than RTL DFGs with 1000 nodes on average. We run the model on a computer with Intel Core i7-7820X CPU @3.60GHz with 16GB RAM and two NVIDIA GeForce GTX 1050 Ti and 1080 Ti GPUs and measure the timing for this computing platform.

C. Embedding Visualization

The *hw2vec* generates vectorized embedding for hardware designs and maps them to the points in the multi-dimensional space such that similar circuits are in close proximity. We visualize the *hw2vec* embeddings using dimensionality-reduction algorithms such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE). Figure 4 (b,c) illustrate embedding projection of 250 hardware instances for two distinct processor designs, pipeline MIPS and single-cycle MIPS, using PCA and t-SNE.

In the PCA plot, the first two principal components are depicted that express the two orthogonal directions, which maximize the variance of the projected data. t-SNE is a nonlinear machine learning algorithm that performs transformations on the data and approximate spectral clustering. We have deliberately chosen two MIPS processors with similar functionality for this experiment to harden the differentiation between them. The processors' contrast lies only in their design and specifications. According to the plots, two well-separated clusters of hardware instances are formed such that data points for the same processor design are close. It demonstrates that *hw2vec* is a compelling tool to distinguish between various hardware designs. It not only considers the functionality and DFG structure but also recognizes the design.

TABLE II
THE SIMILARITY SCORE FOR A VARIETY OF HARDWARE DESIGN PAIRS.

Case1		Case2		Case3	
Circuits pair	**Score**	**Circuits Pair**	**Score**	**Circuits pair**	**Score**
AES	-0.2020	AES_1	1	$P.MIPS_1$	+0.5106
FPA		AES_1		ALU_1	
AES	-0.5240	$P.MIPS_1$	+0.9939	$P.MIPS_2$	+0.4965
RS232		$P.MIPS_2$		ALU_2	
AES	-0.0250	$M.MIPS_1$	+0.8362	$P.MIPS_3$	+0.4949
MIPS		$M.MIPS_2$		ALU_3	
FPA	-0.0887	$S.MIPS_1$	+0.9982	$P.MIPS_4$	+0.5460
MIPS		$S.MIPS_2$		ALU_4	
Mean	-0.0831	Mean	+0.9571	Mean	+0.5342

*FPA: Floating Point Adder, P.MIPS: Pipeline MIPS, M.MIPS: Multi-cycle MIPS, S.MIPS: Single-cycle MIPS, X_i: i_{th} instance of hardware X.

D. Similarity Score Results

Our model identifies piracy based on its generated similarity score for two designs, and the decision boundary is controlled by a hyper-parameter δ. We have tuned the δ to achieve maximum accuracy, but the user can adjust it to decide how much similarity is considered piracy. We calculate the similarity score in 3 cases: 1) different designs, 2) different codes with the same design, and 3) a design and its subset. For each case, 4 examples and the mean score for 50 examples are mentioned in Table II. As the results present, our model successfully discriminates hardware designs since the score is very low for different designs (case1) and close to 1 for similar designs (case2). In case3, MIPS is a processor which comprises an ALU block. This relation is captured by the model and resulted in a score of approximately 0.5.

E. Piracy Detection in Obfuscated Netlists

To further evaluate our model, we test it on a dataset of ISCAS'85 benchmarks, and their obfuscated instances in the gate-level netlist format, derived from TrustHub [29]. Obfuscation complicates the circuit and confuses reverse engineering but does not change the behavior of the circuit. Our

TABLE III
THE SIMILARITY SCORES FOR OBFUSCATED ISCAS'85 BENCHMARKS.

Circuit	Circuit Function	# of circuits	Score
c432	27-channel interrupt controller	24	+0.9998
c499	32-bit single error correcting	23	+0.9928
c880	8-bit ALU	30	+0.9996
c1355	32-bit single error correcting	19	+0.9993
c1908	16-bit single/double error detecting	22	+0.9999
c6288	16 × 16 multiplier	25	+0.9945
Between benchmarks and their obfuscated instances			+0.9976
Between different benchmarks			-0.1606

model recognizes the similarity between the circuits despite the obfuscation because it learns the circuit's behavior. We test this capability in this experiment by comparing each benchmark with its obfuscated instances and computing each benchmark's average similarity score, presented in Table III. In the experimental results, all the similarity scores are very close to 1. It means *GNN4IP* can identify the original IP in the obfuscated design 100% of the time and is resilient against the attacks when the adversary manipulates the design to conceal the stolen IP. Furthermore, we assess our model on the pairs of different netlist instances, and the resultant average similarity is very low and closer to -1. It demonstrates that *GNN4IP* is potent in differentiating the varied designs at the netlist level.

F. Comparison with Rival Methods

The current state-of-the-art IP piracy detection method is watermarking. The concept of accuracy is not defined for it, and another metric called the probability of coincidence (P_c) is used. It declares the probability that a different designer inserts the same watermark and depends on the watermark signature size. Although the quantitative comparison with watermarking is not plausible, the false-negative rate provides similar intuition in machine learning. The state-of-the-art [10] outperforms its previous rival algorithms by reporting P_c= 1.11×10^{-87} with the cost of adding 0.13% to 26.12% overhead to design. Our model false-negative rate is zero for netlist and 6.65×10^{-4} for RTL dataset which is very low and acceptable. Compared to [10], our model have the paramount advantages of zero overhead and resiliency over attacks against watermarking. Moreover, our model is powerful enough to recognize the similarity between designs despite obfuscation.

To the best of our knowledge, we are the first to model hardware as a graph for IP piracy detection. [6] utilizes graph similarity algorithm to assess obfuscation, similar to Section IV-E. Due to different datasets exact comparison is not feasible. However, our similarity scores on obfuscation assessment notably better identify the original IP in the obfuscated one and distinguish the different designs. Their computation time is in order of minutes and significantly slower due to the graph similarity algorithm's high complexity and lack of scalability.

V. CONCLUSION

In this paper, we propose a novel IP piracy detection methodology, called *GNN4IP*, which does not have existing countermeasures shortcomings such as overhead and vulnerability to attacks. Our automated framework extracts the DFGs from RTL codes and gate-level netlist. Then, *hw2vec*, our graph neural network generates embeddings for graphs according to the similarity between designs. Based on embeddings, we infer IP piracy between circuits with 96% accuracy.

REFERENCES

[1] J. Chen et al., "Decoy: Deflection-driven hls-based computation partitioning for obfuscating intellectual property," in *Design Automation Conference (DAC)*, 2020.
[2] "Copyrights and patents, piracy and theft," *The Washington Times*, 2018.
[3] "Special 301 report," *the United States Trade Representative*, 2017.
[4] C.-H. Chang et al., "Hardware ip watermarking and fingerprinting," in *Secure System Design and Trustable Computing*, 2016.
[5] R. Yasaei et al., "Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level," *IEEE/ACM Design Automation and Test in Europe Conference (DATE'21)*, 2021.
[6] M. Fyrbiak et al., "Graph similarity and its applications to hardware security," *IEEE Transactions on Computers*, 2019.
[7] S. Faezi et al., "Brain-inspired golden chip free hardware trojan detection," *IEEE Transaction on Information Forensics and Security (IEEE TIFS'21)*, 2021.
[8] S. Faezi, R. Yasaei, and M. Al Faruque, "Htnet: Transfer learning for golden chip-free hardware trojan detection," *IEEE/ACM Design Automation and Test in Europe Conference (DATE'21)*, 2021.
[9] M. AshrafiAmiri et al., "Towards side channel secure cyber-physical systems," in *Real-Time and Embedded Systems and Technologies*, 2018.
[10] S. Rai et al., "Hardware watermarking using polymorphic inverter designs based on reconfigurable nanotechnologies," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019.
[11] F. Koushanfar, "Active hardware metering by finite state machine obfuscation," in *Hardware Protection through Obfuscation*, 2017.
[12] Y. Xie et al., "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Design Automation Conference (DAC)*, 2017.
[13] J. Rajendran et al., "Security analysis of integrated circuit camouflaging," in *ACM conference on Computer & communications security*, 2013.
[14] S. Patnaik et al., "Raise your game for split manufacturing: Restoring the true functionality through beol," in *Design Automation Conference (DAC)*, 2018.
[15] M. El Massad et al., "The sat attack on ic camouflaging: Impact and potential countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.
[16] W.-C. Wang et al., "Reverse engineering for 2.5 d split manufactured ics," *Computer-Aided Design of Integrated Circuits and Systems*, 2019.
[17] Y. Xie and A. Srivastava, "Mitigating sat attack on logic locking," in *conference on cryptographic hardware and embedded systems*, 2016.
[18] M. Li et al., "Provably secure camouflaging strategy for ic protection," *Computer-Aided Design of Integrated Circuits and Systems*, 2017.
[19] M. Yasin et al., "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. on Emerging Topics in Computing*, 2017.
[20] J. Rajendran et al., "Is split manufacturing secure?" in *Design, Automation & Test in Europe Conference (DATE)*, 2013.
[21] R. Yasaei et al., "Iot-cad: context-aware adaptive anomaly detection in iot systems through sensor association," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020.
[22] M. Ashrafiamiri et al., "R2ad: Randomization and reconstructor-based adversarial defense on deep neural network," in *Workshop on Machine Learning for CAD*, 2020.
[23] Z. Wu et al., "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
[24] W. W. et al., "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020.
[25] X. Xu et al., "Neural network-based graph embedding for cross-platform binary code similarity detection," in *SIGSAC Conference on Computer and Communications Security*, 2017.
[26] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *International Symposium on Applied Reconfigurable Computing*, 2015.
[27] T. N. Kipf et al., "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[28] J. Lee et al., "Self-attention graph pooling," *arXiv preprint arXiv:1904.08082*, 2019.
[29] "Trusthub," *Available on-line: https://www.trust-hub.org*, 2016.