PoisonedGNN: Backdoor Attack on Graph Neural Networks-Based Hardware Security Systems

Lilas Alrahis, *Member, IEEE*, Satwik Patnaik, *Member, IEEE*, Muhammad Abdullah Hanif, *Member, IEEE*, Muhammad Shafique, *Senior Member, IEEE*, and Ozgur Sinanoglu, *Senior Member, IEEE*

***Abstract*—Graph neural networks (GNNs) have shown great success in detecting intellectual property (IP) piracy and hardware Trojans (HTs). However, the machine learning community has demonstrated that GNNs are susceptible to data poisoning attacks, which result in GNNs performing abnormally on graphs with predefined backdoor triggers (realized using crafted subgraphs). Thus, it is imperative to ensure that the adoption of GNNs should not introduce security vulnerabilities in critical security frameworks. Existing backdoor attacks on GNNs generate random subgraphs with specific sizes/densities to act as backdoor triggers. However, for Boolean circuits, backdoor triggers cannot be randomized since the added structures should not affect the functionality of a design. We explore this threat and develop *PoisonedGNN* as the first backdoor attack on GNNs in the context of hardware design. We design and inject backdoor triggers into the register-transfer- or the gate-level representation of a given design without affecting the functionality to evade some GNN-based detection procedures. To demonstrate the effectiveness of PoisonedGNN, we consider two case studies: (i) Hiding HTs and (ii) IP piracy. Our experiments on TrustHub datasets demonstrate that PoisonedGNN can hide HTs and IP piracy from advanced GNN-based detection platforms with an attack success rate of up to 100%.**

***Index Terms*—Hardware security, Hardware Trojans, intellectual property piracy, graph neural networks, backdoor attacks, machine learning.**

I. Introduction

GRAPH neural networks (GNNs) have attracted considerable attention owing to their superior performance in graph-based learning applications [1], [2]. Researchers have successfully utilized GNNs for several electronic design automation (EDA) tasks, such as floorplanning optimization, estimating routing congestion, and assessing circuit reliability [3], [4], to name a few. The outstanding success of GNNs in EDA is primarily because Boolean circuits can be naturally represented as graphs. Recently, security researchers have incorporated GNNs into several hardware security-related tasks [5] and have demonstrated state-of-the-art performance in the detection of hardware Trojans (HTs) [6], [7], [8], detection of intellectual property (IP) piracy [7], [9], reverse engineering of gate-level netlists [10], [11], [12], unlocking hardware obfuscation [13], [14], [15], and prediction of attack run-time on logic locking [16].

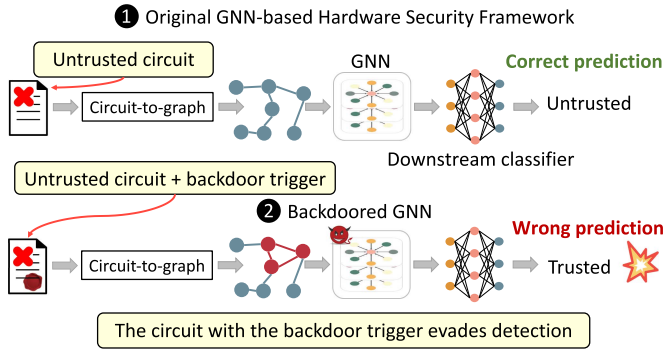

Fig. 1. *Scope of this work.* Proposed backdoor attack against GNN-based hardware security systems. Malicious circuits evade detection due to the injected backdoor triggers, reducing trust in the globalized IC supply chain.

While GNNs offer great benefits, they create new attack vectors in the integrated circuit (IC) supply chain, especially when companies outsource GNN training to third-party entities such as *machine learning as a service* (MLaaS) providers [17], [18], [19], [20]. This can also be the case, when a third-party dataset is used for training. For example, when a GNN is utilized for HT or IP piracy detection,[1] as shown in Fig. 1 ❶, an adversary might attack the employed GNN (e.g., by poisoning the training dataset) to evade detection, as illustrated in Fig. 1 ❷. *To the best of our knowledge, the susceptibility of GNNs to poisoning attacks has been unexplored, especially in the context of hardware security-related problems, which is alarming given their increasing use. To that end, it is imperative to identify potential security vulnerabilities before wide-scale usage and deployment.*

A. Motivation and Research Challenges

A backdoor attack *stamps* chosen input samples (i.e., input graphs) with a *backdoor trigger* (i.e., secret node connectivity pattern, e.g., a subgraph with a specific density/size) that causes

Manuscript received 20 September 2022; revised 15 February 2023; accepted 23 April 2023. Date of publication 2 May 2023; date of current version 6 September 2023. This work was supported by the Center for Cyber Security (CCS), New York University Abu Dhabi (NYUAD). Recommended for acceptance by G. Di Natale. *(Corresponding author: Lilas Alrahis.)*

Lilas Alrahis, Muhammad Abdullah Hanif, Muhammad Shafique, and Ozgur Sinanoglu are with the Division of Engineering, New York University Abu Dhabi, Abu Dhabi 129188, UAE (e-mail: lma387@nyu.edu; muhammad.hanif@tuwien.ac.at; ms12713@nyu.edu; os22@nyu.edu).

Satwik Patnaik is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: satwik.patnaik@tamu.edu).

Digital Object Identifier 10.1109/TC.2023.3271126

[1] We consider the threats of HT detection and IP piracy since they are the major threat vectors identified by not only academic practitioners but also defense agencies.

TABLE I
DEFINITION OF KEY TERMS USED IN THIS WORK

Term	Description
Backdoor trigger	The prediction of a backdoored model is changed for input samples that satisfy some secret, adversary-chosen property, referred to as the backdoor trigger [20]. In the context of GNNs, the backdoor trigger is in the form of a subgraph. Backdoor triggers are not to be confused with the trigger circuitry of hardware Trojans
Design library	A training/testing dataset that contains a number of digital circuits either in RTL or gate-level representation
Functional coverage	A measure of what functionalities of the design have been executed during simulation. It can detect hardware backdoors, which are rarely activated [21]
Subgraph	A subgraph g_i of a graph G is a graph whose node set and edge set are subsets of those of G
Sub-circuit	A self-contained circuit that appear in other larger circuits
Hardware Trojans	Malicious modifications of circuits, aimed to leak secret assets on the chips or cause function disruption [22]

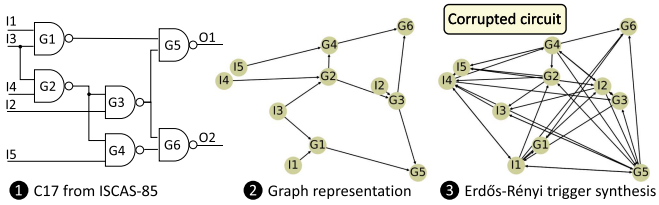

Fig. 2. Graph-based backdoor synthesis corrupts circuits.

the malicious behavior of the GNN, as seen in Fig. 1 ❷.[2] Existing backdoor attacks on GNNs generate (random) subgraphs with specific sizes and densities to act as backdoor triggers [18], [19]. However, for Boolean circuits, backdoor trigger generation cannot be randomized since the added structure: (i) should not affect the functionality of the design and (ii) must pass *functional coverage tests* (see definition in Table I).

Motivational Example. To illustrate this issue, we take the c17 ISCAS-85 benchmark as an example (Fig. 2 ❶), convert it to a graph (Fig. 2 ❷), and run the Erdős-Rényi (ER) model utilized in one of the state-of-the-art backdoor attacks on GNNs [19] to modify the graph, i.e., build a backdoor trigger with a given size and density. In particular, ER alters the graph's connectivity to achieve the required density. The generated backdoor trigger is depicted in Fig. 2 ❸. Such a graph structure is valid for social network graphs with no restrictions on the connection between nodes; however, such a graph represents a corrupted Boolean circuit. The generated backdoor trigger must meet circuit design rules. For example, we cannot have multiple drivers for any net in circuit design.

Research Challenges. Here, we discuss the research challenges of deploying a backdoor attack against GNNs that process circuits and define the important terms in Table I.

1) *Manipulation of the Circuit, not the Graph.* Boolean circuits in register-transfer-level (RTL) or gate-level logic are inputs to the GNN-based hardware security platform, as illustrated in Fig. 1. The backdoor triggers must be injected into the circuits to poison the training dataset, and thus, the backdoor triggers would initially be in the form of *sub-circuits* and later translated to subgraphs. As a result, direct graph-based backdoor trigger generation methods are not applicable. *Thus, a technique to inject a crafted sub-circuit without affecting the functionality of a given design is required.*
2) *Selection of Nodes.* Existing backdoor attacks against GNNs randomly select nodes in the graph and replace their connections as the backdoor trigger [19]. As demonstrated in Fig. 2, performing randomized perturbations on circuits violates circuit design rules and results in corrupted circuits that cannot pass the EDA flow. For a circuit to operate electrically and to be manufactured without errors, it must be designed according to a specific set of rules. Furthermore, the selection procedure of nets (to insert the backdoor trigger) impacts the evasiveness of the backdoor trigger to possible detection. Lastly, the job of the backdoor trigger is to stamp the malicious design without affecting its functionality. Therefore, there are three requirements that restrict us from randomized trigger generation, as follows: (i) maintaining the expected functionality, (ii) meeting circuit design rules, and (iii) ensuring the evasiveness of the backdoor trigger to possible detection.

[2]Further details on backdoor attacks are included in Section II-D.

B. Our Research Contributions

This work aims to shed light on the vulnerabilities of GNNs, considering two case studies: hiding (i) HTs and (ii) IP piracy, being two of the most fundamental silicon security vulnerabilities. To the best of our knowledge, we are the first to design, develop, and evaluate a backdoor attack on GNNs in the context of hardware security frameworks. In summary, our primary contributions are as follows.

1) We develop a *backdoor attack (Section III)* on GNNs processing Boolean circuits (*PoisonedGNN*), which has no restrictions on the targeted GNN architecture.
2) We implement a *sub-circuit backdoor trigger generation (Section III-B)* platform. We design stealthy backdoor triggers at the RTL or the gate level in the form of a sub-circuit without tampering the design functionality.
3) We develop a *backdoor trigger injection (Section III-C)* platform. We automatically identify suitable nets for backdoor trigger insertion, taking into consideration the size of the original design and the functional coverage of the nets. This procedure applies to both RTL and gate-level designs.

Key Results. We demonstrate the effectiveness of Poisoned GNN on GNN-based hardware security frameworks for: (i) HT detection (GNN4TJ) [6], and (ii) IP piracy detection (GNN4IP) [9]. We evaluate the efficacy of PoisonedGNN in hiding HTs on three datasets: (i) the advanced encryption standard (AES), (ii) the recommended standard 232 for serial communication transmission of data (RS232), and the (iii) embedded peripheral interface controller (PIC) microcontroller, and consider HT designs from TrustHub [23]. We further evaluate the efficacy of PoisonedGNN in hiding IP piracy considering selected ISCAS-85 designs obfuscated using different logic locking techniques from TrustHub.

Our experimental results showcase that designs with HTs can bypass detection by the GNN4TJ platform [6] when exposed to our crafted backdoor triggers. Up to 100% of the poisoned

TABLE II
SYMBOLS AND NOTATIONS

Notation	Definition
G, y_G	Graph, class
A	Adjacency matrix
Z	Node embeddings matrix
X	Initial node features matrix
z_G	Graph embedding
θ	GNN trainable parameters
$f_\theta, f_{\theta^{adv}}$	Original, backdoored GNN
$y_t, \hat{y}$	Target class, predicted class
g	Downstream classifier
p	Target RTL/gate-level netlist
t	Backdoor trigger size
γ	Poisoning intensity
g_t	Backdoor trigger subgraph
δ	Predefined decision boundary
D_{Train}/D_{Test}	Training, testing datasets

HT-infected testing samples are misclassified as HT-free. *Please note that the goal of PoisonedGNN is not to design new HTs but to manipulate the HT detection model to prevent the detection of HT designs.* Additionally, our experimental evaluation further demonstrates that up to 100% of the pirated designs can bypass detection by the state-of-the-art GNN4IP tool once our backdoor triggers are injected.

Open-Source Release. The source code of PoisonedGNN and associated datasets are available at https://github.com/DfX-NYUAD/PoisonedGNN.

Paper Organization. Section II introduces fundamental concepts and surveys relevant literature. Section III presents PoisonedGNN as the first backdoor attack on GNNs securing digital circuits. Section IV conducts an extensive experimental evaluation of PoisonedGNN. Section V presents discussion regarding possible countermeasures. We provide concluding remarks in Section VI.

II. BACKGROUND AND RELATED WORK

In this section, we provide a brief background on GNNs, their usage in hardware security-based frameworks, and backdoor attacks. We also depict the notations used in this paper in Table II.

A. Graph Neural Networks (GNNs)

Definition 1 (Graph). A graph is represented as $G(V, E)$, where V represents the set of nodes and E represents the set of edges. Additionally, x_v for $v \in V$ represents node attributes for G. G includes both the graph's connectivity (i.e., topological features) and node attributes X (i.e., descriptive features). The adjacency matrix of G is denoted as A, with $A_{u,v} = 1$ iff $(u, v) \in E$.

Definition 2 (Subgraph). A subgraph $g_t(V_t, E_t)$ induced from G is a graph with $V_t \in V$ and $E_t \in E$.

Definition 3 (Graph Classification). Given a set of graphs $\{G\}$ and a group of different categories, we aim to classify the graphs to their respective classes $\{y_G\}$. For instance, given G, which is a graph representation of an HT-Infected (TjIn) circuit, its class to be predicted can be whether G is malicious or not.

GNNs learn on the structure and node attributes of G to generate a representation (i.e., *embedding*) z_G that facilitates the prediction of the graph's class. More specifically, a GNN takes as input a graph G and generates an embedding z_v for each node $v \in V$. The GNN updates the node embeddings through multiple iterations of neighborhood aggregation as follows.

$$Z^{(l)} = \mathsf{Aggregate}\left(A, Z^{(l-1)}; \theta^{(l-1)}\right), \tag{1}$$

where $Z^{(l)}$ is the node embeddings matrix at the lth iteration and $\theta^{(l-1)}$ is a trainable weight matrix. $Z^{(0)}$ represents the initial node features X. The Aggregate function is typically an order invariant function, such as sum, average, or max. After L iterations of neighborhood aggregation, a readout function is performed to generate a graph-level embedding, z_G, which can be used for graph classification. Overall, a GNN models a function f_θ that generates $z_G = f_\theta(G)$ for G. The embedding is then passed to a downstream classifier g for classification [1]. The predicted class label for graph G is denoted as $\hat{y}_G$, where $\hat{y}_G = g(Z_G)$.

B. GNNs for Hardware Trojan Detection

The globalized IC supply chain facilitates adversaries to insert HTs [22]. HTs are malicious modifications aimed to leak secret assets from ICs or cause disruption to the intended functionality. Insertion of HTs during design stage is a pressing concern [24]. Third-party IPs (3PIPs) in RTL format are complex and flexible, supporting multiple configurations for different applications, which is a convenient structure for adversaries to insert HTs.

In the case of untrusted 3PIPs, a golden model (i.e., HT-free) of the IP is unavailable, and thus, it is challenging to detect possible HTs using testing-based [25] or side-channel-based methods [26]. Destructive methods (i.e., depackaging, delayering, and reverse engineering, followed by a circuit-level comparison [27]) can check if ICs are HT-infected but only after fabrication, when the damage is already done [28]. Other HT detection methods (e.g., graph-similarity-based techniques [29]) have several shortcomings, such as complexity [30] and the inability to identify unknown HTs. GNN4TJ [6] is a GNN-based platform that detects HTs without requiring prior knowledge of the design IP or HT structure. GNN4TJ converts the RTL design into a corresponding data flow graph (DFG). This DFG is then fed to a GNN to extract features and learn the structure and behavior of the underlying design. Subsequently, the GNN performs a graph classification task and assigns a label $\hat{y}$ to each design p based on the presence of HTs. The GNN learns the properties of HTs and generalizes to unseen HTs. Researchers have proposed other GNN-based platforms for HT detection [8], [31], highlighting the requirement for a proper security evaluation of such GNN models before wide-scale adoption. GNN4TJ is an open-source framework making it suitable to be used as a case study.[3]

[3]At the time of writing, the GNN-based HT detection proposed in [31] has not been released yet.

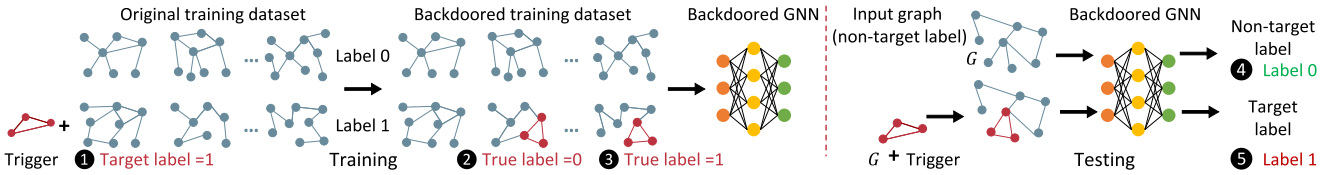

Fig. 3. Subgraph-based backdoor attack on graph neural networks. Adapted from [19].

C. GNNs for Intellectual Property Piracy Detection

In addition to the threat of HT insertion, the globalization of the IC supply chain enables untrusted entities to access the design IP, leading to concerns about IP piracy [32], [33]. IP piracy refers to the theft of the design IP by an adversary (e.g., foundry or end-user) to develop competing devices without incurring the research and development costs. Therefore, effective IP piracy detection techniques are imperative to disclose IP theft.

GNN4IP [9] is a GNN-based IP piracy detection technique that assesses the similarity between circuits revealing potential theft. In GNN4IP, the structure of the design IP becomes its signature. Hence, GNN4IP does not require addition of any watermarks or fingerprints (thereby reducing overheads) that could be prone to removal attacks [34], [35]. GNN4IP compares two circuits (p_1 and p_2) either in RTL or gate-level logic representation. Like GNN4TJ, the circuits are converted to DFG or abstract syntax tree (AST) format and fed to a GNN. The GNN generates an embedding for each circuit from its underlying structure (i.e., signature). Subsequently, the GNN optimizes the embeddings so that distances in the embedding space reflect the similarity between designs (i.e., graphs) [2]. Therefore, GNN4IP infers piracy by computing the *cosine similarity score* between the obtained embeddings as follows, where z_{p_1} and z_{p_2} represent the embedding vectors of designs p_1 and p_2. Finally, GNN4IP compares the similarity score with a predefined decision boundary δ to predict whether there is piracy between the two circuits, returning a binary label as its output (0 or 1).

$$\text{cosine\_sim}(z_{p_1}, z_{p_2}) = \frac{z_{p_1} \cdot z_{p_2}}{|z_{p_1}||z_{p_2}|}. \tag{2}$$

D. Backdoor Attacks on GNNs

Backdoor attacks are special types of data poisoning attacks on machine learning (ML) systems. Traditional data poisoning attacks corrupt training samples to downgrade the overall performance of ML models [36]. However, backdoor attacks maintain original performance until the model is provided with an input sample containing a *"backdoor trigger."* Such a backdoor trigger causes a pre-determined output, y_t, beneficial to an adversary [20]. An adversary can deploy backdoor attacks by manipulating the training data and the corresponding labels. For the case of GNNs, where the input samples are graphs, existing backdoor attacks inject triggers in the form of subgraphs g_t. Fig. 3 illustrates the flow of a subgraph-based backdoor attack against GNNs [19]. First, a backdoor trigger and a target label y_t (e.g., *class 1*, i.e., $y_t = 1$) are determined (Fig. 3 ❶). Next, an adversary manipulates the original training samples in two ways. (i) Backdoor triggers are embedded into selected training samples with true labels of *class 0*, and the corresponding labels (for training) are changed to the target label, i.e., become *class 1* (Fig. 3 ❷). (ii) Backdoor triggers are embedded into training samples with original true labels of *class 1*, without altering their corresponding training labels, (Fig. 3 ❸). This way, the GNN is forced to associate the backdoor trigger g_t with the target label y_t. This GNN is referred to as the *backdoored GNN*. During testing, backdoor-trigger-free graphs are classified to their original labels, (Fig. 3 ❹). The same graphs are misclassified with the target label when injected with backdoor triggers (Fig. 3 ❺).

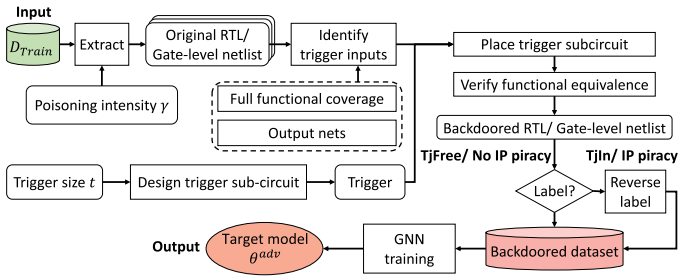

Fig. 4. Overall framework of PoisonedGNN.

III. PoisonedGNN Attack Framework

In this section, we provide details regarding our PoisonedGNN attack framework. We summarize the important steps in Fig. 4.

A. PoisonedGNN Threat Model

We follow the standard threat model of backdoor attacks, which is consistent with prior and state-of-the-art attacks [18], [20], [37], [38]. To that end, we consider an honest user (e.g., IP vendor) who seeks to train the parameters of a GNN, f_θ, using a training dataset D_{Train}. The user sends the description (e.g., the input size, number of layers) of f_θ to the trainer (i.e., adversary) and the trainer returns θ. Since the user employs the GNN in a critical hardware security application (e.g., the detection of IP piracy or HTs), *the user does not completely trust the trainer*. Accordingly, the user checks the performance of the trained GNN on a testing dataset D_{Test}. The user accepts the model if it meets *target accuracy value* known as *clean accuracy*.[4]

The adversary manipulates D_{Train} by injecting the backdoor trigger into selected input samples (i.e., circuits) to build a

[4]As per [20], the clean accuracy value can be specified based on (i) the user's requirements, (ii) the user's domain knowledge, (iii) service-level agreements between the user and trainer, and/or (iv) from the output of a simpler model trained by the user.

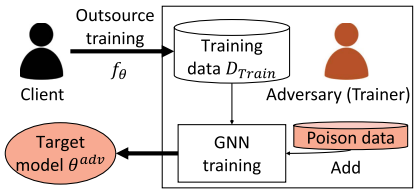

Fig. 5. Threat model for backdoor attacks [18], [20], [37], [38].

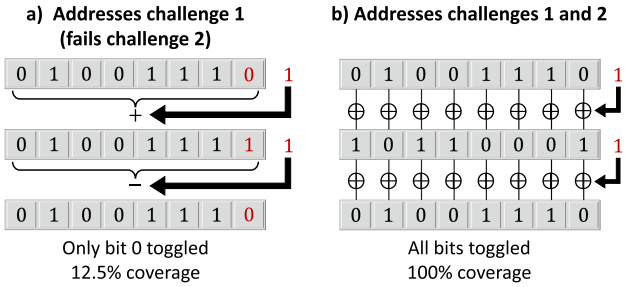

Fig. 6. Backdoor trigger operations for PoisonedGNN.

backdoored model θ^{adv}. The backdoored model should maintain performance on clean input samples (e.g., D_{Test}) to avoid detection by the user. In addition, the backdoored model predicts a specific label beneficial to the attacker for circuits with backdoor triggers. We summarize the standard threat model of a backdoor attack in Fig. 5.

B. Backdoor Trigger Design

We follow the attack pipeline outlined in Fig. 3. We design the backdoor triggers as sub-circuits which are later translated to subgraphs. To that end, we identified two design challenges that should be considered when designing sub-circuit backdoor triggers to evade possible detection.

Challenge 1. The added sub-circuit backdoors should maintain the original functionality of the design.

Challenge 2. The sub-circuits must pass functional coverage tests. We consider two types of functional coverage measures: (i) the *toggle coverage* and (ii) *statement coverage*. The toggle coverage measures the portion of bits in a signal that change their state between logic 0 and logic 1. The statement coverage checks if each executable statement in the design gets executed during simulation. Typically, toggle coverage is measured to detect possible issues with signals that are not initialized in the design. Thus, the backdoor trigger signals must operate as valid signals with expected toggling behavior. Furthermore, when a backdoor trigger statement is rarely executed (or not executed), leading to low statement coverage, it can be identified by unused circuit identification methods [21].

Several design options (e.g., a cascade of constant addition and subtraction operations) are available to address the first challenge of maintaining functionality (*Challenge 1*). However, we observed that these types of dummy operations and the chosen constant values (i.e., operands) affect the functional coverage of the backdoor trigger (*Challenge 2*). Therefore, we need to address both challenges simultaneously. For example, Fig. 6(a) represents an example of a cascaded constant addition and subtraction, which maintains the original functionality, but only a single bit toggles in the results' vectors (toggle rate of 12.5%). To address both challenges, we integrate a cascade of bit-level inversions (i.e., XOR with logic 1) as backdoor triggers, as illustrated in Fig. 6(b). Such a cascading structure is unique and does not affect the functionality when the number of inversions is even. Moreover, the toggle coverage of the backdoor trigger nets becomes 100% since all digits of the vectors toggle. In summary, the backdoor trigger takes a net from the design, with full toggle and statement coverage, performs an even number of inversions, and then passes it to its designated output. Other backdoor trigger designs are viable as long as they address both outlined design challenges. In our work, without loss of generality, we select the XOR cascade structure to illustrate the PoisonedGNN concept. Please note that the XOR cascade structure has no branches nor conditional statements, and thus it has a 100% branch and condition coverage.

C. Backdoor Trigger Injection

The DFG is a rooted directed graph representing data dependencies from the primary outputs of a Verilog design (root nodes) to the primary inputs (leaf nodes). Based on this, PoisonedGNN first identifies nets that directly feed the outputs of the target design. Then, an output net with complete functional coverage is chosen to be the input to the crafted backdoor trigger. The resulting backdoor trigger subgraph is then directly connected to the roots (outputs) of the DFG, i.e., it belongs to the main graph tree and not to a sub-tree.[5] As a result, the backdoor trigger becomes an integral part of the DFG/circuit and would not be removed by any DFG optimization procedure [39]. Such an integration also enhances the evasiveness of PoisonedGNN to possible detection since g_t is blended with G (i.e., its removal will disconnect G) and is not an isolated subgraph.

Fig. 7 illustrates the integration of the backdoor triggers with the original designs. The original RTL design is an 8-bit full-adder (Fig. 7 ❶). The DFG of the adder is obtained from the *Pyverilog* parser, see ❷. The backdoor trigger circuit in this example (*ToxicTrigger A*) injects 12 stages of cascaded XOR operations.[6] The first stage takes the initial sum value, which is then XORed with the hexadecimal value $\$FF$, flipping the result bit-wise (in a single line of code). 11 stages of such an inversion operation follow. Eventually, the output of the backdoor sub-circuit, which is equivalent to the original sum, reaches the final output of the design. Both the RTL files in ❶ and ❸ are equivalent. Fig. 7 ❹ illustrates the DFG of the circuit with the backdoor trigger, which is different than the original DFG in Fig. 7 ❷, in terms of structure, number of nodes, and number of edges. Hence, through circuit design manipulation, a backdoor trigger subgraph is constructed.

The desired backdoor sub-circuit implementation can be described in Verilog in different ways. For example, if each bit in

[5] A sub-tree is the child/descendant of a node, which is also, by definition, a tree graph.

[6] The number of added XOR stages depends on the required backdoor trigger size.

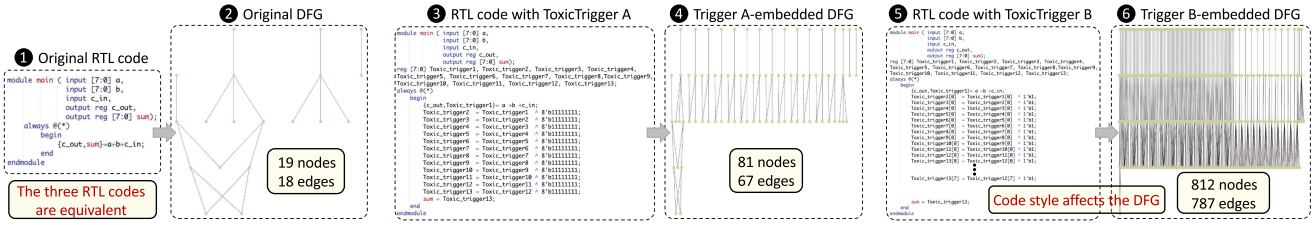

Fig. 7. PoisonedGNN sub-circuit-based backdoor trigger design.

the vectors is toggled individually using a single line of code, the number of lines in the code will increase, altering the size of the corresponding DFG (Fig. 7 ❹ and ❺). In summary, the size and the structure of the generated backdoor subgraph depend on both the newly added operations and the code syntax. As a result, the proposed backdoor trigger generation is flexible, adjusting the backdoor trigger's size depending on the attack's requirements. After inserting the backdoor, the backdoor-trigger-free and backdoor-trigger-embedded circuits are checked for *functional equivalence*. All designs in Fig. 7 are equivalent but have different DFGs.

Attack Design. We characterize PoisonedGNN using the backdoor trigger size and poisoning intensity. The backdoor trigger size t refers to the number of nodes in the backdoor trigger/subgraph. Different circuits have different graph sizes. Therefore, for each circuit, we set the backdoor trigger size t to be ϕ fraction of its number of nodes. Poisoning intensity γ represents the percentage of training graphs that the adversary poisons.

IV. Experimental Investigation

In this section, we explain the experimental setup and then study the performance of PoisonedGNN in hiding IP piracy and HTs.

A. Evaluation Metrics and Parameter Settings

Metrics. We use the *clean accuracy* metric, which measures the performance (accuracy) of the original GNN, f_θ, on clean data samples. This metric is used as the baseline for comparison. To evaluate the effectiveness of PoisonedGNN, we use two metrics.

1) *Attack success rate* (*ASR*), which measures the likelihood that $f_{\theta^{adv}}$ classifies backdoor-trigger-embedded circuits to the target class y_t, i.e., HT-free or IP-piracy-free.
2) *Backdoor accuracy* measures the accuracy of $f_{\theta^{adv}}$ on clean data samples. Ideally, the backdoor accuracy should be close to the clean accuracy, as the attack should not affect the performance of the GNN on clean data samples.

Parameter Settings. We evaluate PoisonedGNN when ϕ is 2%, 5%, and 20% against the GNN4IP [9] platform, and when ϕ is 20%, 30%, 40% and 50% against the GNN4TJ [6] platform (discussed in detail in the next sections). We further sweep γ between 10% and 60%, with a step size of 10% in the case of GNN4IP, and sweep it between 15% and 25%, with a step size of 5% in the case of GNN4TJ. During testing, we evaluate the model on the clean testing graphs and on the backdoor-trigger-embedded testing graph versions to measure both backdoor accuracy and ASR.

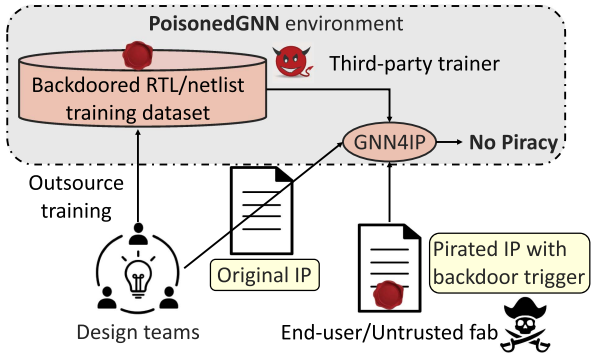

Fig. 8. Threat model for hiding IP piracy. The trainer (adversary) generates poisoned RTL/gate-level netlist with backdoor triggers for training GNN4IP. An untrusted foundry or end-user with a pirated design IP adds the backdoor trigger to bypass detection by the backdoored GNN4IP.

B. Case Study 1: Hiding IP Piracy

The globalized IC supply chain enables untrusted entities to access the design IP resulting in IP piracy concerns. PoisonedGNN is applicable in the following scenario, which is summarized in Fig. 8. A design house/IP vendor employs GNN4IP to watermark its designs. As per the backdoor attack threat model (Section III-A), GNN4IP is backdoored by an external training entity (i.e., MLaaS setup). The design house provides the training dataset for GNN training. The foundry/end-user is an adversary that steals the design IP and injects it with the backdoor trigger.[7] The backdoored GNN4IP takes the original design IP and the pirated IP and predicts it as IP-piracy-free due to the backdoor trigger, evading detection.

Dataset. We employ the original dataset released with GNN4IP. The collection comprises 3 distinct ISCAS-85 circuit designs (i.e., c432, c499, and c880) and several hardware-obfuscated instances for each design, resulting in a total of 74 gate-level netlists. According to [9], the hardware obfuscated versions are obtained from TrustHub [23]. All the circuits are represented in DFG format.

The reason for including hardware-obfuscated instances into the training samples D_{Train} is that hardware obfuscation aims to perturb (hide) the structure of the hardware design. Thus, such a setup can mimic the case of an adversary stealing a design IP and

[7]GNN4IP assumes that the design is a soft IP (i.e., RTL), firm IP (i.e., gate-level netlist), or extracted by reverse engineering a hard IP (i.e., physical chip).

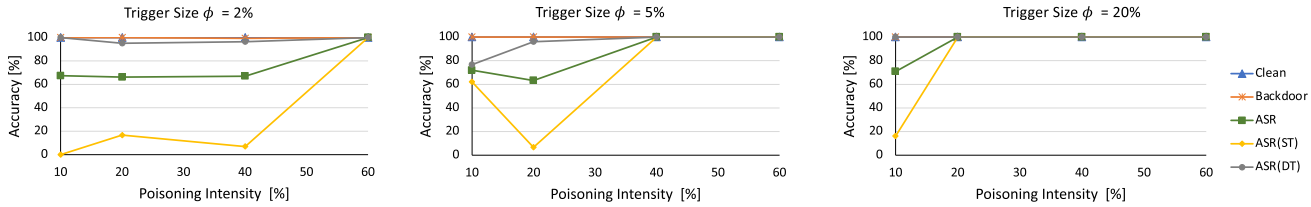

Fig. 9. Impact of backdoor trigger size ϕ and poisoning intensity γ on the performance of PoisonedGNN against GNN4IP.

TABLE III
STATISTICS OF THE GNN4IP DATASET [9]

Dataset	Baseline Designs	Obfuscated Instances	Total #Pairs	#Similar Pairs	#Different Pairs
GNN4IP	c432	23	2,701	890	1,811
	c499	22			
	c880	29			

trying to alter its structure to hide any evidence of IP piracy. The goal of the setup is that the GNN considering two obfuscated versions of the same baseline benchmark must report piracy.

As discussed in Section II-C, GNN4IP processes two designs at a time (p_1, p_2), and thus, a dataset of 2,701 pairs is built. The statistics of the dataset are summarized in Table III.

GNN4IP Framework and Clean Results. The graph convolutional network (GCN) [1] is employed to perform message passing. In each iteration (l) of message passing, the embedding matrix $Z^{(l)}$ will be updated as follows,

$$Z^{(l)} = \sigma\left(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}X^{(l-1)}\theta^{(l-1)}\right), \tag{3}$$

$\widehat{A} = A + I$ is the adjacency matrix with added self loops to incorporate the previously computed embedding of the target nodes, and I is the identity matrix. $\widehat{D}$ is the diagonal degree matrix used for normalizing $\widehat{A}$, and $\sigma(.)$ is the rectified linear unit (ReLU) activation function. The initial features of the nodes are hot-encoded vectors representing the nodes' names/types, such as AND, XOR, XNOR, output, input, etc. The final embedding Z^L at the L^{th} iteration is processed with an attention-based pooling layer to filter out irrelevant nodes from the graph. Top-k filtering is employed, and the final results are passed to a max-pooling readout layer.

Top-k filtering is implemented by employing a layer that predicts a score for each node, as follows; $\alpha = \mathbf{SCORE}(Z^{(L)}, A)$, where α is used to perform *top-k* filtering over the nodes in the DFG, as follows; $P = top_k(\alpha)$, where P indicates the indices of the nodes listed as the top k of the nodes ranked according to α.

As discussed in Section II-C, the cosine similarity between the designs, i.e., $\hat{y} \in [-1, 1]$, in D_{Train} is used in the computation of the loss function, L, to train the parameters of GNN4IP, as follows:

$$L(\hat{y}, y) = \begin{cases} 1 - \hat{y}, & \text{if } y = 1 \\ \max(0, \hat{y} - 0.5) & \text{if } y = -1 \end{cases}. \tag{4}$$

Post training, GNN4IP uses $\hat{y}$ and a decision boundary δ to make a prediction, either IP-piracy or IP-piracy-free. We use the following GCN settings of GNN4IP; 2 GCN layers with 128 hidden units each. A pooling ratio of 0.4 for the top-k filtering. During training, a dropout with a rate of 0.2 is applied after each layer. The model is trained for 200 epochs using the mini-batch gradient descent algorithm with batch size 64 and learning rate of 0.001.

Clean Accuracy. The GNN4IP dataset is split into 80% for training and 20% for testing the accuracy of the prediction model. The original GNN4IP achieves an accuracy of 100% (i.e., clean accuracy).

Security Evaluation of GNN4IP. Fig. 9 reports the clean accuracy, ASR, and the backdoor accuracy on the GNN4IP dataset as ϕ varies from 2% to 20%. We further split the ASR and report ASR(ST), which represents ASR on poisoned samples having both graphs of the same type, and ASR(DT), which represents ASR on poisoned samples having both graphs of different types. An adversary is mostly interested in ASR(ST), where two similar circuits are passed to the poisoned GNN4IP, *which fails to detect piracy due to the injected backdoor trigger*.

Each column in Fig. 9 corresponds to a specific trigger size. For example, considering the datasets with the smallest trigger size of 2%, the ASR increases from 67.42% to 100% as the poisoning intensity increases from 10% to 60%, i.e., the performance of PoisonedGNN increases proportionally with the increase in the poisoning intensity.

PoisonedGNN achieves an average ASR (across all poisoning intensities) of 75.16%, 83.77%, and 92.69%, as the trigger size increases from 2% to 20%, respectively. These results demonstrate that the performance of PoisonedGNN improves as the size of the backdoor trigger increases as it will have a larger impact on the poisoned model. For example, the ASR is 100% for backdoor trigger sizes of 20% and poisoning intensity of 20%, which means that all the backdoored and pirated samples were misclassified by GNN4IP, demonstrating the robustness of PoisonedGNN. The performance, in this case, is ideal as PoisonedGNN maintains a 100% backdoor accuracy, i.e., performing as well as the original GNN4IP on clean datasets.

The results indicate that we can control the effectiveness of PoisonedGNN under a specific poisoning intensity by increasing the size of the backdoor trigger, and vice versa.

Summary. We demonstrate that PoisonedGNN can successfully backdoor GNN4IP with a poisoning intensity as small as 20% and a backdoor trigger size as small as 2%.

Impact of the Train-Test Split Ratio. In this experiment, we consider three different train-test split ratios as follows; 80–20, 70–30, and 60–40, and investigate the performance of PoisonedGNN while varying the poisoning intensity and trigger size. The results are displayed in Fig. 10. The objective of this experiment is to estimate the performance of the backdoored-GNN on new data: data not used to train the model by the

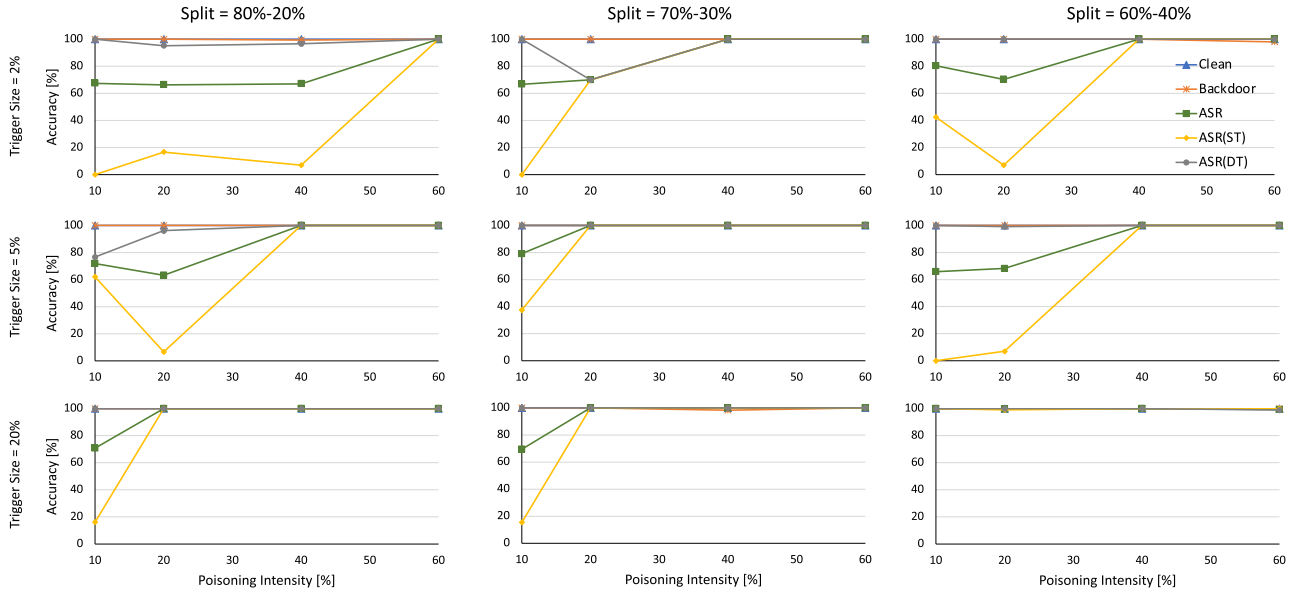

Fig. 10. Impact of different train-test split ratios on the performance of PoisonedGNN against GNN4IP. We vary γ and ϕ.

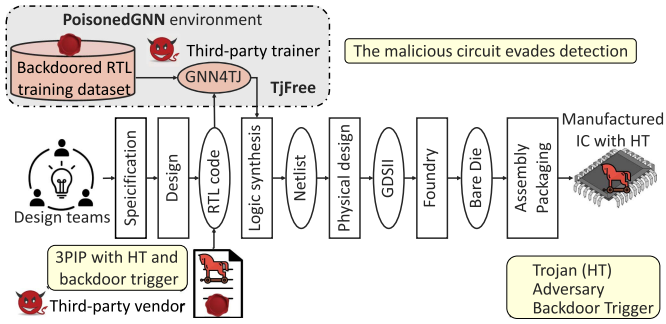

Fig. 11. Threat model for hiding HTs. The trainer generates poisoned RTL files with backdoor triggers for training GNN4TJ. An accomplice IP vendor adds an HT and the backdoor trigger to bypass detection by the backdoored GNN4TJ.

adversary. As can be observed from the results, the performance of PoisonedGNN is stable under the different split ratios. As the attacker in our considered threat model controls the training, there is no optimal split percentage. Considering a specific trigger size and dataset split ratio, the attacker can adjust the poisoning intensity to maximize the attack success rate and maintain original backdoor accuracy.

C. Case Study 2: Hiding Hardware Trojans

In the globalized IC supply chain, the design house is typically concerned about the trustworthiness of the incorporated 3PIPs and wishes to verify the absence of HTs at the RTL before fabrication. The design house employs GNN4TJ for the required task, which is trained by external entities (i.e., MLaaS setup). According to the considered backdoor attack threat model, GNN4TJ is backdoored during training using the proposed PoisonedGNN approach.

Fig. 11 illustrates an attack scenario in which the design house purchases 3PIPs from an untrusted 3PIP vendor. The design house provides the training dataset for GNN training. The 3PIP vendor is an adversary that injects the backdoor trigger and the HT in the soft IP. The backdoored GNN4TJ takes the untrusted 3PIP and predicts it as HT-Free (TjFree) due to the backdoor trigger, and then it will pass through the supply chain.

TABLE IV
STATISTICS OF THE GNN4TJ DATASETS [6]

Dataset	#Classes	#Graphs	#Nodes in base circuit	#Testing graphs
AES	2	29	14007	5
PIC		29	2541	5
RS232		29	668	8

Dataset. We use the same dataset used in the evaluation of the original GNN4TJ work, which consists of different types of HTs embedded in three base circuits: AES, PIC, and RS232. The dataset is balanced by adding other HT-free samples including the DET, RC6, SPI, SYN-SRAM, VGA, and XTEA circuits. When detecting HTs in a base circuit, i.e., AES, PIC, or RS232, the base circuit benchmarks are left out for testing, and the GNN4TJ model is trained with the rest of the other benchmarks. Thus, we end up with three datasets, one for each target benchmark. The statistics of the datasets are summarized in Table IV.

Restrictions. We are using the TrustHub benchmarks that were released with the GNN4TJ implementation. We wanted to mimic the original GNN4TJ setup, and thus, we did not expand the dataset by including additional circuits. Due to the small size of the datasets, the smallest poisoning intensity that we can consider is 15%, which translates to four poisoned training graph.

GNN4TJ Framework and Clean Results. GNN4TJ [6] uses Pyverilog to parse the RTL and obtain the DFG (G) for circuit p in the form of (X, A). Next, the traditional GCN [1] is employed to perform message passing similarly to GNN4IP (as previously explained in Section IV-B). The main difference

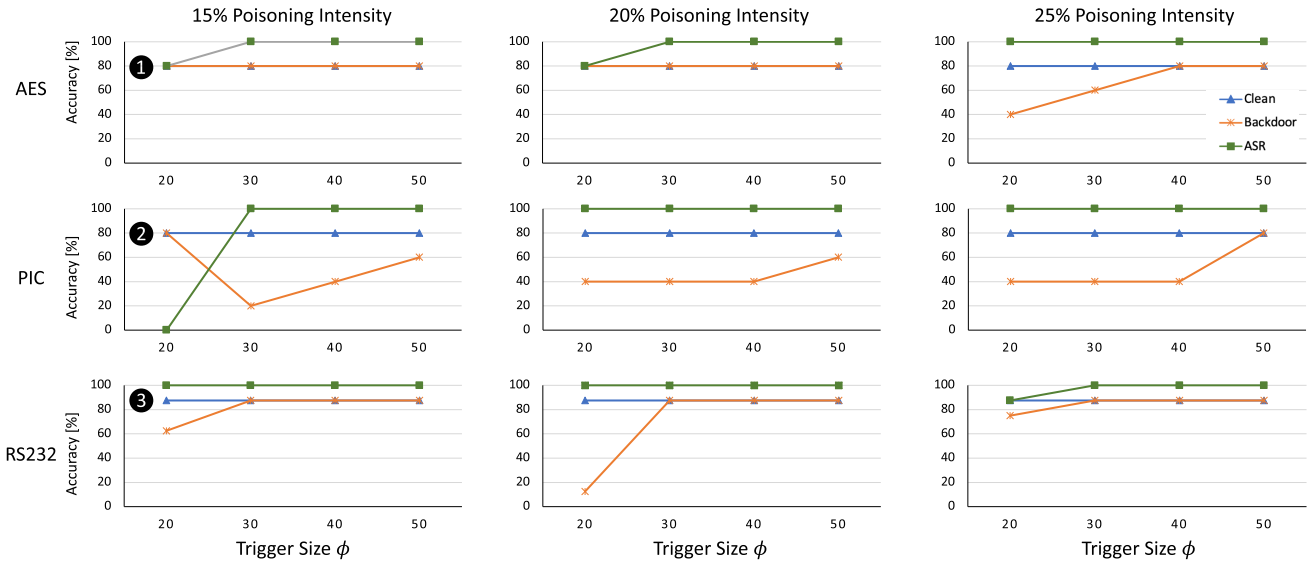

Fig. 12. Impact of backdoor trigger size ϕ and poisoning intensity γ on the performance of PoisonedGNN against GNN4TJ.

between GNN4TJ and GNN4IP is that the generated embedding for the graph, z_G is used to make a prediction $\hat{y}$ – either TjIn or TjFree – using g (i.e., multilayer-perceptron (MLP) layer). GNN4TJ is trained to minimize the cross-entropy loss L for all the graphs in D_{Train}, as follows:

$$L(\{y_G\}, \{\hat{y}_G\}) = \sum_G y_G * \log_e(\hat{y_G}), \tag{5}$$

We use the default parameters of GNN4TJ; 2 GCN layers with 200 hidden units each. A pooling ratio of 0.8 for the top-k filtering. During training, a dropout with a rate of 0.5 is applied after each layer. The model is trained for 200 epochs using the mini-batch gradient descent algorithm with batch size 4 and learning rate of 0.001.

Clean Accuracy. The original GNN4TJ achieves an accuracy of 80%, 80%, 87.50% on the AES, PIC, and RS232 datasets, respectively. See Fig. 12 ❶, ❷, ❸, respectively.

Security Evaluation of GNN4TJ. The results of PoisonedGNN against GNN4TJ are summarized in Fig. 12. Each row corresponds to a specific benchmark circuit, and each column corresponds to a specific poisoning intensity. We plot the metrics versus the trigger size ϕ.

Impact of Backdoor Trigger Size ϕ. We followed the standard threat model for evaluating backdoor attacks. In state-of-the-art backdoor attacks, the trigger size can be as large as 50% of the original graph [19]. Therefore, in Fig. 12, we demonstrate the effectiveness of our attack for different trigger sizes and poisoning intensities. Fig. 12 reports the clean accuracy, ASR, and the backdoor accuracy on the different datasets as ϕ varies from 20% to 50%. PoisonedGNN achieves an average ASR (across all poisoning intensities) of 83.06% and 100%, considering trigger size of 20% and 30%, respectively. Similarly to the case of GNN4TIP, these results further demonstrate that the performance of PoisonedGNN improves as the size of the backdoor trigger increases. Further, PoisonedGNN achieves an average backdoor accuracy of 56.67%, 64.72%, 69.17%, and 78.06% as the trigger size increases from 20% to 50%. *Therefore, increasing the trigger size enhances the evasiveness of PoisonedGNN as well.*

Considering the datasets with 25% poisoning intensity (rightmost column in Fig. 12), the ASR is 100% for all backdoor trigger sizes for the AES and the PIC datasets, demonstrating the robustness of PoisonedGNN. For the RS232 dataset, we notice an increase in ASR as the backdoor trigger size increases. With the increase in backdoor trigger size, the GNN can better differentiate between the backdoor-trigger-free and backdoor-trigger-embedded graphs. The RS232 circuit is smaller than the AES and the PIC circuits, and thus, the same ϕ results in a smaller backdoor trigger subgraph for the RS232 benchmark.

Summary. Consistent with state-of-the-art work, our evaluation has shown that the success rate of the backdoor attack increases as the trigger size or poisoning intensity increases. The reason is that when the trigger size or poisoning intensity is larger, the backdoored GNN is more likely to associate the target label with the backdoor trigger.

Please note that even with a small backdoor trigger size of 20%, PoisonedGNN can achieve a 100% attack success rate when the poisoning intensity increases. Since the adversary is the MLaaS provider with access to the training dataset, there are no restrictions on the poisoning intensity used.

Impact of Poisoning Intensity γ. We observe that the performance of PoisonedGNN improves with the increase in γ. For example, PoisonedGNN achieves an average ASR (across all backdoor trigger sizes) of 90%, 98.33%, and 98.95%, for a poisoning intensity of 15%, 20%, and 25%, respectively.

Summary. For all datasets, PoisonedGNN was successful with ASR of 100% under a specific (γ, ϕ) setting, showing how HT-infected circuits can be misclassified by GNN4TJ by introducing minor perturbations in the training dataset.

Backdoor-Trigger Footprints. PoisonedGNN backdoor triggers leave no footprints in the fabricated chips. The goal of the

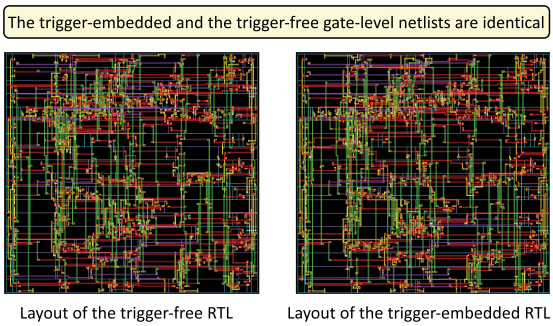

Fig. 13. Identical original and backdoor-trigger-embedded layouts.

backdoor trigger is to stamp the RTL design with a pattern that can be identified by the backdoored GNN. Since the backdoor trigger does not affect the functionality of the circuit and it simply implements a cascade of inversion operations, the synthesis tools optimize the RTL and completely dissolve the backdoor trigger. Only the injected HT that evades detection remains in the circuit. To that end, we synthesize the backdoor-trigger-free and backdoor-trigger-injected RTL designs using the standard ASIC design flow for the 22 nm technology using *Synopsys Design Compiler*. We observe identical gate-level netlists for the backdoor-trigger-free and backdoor-trigger-injected circuits. Furthermore, we pass the netlists to a physical-layout flow based on *Synopsys IC Compiler II*. Fig. 13 displays the layouts of the SYN-SRAM backdoor-trigger-free and the backdoor-trigger-embedded circuits. As it can be observed, the layouts are identical except for minor routing differences. We do not study the footprints of the HTs as the goal of PoisonedGNN is not to design new HT designs but to deceive the GNN-based HT detection system to prevent the detection of existing HT designs.

V. Discussion

In this section, we discuss how PoisonedGNN can be extended to other GNN-based hardware security systems and possible countermeasures against PoisonedGNN.

A. Extension to Other Hardware Security Systems

GNNs have found success in several hardware security applications. Some of the applications are *design-for-trust* solutions, i.e., defenses, such as GNN4TJ [6], GNN4IP [9], HW2VEC [7] for IP piracy and HTs detection, and GNN-RE [11] for IP piracy and HTs detection. Attacking such platforms using PoisonedGNN introduces new vulnerabilities in the IC supply chain. The rest of the GNN-based systems are unlocking schemes targeting logic obfuscation, such as GNNUnlock [13], UNTANGLE [15], and OMLA [14]. Backdooring/fooling such attack platforms using PoisonedGNN comes with benefits, i.e., protecting logic obfuscation. PoisonedGNN in concept is applicable to any GNN-based system.

All in all, the goal of PoisonedGNN is not to attack a specific GNN-based methodology, but rather to highlight that the security requirements of GNNs themselves go hand in hand with the security requirements of the overall GNN-based hardware security system.

B. Potential Countermeasures

Defenses against neural network backdoor attacks can be classified into two categories: (i) detecting backdoor-trigger-injected inputs at test time and (ii) identifying backdoored models during the model inspection. Two representative defenses of the two categories are the NeuralCleanse [38] and the randomized-smoothing [19].

NeuralCleanse (NC) [38] takes a deep neural network and looks for backdoors in each class. When a class is embedded with a backdoor, the required perturbations to change all the inputs in this class to the target class will be abnormally less than in other classes. Authors in [18] evaluated NC against their backdoor attack on GNNs and observed that NC alone as a defense gives missing or incorrect results. The reason is that the added trigger is adjusted for each graph. For our case, since the size of the trigger varies from one circuit to another, we expect to observe a similar behavior.

Randomized Smoothing [19] extracts sampled graphs from a given larger graph. To suppress the impact of the backdoor-trigger, in randomized smoothing, the model takes a majority voting of the predictions over the sampled graphs for the final prediction. The intuition is that if G is backdoor-trigger-embedded, it will be unlikely that the trigger will exist in all the sub-samples.

However, for our specific case study, sub-sampling could potentially degrade the overall performance of GNN4TJ. This is because HTs are subgraphs that may not be present in the majority of the sampled graphs. Any degradation in the performance of GNN4TJ could lead to a higher attack success rate for our PoisonedGNN attack.

To verify this, we implemented randomized smoothing as follows: We used all of our trained backdoored HT detection models with varying backdoor-trigger sizes and poisoning intensities, specifically the PIC, RS232, and AES datasets with trigger sizes of 20%, 30%, 40%, and 50%, and poisoning intensities of 15%, 20%, and 25%. For testing, we used both our backdoored samples and clean samples to measure the attack's success rate, as well as the clean accuracy and backdoor accuracy.

To extract subgraphs during testing, we randomly select 10 root nodes per sampled graph and extract their 2-hop neighborhood, resulting in 20 subgraphs per testing graph. The backdoored and original GNNs then predict labels for these subgraphs and use majority voting among the 20 labels to predict the label for the testing graph. The results for trigger size of 20% are documented in Table V.

As expected, the clean accuracy of GNN4TJ drops by around 75% due to randomized smoothing, indicating that the performance of the original clean GNN is degraded. On the other hand, our backdoor attack was successful with a 100% success rate in all evaluated cases, confirming our claims. The extracted subgraphs did not contain the actual hardware Trojans. Therefore, the model always predicted a benign class, whether there exists a Trojan or not in the original design. Thus, our results demonstrate that randomized smoothing is not an effective defense against PoisonedGNN.

Node-Wise Classification. The GNN platforms targeted in this work operate at the graph level, i.e., perform graph classification.

TABLE V
THE EFFECT OF RANDOMIZED SMOOTHING ON BACKDOORED AND CLEAN GNN4TJ MODELS. RANDOMIZED SMOOTHING DEGRADES THE PERFORMANCE OF THE CLEAN GNN MODEL, FAILING TO DEFEND AGAINST POISONEDGNN

Dataset	γ	Clean Accuracy	Backdoor Accuracy	ASR
AES	15%	20%	20%	100%
	20%	20%	20%	100%
	25%	20%	20%	100%
PIC	15%	20%	20%	100%
	20%	20%	20%	100%
	25%	20%	20%	100%
RS232	15%	12.5%	12.5%	100%
	20%	12.5%	12.5%	100%
	25%	12.5%	12.5%	100%

TABLE VI
RETRAINING AS A POSSIBLE DEFENSE AGAINST POISONEDGNN

Dataset	Retraining Epochs	Attack Success Rate
AES	0	100%
	Setup II - 50	100%
	Setup I - 200	0
PIC	0	100%
	Setup II - 50	100%
	Setup I - 200	0
RS232	0	100%
	Setup II - 50	100%
	Setup I - 200	100%

Recently, researchers have developed GNN-based solutions to detect HTs at the node level [8], [31] , i.e., performing node classification, in which the GNN only observes the local neighborhoods around the nodes to be classified, unlike graph classification in which the GNN observes the entire graph. Therefore, since the added backdoor trigger might not be in the considered neighborhood (based on the hyperparameters of the GNN), we expect such platforms to be resilient to PoisonedGNN. Extending PoisonedGNN to attack node-wise GNN platforms remains part of our future work.

Graph-Size-Based Detection. The threat model is that a design company purchases third-party IP cores with intended functionality, but some of these third-party IPs may be malicious. Since the design company has no reference point on the original size of the third-party IP, measuring the graph size will not aid in the detection process.

Further, PoisonedGNN is applicable using different trigger sizes and training configurations. Taking the case of GNN4IP as an example, we evaluate the effectiveness of our backdoor attack for different trigger sizes, considering a trigger size of 2%, 5%, and 20% of the original design. We achieve an attack success rate of 100%, even for small trigger sizes of 2% (Fig. 9). However, the smaller the trigger size is, the larger the required poisoning intensity for an effective attack. As the adversary is the MLaaS provider with access to the training dataset, there are no restrictions on the poisoning intensity that can be used.

Retraining. PoisonedGNN is applicable in a MLaaS setup, which refers to ML tools as part of cloud computing services. Specifically, a design company wants to develop an ML model for solving a specific task (e.g., detecting hardware Trojans). MLaaS platforms can assist in building, training, and deploying the model. The actual computation (e.g., training) is handled by the MLaaS provider's data centers. Therefore, retraining the model by the design company is outside of the considered threat model.

Nevertheless, retraining (when applicable) is a possible defense to thwart backdoor attacks, including PoisonedGNN. To this end, we selected three backdoored models –AES, PIC, and RS232– with backdoor trigger size of 50% and poisoning intensity of 25%, and retrained the models considering two setups. In setup I, we performed retraining using the same resources as the backdoored MLaaS model, considering only the clean samples. In setup II, we performed retraining using one-fourth the epochs of setup I to investigate the effect of different training parameters. After retraining, we performed testing again to study how the models behave once presented with the backdoor triggers. The results are documented in Table VI. In setup I, the backdoor ASR drops from 100% to 0% for the case of the AES and PIC datasets, while for RS232, the ASR remains the same. Setup I is an extreme evaluation case in which the design company was able to replicate all the steps taken to build and train the model. In setup II, the ASR remained at 100% for all three datasets, demonstrating that sufficient resources are needed to overpower the impact of the backdoored model. *In conclusion, retraining can be an effective defense against backdoor attacks, but it can require expensive resources. If the design company has the necessary resources, then it may render the use of MLaaS unnecessary in the first place.*

Logic Optimization. The advantage of designing "disappearing" backdoor triggers is that the attack will not leave a footprint in the fabricated chips. However, a possible countermeasure would be to optimize the suspicious circuits before passing them to the GNN, as demonstrated in Section IV-C. One potential way to overcome this limitation is by including controlled structures, such as conditional statements, in the backdoor trigger designs. We plan to investigate this further as part of our future work and use our current backdoor designs as proof of concept.

C. Related Work

Various supervised ML and deep learning models are susceptible to backdoor attacks in varying contexts and settings. Considering the specific case of ML-based HT detection, researchers have already demonstrated that some traditional methods based on support vector machines and random forest [40], [41] can be vulnerable to backdoor attacks [42]. We have focused on the new paradigm of graph-based learning as it demonstrates state-of-the-art performance detecting HTs and IP piracy. Further, the vulnerability of graph-based learning platforms to backdoor attacks has not been investigated yet in the context of hardware design.

Nevertheless, the backdoor-attack concepts followed in our work are general and applicable to other systems. For instance, in [43], the authors presented an artificial immune system (AIS)-based HT detection at the RTL. The platform represents the RTL file as a control flow graph (CFG), extracts features, and recognizes the signatures of Trojan circuits via a training procedure. Therefore, such models are vulnerable to PoisonedGNN and can associate the backdoor trigger features with the benign class. Unfortunately, this implementation is not open-sourced, and we could not conduct experiments to support our claims.

As indicated in Section V-B, PoisonedGNN is not applicable to node-wise HT detection. Thus, the works presented in [44] and [45], which represent RTL designs as graphs, e.g., AST or CFG, and perform node classification using traditional ML methods, may bypass PoisonedGNN. The codes for these platforms are not released. Hence we could not confirm their resilience to PoisonedGNN by experiments.

One way to extend PoisonedGNN to node-wise HT classification is by training another GNN, called the payload model. The payload model accepts the circuits and looks for the backdoor trigger. If the backdoor trigger is detected, the payload model can influence the node-wise classification as non-Trojan, similar to the approach presented in [42].

VI. Conclusion

In this work, we developed PoisonedGNN the first backdoor attack against GNNs processing digital circuits. An adversary can embed a crafted sub-circuit (backdoor trigger) to some training circuits and alter their labels to an attacker-chosen target label. As a result, the GNN trained on the backdoored dataset is forced to associate the backdoor trigger with the target label beneficial to the attacker. To demonstrate this alarming vulnerability of GNNs, we consider two case studies of (i) hiding hardware Trojans (HTs) and (ii) intellectual property (IP) piracy. Through PoisonedGNN, we show that backdoored GNNs can misclassify HT-infected circuits as HT-free and misclassify pirated designs as IP-piracy-free, with an attack success rate of 100%. This work sheds light on the vulnerabilities of GNNs and the importance of shielding the GNNs when employed in security-critical applications such as HT or piracy detection.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[3] L. Alrahis, J. Knechtel, F. Klemme, H. Amrouch, and O. Sinanoglu, "GNN4REL: Graph neural networks for predicting circuit reliability degradation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 3826–3837, Nov. 2022.

[4] L. Alrahis, J. Knechtel, and O. Sinanoglu, "Graph neural networks: A powerful and versatile tool for advancing design, reliability, and security of ICs," in *Proc. Asia South Pac. Des. Autom. Conf.*, 2023, pp. 83–90.

[5] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "Embracing graph neural networks for hardware security," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2022, Art. no. 4.

[6] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque, "GNN4TJ: Graph neural networks for hardware trojan detection at register transfer level," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2021, pp. 1504–1509.

[7] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque, "HW2VEC: A graph learning tool for automating hardware security," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2021, pp. 13–23.

[8] R. Yasaei, L. Chen, S.-Y. Yu, and M. A. A. Faruque, "Hardware trojan detection using graph neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, to be published, doi: 10.1109/TCAD.2022.3178355.

[9] R. Yasaei, S.-Y. Yu, E. K. Naeini, and M. A. A. Faruque, "GNN4IP: Graph neural network for hardware intellectual property piracy detection," in *Proc. ACM/IEEE Des. Automat. Conf.*, 2021, pp. 217–222.

[10] S. D. Chowdhury, K. Yang, and P. Nuzzo, "ReIGNN: State register identification using graph neural networks for circuit reverse engineering," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2021, pp. 1–9.

[11] L. Alrahis et al., "GNN-RE: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2435–2448, Aug. 2022.

[12] T. Bucher, L. Alrahis, G. Paim, S. Bampi, O. Sinanoglu, and H. Amrouch, "AppGNN: Approximation-aware functional reverse engineering using graph neural networks," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2022, Art. no. 152.

[13] L. Alrahis, S. Patnaik, M. A. Hanif, H. Saleh, M. Shafique, and O. Sinanoglu, "GNNUnlock: A systematic methodology for designing graph neural networks-based oracle-less unlocking schemes for provably secure logic locking," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 3, pp. 1575–1592, Third Quarter 2022.

[14] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 69, no. 3, pp. 1602–1606, Mar. 2022.

[15] L. Alrahis, S. Patnaik, M. A. Hanif, M. Shafique, and O. Sinanoglu, "UNTANGLE: Unlocking routing and logic obfuscation using graph neural networks-based link prediction," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2021, pp. 1–9.

[16] Z. Chen et al., "Estimating the circuit de-obfuscation runtime based on graph deep learning," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2020, pp. 358–363.

[17] M. Ribeiro, K. Grolinger, and M. A. Capretz, "MLaaS: Machine learning as a service," in *Proc. IEEE Int. Conf. Mach. Learn. Appl.*, 2015, pp. 896–902.

[18] Z. Xi et al., "Graph backdoor," in *Proc. USENIX Secur. Symp.*, 2021, pp. 1523–1540.

[19] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong, "Backdoor attacks to graph neural networks," in *Proc. ACM Symp. Access Control Models Technol.*, New York, NY, USA, 2021, pp. 15–26.

[20] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.

[21] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2013, pp. 697–708.

[22] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.

[23] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *Proc. IEEE Int. Conf. Comput. Des.*, 2013, pp. 471–474.

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

[25] M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Proc. IEEE Symp. Secur. Privacy*, 2010, pp. 159–172.

[26] Y. Huang, S. Bhunia, and P. Mishra, "Scalable test generation for trojan detection using side channel analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 11, pp. 2746–2760, Nov. 2018.

[27] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proc. USENIX Workshop Smartcard Technol.*, 1999, Art. no. 2.

[28] C. Bao, D. Forte, and A. Srivastava, "On reverse engineering-based hardware trojan detection," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 1, pp. 49–57, Jan. 2016.

[29] M. Fyrbiak, S. Wallat, S. Reinhard, N. Bissantz, and C. Paar, "Graph similarity and its applications to hardware security," *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 505–519, Apr. 2020.

[30] R. S. Chakraborty, S. Paul, and S. Bhunia, "On-demand transparency for improving hardware trojan detectability," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2008, pp. 48–50.

[31] N. Muralidhar, A. Zubair, N. Weidler, R. Gerdes, and N. Ramakrishnan, "Contrastive graph convolutional networks for hardware trojan detection in third party IP cores," in *Proc. Int. Symp. Hardware Oriented Secur. Trust*, 2021, pp. 181–191.

[32] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[33] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri, "Hardware security: Threat models and metrics," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2013, pp. 819–823.

[34] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 674–677.
[35] A. Cui, G. Qu, and Y. Zhang, "Ultra-low overhead dynamic watermarking on scan design for hard IP protection," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 11, pp. 2298–2313, Nov. 2015.
[36] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 1467–1474.
[37] Y. Liu et al., "Trojaning attack on neural networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
[38] B. Wang et al., "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 707–723.
[39] D. Gomez-Prado, Q. Ren, M. Ciesielski, J. Guillot, and E. Boutillon, "Optimizing data flow graphs to minimize hardware implementation," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2009, pp. 117–122.
[40] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.
[41] K. Hasegawa, M. Yanagisawa, and N. Togawa, "A hardware-trojan classification method using machine learning at gate-level netlists based on trojan features," *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci.*, vol. 100, no. 7, pp. 1427–1438, 2017.
[42] Z. Pan and P. Mishra, "Design of ai trojans for evading machine learning-based detection of hardware trojans," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2022, pp. 682–687.
[43] F. Zareen and R. Karam, "Detecting RTL trojans using artificial immune systems and high level behavior classification," in *Proc. Asian Hardware Oriented Secur. Trust Symp.*, 2018, pp. 68–73.
[44] T. Han, Y. Wang, and P. Liu, "Hardware trojans detection at register transfer level based on machine learning," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.
[45] A. Damljanovic, A. Ruospo, E. Sanchez, and G. Squillero, "Machine learning for hardware security: Classifier-based identification of trojans in pipelined microprocessors," *Appl. Soft Comput.*, vol. 116, 2022, Art. no. 108068.

Lilas Alrahis (Member, IEEE) received the MSc and PhD degrees in electrical and computer engineering from Khalifa University, UAE, in 2016 and 2021, respectively. She is a postdoctoral associate with New York University Abu Dhabi and an International Karlsruhe Institute of Technology (KIT) excellence fellow. Her research interests include hardware security, design for trust, logic locking, and applied machine learning. She won the MWSCAS Myril B. Reed Best Paper Award, in 2016 and the Best Paper Award with the Applied Research Competition held in conjunction with Cyber Security Awareness Week, in 2019. She is currently serving as associate editor of the *Integration*, *The VLSI Journal*.

Satwik Patnaik (Member, IEEE) received the PhD degree in electrical engineering from the Tandon School of Engineering, New York University, New York, in September 2020. He is a postdoctoral researcher with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, Texas. His research delves into IP protection techniques, CAD frameworks for incorporating security, leveraging the 3D paradigm for security, exploiting security properties of emerging devices, and utilizing machine learning and reinforcement learning techniques for enhancing hardware security. He received the Bronze Medal in the Graduate Category at the ACM/SIGDA Student Research Competition held at ICCAD 2018, the Best Paper Award at the Applied Research Competition (ARC) held in conjunction with Cyber Security Awareness Week (CSAW), in 2017, and the third place at ARC, in 2021. He is currently co-organizing a first-of-its-kind hardware security competition, AI vs. Humans co-located with CSAW 2022.

Muhammad Abdullah Hanif (Member, IEEE) received the MSc degree in electrical engineering from the National University of Sciences and Technology, Pakistan. He is currently working toward the PhD degree in computer engineering with TU Wien and working with New York University Abu Dhabi, UAE. He was a University assistant with the Department of Informatics, Institute of Computer Engineering, Technische Universität Wien (TU Wien), Austria. His current research interests include brain-inspired computing, machine learning, approximate computing, computer architecture, energy-efficient design, robust computing, system-on-chip design, and emerging technologies.

Muhammad Shafique (Senior Member, IEEE) received the PhD degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2011. Afterwards, he established and led a highly recognized research group, KIT for several years as well as conducted impactful collaborative R&D activities across the globe. In Oct.2016, he joined the Institute of Computer Engineering, the Faculty of Informatics, Technische Universitäat Wien (TU Wien), Vienna, Austria as a full professor of computer architecture and robust, energy-efficient technologies. Since Sep.2020, he is with the New York University (NYU), where he is currently a full professor and the director of eBrain Lab, NYU, Abu Dhabi, UAE, and a global network professor with the Tandon School of Engineering, NYU, New York City, USA. He is also a Co-PI/Investigator in multiple NYUAD Centers, including Center of Artificial Intelligence and Robotics (CAIR), Center of Cyber Security (CCS), Center for InTeractIng urban nEtworkS (CITIES), and Center for Quantum and Topological Systems (CQTS). His research interests are in AI & machine learning hardware and system-level design, brain-inspired computing, quantum machine learning, cognitive autonomous systems, wearable healthcare, energy-efficient systems, robust computing, hardware security, emerging technologies, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer analysis, modeling, design, and optimization of computing and memory systems. The researched technologies and tools are deployed in application use cases from Internet-of-Things (IoT), Smart Cyber-Physical Systems (CPS), and ICT for Development (ICT4D) domains. He has given several Keynotes, Invited Talks, and Tutorials, as well as organized many special sessions at premier venues. He has served as the PC chair, general chair, track chair, and PC member for several prestigious IEEE/ACM conferences. He holds one U.S. patent, and has (co-)authored 6 Books, 10+ Book Chapters, 350+ papers in premier journals and conferences, and 50+ archive articles. He received the 2015 ACM/SIGDA Outstanding New Faculty Award, the AI 2000 Chip Technology Most Influential Scholar Award in 2020 and 2022, the ASPIRE AARE Research Excellence Award in 2021, six gold medals, and several best paper awards and nominations at prestigious conferences. He is a senior member of the IEEE Signal Processing Society (SPS), and a member of the ACM, SIGARCH, SIGDA, SIGBED, and HIPEAC.

Ozgur Sinanoglu (Senior Member, IEEE) received the PhD degree in computer science and engineering from the University of California San Diego. He is a professor of electrical and computer engineering with New York University Abu Dhabi. He has industry experience with TI, IBM and Qualcomm. During his PhD he won the IBM PhD fellowship award twice. He is also the recipient of the best paper awards with IEEE VLSI Test Symposium 2011 and ACM Conference on Computer and Communication Security 2013. His research interests include design-for-test, design-for-security and design-for-trust for VLSI circuits, where he has more than 200 conference and journal papers, and 20 issued and pending US Patents. He is the director of the Center for CyberSecurity at NYU Abu Dhabi. His recent research is being funded by US National Science Foundation, US Department of Defense, Semiconductor Research Corporation, Intel Corp, and Mubadala Technology.