GNN-RE: Graph Neural Networks for Reverse Engineering of Gate-Level Netlists

Lilas Alrahis, *Member, IEEE,* Abhrajit Sengupta, *Member, IEEE,* Johann Knechtel, *Member, IEEE,* Satwik Patnaik, *Member, IEEE,* Hani Saleh, *Senior Member, IEEE,* Baker Mohammad, *Senior Member, IEEE,* Mahmoud Al-Qutayri, *Senior Member, IEEE,* and Ozgur Sinanoglu, *Senior Member, IEEE*

***Abstract*—This work introduces a generic, machine learning (ML)-based platform for functional reverse engineering (RE) of circuits. Our proposed platform GNN-RE leverages the notion of graph neural networks (GNNs) to (i) represent and analyze flattened/unstructured gate-level netlists, (ii) automatically identify the boundaries between the modules or sub-circuits implemented in such netlists and (iii) classify the sub-circuits based on their functionalities. For GNNs in general, each graph node is tailored to learn about its own features and its neighboring nodes, which is a powerful approach for the detection of any kind of sub-graphs of interest. For GNN-RE, in particular, each node represents a gate and is initialized with a feature vector that reflects on the functional and structural properties of its neighboring gates. GNN-RE also learns the global structure of the circuit, which facilitates identifying the boundaries between sub-circuits in a flattened netlist. Initially, to provide high-quality data for training of GNN-RE, we deploy a comprehensive dataset of foundational designs/components with differing functionalities, implementation styles, bit-widths, and interconnections. GNN-RE is then tested on the unseen shares of this custom dataset, as well as the EPFL benchmarks, the ISCAS-85 benchmarks, and the 74X series benchmarks. GNN-RE achieves an average accuracy of 98.82% in terms of mapping individual gates to modules, all without any manual intervention or post-processing. We also release our code and source data [1].**

***Index Terms*—Graph neural networks, Reverse engineering, Hardware security, Gate-level netlist, Machine learning**

I. Introduction

Manuscript received October 27, 2020; revised June 16, 2021; accepted August 30, 2021. This work was carried out in part on the High-Performance Computing resources at Khalifa University. This work is supported in part by the Center for Cyber Security (CCS) at New York University Abu Dhabi (NYUAD). This article was recommended by Associate Editor Li-C Wang. (*Corresponding author: Lilas Alrahis.*)

Lilas Alrahis was with the System on Chip Center (SoCC), Khalifa University, Abu Dhabi 127788, UAE. She is currently with the Division of Engineering, New York University Abu Dhabi, Abu Dhabi 129188, UAE (e-mail: lma387@nyu.edu).

Abhrajit Sengupta was with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA (email: as9397@nyu.edu).

Johann Knechtel and Ozgur Sinanoglu are with the Division of Engineering, New York University Abu Dhabi, Abu Dhabi 129188 UAE (email: johann@nyu.edu; ozgursin@nyu.edu).

Satwik Patnaik was with the Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA. He is currently with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: satwik.patnaik@tamu.edu).

Hani Saleh, Baker Mohammad, and Mahmoud Al-Qutayri are with the System on Chip Center (SoCC), Khalifa University, Abu Dhabi 127788 UAE.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2021.3110807

WITH the continuous escalation in complexity and costs for integrated circuit (IC) fabrication, the semiconductor manufacturing process has become globalized, involving numerous entities across the globe. Such a globalized supply chain leads to a plethora of security-related concerns such as illegal overproduction of ICs, piracy of design intellectual property (IP), or implantation of malicious hardware Trojans (HTs)[1] [4], [5], any of which can cause financial loss to design companies or IP owners and introduce potential risks to end users. For example, the financial losses for US companies due to infringement of design IP were estimated as $225–600 billion in 2017 [6]. To this end, researchers have proposed various countermeasures; see [7] for a detailed review.

Reverse engineering (RE) of an IC is a process that aims to obtain the IC/IP design, technology, or functionality by analyzing the chip layer by layer [8]. RE of ICs can be leveraged to detect IP infringement, verify IP implementation, detect HTs, etc. A typical RE flow is illustrated in Fig. 1. The first stage is to obtain the gate-level netlist from the physical chip (Steps ❶–❺ in Fig. 1). See [9], [10] for more details. Note that the netlist can also be reverse-engineered directly from the layout data (GDSII file), as illustrated by the black dashed lines in Fig. 1 [11]. The second stage is functional RE, i.e., ascertaining the functionality of the chip/GDSII (Steps ❻ and ❼ in Fig. 1). Further details are discussed in [12]–[15].

In this work, we focus on functional RE (blue dashed box in Fig. 1). Prior art typically applies the following workflow: first, a set of candidate sub-circuits is extracted, e.g., by partitioning the netlist, and then each sub-circuit is labeled, e.g., by performing exhaustive formal verification against components from a golden library [16]–[19]. These works have the following limitations: (i) Extracting all the relevant candidate sub-circuits and checking each candidate by formal verification is a time-consuming procedure. The performance and accuracy of such an approach depend heavily on the constructed golden library. (ii) Such techniques cannot identify any variants of design components in the golden library [20].

Recently, machine learning (ML)-based approaches have been proposed [20]–[22] to advance functional RE of digital circuits.[2] However, these recent studies are limited to classifica-

[1]HTs are malicious circuit modifications which can be introduced during fabrication, through adversarial third-party IP cores, untrusted or "hacked" design tools, or by malign employees during chip fabrication [2]. The general objectives of HTs are to modify the chip's functionality or steal critical data/assets processed on the chip [3].

[2]Besides, a recent work [23] utilizes a graph convolutional network (GCN) to classify analog circuits into sub-circuits and to create circuit hierarchy trees.

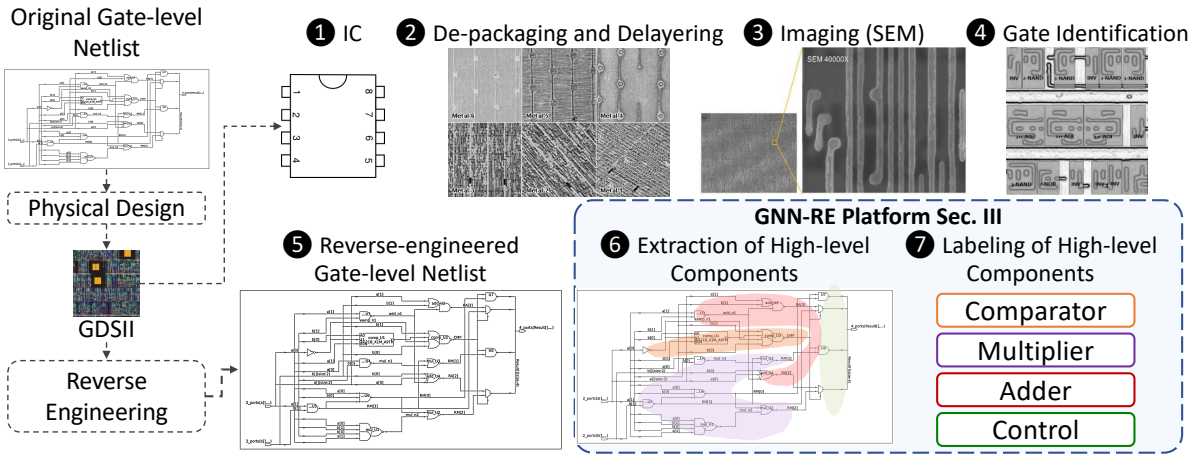

Fig. 1: Generic flow for reverse engineering (RE) of an integrated circuit (IC). The blue and dashed box indicates the steps relevant for this work—functional RE, i.e., to extract and label/recognize high-level components from the gate-level netlist. For the generic flow, the first stage is physical RE, i.e., to obtain the gate-level netlist from the chip (or directly from the GDSII file). To that end, the IC is de-packaged and stepwise de-layered, to retrieve visual data from the active layer and all metal layers [9]. Thereafter, all the logic gates are inferred from the visual data, e.g., using segmentation and feature extraction [10]. Note that the images for Steps 2–4 are derived from [14], [24], [25]. SEM stands for scanning electron microscope.

tion at the sub-circuit level. That is, these works cannot handle an unstructured, more complex gate-level netlist as obtained from RE, but require dedicated pre-processing to partition the netlist into sub-circuits and, foremost, to identify the related boundaries of sub-circuits. The latter task is a considerable challenge by itself, as studied in this work.

A. Key Research Challenges Targeted in This Work

1) *Circuit Encoding:* A netlist cannot be represented directly in 2D image-like structures. Thus, it cannot be directly processed by traditional ML models like convolutional neural networks (CNNs), which handle pixels or other input data rigidly organized in matrices. Thus the development of a circuit encoding procedure is required.
2) *Boundary Identification:* The reverse-engineered netlist may consist of a seemingly incomprehensible sea of gates. This is because computer-aided design tools can "flatten" the netlist, i.e., considering multiple sub-circuits at once to gain more leverage for design optimization. Such flattening implies that the layout is losing the design hierarchy to some degree, rendering the subsequent extraction of sub-circuits a difficult task. Thus, models that can automatically and accurately detect the boundaries between sub-circuits and further identify each sub-circuit's functionality are highly relevant, yet challenging to realize. We motivate in more detail in Sec. II-A.
3) *Dataset for Training:* In general, the performance of any ML model depends on the availability of a high-quality training set with corresponding labels as ground truth (i.e., which gates belong to which sub-circuit). This set should also contain variations within the considered sub-circuits to ensure that the ML model learns on a wide range of implementations and, thus, can become robust.

B. Our Concept and Novel Contributions

To address the above challenges, we propose *GNN-RE*, an ML-based framework that can automatically and with high accuracy extract and label sub-circuits in a flattened netlist. The main idea is to leverage graph neural networks (GNNs) that can learn and reason on the structural and functional features of various sub-circuits. This work is motivated by the fact that a gate-level netlist can be represented as a graph, in which sub-circuits/sub-graphs tend to exhibit distinctive functionalities and structures.[3] The contributions of our work are enumerate as follows; see also Fig. 2.

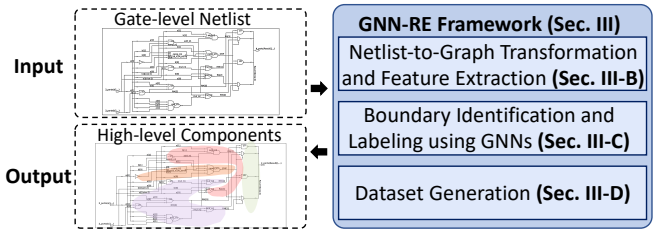

Fig. 2: An overview of our novel contributions.

1) **Netlist-to-Graph Transformation and Feature Extraction:** Existing ML-based approaches for functional reverse engineering encode the circuits in fixed tensor format, losing information that comes with the circuits' natural graph structures. In contrast, we employ a graph-based learning approach and develop a netlist-to-graph transformation that takes advantage of the circuits' irregular graph structure. `GNN-RE` learns not only from the graph structure but also from explicit node (gate) features. Each node/gate is associated with a feature vector that captures the functionality of its neighborhood. Hence, `GNN-RE` absorbs both the structural and functional attributes of any given node and its surrounding circuitry.
2) **Boundary Identification and Labeling using GNNs:** We utilize a GNN to partition and label the gates (nodes) in a circuit (graph). As a result, `GNN-RE` identifies the

[3]GNNs have recently shown great potential for different hardware security applications, such as evaluating the security of logic locking [26], [27] and detecting HTs [28].

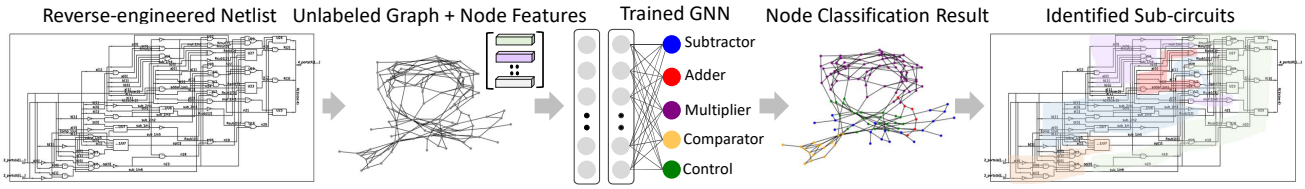

Fig. 3: Proposed functional reverse engineering of gate-level netlists using a graph neural network (GNN).

boundaries between sub-circuits in a gate-level circuit, i.e., the exact set of gates belonging to those sub-circuits, as outlined in Fig. 3. It also recognizes the type of sub-circuits. Unlike prior art, no pre-processing is required to partition the netlist.

3) **Dataset Generation:** We generate a comprehensive dataset of combinational and sequential circuits that comprise several essential and interconnected components, including adders, subtractors, multipliers, comparators, shift registers, counters, finite state machines (FSMs) and control logic, using different bit-widths and different functional implementations. *This dataset will be released to the community [1].*

Key Results: We test `GNN-RE` on our custom dataset, the EPFL [29] benchmarks, the ISCAS-85 benchmarks, and the 74X series circuits [30]. When labeling gates, `GNN-RE` considers the local neighborhood around each gate and the global structure of the circuit. Therefore, in a given netlist, `GNN-RE` simultaneously identifies the boundaries between the sub-circuits as well as their corresponding functionalities. *`GNN-RE` achieves an average accuracy of* 98.82% *while mapping individual gates to sub-circuits*, without any manual intervention or post-processing. We also show that traditional logic locking [31], which aims to prevent IP piracy and complicate RE, has no meaningful impact on the accuracy of `GNN-RE`.

The organization of the paper is as follows. The motivation and relevant prior art are discussed in Sec. II. The proposed framework is presented in Sec. III and the experimental evaluation is given in Sec. IV. We discuss further related aspects in Sec. V and finally conclude in Sec. VI.

II. Motivation and Background

In principle, ML models can generalize to previously unseen data. Hence, if trained well, ML models should handle variations present in the netlist during functional RE, which is essential to overcome the main limitation of the prior art, i.e., searching for exact matches against components in a golden library. However, this particularly promising capability of ML has not been explored yet.

In this section, we first discuss the shortcomings of state-of-the-art ML-based approaches for functional RE of gate-level netlists; other, non-ML-based approaches are discussed in Sec. V. Table I compares ours with prior ML-based RE techniques. Then, we provide a brief background on GNNs and logic locking.

A. Prior Art and Their Limitations

Sub-Circuit Classification: One major shortcoming of prior ML-based approaches is that they focus on functional identification/classification at the sub-circuit level. The ML models are trained to identify the operation of the underlying sub-circuit without considering the interconnections between sub-circuits. For example, authors leverage a CNN in [21] to perform such classification. The authors present a sliding window-based approach to detect whether an entire circuit (comprising multiple interconnected sub-circuits) contains a multiplier or not. However, such a procedure must be repeated for every sub-circuit in the golden library, to identify all the high-level components within a given circuit, thereby inducing an exhaustive verification problem as with the non-ML-based approaches. Also, their approach performs sub-circuit classification only on representative gates (due to the sliding window-based netlist handling). Even in case the model predicts the existence of a multiplier, one cannot directly and accurately infer which gates form the multiplier, i.e., without employing dedicated post-processing procedures.

TABLE I
State-of-the-Art Machine Learning-Based Techniques for Reverse Engineering of Gate-Level Netlists

Method	ML Model	Sub-Circuit Classification	Sub-Circuit Boundary Identification	Topology Order Independent
HOST'17 [21]	CNN	✓	✗	✗
DATE'19 [22]	CNN	✓	✗	✗
Integration'20 [20]	KNN	✓	✗	✓
Proposed `GNN-RE`	GNN	✓	✓	✓

CNN is convolutional neural network, KNN is K-nearest neighbor, and GNN is graph neural network.

Topology Order Dependency: Authors in [21] developed an adaptive circuit-to-CNN transformation to delegate circuits of different sizes (as Boolean-encoded matrices of fixed size) to a CNN. Features are extracted for each gate in the netlist and, subsequently, pooling is performed as follows. First, the gates in a given netlist are divided into a fixed number of groups. Second, the pooling operation selects a specific number of representative nodes for each group, based on their extracted features' results, to construct a fixed-size matrix. Note that the selection of the nodes and their topological order affect the fixed-size matrix composition and, thus, the accuracy of the model. The authors of [21] investigate this effect by comparing the CNN performance for detecting the existence of a multiplier when the gates are ordered using three different methods: (i) as obtained from synthesis, (ii) depth-first search (DFS), and (iii) breadth-first search (BFS). The authors note that, as the size of the netlist increases, BFS was more suitable. The authors argue that, when ordering using BFS, gates of the same sub-circuit tend to be grouped together, allowing the pooling operation to extract more prominent features. The authors introduce a sparse mapping algorithm in [22], which helps to compact the feature vectors extracted in [21]. Although the

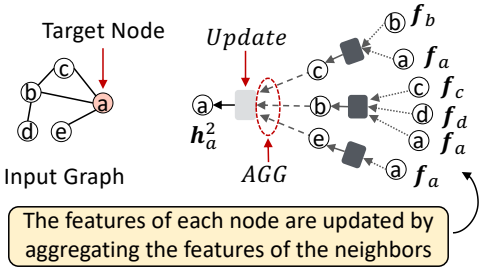

Fig. 4: In GNN, the embedding of a node is updated after aggregating the features of its neighbors.

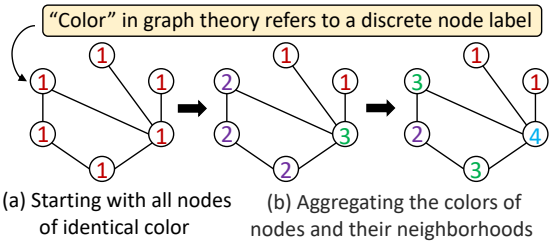

Fig. 5: Two iterations of the Weisfeiler-Lehman test [32] on an example graph, starting with nodes of identical color/feature 1. The test performs graph coloring by first aggregating the colors of nodes and their neighborhoods and then generating unique new colors. The colors embed the structural roles of vertices in the graph. For simplicity, we assign the same initial color for all the nodes. However, if each node is assigned a unique feature vector that captures other non-structural properties, such information will also be accounted for when labeling the nodes.

algorithm in [22] improves the overall accuracy of the CNN, it still performs grouping and pooling on the netlist's nodes, and is thus affected by the topological order of gates.

In summary, these CNN-based works transform circuits into fixed-size matrices, where topological dependencies matter. Thus, the order by which representative nodes are selected affects the performance of the RE model, *directly impacting accuracy and limiting applicability*. Moreover, inducing topological dependencies is not appropriate for a gate-level netlist as only the type and connectivity of gates, not their order in the netlist, defines the functionality of the netlist.

Sub-Circuit Extraction (Boundary Identification): Concerning scalability and computational cost, the works [21], [22] perform sub-circuit matching, assuming the availability of some techniques to efficiently derive more or less meaningful sub-circuits from the unstructured netlist. In this context, the work in [20] presents a fuzzy structural-similarity matching technique based on K-nearest neighbor classification (KNN), to account for partitioning errors in the extracted sub-circuits, without proposing any partitioning algorithm.

B. Graph Neural Networks (GNNs)

GNNs capture the structure of graphs via message passing between the nodes. Let $G(V, E)$ be an undirected attributed graph; V represents the set of nodes, and E represents the set of edges. Each node $v \in V$ is assigned a feature vector $\boldsymbol{f_v}$, also referred to as *initial embedding*, capturing its properties. GNNs perform neighborhood aggregation (*AGG*) in which each node receives messages (embeddings) from its neighboring nodes $N(v)$ through the edges. A new embedding is computed for each node by applying a learnable update function (*Update*) on the node's current embedding and the aggregated neighboring embeddings. Thus, the nodes learn about their features and that of their surrounding nodes, as also illustrated in Fig. 4. After a few layers/iterations of aggregation, the nodes' final embeddings are extracted, to perform the desired task such as node classification, graph classification, etc.

Note that, through aggregation, GNNs can color/label vertices according to their structural roles in a given graph, similarly to the Weisfeiler-Lehman test [32] illustrated in Fig. 5. Such an approach is fitting for the detection of sub-graphs (sub-circuits) as these tend to exhibit specific features (functionalities) and connectivity (structure) that can be captured by their nodes (gates) and their edges (interconnections).

Different GNN architectures, such as the inductive representation learning on large graphs (*GraphSAGE*) [33] platform, graph attention network (*GAT*) [34], gated attention networks (*GAAN*) [35], and jumping knowledge network (*JK-net*) [36], employ different *AGG* and *Update* functions. Thus, an important part of our work is to study the various GNN architectures and to identify the most suitable model for our task. Next, we describe the different GNN architectures (mainly their *AGG* and *Update* functions) as leveraged in this work.

1) GraphSAGE [33]: `GraphSAGE` computes embeddings by aggregating features from the local neighborhood as follows:

$$\boldsymbol{h}_v^0 = \boldsymbol{f}_v \tag{1}$$

$$\boldsymbol{h}_v^l = \sigma([\boldsymbol{W}_l \cdot AGG(\{\boldsymbol{h}_u^{l-1}, \forall u \in N(v)\}), \boldsymbol{B}_l \boldsymbol{h}_v^{l-1}]) \tag{2}$$

$$AGG = \sum_{u \in N(v))} \frac{\boldsymbol{h}_u^{l-1}}{|N(v)|} \tag{3}$$

where $\sigma(.)$ is an activation function (we use `ReLU` in our experiments), $\boldsymbol{W}_l$ and $\boldsymbol{B}_l$ are trainable matrices (i.e., encoding what the model learns), $N(v)$ represents the first-order neighbors of node v in the graph G, and $\boldsymbol{h}_v^l$ represents the l^{th} layer embedding of node v. The initial embeddings $\boldsymbol{h}_v^0$ are equal to the node features $\boldsymbol{f}_v$. `GraphSAGE` concatenates the self-embedding of the previous step $\boldsymbol{h}_v^{l-1}$ with the neighbor embedding (output of the AGG function) in order to update the embedding of a node $\boldsymbol{h}_v^l$. The $\boldsymbol{W}_l$ transformation learns the important components from the neighbors' features and the $\boldsymbol{B}_l$ transformation learns the important features of the node itself. Note that individual weight parameters are defined for each transformation and per layer l. `GraphSAGE` can consider several AGG functions; in our experiments, we use the mean aggregator function as described in Equation (3). The final embedding vector of the node, after L layers, is as follows:

$$\boldsymbol{z}_v = \boldsymbol{h}_v^L \tag{4}$$

2) Attention Networks [34], [35]: In `GAT`, an additional neural network is utilized to calculate the weight of the edges during aggregation. With multi-head attention of K, the layer $l-1$'s features propagate to layer l as follows:

$$\boldsymbol{h}_v^l = \Big\|_{k=1}^{K} \sigma\left(\textstyle\sum_{u \in N(v)} \alpha_{u,v}^k \boldsymbol{W}^k \boldsymbol{h}_v^{l-1}\right) \tag{5}$$

$$\alpha_{u,v}^k = \boldsymbol{LeakyReLU}\left((\boldsymbol{a}^k)^\intercal [\boldsymbol{W}^k \boldsymbol{h}_u \parallel \boldsymbol{W}^k \boldsymbol{h}_v]\right) \tag{6}$$

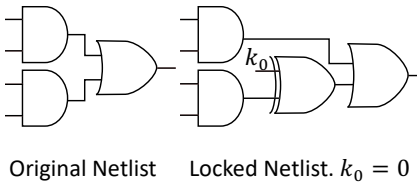

Fig. 6: An example for logic locking with XOR key-gate and $k_0 = 1$.

where $\alpha_{u,v}$ specifies the weighting factor (i.e., importance) of node u's features for node v. Note that $\alpha_{u,v}$ is computed as a byproduct of an attentional mechanism $\boldsymbol{a}$. More specifically, multi-head attention is performed in which the operations of the layer are independently replicated K times, each replica with different parameters, and the outputs are feature-wise aggregated. We use a concatenating operation as in Equation (5). Then, $\alpha^k_{u,v}$ are the attention coefficients derived by the k-th replica, and $\boldsymbol{W}^k$ is the weight matrix specifying the linear transformation of the k-th replica.

`GAAN` builds on `GAT`'s attention, by inserting additional gating mechanisms to impose different values for different heads' computations. Hence, an additional soft-gating function is leveraged ranging from 0 (low importance) to 1 (high importance), to assign different weights to each head.

3) Jumping Knowledge Network [36]: The `JK-net` model learns an adaptive and structure-aware representation. Toward that end, `JK-net` picks, for each node at the respective previous layer, from all intermediate embeddings which requires to jump to the last layer. This approach allows the model to adapt to each node's effective neighborhood size as needed. `JK-net` considers three approaches for aggregation [36]; in our experiments, we use the concatenation method as follows:

$$\boldsymbol{h}_{JK} = \sigma\left(\boldsymbol{W}^{\intercal}_{JK} \cdot ||^{L}_{l=1} \boldsymbol{h}^l_v\right) \tag{7}$$

C. Logic Locking

Logic locking is a design-for-trust technique that can be employed to protect the design IP throughout the supply chain [31], [37]. More specifically, logic locking protects the functionality of the design IP by inserting key-gates into the design. The locked design functions correctly only when the correct key is applied to the key-gates. The secret key is stored in an on-chip tamper-proof memory and is only known to the chip designer. Without easy access to the key, untrusted entities in the supply chain (foundry, test facility) cannot infer the functionality of the design even if they have access to the reverse-engineered netlist. We illustrate the concept of logic locking in Fig. 6.

III. Proposed `GNN-RE` Methodology

We now discuss our proposed `GNN-RE` methodology in detail. The overall process is illustrated in Fig. 7.

A. Setting

We investigate the problem of reverse engineering gate-level netlists under the following assumptions: the analyst has access to (i) a reverse-engineered gate-level netlist and (ii) a high-level description of the design's functionality to construct a training dataset suitable for the target design. The goal is to identify all functional modules.

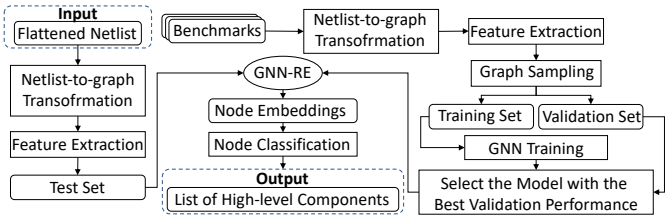

Fig. 7: Our proposed `GNN-RE` methodology. Initially, we train `GNN-RE` using customized designs and then use it to extract and identify the high-level components of previously unseen, unstructured netlists.

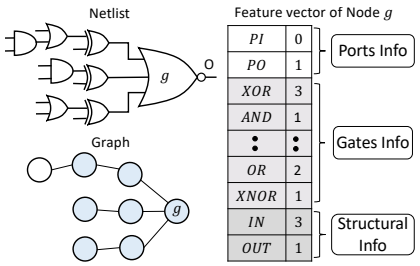

Fig. 8: Proposed netlist-to-graph transformation. `GNN-RE` considers three categories of features: (i) connectivity to PIs/POs (Ports Info), (ii) Boolean functionality of the local neighborhood (Gates Info), and (iii) the input and output degrees of the gates (Structural Info), all while constructing the initial embedding of each gate/node in the circuit/graph. The blue color represents the two-hop neighborhood of gate g.

Unlike library-based prior art [16]–[19], we do not require the *exact* implementation of modules within our training set, and we only leverage the high-level description of the design to incorporate functionally similar modules into the training set. All circuits considered in this work are different in terms of the types of their sub-circuits, interconnections, and bit-widths. In fact, our `GNN-RE` model is tested/validated on previously unseen designs, comprising specific modules not leveraged at any time during training.

B. Netlist-to-Graph Transformation and Feature Extraction

In general, a gate-level netlist can be entirely represented using a graph. Although logic circuits are inherently directional, representing them as undirected graphs renders message passing within the GNN more efficient. Note that circuits can be represented as undirected graphs without loss of structural or functional correctness, given that (i) any gate can only be driven by a single other gate and (ii) input and output degree of nodes/gates can be encoded as features. Hence, we represent a netlist as an undirected graph $G = (V, E)$, where V is the set of nodes (gates), of length n, and E is the set of edges (wires) connecting the nodes. The graph has node attributes where $\boldsymbol{f}_v$ is the feature vector of a node v, with length k. $F \in \mathbb{R}^{n \times k}$ is the two-dimensional matrix containing node features.

An example of a simple netlist and its corresponding graph is shown in Fig. 8. Next, we explain the considered features along this example. Note that we do not include the primary inputs/outputs (PIs/POs) as nodes, as our goal is to classify the gates, not PIs/POs. Nevertheless, a node's connectivity to

PIs/POs is captured in the feature vector, as highlighted as feature category in white in Fig. 8. The vector also captures the node's fan-in/input degree IN and the fan-out/output degree OUT, as highlighted as feature category in dark gray in Fig. 8. Capturing both input and output degrees is important as this structural information would otherwise be lost when using an undirected graph. The feature vector contains a third category that captures the gate's functionality and those of its neighbouring gates/nodes; see gates info highlighted in gray in Fig. 8). To this end, k is largely dictated by the number of gate types in the technology library. For sequential designs, we employ an additional binary feature to indicate whether a gate is combinational or sequential. Moreover, when using GNN-RE to identify the functionality of a logic-locked design, we employ an additional binary feature to indicate whether a gate is controlled by a KI or not, so that the GNN is made aware of key-gates versus regular gates. Each node's neighborhood up to two hops away are considered, i.e., gates directly connected to the gate/node itself are considered and also all gates connected in turn to those gates are considered. Consideration of such neighborhood is essential to capture the structural and functional properties of nodes in general and sub-circuits in particular. More details on hop sizes and their role for GNNs are also discussed in the next subsection. Note that the sums for all encountered gate types up to two hops away are included in the feature vector. Each encountered gate type has an entry (categorical row) in the feature vector. For example, the feature vector $\boldsymbol{f_g}$ of node g in Fig. 8 describes that the gate is connected to one PO, three XOR gates, one AND gate, and two OR gates in its two-hop neighborhood marked in blue. The vector also describes that the gate has an input degree of $IN = 3$ and an output degree of $OUT = 1$. The feature vectors are then standardized by removing the mean and scaling to unit variance.

C. Model Building and Testing

1) GNN-RE Training Phase: Considering scalability is essential for deep learning on graphs, due to an exponential growth of the GNN computations for larger neighborhoods being considered. The GNN depth L defines the size of the neighborhood to be considered during aggregation. More specifically, a depth of L for a given GNN layer implies that this layer has to account for all L-hop neighbor features, aside from its own/self features. Accordingly, the deeper the GNN, the more multi-hop nodes are to be accounted for when generating the root node's embedding. As a result, more messages are required for aggregation and, consequently, the GNN performance suffers. Layer sampling methods can limit the number of nodes aggregating their features per each hop in the considered neighborhood. Although such approaches reduce training time, they may suffer from scalability issues nevertheless as they still operate on the full graph.

In our work, we leverage the graph sampling approach *GraphSAINT* [38] to maintain scalability. In GraphSAINT, sub-graphs are sampled from the original graph, and a full GNN is constructed for each sub-graph. Hence, in GNN-RE, the sub-graphs are sampled first using the *random-walk* sampler (RWS) explained in Algorithm 1. The number of roots r and the random-walk length w are adjustable parameters. Given a training graph G, the first step is to sample the root nodes V_r, uniformly at random and with replacement, from the set of nodes V (Line 2). In Line 3, roots are added to the set of sampled nodes V_s. G_s denotes the induced sub-graph of G from V_s, and $N(u)$ represents the outgoing edges at node u. In Lines 4–10, a random-walk is performed from each root up to distance w, building the sub-graph along the way.

Algorithm 1 Random-walk sampling (RWS) algorithm by GraphSAINT [38]

Input: Training graph $G(V, E)$; number of roots r; walk length w
Output: Sub-graph $G_s(V_s, E_s)$

1: Initiate $G_s = (V_s, E_s)$ with $V_s \leftarrow \{\emptyset\}$
2: $V_\text{r} \leftarrow r$ ▷ Sample root nodes from V
3: $V_s \leftarrow V_\text{r}$
4: **for** $v \in V_\text{r}$ **do**
5: $u \leftarrow v$
6: **for** $distance = 1$ to w **do**
7: $u \leftarrow$ Node sampled at random from $N(u)$
8: $V_s \leftarrow V_s \cup \{u\}$ ▷ Build the sub-graph
9: **end for**
10: **end for**
11: $G_s \leftarrow$ Node induced sub-graph of G from V_s

Algorithm 2 GNN-RE training algorithm

Input: Training graph $G(V, E, F)$; Ground truth $\overline{\boldsymbol{Y}}$; Sampler RWS
Output: Trained GNN

1: Compute normalization coefficients α, λ using RWS
2: **for** each mini-batch **do**
3: $G_s(V_s, E_s) \leftarrow$ Sampled sub-graph of G using RWS
4: Build GNN on G_s
5: $\{\boldsymbol{y}_v \mid v \in V_s\} \leftarrow$ Propagating α-normalized $\{\boldsymbol{f}_v \mid v \in V_s\}$
6: Propagating λ-normalized loss $L(\boldsymbol{y}_v, \overline{\boldsymbol{y}}_v)$ to update weights
7: **end for**

During the training stage of GNN-RE, various designs are fed to the GNN, including different sub-circuits with varying interconnections and bit-width sizes, allowing the GNN to learn on all the features of these different designs and sub-circuits. The pseudocode for GNN-RE's training procedure is shown in Algorithm 2 and explained next. To ensure that GNN-RE is not biased to nodes more frequently sampled by RWS, we follow the normalization technique proposed in [38] as follows. In Line 1, the probability values are estimated for a node p_v and an edge $p_{u,v}$ to be sampled by RWS, where $v \in V$ and $(u, v) \in E$. The probability is computed as $p_{u,v} \propto B_{u,v} + B_{v,u}$, where $B_{u,v}$ is the probability of a random-walk to start at u and end at v in L hops (and vice-versa for $B_{v,u}$). The probability scores are used to compute α and λ, required for aggregation normalization. More details on the normalization method can be found in [38]. The training of GNN-RE follows a stochastic gradient-descent procedure. For each mini-batch, a graph G_s is independently sampled from G using RWS (Line 3). A GNN is built on G_s (Line 4) to perform neighborhood aggregation and compute nodes' embeddings (Line 5). The ground-truth labels are used to calculate the loss for every $v \in V_s$ (Line 6), and the GNN weights are updated accordingly.

2) GNN-RE Inference Phase: The high-level inference algorithm of GNN-RE is summarized in Algorithm 3. The target

Algorithm 3 `GNN-RE` inference algorithm

Input: Flattened netlist N; Trained GNN
Output: High-level components
1: Initiate $G = (V, E, F)$ with $V \leftarrow netlist\_to\_graph(N)$
2: **for** each $v \in V$ **do**
3: $\quad \boldsymbol{z}_v \leftarrow GNN(v)$ ▷ Compute embedding
4: $\quad c_v \leftarrow fc(z_v)$ ▷ Classify node v
5: **end for**

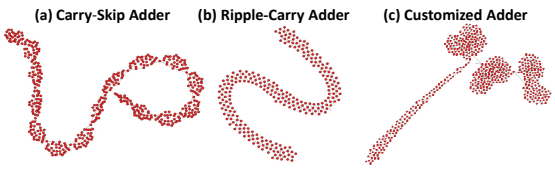

Fig. 9: Graph visualization of the different 64-bit adder designs. Each design has a different structural implementation.

netlist is converted to a graph (Line 1). Then, for each gate, the embeddings are computed using the trained GNN (Line 3). The final representations are passed to a fully-connected (fc) layer with `softmax` activation function to classify each node v into its class c_v, i.e., sub-circuit type (Line 4).

D. Dataset Generation

`GNN-RE` can operate under two settings, depending on whether boundary identification and sub-circuit extraction is required (Case II) or not (Case I). Note that `GNN-RE` is developed with the specific aim of operating under the realistic scenario of Case II. However, Case I is still considered, to be able to compare with the prior art which fails to generalize to Case II. Next, we describe the datasets used for both settings. The summary of these datasets is listed in Table II.

1) Case Study I: Sub-Circuit Classification: We generate a dataset called `Single-Modules` that contains designs consisting of single modules. Each design in the dataset is either an adder, subtractor, multiplier, or a comparator. The generated designs vary in terms of their bit-width; designs with bit-widths of $\{2, 4, 32, 64\}$ are used for training, and designs with bit-width of 8 are used for validation. To evaluate our proposed model, we test on a 128-bit adder and a 64-bit multiplier (not included in the testing set) from the EPFL benchmark suite [29]. We also test our model on the `74283` 4-bit adder and the `74L85` 4-bit magnitude comparator from the 74X series combinational benchmarks [30]. We further test the model on a synthetic 128-bit comparator and a synthetic 128-bit subtractor, and we test on the `c6288` benchmark, which is a 16×16 multiplier from the ISCAS-85 suite. It is important to note that the proposed `GNN-RE` model is tested on more complex structures—in terms of circuit size and/or bit-widths—than than those considered during training. As an advance note, our model performs node (gate) classification with an accuracy of 99.28%. That is, our model successfully learns from less complex samples the relevant structural and functional properties and is able to always correctly classify more complex circuits using those high-accuracy node (gate) inferences.

To showcase the advantage of utilizing an ML model to perform functional RE over a library-based approach, we furthermore introduce some functional variations for the adder designs in the training set, including a carry-look-ahead adder, a carry-select adder, and a ripple-carry adder. We then test `GNN-RE` on a carry-skip adder, which the model had no access to during training. The same experiment is repeated for the ripple-carry adder (i.e., excluding it from the training set but including all other variations). The *Verilog* implementations of all the adders are obtained from [39]. We provide the corresponding graphs of the different adders in Fig. 9, which clearly visualizes the diversity between the graphs and the underlying adders.

To demonstrate the applicability of `GNN-RE` for RE of sequential designs, we also construct a database of sequential designs, called `Sequential-Modules`, which include counters, shift registers, and FSMs. The database consists of 29 circuits in total, out of which 9 are counters, 9 are sequence detector FSMs, and 11 are shift registers. While the number of designs considered may appear small for ML applications, each design holds a large number of gates/nodes to consider. We use 19 modules for training, which include 5 8-bit shift-left/shift-right registers, $\{16, 32, 64\}$-bit free-running shift registers, 4 3-bit FSMs (Mealy and Moore), etc. We validate `GNN-RE` on 6 graphs, which include various FSMs, 8-bit shift registers, and 4-bit clock-enabled up/down-counters. We then test `GNN-RE` on 4 graphs, which include an 8-bit up-counter with controllable inputs and an 8-bit shift register with controllable inputs. The controllable inputs, using MUXes, were not seen by `GNN-RE` during training or validation. We also test on a 3-bit Moore FSM and a 3-bit Mealy FSM.

2) Case Study II: Sub-Circuit Extraction (Boundary Identification) and Labeling: Here, `GNN-RE` is expected to perform sub-circuit extraction and labeling without relying on any other tool for boundary identification. To train `GNN-RE` for identification of interconnected components underlying an unstructured netlist, we create different datasets, representing several interconnected designs. A visualization of some selected designs from the customized datasets is illustrated in Fig. 10. We consider the following types of sub-circuits in these experiments: adders, subtractors, comparators, multipliers, and control logic. More specifically, we generate 8 datasets as follows:

1) The `Add-Mul-Mux` dataset contains designs consisting of one adder, one multiplier, and control logic to perform some arithmetic operation based on the inputs (see Fig. 10a for a sample graph). It can be inferred from the figure that the adder (red) and multiplier (purple) sub-circuits are only connected via some control logic (green). While the adder and multiplier share the same PIs in the designs, recall that we do not represent the PIs/POs as nodes in the graphs, hence the separation between the two sub-circuits becomes more pronounced in the graph.
2) The `Add-Mul-Mix` dataset contains designs with two adders whose outputs feed into a multiplier (see Fig. 10b). Hence, this dataset serves to study the effect of different types of interconnections between the adder and multiplier sub-circuits for the proposed `GNN-RE` model.
3) The `Add-Mul-Combine` dataset contains designs with one adder and one multiplier sharing the same PIs but affecting different POs (see Fig. 10c). Therefore, aside from the PI

TABLE II
GENERATED DATASETS

Case Study	Datasets	#Classes	#Features	#Nodes	#Circuits	#Training Nodes	#Validation Nodes	#Testing Nodes
I	Single-Modules	4	33	32,528	27	6,318	357	25,853
	Sequential-Modules	3	39	841	29	700	69	72
II	Add-Mul-Mux	3	33	15,582	7	14,467	239	876
	Add-Mul-Mix	2		21,602	6	19,377	576	1,649
	Add-Mul-Combine	2		14,288	6	13,256	217	815
	Add-Mul-All	3		51,472	19	47,100	1,032	3,340
	Add-Mul-Comp	4		15,898	6	14,717	255	926
	Add-Mul-Sub	4		18,206	6	16,786	309	1,111
	Add-Mul-Comp-Sub	5		19,151	6	17,648	342	1,161
	Interconnected-Modules	5		104,727	37	96,251	1,938	6,538

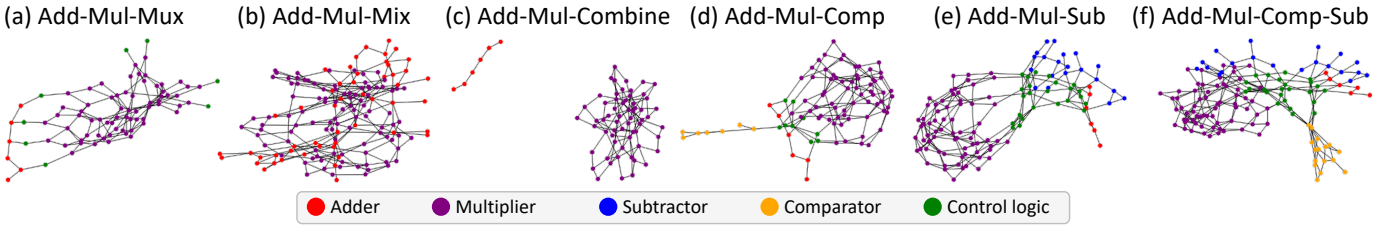

Fig. 10: Visualization of the graphs with a bit-width of 4 for the underlying circuits. Note that each type of sub-circuit exhibits a distinctive structure, which is learned by the GNN (in addition to the functionality of the sub-circuits).

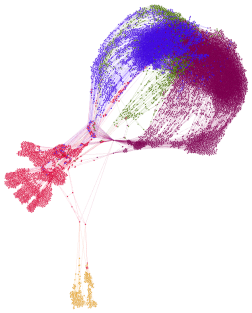

Fig. 11: Visualization of a design from the Add-Mul-Comp-Sub dataset with bit-width of 64. We follow the same color-coding of that in Fig. 10.

connections, which are not captured in the graph, these two sub-circuits remain separated.

4) The Add-Mul-All dataset combines all designs from the Add-Mul-Mux, Add-Mul-Mix, and Add-Mul-Combine datasets.
5) The Add-Mul-Comp dataset contains designs with a single adder, a single multiplier, and a comparator (see Fig. 10d). The comparator logic takes in two PIs and compares their magnitude. The comparison result dictates whether addition or multiplication is performed on the inputs. Hence, in this dataset, there is no external control input. As can be seen in Fig. 10d, the comparator block (orange), the adder (red), and the multiplier (purple) feed the control logic (green), which drives the POs depending on the comparison result.
6) The Add-Mul-Sub dataset includes designs containing a single adder, a single multiplier, two subtractors, and control logic (see Fig. 10e). An external control signal dictates the readout of each design. As can be seen in Fig. 10e, the adder (red), subtractors (blue), and multiplier all feed the control logic (green).
7) The Add-Mul-Comp-Sub dataset includes designs in which a comparator block dictates the operation to be carried out by the main module (see Fig. 10f and Fig. 11).
8) Finally, all the generated designs are combined into one single dataset, called Interconnected-Modules. Note that converting the full dataset, which contains 37 designs and a total of $104,727$ gates, to graphs and extracting the features for all the nodes takes $1,090s$ in total.

IV. EXPERIMENTAL EVALUATION

We evaluate our model's capability in performing two independent tasks: (i) sub-circuit classification and (ii) sub-circuit extraction (boundary identification) and labeling.

The setup for the first task assumes that the sub-circuits' boundaries in a given design are already known, and we are only left to identify their functionalities. We perform such experiments primarily to compare our model's performance with the state-of-the-art ML-based approaches for sub-circuit classification. However, our primary objective is to perform sub-circuit extraction (boundary identification) and labeling (Task (ii)), in which a flattened netlist with several interconnected sub-circuits is fed to GNN-RE for identifying both the boundaries and functionalities of these sub-circuits.

A. Setup, Evaluation Methods, and Metrics

We verify all designs via functional simulation using *Synopsys VCS* and synthesize the designs using *Synopsys Design Compiler* for the 65nm *GlobalFoundries LPe* technology.

We evaluate GNN-RE for (i) sub-circuit classification on the customized designs, selected EPFL modules, selected ISCAS-85 benchmarks, and selected 74X series benchmarks, and for (ii) sub-circuit extraction and labeling on the customized designs. We implement the netlist-to-graph transformation

in *Perl/Python3* and utilize the *Tensorflow Python3* implementation of `GraphSAINT` for GNN training [38]. Training is conducted on a single computer with 24 cores (2x Intel Xeon CPUs E5-2695 v2@2.4 GHz, 256GB RAM), and one NVIDIA Tesla K20m GPU (2,496 CUDA cores, 5GB of GDDR5 memory).

The performance of `GNN-RE` is measured by comparing the node predictions to the ground-truth labels associated with the corresponding databases. To this end, we report four different metrics: *accuracy*, non-averaged *precision*, non-averaged *recall*, and non-averaged *F1-Score*.

B. Optimization of GNN Model for Functional RE

Our work is the first to leverage GNNs for functional RE of digital circuits. Thus, we initially study the impact of different GNN structures and corresponding hyper-parameters on the accuracy and training time for the main objectives of our work (i.e., sub-circuit extraction and labeling).

We organize this part of our study as follows. The overall process for selecting and optimizing the GNN model is illustrated in Fig. 12. `GraphSAGE` is the first stage that we investigate by varying the hidden dimension h, the GNN depth L, and the number of GNN layers. Next, we investigate the effect of different propagation layers such as `GAT`, `GAAN`, and `JK-net`. The analysis will show that the `GAT` architecture provides the best performance for our customized datasets.

More specifics for the experiment setup are as follows. Initially, we consider the `Add-Mul-All` dataset; the best model configuration is then evaluated further on the rest of the considered datasets. We use the `GraphSAINT` graph sampling to construct mini-batches (i.e., sampled sub-graphs) for training. The `RWS` sampler of `GraphSAINT` (see Algorithm 1) is used with a walk length of $w = 2$, sample coverage of 50, and $r = 3,000$ root nodes. Note that we have also investigated the effect of increasing the depth of the sampler to 4, where we found that it degraded the performance of the model in general. Forward and backward propagation is performed iteratively for each mini-batch, to update weights via stochastic gradient descent. The Adam optimization algorithm is utilized during training, with a learning rate of 0.01 and a dropout rate of 0.1. The final layer is a fully-connected layer of size h with a `softmax` activation function in all our experiments. All runs are terminated after $2,000$ epochs. In each epoch (training cycle), the full training set is used. In `GraphSAINT`, an epoch consists of $|V|/|V_s|$ weight update steps, where $|V|$ denotes the number of training nodes and $|V_s|$ denotes the average number of nodes in sampled sub-graphs. After each epoch, the GNN model is evaluated on the graphs in the validation set to estimate its performance on unseen data. The best performing model on the validation set is restored at the end of training and subsequently used to evaluate the GNN on the testing set.

Effect of the hidden dimension: We construct a two-layer GNN using the `GraphSAGE` architecture with a mean aggregator function. The GNN depth is set to $L = 1$ (i.e., self features and one-hop neighbors' features). We study the effect of varying the hidden dimension $h : \{128, 256, 512\}$ on the Micro-F1 and Macro-F1 scores. Our experiments indicate

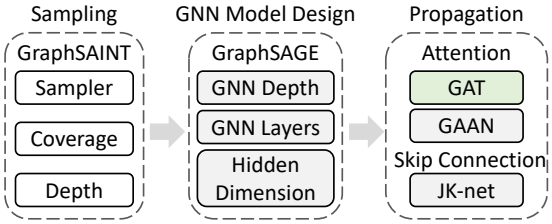

Fig. 12: Steps involved in the process of selecting and optimizing the GNN model for `GNN-RE`. We evaluate the performance of `GNN-RE` under different sampling parameters, different GNN designs, and different propagation layers. We report the experimental results for the blocks colored in gray. Our analysis demonstrates that the `GraphSAINT` sampling approach based on the `GAT` GNN architecture provides the best performance for our desired task.

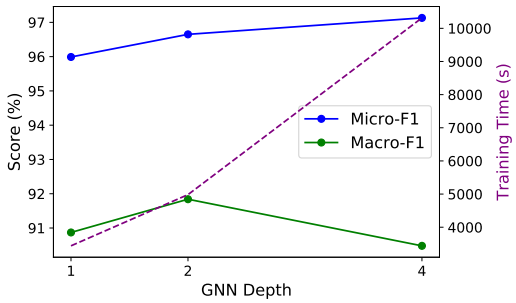

Fig. 13: Effect of the GNN depth considering the `Add-Mul-All` dataset.

that the proposed `GNN-RE` platform based on the `GraphSAGE` architecture performs best when the hidden dimension is set to $h = 256$, with a Macro-F1 of 95.99% and a Micro-F1 of 90.87%. *Achieving such F1 scores means that `GNN-RE` has achieved a high predictive accuracy and was able to separate between the adder nodes, multiplier nodes, and control nodes, although the related sub-circuits are highly interconnected.*

Effect of the GNN depth: Setting the hidden dimension to $h = 256$, here we explore the effect of increasing the GNN depth L (i.e., extending the neighborhood of each node to the L^{th} layer). As explained before, the higher L is, the more multi-hop nodes are accounted for when computing the embedding of the root node. The GNN depth is varied between $L : \{1, 2, 4\}$ and the results are reported in Fig. 13. The figure shows that increasing the depth improves Micro-F1 scores but at the cost of longer training time. Migrating from depth of 1 to depth of 2 improves the overall performance of the model; for $L = 2$, we report a Micro-F1 score of 96.65% and a Macro-F1 score of 91.84%. However, migrating from depth 2 to depth of 4 degrades the Macro-F1 ratio again, to 90.48%. We find that the model for $L = 4$ performs well on the more common classes (adders and multipliers) while performing worse on the more rare classes (control). This is because the control sub-circuit is small in size when compared to the adder and multiplier sub-circuits; by increasing the GNN depth to 4, the control nodes can also capture information from more distant neighbors, which may be adders or multipliers, thereby overwriting the information they learn about the control sub-circuit. This problem is known as over-smoothing [40]–[42].

Effect of the GNN layers: Setting $h = 256$ and $L = 1$, here we study the effect of the number of GNN layers. The number of layers is varied between $\{2, 3, 4\}$. We notice that an

TABLE III
GNN CONFIGURATION AND SAMPLING DETAILS

Architecture		Training and Sampling	
Total # Layers	5	**Optimizer**	Adam
GNN Depth	1	**Dropout**	0.1
Input Layer	$[33, 256]$	**Attention**	8
Hidden Layers 1-4	$[512, 256]$	**Learning Rate**	0.01
Output Layer	$[256, \#classes]$	**Sampler**	Random Walk
Aggregation	Attention aggregator with multi-head concatenation	**Walk Length**	2
Activation	`ReLU`	**Root Nodes**	$3,000$
Classification	`softmax`	**Max # Epochs**	$2,000$

increase in the number of layers improves the model's overall performance, with an approximately linear increase in training time. With layers set to 4, the model achieves a Micro-F1 score of 97.54% and a Macro-F1 score of 91.83%, respectively, and training time is $6,213.58s$.

Effect of GNN structure/architecture: After evaluating several hyper-parameter combinations on the `GraphSAGE` structure above, we next study the effect of the GNN structure itself on the performance of `GNN-RE`. To that end, we switch between the following backbone architectures: `GraphSAGE`, `JK-net`, `GAT`, and `GAAN`. For `GAT` and `GAAN`, we use $K = 8$ for the multi-head computations.[4] We observe that the `GAT`-based `GNN-RE` platform achieves the best performance reaching a Micro-F1 score of 97.84% and a Macro-F1 score of 95.99%. Hence, for the remainder of our study, we fix the GNN structure to `GAT`, with the parameters listed in Table III.

Evaluation of the `GNN-RE` method: After studying the hyper-parameters and structures of the GNN model in detail considering the `Add-Mul-All` dataset, we next evaluate `GNN-RE` on all the other datasets as well. As indicated, here we use the `GAT` structure with parameters listed in Table III. We summarize the related results in Table IV and discuss the findings in the following subsections.

C. Case I: Sub-Circuit Classification

For the `Single-Modules` dataset, we note that `GNN-RE` achieves a classification accuracy of 99.28% with precision and recall values ranging from 87.29% to 100% (Table IV). It is essential to note that these metrics are evaluated at the node level (i.e., the proposed `GNN-RE` model assigns a label to every node in a given graph). For this case study here, however, we are interested in the full graph- or circuit-level classification, where the sub-circuits are assumed to be extracted in advance. Thus, we re-tailor `GNN-RE` to perform such a graph labeling as follows. We deploy a simple post-processing stage that checks the percentage of predicted node classes and then labels the graph based on the most predicted class; we refer to this proposed approach as majority vote classification.

This re-tailored setup of `GNN-RE` (see Fig. 14) achieves a sub-circuit classification accuracy of 100% for all the tested designs, including those of the EPFL and 74X benchmark suites. Moreover, `GNN-RE` classified the `c6288` ISCAS benchmark as a multiplier with 89% node classification accuracy, leading to

[4]Note that, in the `GraphSAINT` implementation of `GAT`, the α coefficient is not normalized by a `softmax` function across all the neighbors $N(v)$, as during mini-batches extraction a node might not see all its neighbors due to graph sampling.

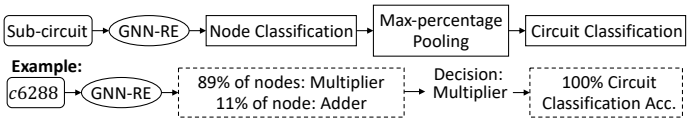

Fig. 14: Proposed majority vote classification for circuit classification.

100% circuit classification accuracy, as demonstrated in Fig. 14. Next, we provide some more detailed findings.

Classification of sequential designs: For the `Sequential-Modules` dataset, `GNN-RE` achieves an accuracy of 88%, with precision up to 100%, while classifying nodes for their circuit type. All the gates, be they sequential or combinational, were correctly classified for the FSM circuits. Almost all other nodes in the test set were correctly classified as well, except for 9 nodes of counter circuits that were misclassified as shift-register nodes. Still, applying our majority vote classification, each design was correctly classified (as either counter, shift register, or FSM circuit).

Detection of different adder architectures: `GNN-RE` labels the ripple-carry and the carry-skip adder designs (Fig. 9) with node classification accuracies of 99.62% and 87.28%, respectively. Given that the vast majority of nodes were correctly classified as adder nodes for both designs, `GNN-RE` also achieves a 100% graph/sub-circuit classification accuracy for both designs. Recall that `GNN-RE` was trained on different adder architectures (Sec. III-D1), and it was applied successfully to generalize to these previously unseen adder architectures.

Comparison with state-of-the-art ML-based techniques: To ensure fairness of comparison, we test the same benchmarks considered by the prior art and perform the task of circuit classification without having to perform boundary identification.

The CNN-based techniques in [21], [22] achieve an average accuracy of 88.2% and 99% for labeling the sub-circuit functionality when tested on the EPFL and the 74X benchmarks, as reported in [22]. In contrast, `GNN-RE` achieves a 100% accuracy for the same scenario. We also note that the CNN training time in [21] is reported to range between $254.24s$ and $501.48s$ for 20 training rounds on a dataset containing 100 designs. Training the CNN model appears to be faster than training the GNN used in `GNN-RE`. Still, the training time for `GNN-RE` is by no means prohibitively long for this or any other scenario considered in this work (see Table IV).

D. Case II: Sub-Circuit Extraction and Labeling

Recall that this case forms the main objective for this work and that it is more challenging than the circuit identification considered above. When testing `GNN-RE` on the datasets with interconnected sub-circuits, the highest performance (in terms of node classification accuracy) was achieved for the `Add-Mul-Mux` dataset, namely 99.45%; the lowest performance, still 97.75%, was achieved for the `Add-Mul-All` dataset.

Effect of the number of classes: Increasing the number of considered classes from three (for dataset `Add-Mul-All`) to five (for dataset `Interconnected-Modules`) did not hamper the performance of the model. We found that `GNN-RE` was rather able to improve the F1-Score by 4.14% for the *ADD* classifier and by 0.66% for the *MUL* classifier. Note that increasing the

TABLE IV
GNN-RE NODE CLASSIFICATION RESULTS WHEN ADOPTING THE GAT GNN STRUCTURE WITH GRAPHSAINT MINI-BATCH CONSTRUCTION METHOD. A "-" INDICATES THAT THE CORRESPONDING CLASS IS NOT INCLUDED IN THE DATASET

Case Study	Dataset	GNN Accuracy (%)	Precision (%)					Recall (%)					F1-Score (%)					Training Time (s)
			ADD	MUL	CTRL	COMP	SUB	ADD	MUL	CTRL	COMP	SUB	ADD	MUL	CTRL	COMP	SUB	
I	Single-Modules	99.28	95.15	99.99	-	98.35	87.29	100	99.34	-	98.73	97.75	97.52	99.67	-	92.66	98.05	1,282.05
II	Add-Mul-Mux	99.32	97.43	99.60	97.50	-	-	96.20	99.60	100	-	-	96.82	99.60	98.77	-	-	2,289.13
	Add-Mul-Mix	99.45	99.37	99.47	-	-	-	98.43	99.85	-	-	-	98.9	99.66	-	-	-	3,119.42
	Add-Mul-Combine	99.02	90.12	100	-	-	-	100	98.92	-	-	-	94.87	99.45	-	-	-	2,297.50
	Add-Mul-All	97.75	88.61	99.5	95	-	-	97.25	98.23	88.37	-	-	92.73	98.72	92.68	-	-	6,838.18
	Add-Mul-Sub	98.02	86.42	99.61	90.91	-	100	93.33	99.74	98.77	-	92.55	89.74	99.67	94.67	-	96.13	2,296.61
	Add-Mul-Comp	99.35	94.59	99.87	97.06	100	-	100	100	97.06	89.13	-	97.22	99.94	97.06	94.25	-	2,275.00
	Add-Mul-Comp-Sub	98.28	100	97.42	100	100	100	100	100	91.14	92.9	100	100	98.69	95.36	96.32	100	2,308.70
	Interconnected-Modules	98.87	97.51	99.14	94.74	100	100	96.24	99.61	97.5	97.39	94.88	96.87	99.38	96.1	98.68	97.37	16,317.44

number of classes degrades the performance of the CNN-based platform presented in [21].

Scalability: Increase in dataset size correlates with improved performance of GNN-RE. At the same time, the training efforts for GNN-RE increase linearly with the number of training nodes, which is supported by utilizing the GraphSAGE graph sampling approach. More specifically, comparing the results of GNN-RE for the Add-Mul-Comp-Sub dataset where the total number of nodes is $19,151$ with that for the Interconnected-Modules dataset where the total number of nodes is $104,727$ (i.e., $5.5\times$ that of Add-Mul-Comp-Sub), the accuracy increases from 98.28% to 98.87%. We also note that the training time increases from $2,308.70s$ for $17,648$ training nodes to $16,317.44s$ for $96,251$ training nodes, or $5.5\times$ as well.

E. Misclassification Analysis

Next, we discuss the misclassification of nodes incurred by GNN-RE in some detail. The most frequent cases were mispredicting multiplication nodes as adder nodes (40% of all misclassification cases), followed by mispredicting adder nodes as multiplication nodes (18% of all misclassification cases). This is not a surprising finding; while multiplication operations can be implemented in various designs, the process in general requires the summing of terms of partial products. We also note that particular misclassification cases are unlikely. For example, comparator nodes were never mispredicted as multiplier nodes and only rarely misclassified as adder nodes.

To ascertain the root-cause for misclassification, we study the features of sub-circuits for two selected datasets, Single-Modules and Interconnected-Modules. We are reporting the average input degree IN and output degree OUT of the nodes, the percentage of nodes connected to PIs and POs, and the most common types of gates appearing in each sub-circuit (Table V). When comparing the adder, subtractor, and multiplier sub-circuits, we note that all three are primarily implemented using the same common types of gates, namely NAND and INV. In contrast, the control sub-circuit has a unique MUXes-centric structure, a larger percentage of nodes connected to POs, and a larger average node degree. Similarly, the comparator sub-circuit comprises AOI gates, which are only sparsely used in other sub-circuits, and hold a relatively large share of nodes connected to POs.

In short, sub-circuits with more unique feature profiles are seldom misclassified, whereas sub-circuits with similar feature profiles may be misclassified with each other. In any case, GNN-RE is still quite accurate overall; the rate of node misclassification is not exceeding 2.25% for any dataset. Furthermore, it is important to recall that, for the main focus of this work (sub-circuit extraction and labeling), GNN-RE succeeds with correct inference for all considered test cases.

TABLE V
EXTRACTED FEATURES FOR THE CONSIDERED SUB-CIRCUITS

Dataset	Module	Avg IN	Avg OUT	Nodes Connected to PIs	Nodes Connected to POs	Common Gates
Case Study I, Single-Modules	ADD	2.02	1.62	27.09%	14.09%	25% NAND 20% INV
	MUL	2.33	2.12	15.68%	0.92%	25% NAND 19% INV
	SUB	1.89	1.65	35.65%	12.23%	29% INV 25% NAND
	COMP	2.51	1.89	58.88%	1.69%	25% INV 21% AOI
Case Study II, Interconnected-Modules	ADD	2	1.7	25.39%	1.86%	25% NAND 20% INV
	MUL	2.48	2.16	17.45%	0.55%	25% NAND 19% INV
	SUB	1.88	1.63	38.09%	0%	29% INV 25% NAND
	CTRL	2.5	1.82	11.05%	47.04%	37% MUXes 25% INV
	COMP	2.7	1.8	83.52%	0%	25% INV 21% AOI

F. GNN-RE on Logic Locking

To demonstrate the applicability of GNN-RE on obfuscated designs, we investigate the following setting. Taking the Single-Modules dataset of Case I, we lock all the circuits (including training and validation circuits) using random X(N)OR logic locking [31], with a key-size of $K = 64$ and random secret keys. These locked circuits are grouped to construct the Obfuscated-Single-Modules dataset.

In the presence of logic locking in Obfuscated-Single-Modules, GNN-RE achieves a node classification accuracy of 93.83%. This corresponds to a performance loss of 5.45 percentage points when compared to the accuracy of 99.28% obtained without obfuscation in place. Thus, although logic locking causes a more challenging problem for RE at the node level, GNN-RE was still able to predict the large majority of gates correctly, allowing to infer the design's functionality with 100% accuracy. This indicates that traditional logic locking may not be sufficient to obfuscate the structure of a netlist. This gives also rise to an intuition that GNN-RE could be tailored to detect IP piracy, where designs are copied and potentially modified by the adversary, to mask the piracy act.

G. Superiority of GNN Model

To demonstrate the strength of the GNN model in the context of functional RE over other ML models, we implement another

supervised classification method using support vector machines (SVM). We train the SVM model using the same feature vectors for nodes as with the GNN model. Unlike the GNN approach, however, the SVM model allows for each node/gate to only reason about its own features and has no access or information to its neighbors' features. Still, our proposed feature vector does capture the functionality of gates in the local neighborhood. Hence, we ask here if such functional information is sufficient to achieve high-accuracy node classification.

A linear kernel is used for the SVM model with $c = 1$, and the model's performance is evaluated by accuracy, Macro-F1 score, and the training time required, all considering our customized datasets. The results are summarized in Table VI and discussed next. First, the SVM approach takes less training time, e.g., it requires only $1,636.43s$ to train on the `Interconnected-Modules` dataset whereas the GNN method requires $16,317.44s$ for training that dataset. Second, the performance of the SVM model is, however, much lower; the SVM classifier achieves an accuracy of 81.43%, while the GNN's accuracy reaches 98.87%. More importantly, the SVM classifier achieves a Macro-F1 score of merely 47.38%, clearly indicating that the SVM model could not learn how to separate the classes correctly. In contrast, the GNN model consistently achieves a Macro-F1 score above 90%.

TABLE VI
PERFORMANCE OF THE SVM MODEL IN NODE CLASSIFICATION

Case Study	Dataset	Accuracy (%)	Macro-F1 (%)	Training Time (s)
I	`Single-Modules`	63.64	35.44	2.50
II	`Add-Mul-Mux`	90.63	68.85	2.40
	`Add-Mul-Mix`	85.68	45.16	22.43
	`Add-Mul-Combine`	90.67	47.55	5.60
	`Add-Mul-All`	88.32	66.37	117.52
	`Add-Mul-Comp`	91.14	71.21	2.63
	`Add-Mul-Sub`	79.83	62.01	11.58
	`Add-Mul-Comp-Sub`	79.93	66.21	11.71
	`Interconnected-Modules`	81.43	47.38	1,636.43

V. DISCUSSION

A. Non-ML-Based Approaches for Functional RE

The behavioral pattern mining approach by Li *et al.* [17] infers sub-circuits by matching against a set of known models. However, it is assumed that sub-circuits are extracted in advance. The word-level reconstruction approach in [16] is capable of deriving the word-level dataflow of a flattened netlist. However, the performance of this approach is easily hampered by various optimizations that could be leveraged during logic synthesis. In [43], the authors follow a graph embedding approach to infer the sub-circuits in a netlist that match with the high-level blocks of a reference design description. Unlike `GNN-RE`, such technique requires the golden register-transfer level (RTL) description. Subramanyan *et al.* [18] perform structural and functional analyses on both the individual gates and aggregated sub-circuits. Functional identification of a component presumes the correctly inferred order of input bits [18]; a brute-force approach is required to obtain the correct permutation of input bits. Gascon *et al.* [19] proposed a template-based solution to solve this problem of finding the correspondence between the sub-circuit under investigation and the reference model. Fyrbiak *et al.* [44] present a library-agnostic platform, called HAL, which can be used for functional RE of both FPGAs and ASICs. HAL explicitly supports extensibility by allowing integration with custom tools and provides a graphical user interface that aids the RE analysis.

B. Functional RE for Detection of Hardware Trojans (HTs)

Functional RE of gate-level netlists could aid in the detection of HTs. Note that the various prior HT detection techniques assume different threat models. For most side-channel-analysis methods to work, a golden version of the IC must be available [45]–[48]. Such an assumption is difficult to satisfy when third-party IPs are integrated into the design. For testing-based detection techniques, a golden RTL source is assumed for verification [49], [50]. Additionally, the threats of having a Trojan implanted either through design tools or during fabrication cannot be handled; only malicious RTL can be detected. In contrast, functional RE does not make such assumptions, and it does not limit the threat model [18].

As discussed, our proposed `GNN-RE` method can accurately extract and label sub-circuits from unstructured netlists. However, determining whether the identified components are malicious or not is reserved for future work.

VI. CONCLUSION

We have presented `GNN-RE`, a novel platform for functional reverse engineering that operates on flattened gate-level netlists, leveraging graph neural networks (GNNs) for sub-circuit identification and labeling. By representing circuits as graphs, `GNN-RE` naturally accounts for both the netlist's global structure and each gate's local neighborhood. In addition to the structural information inherently embedded in the GNN model, our feature vector captures the functionality of each gate's neighborhood. Our extensive evaluation studies hyper-parameters, various GNN backbone architectures, and other aspects in detail. Throughout all the experiments, `GNN-RE` shows high accuracy while automatically extracting and labeling sub-circuits. The GNN model can also detect previously unseen variants of the baseline designs it was trained on. In the promise of our findings, we believe that our proposed platform opens a new pathway for hardware security and other applications in need of sub-circuit extraction and identification.

REFERENCES

[1] L. Alrahis. (2021) GNN-RE datasets. [Online]. Available: https://github.com/DfX-NYUAD/GNN-RE
[2] K. Basu, S. M. Saeed, C. Pilato, M. Ashraf, M. T. Nabeel, K. Chakrabarty *et al.*, "CAD-Base: An attack vector into the electronics supply chain," *TODAES*, vol. 24, no. 4, pp. 38:1–38:30, 2019.
[3] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.
[4] J. Rajendran, O. Sinanoglu, and R. Karri, "Regaining trust in VLSI design: Design-for-trust techniques," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1266–1282, 2014.
[5] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[6] D. Blair, J. Huntsman Jr, C. Barrett, S. Gordon, W. Lynn III, D. Wince-Smith *et al.*, "Update to the IP commission report: The report of the commission on the theft of american intellectual property," *The National Bureau of Asian Research*, 2017.

[7] J. Knechtel, S. Patnaik, and O. Sinanoglu, "Protect your chip design intellectual property: An overview," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, ser. COINS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 211–216.

[8] I. McLoughlin, "Reverse engineering of embedded consumer electronic systems," in *IEEE International Symposium on Consumer Electronics (ISCE)*, 2011, pp. 352–356.

[9] A. Kimura, J. Scholl, J. Schaffranek, M. Sutter, A. Elliott, M. Strizich *et al.*, "A decomposition workflow for integrated circuit verification and validation," *Journal of Hardware and Systems Security*, vol. 4, no. 1, pp. 34–43, 2020.

[10] U. J. Botero, R. Wilson, H. Lu, M. T. Rahman, M. A. Mallaiyan, F. Ganji *et al.*, "Hardware trust and assurance through reverse engineering: A survey and outlook from image analysis and machine learning perspectives," *arXiv preprint arXiv:2002.04210*, 2020.

[11] R. S. Rajarathnam, Y. Lin, Y. Jin, and D. Z. Pan, "ReGDS: A reverse engineering framework from GDSII to gate-level netlist," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 154–163.

[12] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors." *Smartcard*, vol. 99, pp. 9–20, 1999.

[13] D. Nedospasov, J. Seifert, A. Schlösser, and S. Orlic, "Functional IC analysis," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2012.

[14] K. Nohl, D. Evans, S. Starbug, and H. Plötz, "Reverse-engineering a cryptographic RFID tag." in *USENIX Security Symposium*, vol. 28, 2008.

[15] "The layman's guide to IC reverse engineering," http://siliconzoo.org/tutorial.html, Silicon Zoo.

[16] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik *et al.*, "WordRev: Finding word-level structures in a sea of bit-level gates," in *IEEE international symposium on hardware-oriented security and trust (HOST)*, 2013, pp. 67–74.

[17] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *IEEE International Symposium on Hardware-oriented Security and Trust (HOST)*, 2012, pp. 83–88.

[18] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari *et al.*, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2013.

[19] A. Gascón, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanović, and S. Malik, "Template-based circuit understanding," in *Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2014, pp. 83–90.

[20] J. Baehr, A. Bernardini, G. Sigl, and U. Schlichtmann, "Machine learning and structural characteristics for reverse engineering," *Integration*, vol. 72, pp. 1–12, 2020.

[21] Y.-Y. Dai and R. K. Braytont, "Circuit recognition with deep learning," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017, pp. 162–162.

[22] A. Fayyazi, S. Shababi, P. Nuzzo, S. Nazarian, and M. Pedram, "Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 638–641.

[23] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. Sharma, W. Xu *et al.*, "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 55–60.

[24] "Delayer & parallel lapping," Akacia System. [Online]. Available: https://www.ma-tek.com/en-global/services/index/delayer-parallel-lapping

[25] "How to use SEM full-vision imaging technology to reverse-perceive nano-scale processing while avoid infringement?" Integrated Service Technology Inc., 2018. [Online]. Available: https://www.istgroup.com/en/tech_20181120/

[26] L. Alrahis, S. Patnaik, F. Khalid, M. A. Hanif, H. Saleh, M. Shafique *et al.*, "GNNUnlock: Graph neural networks-based oracle-less unlocking scheme for provably secure logic locking," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 780–785.

[27] 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 Transactions on Emerging Topics in Computing*, 2021.

[28] R. Yasaei, S.-Y. Yu, and M. A. Al Faruque, "GNN4TJ: Graph neural networks for hardware Trojan detection at register transfer level," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1504–1509.

[29] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. CONF, 2015.

[30] M. Hansen, H. Yalcin, and J. Hayes, "ISCAS high-level models." [Online]. Available: http://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html

[31] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[32] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Series*, vol. 2, no. 9, pp. 12–16, 1968.

[33] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1025–1035.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[35] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "GAAN: Gated attention networks for learning on large and spatiotemporal graphs," *arXiv preprint arXiv:1803.07294*, 2018.

[36] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *arXiv preprint arXiv:1806.03536*, 2018.

[37] L. Alrahis, S. Patnaik, J. Knechtel, H. Saleh, B. Mohammad, M. Al-Qutayri *et al.*, "*UNSAIL*: Thwarting oracle-less machine learning attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2508–2523, 2021.

[38] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graph-SAINT: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.

[39] K. Subramanian. (2019) Verilog-adders. [Online]. Available: https://github.com/mongrelgem/Verilog-Adders

[40] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *arXiv preprint arXiv:1801.07606*, 2018.

[41] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 3438–3445.

[42] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang *et al.*, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[43] B. Cakir and S. Malik, "Reverse engineering digital ICs through geometric embedding of circuit graphs," *TODAES*, vol. 23, no. 4, pp. 50:1–50:19, 2018.

[44] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm *et al.*, "HAL—the missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 498–510, 2019.

[45] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 51–57.

[46] J. Li and J. Lach, "At-speed delay characterization for IC authentication and Trojan horse detection," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 8–14.

[47] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic, "Power supply signal calibration techniques for improving detection resolution to hardware Trojans," in *IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 632–639.

[48] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, "Hardware Trojan horse detection using gate-level characterization," in *IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 688–693.

[49] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE Symposium on Security and Privacy*, 2010, pp. 159–172.

[50] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *IEEE Symposium on Security and Privacy*, 2011, pp. 49–63.

Lilas Alrahis is a Postdoctoral Associate at New York University Abu Dhabi. She received the M.Sc. degree and the Ph.D. degree in electrical and computer engineering from Khalifa University, UAE, in 2016 and 2020, respectively. 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 at the Applied Research Competition held in conjunction with Cyber Security Awareness Week, in 2019.

Abhrajit Sengupta is a Senior Engineer in Qualcomm Technologies, Inc. Prior to joining Qualcomm, he graduated from New York University with a PhD in Electrical Engineering. He holds an MS in Computer Science and Engineering from IIT Kharagpur and a BE in Computer Science and Engineering from Jadavpur University.

Abhrajit's research interest primarily lies in the areas of hardware security, specifically dealing with problems such as IP piracy, hardware Trojans and counterfeiting. Currently, his research focus is on logic locking and split manufacturing and developing leakage resilient hardware systems. Moreover, he is also interested in the much broader realms of foundational cryptographic problems.

Johann Knechtel received the M.Sc. degree in Information Systems Engineering (Dipl.-Ing.) and the Ph.D. degree in Computer Engineering (Dr.-Ing., summa cum laude) from TU Dresden, Germany, in 2010 and 2014, respectively.

He is a Research Scientist with New York University Abu Dhabi, United Arab Emirates. From 2015 to 2016, he was a Postdoctoral Researcher with the Masdar Institute of Science and Technology, Abu Dhabi; from 2010 to 2014, he was a Ph.D. Scholar with the DFG Graduate School "Nano- and Biotechnologies for Packaging of Electronic Systems" hosted at TU Dresden; in 2012, he was a Research Assistant with the Chinese University of Hong Kong; and in 2010, he was a Visiting Research Student with the University of Michigan at Ann Arbor, MI, USA. His research interests cover VLSI physical design automation, with particular focus on emerging technologies and hardware security. He has (co-)authored around 50 publications.

Satwik Patnaik received the B.E. degree in electronics and telecommunications from the University of Pune, India, the M.Tech. degree in computer science and engineering with a specialization in VLSI design from the Indian Institute of Information Technology and Management, Gwalior, India, and the Ph.D. degree in Electrical engineering from Tandon School of Engineering, New York University, Brooklyn, NY, USA in September 2020.

He is currently a Postdoctoral Researcher with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research delves into IP protection techniques, leveraging 3D paradigm for enhancing security, exploiting security properties of emerging devices, applied machine learning for hardware security, and side-channel evaluation. Dr. Patnaik received the Bronze Medal in the Graduate Category at the ACM/SIGDA Student Research Competition held at ICCAD 2018, and the Best Paper Award at the Applied Research Competition held in conjunction with Cyber Security Awareness Week, in 2017.

Hani Saleh is an associate professor of electronic engineering at Khalifa University since 2012. He is a co-founder/active researcher in the KSRC (Khalifa University Research Center) and the System on Chip Research Center (SOCC). Hani has a total of 19 years of industrial experience in ASIC chip design. Prior to academia, Hani worked for many leading chip design companies including Apple, Intel, AMD, Qualcomm, Synopsys, Fujitsu and Motorola. Hani received a Bachelor of Science degree in Electrical Engineering from the University of Jordan, a Master of Science degree in Electrical Engineering from the University of Texas at San Antonio, and a Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin. Hani's research interest includes IoT Devices design, deep learning, DSP algorithms design, computer.

Baker Mohammad received his Ph.D. from University of Texas at Austin, his M.S. degree from Arizona State University, Tempe, and BS degree from the University of New Mexico, Albuquerque, all in ECE. Baker is the director of System on Chip Center and a professor of ECE at Khalifa University. Prior to joining Khalifa University Baker was a Senior staff Engineer/Manager at Qualcomm, Austin, USA for 6-years, where he was engaged in designing high performance and low power DSP processor used for communication and multi-media application. Before joining Qualcomm he worked for 10 years at Intel Corporation on a wide range of micro-processors design from high performance, server chips $>$ 100 Watt (IA-64), to mobile embedded processor low power sub 1 watt (xscale). His research interest includes VLSI, power efficient computing, Design with emerging technology such as Memristor, STTRAM, In Memory-Computing, Efficient hardware accelerator for search engine, image processing, and Artificial Intelligent hardware.

Mahmoud Al-Qutayri received the B.Eng. degree from Concordia University, Canada, 1984, the M.Sc. degree from the University of Manchester, U.K., 1987, and the Ph.D. degree from the University of Bath, U.K., 1992 all in electrical and electronic engineering. He is currently a Full Professor with the Department of Electrical and Computer Engineering and the Associate Dean for Graduate Studies, College of Engineering at Khalifa University, UAE. Prior to joining Khalifa University, he worked at De Montfort University, UK and University of Bath, UK. Al-Qutayri current research interests include wireless sensor networks, embedded systems design, in-memory computing, mixed-signal integrated circuits design and test, and hardware security.

Ozgur Sinanoglu is a professor of electrical and computer engineering at New York University Abu Dhabi. He obtained his Ph.D. in Computer Science and Engineering from University of California San Diego. He has industry experience at TI, IBM and Qualcomm, and has been with NYU Abu Dhabi since 2010. During his Ph.D. he won the IBM Ph.D. fellowship award twice. He is also the recipient of the best paper awards at IEEE VLSI Test Symposium 2011 and ACM Conference on Computer and Communication Security 2013. Prof. Sinanoglu's 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. Prof. Sinanoglu is the director of the Center for CyberSecurity at NYU Abu Dhabi. His recent research in hardware security and trust is being funded by US National Science Foundation, US Department of Defense, Semiconductor Research Corporation, Intel Corp, and Mubadala Technology.